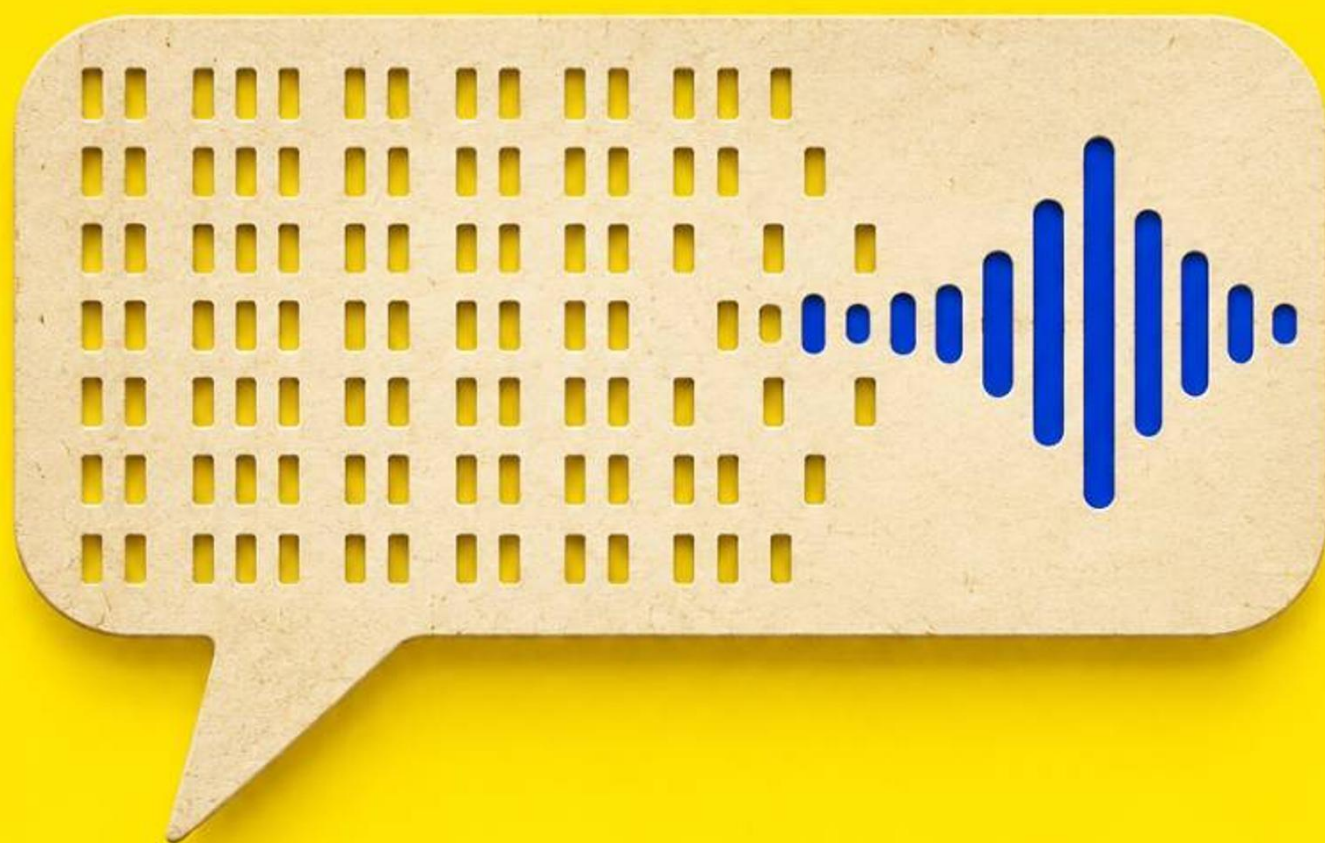


Струков Максим

ИИ.

**История машины,
которая научилась
говорить**



Максим Струков

**ИИ. История машины,
которая научилась говорить**

«Автор»

2026

Струков М.

ИИ. История машины, которая научилась говорить /
М. Струков — «Автор», 2026

Машина заговорила с нами так убедительно, что мы снова спорим, понимает ли она хоть что-нибудь. Но история искусственного интеллекта началась задолго до ChatGPT. От первых попыток описать мышление правилами — через шахматные матчи, «зимы ИИ» и прорыв нейросетей — книга ведёт к современным генеративным моделям. За каждым скачком стоят люди, данные, труд, вычислительные мощности и решения компаний. Как появились эти системы, почему они способны поражать и уверенно ошибаться и кто отвечает за последствия их работы? Документальная история ИИ, которая отделяет реальные достижения от громких обещаний.

Содержание

Вступление. Что мы называем искусственным интеллектом	8
Глава 1. До того, как машинам дали имя	11
Мечта о рассуждении без спора	12
Машины, которые считали	14
Разность между идеей и механизмом	14
Люди-компьютеры	17
Логика получила выключатель	20
Что можно вычислить?	21
Не всякая задача имеет рецепт	23
Нейрон как математический элемент	25
Машины, которые слушали ответ мира	27
Формула не выбирает задачу	29
Тьюринг меняет вопрос	31
Игра, в которой судят по ответам	31
Учиться вместо того, чтобы заранее знать всё	32
Нить, которая тянется дальше	34
Глава 2. Лето, которому дали имя	35
Восемнадцать страниц и один большой глагол	36
Четверо подписантов, четыре рабочих стола	37
Список задач, который оказался шире одной науки	39
Зачем Маккарти понадобилось новое название	41
У проекта были деньги, сроки и список приглашённых	43
Лето 1956 года не исполнило план целиком	45
Три программы — три способа проверить успех	47
Один сезон не мог вместить все эти вопросы	49
Гипотеза не была обещанием готового разума	51
Как превратить слово «интеллект» в исследовательскую задачу	53
Слово «симулировать» оставляло выбор за исследователем	55
Что именно началось в Дартмуте	56
Глава 3. Логика учится искать	58
Задача, у которой можно проверить каждый шаг	59
От шахматной доски к логическим формулам	60
Три участника, три разных вида работы	61
Сначала алгоритм исполнили вручную	62
Прежде чем машина могла искать, ей нужен был язык	63
Машина в комнате RAND	65
Знать правила — не значит знать, какой ход сделать	66
Как выглядит один доказанный результат	68
Британский музей и цена полного перебора	70
Четыре способа искать дорогу к теореме	72
Когда компьютер пропускает решение, а находит другое	73
Программа была и моделью человеческого решения	74
Что именно показали 38 доказательств	75
Сильная демонстрация в очень узком мире	76
После доказательства остаётся вопрос о разуме	77
От программы к исследовательской программе	78

Машина вывела формулу. Учёные должны были вывести границу	79
Глава 4. Нейрон без мистики	80
От доказательства к обучению	81
Как простая модель выбирает	82
Психолог в авиационной лаборатории	83
IBM 704 и стопка карточек	84
Маршрут сигнала внутри Mark I	85
Что значило обучаться без примеров от всего мира	86
Мозг в заголовке	87
Тобермори, который должен был услышать	88
Где проходит предел одной границы	89
Минский, Пейперт и математическая критика	90
Почему простота работала против обещаний	91
От нервной клетки к вычислительному элементу	92
Что именно меняется после ошибки	93
Демонстрация как эксперимент, а не чудо	94
Военные деньги и гражданская наука	95
Почему правила снова казались лучшим ответом	96
Восприятие как задача, а не метафора	97
Обещание и масштаб реального результата	98
Чему учит карточка	99
Розенблатт и жизнь исследования	100
От лабораторного времени к историческому	101
После карточки	102
Глава 5. Программа, которая умела слушать	103
ELIZA — это программа, DOCTOR — роль	104
Что программа делала с фразой	105
Почему психотерапевт был удачной ролью	106
Интерфейс был бумажным и общим	107
ELIZA среди ранних языковых программ	108
Правило под микроскопом	109
Что делал пользователь	110
Психотерапия — больше, чем набор вопросов	111
Имя, голос и старые социальные роли	112
Не один диалог, а множество версий	113
Почему люди продолжали говорить	114
От эксперимента к обещанию терапии	115
Джозеф Вейценбаум: инженер, ставший критиком	116
Кто решает, что можно автоматизировать	117
Почему ELIZA не была ранней языковой моделью	118
Личный разговор и публичное обещание	119
Не экзамен на человечность	120
Разговор становится частью машины	121
У терминала не было человеческого лица	122
Какой успех можно было считать успехом	123
Личный текст не всегда требует веры	124
От впечатления — к ответственности	125
Программа умела отвечать. Слушать — другое дело	126

Глава 6. Зима, которую нельзя объяснить одной причиной	127
Демонстрация становится обещанием	128
Переводчик в центре государственной задачи	129
Перевод — это процесс, а не кнопка	130
Что ALPAC написал — и чего не писал	131
Невозможность, неудача и неподходящая цена	132
У каждого результата свой метр	133
Комбинаторный взрыв: не один враг, а свойство масштаба	134
Британия искала собственный масштаб	135
Чистый мир и настоящая комната	136
Лайтхилл получает поручение	137
Буквы А, В и С	138
Ответы и публичная сцена	140
Американская переоценка шла своим путём	141
Лайтхилл не закрыл американские лаборатории	142
У слова «зима» нет единственной даты	143
Зима была неодинаковой	144
Когда меняется грант, меняется и биография	145
Наука и отчётность: конфликт, который никуда не исчез	146
Компьютеру не хватало не только скорости	147
Под одной вывеской скрывались разные дисциплины	148
Любая оценка описывает свою эпоху	149
Чего не произошло	150
Почему после зимы выбрали узкую задачу	151
Глава 7. Если эксперт — это правила	152
Химическая задача, которую нельзя было решить перебором	153
Знание как способ не искать всё подряд	154
Meta-DENDRAL: можно ли получить правило из примеров?	156
От молекул к инфекциям	157
Как работает правило, если показать его без жаргона	158
Неопределённость без притворства точности	160
«Почему вы спрашиваете?» — и «почему вы так решили?»	161
Проверка системы — не только сравнение ответов	162
Оболочка без медицинского содержимого	164
Что знает эксперт, когда не может объяснить	165
Когда исключение рождает новое исключение	166
XCON: когда правила попали в производственный процесс	167
В компании обнаружилось второе производство — производство правил	169
Когда язык правил становится отдельным рынком	170
Узкая система и большая тень за её пределами	171
Проверка правил становится работой исследователя	172
Что осталось после обещания заменить специалиста	173
Глава 8. Здравый смысл в коробке	175
До комнаты блоков: советчик, которому давали сведения	176
SHRDLU: разговор в комнате с цветными блоками	177
Что скрывалось за простой командой	179
Почему маленький мир так легко принять за большой	180
Что человек знает, не замечая	181

Фреймы: знакомая сцена с пустыми ячейками	182
«Обычно» не значит «всегда»	183
Проблема рамки: что изменилось, а что осталось	184
От базы фактов к карте понятий	185
Наивная физика: мир до учебника	186
Сус: проект, которому досталась повседневность	187
МСС: где повседневное знание стало долгим проектом	188
Что делать с противоречивым миром	189
Город, который нужно не только строить, но и содержать	190
Знание о людях тоже нужно формализовать	191
Проблема не в том, что люди знают слишком много	192
Чей именно здравый смысл?	193
Правила, примеры и предел полноты	194
Как проверить то, что по определению выходит за рамки	195
Неизвестно — не значит «нет»	196
От здравого смысла к языку	197
Глава 9. Язык, который не помещался в правила	198
Перевод становится государственной задачей	199
До электронных компьютеров: механическая идея Троянского	200
Уоррен Уивер предлагает читать язык как код	201
Что именно делала машина IBM 701	202
Демонстрация как тщательно поставленный опыт	203
Слово — не маленькая коробка со значением	204
Как предложение превращалось в программу	205
Прямая дорога, перевод через посредника и общий язык	206
Спор о том, что именно нужно знать машине	207
Советская линия: не сноска к американской демонстрации	208
Вычислительная машина не равна исследовательской программе	210
Почему язык не поддавался окончательному словарю	211
Перевод не останавливается на точке	212
Бар-Хиллел и «коробка в загоне»	213
Цена правила: кто переводит до и после компьютера	214
Узкая область — разумный инженерный компромисс	215
Один результат — несколько разных оценок	216
Оценка качества превращается в отдельную проблему	217
После большого разочарования осталась прикладная работа	218
Машинный перевод не закрылся вместе с финансированием	219
Язык потребовал разговора о понимании	220
Глава 10. Вторая зима не была повтором первой	221
Специализированный компьютер для новой профессии	222
XCON: полезная система, у которой появились обязанности	224
Конец ознакомительного фрагмента.	227

Максим Струков

ИИ. История машины, которая научилась говорить

Вступление. Что мы называем искусственным интеллектом

Слово «искусственный интеллект» звучит так, будто за ним стоит одна технология с ясными границами. В действительности это скорее имя для множества разных способов поручить машине работу, которую люди связывают с мышлением: искать ход в игре, распознавать речь, находить объекты на снимке, переводить фразу, рекомендовать фильм, продолжать текст. Одни системы следуют заранее заданным правилам, другие находят закономерности в примерах, третьи соединяют несколько методов и инструментов. Общее название удобно — и именно поэтому оно способно запутать.

Когда говорят «ИИ умеет», полезно остановиться и уточнить: какая система, какая задача, в каких условиях? Успех в одной области не переносится автоматически в другую. Шахматная программа может исследовать огромное число вариантов и играть сильнее чемпиона, не становясь от этого универсальным собеседником. Модель может составить связный абзац и одновременно ошибиться в простом факте. Автоматическое распознавание изображения может быть точным на тестовом наборе и ненадёжным при другом освещении, ракурсе или составе данных. Слово «интеллект» словно обещает единую шкалу способностей. Практика предлагает целый набор разных шкал — и спор о том, какие из них важны.

Эта книга начинается с различия между результатом и объяснением. Если машина выполнила задачу, мы знаем, что она выполнила именно эту задачу при этих условиях. Но что доказал успех? Что система нашла полезный способ решить ограниченную проблему? Что найденный метод переносится на новые ситуации? Что машина поняла смысл происходящего? Это не один и тот же вывод. Чем эффективнее результат, тем легче незаметно перейти от первого утверждения ко второму, а затем к третьему.

Вопрос не нов. В 1950 году Алан Тьюринг предложил обсуждать машинное мышление через игру, где судят по ответам, а не по недоступному наблюдателю внутреннему состоянию. Это был способ переформулировать трудный спор, не волшебный прибор для измерения разума. В 1956 году исследователи, собравшиеся на летний проект в Дартмуте, закрепили выражение «искусственный интеллект» как имя исследовательской программы. И Тьюринг, и Дартмут важны для этой истории. Но ни один эпизод не отменяет более ранних попыток описать рассуждение, вычисление и управление как процедуры — и не закрывает вопрос, что именно мы вправе заключить из поведения машины.

Нам хочется найти момент рождения. Так история получает понятное начало: несколько людей, одна лаборатория, прорыв, после которого будущее будто бы стало неизбежным. Но техника редко подчиняется такой драматургии. В ней сходятся идеи, которые долго развивались отдельно; устройства, которые наконец становятся достаточно быстрыми; данные, собранные для конкретных задач; деньги и институты, поддерживающие долгую работу; а также труд, который не всегда виден в рассказе об изобретателе. То, что позднее выглядит прямой дорогой, в моменте было выбором между несовершенными вариантами.

Поэтому мы будем возвращаться к одним и тем же вопросам. Какую задачу люди хотели решить? Как они описали её машине? Что именно считали успехом? Какие ресурсы и чья работа понадобились? Где система справлялась, а где её возможности заканчивались? И что

изменилось после того, как результат вышел из лаборатории и стал частью чужой повседневной жизни?

Эти вопросы помогут пройти через разные эпохи, не сваливая всё в одну историю непрерывного прогресса. В одни периоды исследователи пытались описать знания набором символов и правил. В другие ставка делалась на обучение по примерам. Позже важными стали большие массивы данных, более мощные вычисления и модели, которые находят статистические связи в тексте, изображениях и других материалах. Ни один подход не просто вытеснил предыдущие. Они по-разному решают задачи, требуют разных условий и терпят неудачу по разным причинам. История становится понятнее, если видеть не смену одного «разума» другим, а смену способов формулировать проблему и оплачивать её решение.

Здесь будут заметные победы: программа обыгрывает чемпиона, нейросеть резко улучшает распознавание изображений, новая архитектура меняет направление исследований, разговорный интерфейс делает генеративные модели привычными для широкой публики. Но победа — начало разбора, а не его конец. Надо спросить, что измерялось, на каком наборе задач и какой ценой достигнут результат. Система может быть выдающейся в узком смысле и при этом не обладать теми качествами, которые читатель невольно приписывает ей по одному удачному ответу. Признать достижение и очертить его границы — не взаимно исключающие жесты.

Эта точность нужна не только специалистам. Автоматизированные решения участвуют в отборе материалов, распределении внимания, работе организаций и доступе людей к услугам. Даже когда система ничего не «понимает» по-человечески, последствия её классификаций и рекомендаций вполне реальны. Вопрос о машине быстро становится вопросом о тех, кто поставил задачу, выбрал критерий, собрал данные, проверил результат и решил внедрить его. Ответственность не растворяется в модели. Она распределена между людьми и учреждениями — иногда так широко, что её трудно заметить.

Так же устроена и цена автоматизации. Модели и программы не возникают сами собой: их проектируют, обучают, оценивают, поддерживают и включают в конкретные продукты. За гладким результатом могут стоять вычислительные ресурсы, инженерные решения и большой объём человеческого труда — от подготовки данных до проверки опасных сбоев. История ИИ становится неполной, если в ней остаются только знаменитые авторы алгоритмов и исчезают все, кто делал систему работоспособной, полезной или хотя бы безопаснее.

Эта книга не собирается объявлять машину человеком на основании убедительной беседы. Но ответ «это всего лишь программа» тоже ничего не объясняет: программа может менять то, как распределяются работа, власть, деньги и внимание. Нам понадобится язык точнее двух привычных крайностей. Различать способность выполнить задачу, устойчивость результата, перенос на новые условия, понимание в человеческом смысле и сознательный опыт — значит не уклоняться от ответа, а выяснять, о каком именно ответе идёт речь.

Хронология книги доведена до **29 сентября 2026 года**; сведения о быстро меняющихся системах имеют смысл только вместе с датой и условиями, при которых они были верны. Это срез, а не финальная точка. Некоторые технические детали устареют, но вопросы о доказательствах, границах способностей и человеческих решениях останутся пригодны для следующей версии технологии.

Дальше мы начнём с более старой и более широкой истории, чем история компьютеров: с попыток вынести рассуждение наружу, представить его правилами, числами и механизмами. Затем увидим, как отдельные замыслы превратились в исследовательскую дисциплину, почему обещания регулярно опережали возможности и что изменилось, когда машины стали учиться на примерах и работать в масштабах, прежде недоступных. Путь приведёт к нынешним системам, но цель не в том, чтобы объявить их последним словом.

Цель — научиться читать их результаты без благоговения и без пренебрежения. Спросить, что они сделали, как это стало возможным и кому теперь приходится жить с последствиями. Возможно, тогда вопрос «думает ли машина?» станет не проще, зато станет точнее: мы сможем обсуждать не мифическую способность вообще, а конкретные доказательства, ограничения и выборы, из которых складывается искусственный интеллект.

Глава 1. До того, как машинам дали имя

В 1950 году Алан Тьюринг предложил читателям мысленный эксперимент. Представьте: вы задаёте вопросы, а ответы приходят только на бумаге. По ним нужно понять, кто собеседник — человек или машина. На первый взгляд, это игра про ловкость: сумеет ли устройство подобрать убедительные слова? Но за ней скрывался вопрос посложнее. Что именно мы называем мышлением — то, что происходит внутри, или работу, которую можно наблюдать?

Нам знакома эта трудность и без компьютеров. Мы судим о чужом мышлении по тому, что человек говорит и делает, хотя не можем напрямую заглянуть в его сознание. С машиной возникает соблазн применить двойной стандарт: потребовать от неё доказательств, которых мы не требуем от собеседника, или, наоборот, принять гладкий ответ за доказательство понимания. Тьюринг предложил не готовое решение этой дилеммы, а способ сделать её предметом разговора.

Чтобы ответить, Тьюринг не мог просто попросить машину пройти проверку: компьютера, способного вести такой разговор, ещё не существовало. Он опирался на десятилетия исследований и более длинную историю попыток описать разум через правила и действия. Люди научились разбирать отдельные выводы, рассчитывать таблицы по повторяемой процедуре, строить машины с программируемой последовательностью шагов, а затем — задавать вопросы о границах самого вычисления.

Это не история о том, как Аристотель «изобрёл ИИ», а Бэббидж построил компьютер раньше всех. Такие формулы звучат эффектно и стирают различия между идеями, которые возникали в разное время и решали разные задачи. Перед нами несколько дорог: делать выводы проверяемыми; снять с людей утомительные повторяющиеся расчёты; научиться управлять устройством по сигналам о том, что происходит вокруг; понять, какие задачи можно описать конечным набором правил. Ни одна дорога сама по себе не привела к искусственному интеллекту. Их пересечение изменило сам вопрос: какую часть умственной работы можно поручить машине?

По пути мы встретим не только философов и знаменитых изобретателей, но и людей, чью работу машина должна была заменить или продолжить: вычислителей, астрономов, инженеров и тех, кто собирал механизмы с невиданной точностью. Их истории помогают увидеть главное: программа никогда не возникает в пустоте. Кто-то выбирает задачу, кто-то описывает правила, кто-то собирает устройство, а кто-то решает, считать ли полученный результат успехом.

Почти в каждом эпизоде важно отделить замысел от результата: что авторы хотели построить, что действительно смогли запустить и какие выводы оправдывал эксперимент. Это простое различие убережёт нас от красивой истории, в которой проект задним числом превращается в готовое изобретение.

Мечта о рассуждении без спора

В повседневном разговоре мы редко останавливаемся, чтобы разобрать форму каждого вывода. Услышав «все люди смертны» и «Сократ — человек», мы без долгих вычислений приходим к заключению, что Сократ смертен. Но можно отделить содержание от формы: если все представители некоторого класса обладают свойством, а конкретный объект принадлежит этому классу, то свойство относится и к нему. В «Первой аналитике» Аристотель систематизировал виды умозаключений, включая силлогизмы, и тем самым дал рассуждению предмет, который можно было изучать отдельно от частного спора. Силлогизм — это вывод по заданной схеме: несколько исходных утверждений, посылок, ведут к заключению. В примере с Сократом посылки — «все люди смертны» и «Сократ — человек», а заключение — «Сократ смертен». Логика проверяет переход между ними, а не биографию Сократа и не истинность исходных утверждений.

Это был не замысел мыслящего автомата. Аристотель не проектировал вычислитель, а его логика не была чертежом компьютера. Но появилось важное различие: вывод иногда можно оценивать не по красноречию говорящего, а по устройству аргумента. Если принять посылки и правила, заключение следует или не следует из них. Логика не обещала, что принятые посылки истинны, что вопрос поставлен правильно или что человек выбрал важную тему. Она могла проверить форму перехода. Именно эта ограниченность и делала её полезной: рассуждение переставало быть полностью неуловимым.

Спустя много веков Раймунд Луллий искал способ систематизировать рассуждения о философских и религиозных вопросах. Его *Ars Magna* — «Великое искусство» — была не компьютерной программой и не нейтральной машиной, разрешающей любой спор. Она принадлежала богословскому миру Луллия и должна была помогать организовывать комбинации понятий и аргументов, в том числе в межрелигиозной полемике. Круги, таблицы и схемы позволяли сочетать категории по заданным правилам. Для современного взгляда здесь заметен знакомый жест: сложную область пытаются представить как конечное число элементов, отношения между которыми можно исследовать систематически.

Но инструмент не становился от этого объективным судьёй. Результат зависел от того, какие понятия были включены, как они определялись и какие выводы автор считал допустимыми. Луллий не устранял убеждения из рассуждения; он строил процедуру внутри собственной системы убеждений. Это различие будет преследовать историю ИИ: вычисление может быть строгим, даже если выбор задачи и исходных предпосылок остаётся человеческим.

В XVII веке Готфрид Вильгельм Лейбниц мечтал о формальном языке, в котором сложные идеи можно было бы выражать ясными знаками, а выводы — получать по правилам. В его проектах переплетались несколько разных замыслов: универсальная характеристика, которая точнее представляла бы понятия, и исчисление рассуждений, способное выполнять операции над ними. В современной популярной версии это часто превращается в одну эффектную цитату: «Давайте посчитаем». Её передают как предсказание компьютера или обещание, что любая человеческая ссора однажды закончится результатом на экране. Но такой пересказ сжимает программу Лейбница до лозунга. Он стремился сделать знания и аргументы достаточно ясными для формального анализа, а не утверждал, что всякая истина, ценность или человеческий конфликт сводятся к арифметике.

Тем не менее в этой мечте уже содержался переход от спора к операции. Если две стороны используют один и тот же язык, если понятия определены, если правила вывода приняты, то часть разногласий можно разбирать шаг за шагом. Но слово «если» здесь несёт почти весь вес конструкции. Реальные споры редко начинаются с общих определений; люди спорят о том,

какие факты важны, кому доверять и какую цену считать приемлемой. Формальная процедура может помочь обнаружить противоречие, но сама по себе не выбирает справедливую цель.

И всё же идея оказалась живучей. Сначала рассуждение описывают как систему отношений между знаками. Затем возникает соблазн поручить работу над знаками устройству. Чтобы понять, как этот переход стал инженерной задачей, нужно на время отложить философов и посмотреть на таблицы чисел.

Машины, которые считали

До электронных компьютеров слово «компьютер» обозначало прежде всего человека, который вычисляет. В учреждениях и научных проектах работа делилась между сотрудниками: одни выполняли отдельные операции, другие переносили числа в таблицы, третьи проверяли результат. Так рассчитывали астрономические и навигационные таблицы, нужные морякам, инженерам и наблюдателям неба. Число могло пройти длинную цепочку рук, прежде чем стать готовой строкой в справочнике.

Таблица не выглядела интеллектуальным триумфом. Она была инфраструктурой, незаметной, пока всё сходилось, и опасной, когда в вычислении закрадывалась ошибка. Утомление человека — не единственная причина ошибок: могла сбиться исходная запись, перепутаться знак, потеряться перенос разряда, не сработать проверка. Исправить одну неверную величину означало иногда перепроверить целую последовательность. Расчёт становился не только математической, но и организационной задачей: как разделить работу, сделать её повторяемой и заметить сбой до того, как он попадёт в печатный справочник?

Чарльз Бэббидж хорошо знал, сколько человеческого труда требовали вычислительные таблицы. Британский математик и инженер начал с проекта разностной машины для автоматизации повторяющихся расчётов, а затем задумал аналитическую машину, которой можно было бы задавать разные последовательности операций. Первая должна была механизировать определённый класс вычислений; вторая открывала куда более широкую возможность — менять программу, не перестраивая весь механизм.

Разность между идеей и механизмом

Чтобы понять, что именно обещала разностная машина, полезно увидеть её принцип на небольшом примере. Возьмём квадраты целых чисел: 1, 4, 9, 16, 25. Если вычитать каждое число из следующего, получатся первые разности: 3, 5, 7, 9. Разности между этими разностями равны 2, 2, 2. Для такого ряда после начальных значений можно продолжать вычисление повторяющимся сложением: следующая разность увеличивается на 2, а следующее число — на новую разность. Механизм не обязан каждый раз заново вычислять квадрат; он использует структуру ряда и выполняет однотипные операции.

В этом и заключалась практическая привлекательность метода конечных разностей: сложную таблицу можно было получать серией простых шагов. Но у метода есть границы. Разностная машина не решала любую математическую задачу; она была задумана для определённых видов последовательностей и могла автоматизировать вычисления, если требуемую процедуру удавалось привести к подходящей форме. Это не обесценивает проект. Напротив, ограничение показывает, что инженерный замысел включал конкретный выбор задач. Машину проектируют не для абстрактного «интеллекта», а для определённой работы с её условиями, точностью и приемлемой стоимостью.

Задуманный способ печати тоже имел значение. Если вычисленные цифры выводятся на бумагу напрямую, уменьшается риск, что человек переписет их с ошибкой. Но автоматическая печать не делает безошибочным весь процесс. Неверная таблица исходных значений, ошибка в проектировании или неисправная деталь способны дать неверный результат. Автоматизация устраняет определённый класс ошибок и создаёт другие места отказа. Это наблюдение справедливо и для современных систем: результат может быть произведён без ручного ввода на последнем шаге, но цепочка до этого результата всё равно состоит из решений и предположений.

Строительство машины оказалось труднее, чем её описание в виде последовательности операций. Бэббидж работал с мастером-инструментальщиком Джозефом Клементом, от которого требовались детали необычайной точности. Работа над разностной машиной остановилась в начале 1830-х после конфликта между ними, а проект оказался связан с расходами, государственным финансированием и вопросом о том, кто контролирует мастерскую и инструменты. Здесь легко написать драму о гении, которого не поняли чиновники. Реальность была менее аккуратной: проект зависел от сложного сотрудничества изобретателя, квалифицированного мастера и покровителей, причём у каждого были свои условия и интересы. Технология существует не отдельно от трудовых отношений и денег.

В 1834 году, после приостановки строительства разностной машины, Бэббидж задумал аналитическую — гораздо более амбициозную. Это не была просто следующая модель с большим числом функций. Разностная машина исполняла заранее ограниченную последовательность арифметических операций; аналитическая должна была получать инструкцию через карты и менять последовательность действий. В ней различались «склад», где удерживались числа, и «мельница», где выполнялись вычисления. Эти названия не нужно воспринимать как буквальные аналоги современной памяти и процессора во всех деталях. Это способ представить принципиально важное разделение: данные хранятся отдельно от механизма, который их обрабатывает.

Связь с жаккардовым станком делает программируемость наглядной. Карточки с отверстиями направляли станок так, чтобы тот воспроизводил выбранный узор; последовательность карт организовывала работу механизма. Бэббидж перенёс этот принцип в проект вычислительной машины. Карта задавала не декоративный рисунок, а операцию или порядок операций. Аналогия Лавлейс — аналитическая машина «ткёт алгебраические узоры» так же, как станок ткёт цветы и листья — была не просто красивой метафорой. Она описывала программу как структуру, управляющую работой универсального механизма.

При этом перфокарта не обладала смыслом сама по себе. Машина не могла догадаться, что отверстия обозначают «сложи эти числа»; смысл задавался соответствием между формой записи, устройством и правилами исполнения. Программа — это не только последовательность символов. Нужна машина, способная трактовать их предсказуемым образом. В этом месте механическая история Бэббиджа встречается с более поздними формальными теориями вычисления: процедуру можно отделить от результата, но только если существует исполнитель, способный следовать ей.

Аналитическую машину не построили в полном виде в XIX веке. В Музее науки Лондона хранится фрагмент механизма, который был изготовлен в мастерской Бэббиджа; это материальное свидетельство работы, но не готовый универсальный вычислитель. В XX и XXI веках исследователи и музейные инженеры смогли построить работающие разностные машины по чертежам Бэббиджа. Это подтвердило инженерную возможность некоторых его проектов, но не следует переносить такую проверку на аналитическую машину: её полный проект оказался куда сложнее и не был воплощён тем же способом. Исторический вес идеи не зависит от того, работал ли в Лондоне 1840-х годов полностью собранный механизм. Однако читатель заслуживает знать разницу между тем, что было построено, тем, что осталось чертежом, и тем, что позднее реконструировали.

Бэббиджевская история показывает, что программируемость появилась не из одной удачной идеи. Её потребовали математический метод, точное производство, управление последовательностью операций и представление данных в физической форме. Если убрать любой из этих слоёв, устройство теряет универсальность или надёжность. Машина на бумаге может быть бесконечно точной; машина в мастерской должна пережить трение, люфт, стоимость металла и спор людей, которые её строят.

С машиной связан и другой часто упрощаемый сюжет — история Ады Лавлейс. В 1843 году она перевела статью итальянского инженера Луиджи Менабреа об аналитической машине и добавила к переводу собственные примечания, по объёму и содержанию значительно превосходившие исходный текст. В одном из них Лавлейс подробно описала операции, необходимые для вычисления чисел Бернулли. Поэтому её часто называют первым программистом. Само звание зависит от того, что именно мы считаем программой и как распределяем авторство в совместной работе; оно не должно превращать историю в соревнование за единственную табличку «первый».

Гораздо важнее, что Лавлейс поняла различие между устройством, которое выполняет операции, и тем, что определяет их порядок. В её примечаниях аналитическая машина предстаёт не просто калькулятором: если задать соответствующие правила, она могла бы работать с символами, связанными не только с числами; Лавлейс даже рассуждала о возможности манипулировать музыкальными отношениями при условии, что их можно будет формализовать. И одновременно она проводила границу: машина делает то, что мы знаем, как предписать ей делать. Она не создаёт собственные правила и не выбирает цель по собственной воле.

Эту позицию позднее стали называть «возражением Лавлейс», хотя её исходный контекст тоньше, чем популярная формула «машины не способны творить». Лавлейс говорила о возможностях конкретного проекта и о том, что машина не производит новизну сама по себе; её замечания не были экспериментальным доказательством невозможности всякого машинного творчества. Но вопрос она поставила точно: откуда берётся то, что устройство исполняет? В перфокартах, алгоритме, исходных данных, замысле человека — или в самой машине? Ответ будет меняться вместе с технологиями, но вопрос не исчезнет.

Таблицы, Бэббидж и Лавлейс помогают увидеть раннюю форму спора о машинном разуме. Автоматизация может быть впечатляющей без понимания задачи. Машина может выполнить расчёт, который человеку занял бы много времени, но способность получить результат не сообщает, что именно результат означает и нужен ли он. Уже в XIX веке появились две мысли, которые современная реклама ИИ регулярно смешивает: устройство способно производить сложный результат; следовательно, устройство обладает человеческим пониманием. Между этими утверждениями нет автоматического перехода.

Люди-компьютеры

Когда слово «компьютер» звучало как название профессии, вычисление было не только действием одного человека над листом бумаги. Это была организованная работа, связанная с расписанием, обучением и проверкой. Большие расчёты разбивали на части, чтобы их можно было распределить между сотрудниками. Такая организация позволяла выполнять больше вычислений, но создавала собственные риски: результат одного человека становился исходными данными для следующего, а ошибка могла незаметно перейти через всю цепочку.

Чтобы такая цепочка не превращалась в игру «испорченный телефон» с числами, работу нужно было организовать. Один человек считал отдельный фрагмент, другой переносил значения в сводную таблицу, третий проверял расчёт независимым способом. При сбое приходилось выяснять, на каком именно шаге он возник: ошибка в исходном наблюдении и ошибка при переписывании требовали разного исправления. Поэтому вычислительный труд включал не только арифметику, но и процедуру контроля — ту самую невидимую часть, без которой готовая таблица не заслуживала доверия.

Человеческая команда не была просто медленной версией будущей машины. Люди могли заметить странный результат, восстановить пропущенный шаг, изменить способ вычисления, если задача оказалась устроена не так, как предполагалось. Они понимали, зачем нужна таблица, для кого она предназначена и какие отклонения выглядят подозрительно. Машинная автоматизация могла ускорить однотипную операцию, но не автоматически переносила весь этот контекст. Если механизм повторял ошибочное действие быстрее человека, ошибка не становилась менее ошибочной — она становилась производительнее.

Такое перераспределение труда будет повторяться всю историю вычислений. Электронные компьютеры автоматизируют расчёты, но потребуют операторов и программистов. Системы распознавания изображений смогут обрабатывать массивы снимков, но сначала понадобятся примеры, которые люди отобрали и подписали. Модель, которая отвечает на вопрос, может скрывать за гладкой репликой труд по сбору данных, настройке и оценке результатов. Поэтому история ИИ — не последовательный марш машин, которые вытесняют людей. Это история изменения границы между человеческой работой и работой устройства, где часть прежних операций исчезает, часть перемещается, а часть появляется только потому, что система существует.

Переход от людей к устройству был связан и с тем, что значит «задать задачу». Человек в вычислительной команде нередко получал инструкцию, но при этом мог применять понимание к исключениям. Механизм должен был получить процедуру, достаточно точную, чтобы не нуждаться в подобном импровизированном суждении. Чем больше исключений, тем труднее формализовать задачу. Это не просто недостаток старой техники: это первая версия проблемы, с которой исследователи столкнутся, когда захотят научить машину читать текст или видеть сцену. Операция над числами легко задаётся, если порядок шагов известен; распознавание важного в незнакомой ситуации устроено гораздо менее прозрачно.

В конце XIX века эта профессия обрела особенно наглядную форму в обсерватории Гарвардского колледжа. Там фотографические стеклянные пластины хранили снимки звёздного неба, которые требовалось сравнивать, каталогизировать и измерять. С 1881 года обсерватория нанимала женщин для работы с растущей коллекцией: среди них были Анна Уинлок, Селина Бонд, Нетти Фаррар и Уильямина Флеминг. Их должность называлась «компьютер» — и это было профессиональное название, не шутка о человеке с калькулятором.

Работа с пластинами требовала больше, чем переписать несколько чисел. По снимкам определяли положение и яркость звёзд, изучали спектры, искали переменные звёзды и готовили каталоги. Для современной аудитории полезно помнить: телескоп мог собрать изобра-

жения ночного неба, но сами изображения не превращались автоматически в астрономические знания. Нужны были люди, которые научились читать пластины, сравнивать наблюдения и превращать видимые следы света в последовательные записи. Именно такая работа помогала стандартизировать наблюдения и делать небесные объекты сопоставимыми для разных исследователей.

Поиск переменной звезды хорошо показывает, почему одной фотографии было недостаточно. Исследователь сравнивал снимки одного участка неба, сделанные в разное время, и искал объекты, чья яркость менялась. След света на пластине приходилось измерять и сопоставлять с другими звёздами и наблюдениями. Речь шла не о том, чтобы взглянуть на снимок и «увидеть ответ»: нужно было превратить множество отдельных изображений в последовательность данных, где изменения можно заметить, проверить и обсудить.

Эти сотрудники не составляли однородную группу. У некоторых была подготовка по математике и астрономии, другие выполняли расчёты и поддерживали исследовательскую инфраструктуру; отдельные женщины вели собственные исследования и получили известность как астрономы. Гарвардская коллекция связывает с этой работой имена Флеминг, Энни Джамп Кэннон, Антонии Мори, Генриетты Ливитт и Сесилии Пейн-Гапошкиной. Называть всех их одинаково «помощницами» означало бы повторить иерархию, которая определяла их положение в обсерватории, а не полностью описывала их вклад. Особенно наглядно это видно в карьере Флеминг. Она приехала из Шотландии в США и сначала работала у директора обсерватории Эдварда Пикеринга; в 1881 году он пригласил её в группу вычислителей. За следующие десятилетия Флеминг каталогизировала более десяти тысяч звёзд, обнаружила 310 переменных звёзд и 52 туманности. Среди них была туманность Конская Голова, распознанная на фотопластине. В 1899 году Флеминг стала первым куратором астрономических фотографий в Гарварде. Один и тот же труд соединял повторяемые измерения, руководство большой коллекцией и научные открытия. В 1890 году вышел первый каталог звёздных спектров, подготовленный в рамках мемориального проекта Генри Дрейпера. Директор Эдвард Пикеринг указал, что Флеминг выполнила основную часть измерений и классификации и подготовила материалы к публикации. Она не только анализировала пластины, но и организовывала работу коллег, редактировала научные публикации и обучала новых сотрудников. Научное открытие здесь не отделено от кропотливой инфраструктуры: каталог становился возможен благодаря сотням измерений, единой системе записи и людям, которые удерживали порядок в огромном массиве наблюдений. Стеклопластина сохраняла исходное наблюдение, а не только число, уже внесённое в каталог. Позднее к ней можно было вернуться, заново измерить изображение или сопоставить его с другим снимком. Для науки это важное различие: таблица хранит прежний вывод, а архив оставляет возможность перепроверить данные и задать новый вопрос к старому материалу. Сохранение и учёт такой коллекции тоже были частью исследовательской работы.

При этом доступ к профессии не означал равенства. Женщин нанимали в том числе потому, что руководство считало их пригодными для внимательной, повторяющейся работы и могло платить им меньше, чем мужчинам. В дневнике 1900 года Флеминг сравнивала свою зарплату в 1 500 долларов с 2 500 долларами, которые получали мужчины за сопоставимую работу, и писала о необходимости содержать дом и семью. Женщинам также были ограничены возможности наблюдать через телескоп в Кембридже, хотя конкретные роли и возможности различались. Это не украшение истории вычислений социальной драмой. Это объяснение того, как научная инфраструктура распределяла доступ к инструментам, оплате и признанию.

История Гарварда показывает, что человеческий компьютер мог заниматься не только механической арифметикой. В центре работы находились измерение, суждение и классификация — задачи, где опыт позволял отличить полезное наблюдение от ошибки или сомнительного следа. Когда позднее вычислительные машины возьмут на себя расчёты, они не унасле-

дуют автоматически этот опыт. Его придётся превратить в правила, данные или процедуры, а затем проверить, какие части человеческого решения были действительно формализованы.

Логика получила выключатель

В аналитической машине числа и операции были разделены конструкцией механизма. Но чтобы вообще поручить машине логический вывод, требовалось ещё одно преобразование: абстрактные знаки должны были получить физическое представление. На бумаге «истина» и «ложь» — слова или символы. В электрической цепи можно различить два состояния: ток проходит или не проходит. Само по себе это сходство ничего не доказывает о разуме. Зато оно даёт инженеру способ построить устройство, которое будет переключаться по заданным правилам.

В середине XIX века Джордж Буль исследовал логику с помощью алгебраического языка. Его работа не описывала электрические компьютеры, которых ещё не существовало; она показывала, что отношения между логическими высказываниями можно выражать формально и преобразовывать. Позже, в 1930-х годах, Клод Шеннон увидел, что такой язык подходит для анализа схем реле и переключателей. В магистерской работе 1937 года он связал булеву алгебру с устройством релейных цепей. То, что раньше было способом рассуждать о высказываниях, стало практическим инструментом проектирования электрической системы.

Представим простую схему управления светом: кнопка замыкает цепь, лампа загорается; размыкает — гаснет. Это ещё не логический процессор. Но если объединить несколько контактов, можно построить условия типа «оба переключателя должны быть включены» или «достаточно одного». В языке логики это знакомые операции «и» и «или». Реле при этом ничего не знают о значении слов «оба» и «достаточно». Они переключают ток. Человек задаёт соответствие между состояниями цепи и тем, что система должна считать входом, условием или результатом.

Здесь срастаются два уровня. Булева алгебра помогает определить, какие состояния должны быть связаны; инженерная схема реализует переход физически. При сложном проектировании алгебра позволяет упрощать цепи и находить эквивалентные конфигурации. Ошибка на одном уровне может проявиться на другом: неверно записанное условие останется неверным, даже если все реле работают безупречно; плохо собранная схема не исполнит правильную формулу. Надёжность вычисления всегда зависит от того, как связаны описание и устройство.

Эта история важна, потому что компьютер не появился в результате одного философского прозрения. Между мечтой превратить рассуждение в процедуру и машиной, способной исполнять программу, стояла длинная работа по передаче состояний: через механические детали, реле, перфокарты, электрические цепи и позднее электронные компоненты. ИИ часто описывают так, будто алгоритм — это вся система. На деле алгоритм не включается сам по себе. Ему нужно физическое устройство, способное сохранять и менять состояние, и люди, которые решат, какие состояния что означают.

Булева алгебра не сделала релейные схемы разумными. Она сделала их вычислимой конструкцией: поведение цепи можно было проектировать заранее, описывать математически и проверять по условиям. Это был важный шаг, но он оставлял следующий вопрос открытым. Если машина может выполнять заданный набор правил, может ли она менять поведение по результатам своих действий?

Что можно вычислить?

В 1930-е годы математики и логики обсуждали не только то, как выполнить конкретный расчёт, но и то, какие задачи вообще имеют решение в виде чёткой процедуры. Можно ли сформулировать метод, который за конечное число ясных шагов найдёт ответ? Существуют ли математические вопросы, для которых такого метода не может быть? Чтобы обсуждать эти границы не на уровне интуиции, требовалась модель вычисления.

В работе «О вычислимых числах», опубликованной в 1936–1937 годах, Алан Тьюринг описал абстрактное устройство, которое стало известно как машина Тьюринга. Это не был проект металлического компьютера и не попытка воспроизвести устройство мозга. Модель состояла из простых элементов: ленты, разделённой на ячейки, символов, которые можно читать и записывать, и набора правил, определяющих, что сделать дальше. Машина может передвигаться по ленте, изменить символ, перейти в другое состояние или остановиться.

Сила этой модели в предельной простоте. Чтобы представить сложное вычисление, не нужно воображать хитрый механизм с тысячами шестерёнок. Достаточно выяснить, можно ли разложить задачу на маленькие однозначные операции, которые выполняются по конечным правилам. Лента служит памятью; состояние и таблица правил определяют следующий шаг. Это очень далеко от человеческого мышления: модель не видит, не слышит и не понимает, о чём говорят символы. Она помогает ответить на более узкий вопрос — что считать механическим вычислением.

Можно представить крошечную задачу на такой ленте. Пусть число записано палочками: три палочки означают три. Чтобы прибавить одну, машина должна найти конец ряда, поставить ещё одну палочку и остановиться. Для нас это почти смешно просто. Для теории важен не масштаб задачи, а возможность точно сказать, что машина видит, какое правило применяет и когда прекращает работу. Если шаг нельзя сформулировать без слов «ну, дальше понятно», значит, процедуру ещё не удалось полностью описать.

Тьюринг работал над математической проблемой разрешимости, а не над «искусственным интеллектом»: этого названия ещё не было. Но его анализ сделал важное различие возможным. Одно дело — задача, для которой можно в принципе описать процедуру. Другое — конкретная машина, на которой эту процедуру можно выполнить с нужной скоростью и объёмом памяти. И третье — вопрос о том, обладает ли исполнитель человеческим пониманием. Эти три уровня часто смешивают, когда говорят: «компьютер решил задачу». Математическая вычислимость не гарантирует практическую выполнимость; успешное выполнение не доказывает наличия опыта или сознания.

Модель Тьюринга также помогает понять, почему слово «программа» оказалось столь мощным. Если машина может выполнять простые операции, а последовательность операций хранится в форме, которую сама машина может читать, одно устройство способно решать разные задачи — в зависимости от того, какую программу ему дали. Аналитическая машина Бэббиджа предвосхитила эту идею в механическом проекте, но у Тьюринга была формальная модель, созданная для анализа границ вычисления. Они не одно и то же и не образуют прямую техническую родословную «от чертежа Бэббиджа к современному ноутбуку». Они показывают разные стороны одного сдвига: операция может быть отделена от конкретной задачи и описана достаточно точно, чтобы исполнитель следовал ей шаг за шагом.

Война резко повысила практическую ценность вычисления. Государствам требовались расчёты и обработка информации в масштабах, недоступных отдельному человеку; военные проекты ускорили развитие электронных устройств, криптоанализа, радиолокации и инженерных методов. Тьюринг участвовал в британской работе по криптоанализу в Блетчли-парке, а его теоретические исследования вычислимости были отдельной линией его карьеры. Важно не

превращать эти сюжеты в упрощённую легенду, будто одна военная задача «изобрела компьютер» или сразу породила ИИ. Война обеспечила ресурсы и давление, которые помогали строить вычислительную инфраструктуру; математические вопросы, инженерные конструкции и государственные цели пересекались, но не сливались в одну причину.

К 1940-м годам стало возможно говорить о вычислении одновременно как о математической процедуре, работе человеческих групп и работе электронного устройства. Машина уже не была только философской метафорой. Её можно было проектировать, собирать и запускать под конкретные задачи. Но вопрос, может ли она мыслить, оставался открытым — и, возможно, именно поэтому Тьюринг позднее попробовал изменить его форму.

Не всякая задача имеет рецепт

Теория вычисления нужна не только для того, чтобы показать, что машина может выполнить задачу. Она помогает увидеть и предел: некоторые вопросы нельзя решить общим алгоритмом, который всегда выдаёт правильный ответ. Это не означает, что любую трудную задачу следует объявить неразрешимой, стоит лишь компьютеру задуматься подольше. Сложность и невозможность — разные вещи. Первое означает, что решение может требовать слишком много времени или памяти; второе — что определённого универсального метода в принципе не существует.

Разницу можно представить на примере огромного архива. Если нужное письмо лежит где-то в неупорядоченной стопке, мы можем перебирать листы один за другим: способ есть, но поиск способен занять неприемлемо долго. Если же сам вопрос сформулирован так, что никакое правило не сможет всегда давать правильный ответ для всех возможных случаев, дополнительная скорость не поможет. Теория вычисления изучает обе границы, но не смешивает их: долгий поиск — ещё не невозможность, а невозможность — не просто очень долгий поиск.

Один из результатов Тьюринга касался проблемы остановки. Можно ли написать процедуру, которая для любой программы и любых её входных данных заранее определит, завершится ли выполнение или будет продолжаться бесконечно? Тьюринг показал, что общего решателя такого рода не может быть. Это не хитрая техническая оговорка, интересная лишь математикам. Она ломает наивное представление, будто достаточно дать машине достаточно времени и мощности, и любая задача превратится в ответ. У вычисления есть границы, которые нельзя преодолеть простым увеличением скорости.

Пример можно представить так: мы хотим создать универсального проверяющего, который читает любую инструкцию и любой набор исходных данных, после чего безошибочно говорит, остановится ли исполнение. Если сам проверяющий получает описание программы, в которое встроен вопрос о результате его собственного анализа, возникает противоречие: что бы проверяющий ни предсказал для специально сконструированной программы, её можно определить так, чтобы она поступила наоборот. Детали доказательства формальны, но идея знакома: некоторые обещания самоссылки рушатся, когда система претендует предсказывать поведение абсолютно всех систем своего класса, включая собственное.

Эта граница не отвечает на вопрос, способен ли компьютер мыслить. Она отвечает на более узкий вопрос о том, какие процедуры могут быть универсально реализованы. Иногда разговор о машинном интеллекте перескакивает от «мы пока не умеем решить» к «это невозможно в принципе» или в обратную сторону — от «машина уже это делает» к «значит, все барьеры исчезли». Теория вычисления удерживает разговор от обоих скачков. Она учит различать то, что можно выразить в виде алгоритма, то, что можно практически посчитать, и то, что вообще означает получить осмысленный ответ.

Универсальная машина Тьюринга — тоже математическая конструкция, но она показывает обратную сторону предела. Если описание другой машины и входные данные представить в подходящем виде, одна универсальная машина может имитировать работу этой машины. В этом смысле не обязательно строить отдельное железо для каждой вычислимой задачи: можно менять описание операций. Это не означает, что любая задача доступна одной машине в любой момент или с приемлемыми затратами. Модель говорит о принципе универсальности, а не о скорости и экономике реальных вычислительных систем.

Связь с аналитической машиной Бэббиджа здесь соблазнительна: оба проекта подталкивают к идее устройства, способного исполнять разные программы. Но Тьюринг не просто переписал чертежи Бэббиджа в математических терминах. Он ставил иной вопрос — что считать

эффективной процедурой и каковы её границы. Историческое сходство не отменяет разницу между механическим проектом XIX века и формальной моделью вычисления XX века.

Когда позднее компьютер запускает программу, пользователь видит результат, а не доказательство, что задача была вычислима в требуемом смысле. Когда система отвечает связным текстом, он может увидеть нечто похожее на суждение, но за внешним поведением стоят конкретные процедуры. Теория вычисления не говорит, что значение результата исчерпывается его алгоритмом. Она позволяет точнее спросить, что именно делает машина, и не спутать выполнимость с пониманием.

Нейрон как математический элемент

Параллельно с исследованиями логики и вычисления возникал другой способ говорить о мышлении: через нервную систему. Человеческий мозг не похож на таблицу правил, аккуратно записанную в книге. В нём есть миллиарды клеток, которые связаны между собой, получают сигналы и меняют активность. Для математика и нейрофизиолога это была возможность задать вопрос иначе: можно ли описать хотя бы некоторые стороны нервной активности через простые элементы и связи между ними?

В 1943 году Уоррен Маккаллоу, психиатр и нейрофизиолог, и математик Уолтер Питтс опубликовали статью, в которой предложили формальную модель нервной активности. Их искусственный нейрон был идеализацией: он получал сигналы от других элементов и переходил в активное состояние, если выполнялось заданное условие. Сеть таких элементов можно было описывать с помощью логических операций. Авторы исследовали связь между нейронной активностью и вычислением, а не пытались нарисовать точный портрет одной живой клетки.

Чтобы почувствовать идею, представьте охранника, который открывает дверь, только если сработали два датчика, или если сработал третий, независимо от остальных. Такой механизм можно описать логическим правилом. Теперь замените охранника простым вычислительным элементом, а датчики — входными сигналами. Сеть элементов способна реализовать комбинации условий. Но аналогия заканчивается там, где начинается биология: настоящие нейроны не являются идеальными переключателями, а мозг — не аккуратный логический шкаф, в котором каждый элемент выполняет отдельное «и» или «или». Модель оставляет за бортом огромное количество деталей, чтобы исследовать ограниченный вопрос.

В этом заключается и ценность упрощения, и его опасность. Если модель претендует объяснить всё мышление, её простота превращается в обещание без основания. Если она заявляет более скромную цель — проверить, какие виды вычисления возможны в сети идеализированных элементов, — её можно оценивать по тому, что она действительно описывает. Математическая модель не должна быть копией мозга, чтобы оказаться полезной. Но сходство слов «нейрон» и «нейронная сеть» не доказывает, что современная модель учится тем же способом, что человек или животное.

Если выход одного элемента подключить ко входу другого, сигнал может пройти через несколько шагов и изменить общий результат сети. Но это не значит, что сеть сама научилась нужному поведению: в модели 1943 года устройство связей и правила перехода задавались заранее. Чтобы система менялась на примерах, требовалось отдельно описать механизм обучения. Такое различие легко упустить, услышав слово «нейрон»: биологическая метафора звучит близко, но за ней стояла строгая и очень ограниченная математическая конструкция.

Работа Маккаллоу и Питтса не описывала обучение по примерам в том смысле, который позднее станет центральным для машинного обучения. Элементы и связи задавались в формальной конструкции. Чтобы система меняла свои параметры на основании опыта, требовалось придумать отдельный механизм изменения. Позже исследователи будут разрабатывать перцептроны и способы обучения, но между логической сетью 1943 года и практическими обучаемыми моделями пройдут годы новых идей, споров и аппаратных ограничений.

Тем не менее в этой работе встретились два языка: язык логики и язык нервной активности. Благодаря этому возникла продуктивная возможность: рассматривать сеть простых элементов как систему, которая может выполнять вычисления. Это не означает, что мозг свели к машине; скорее, машина стала способом проверять предположения о мозге. Вместе с работами о формальном вычислении возник ещё один путь к будущему ИИ — не кодировать каждый вывод вручную, а создавать систему из компонентов, чьё совместное поведение будет иметь интересные свойства.

Здесь уже видны две конкурирующие интуиции, которые будут возвращаться десятилетиями. Первая: чтобы машина действовала разумно, нужно достаточно точно описать знания и правила. Вторая: возможно, нужное поведение легче получить из множества простых взаимодействующих элементов, чем записать вручную. Первая приведёт к символическим программам и экспертным системам; вторая — к нейросетям и обучению на данных. К 1940-м годам эти пути ещё не превратились в два лагеря, но вопрос о том, что именно нужно строить — правила или обучаемую систему, — уже начал проступать.

Машины, которые слушали ответ мира

У переключателя есть положение; у вычислительной схемы — входы, выход и правило, связывающее одно с другим. Но многие устройства должны работать в мире, который не остаётся прежним на протяжении выполнения инструкции. Самолёт отклоняется от курса под действием ветра. Печь нагревает помещение, температура в котором меняется. Ракета получает сведения о собственном движении и должна корректировать траекторию. Команда «делай так» здесь быстро устаревает, если система не проверяет, что происходит после её действия.

В таких задачах действует обратная связь. Система измеряет состояние среды, сравнивает его с некоторой целью и использует разницу как сигнал для корректировки. Представим термостат: задана температура, датчик показывает текущую, механизм включает или выключает нагрев. Если контур устроен плохо, нагреватель может включаться слишком поздно или слишком часто. Если датчик врёт, система будет исправно регулировать не ту температуру, которая есть в комнате, а ту, которую он ошибочно сообщает. Устройство может выполнять функцию надёжно и всё равно приводить к плохому результату, когда измерение, цель или сама модель процесса выбраны неверно.

В 1948 году Норберт Винер опубликовал «Кибернетику», книгу, в которой рассматривал управление и передачу информации в живых организмах и технических системах. Кибернетика не была ранним названием искусственного интеллекта и не сводилась к разговору о роботах. Она предлагала общий язык для вопросов о том, как система получает сведения о среде, регулирует поведение и сохраняет устойчивость. Винер писал в момент, когда военная техника и автоматические устройства сделали проблемы управления особенно наглядными. Успешная система должна была не просто произвести вычисление, а действовать в изменяющемся контексте и учитывать последствия.

Слово «цель» в описании обратной связи иногда звучит так, будто в приборе есть намерение. Но цель в инженерной схеме задана извне: нужное давление, температура, высота или направление. Система сравнивает измерение с установленным ориентиром и выполняет операцию. Термостат не понимает, почему человеку холодно; ракета не желает достичь цели. Однако вокруг этих устройств возникает практический вопрос, который нельзя решить одним кодом: кто выбрал желаемое состояние и кто несёт ответственность, если цена ошибки распределена между разными людьми?

Обратная связь также осложняет привычную границу между машиной и её средой. Пока устройство только считает на бумаге, кажется, что оно наблюдает мир со стороны. Но система управления меняет объект, который измеряет. Результат действия становится новым входом; затем новое измерение вызывает следующее действие. Если контур работает с людьми, этот эффект может стать ещё сложнее: решение влияет на поведение людей, их поведение меняет данные, а система получает уже изменённую ею же картину. Такие петли будут особенно важны для позднейших алгоритмов прогнозирования и распределения ресурсов.

Кибернетика не была рецептом, из которого прямо выросло современное машинное обучение. Она не показывала, как из примеров выучить перевод или распознавание лиц. Но она помогла расширить образ машины. Машина могла быть не только исполнителем заранее фиксированной команды; её можно было рассматривать как часть контура, где измерение, действие и результат связаны. Рядом с формальной логикой возникла другая линия: поведение меняется в ответ на обратную связь.

К середине XX века эти линии ещё не образовывали единой дисциплины. Одни математики изучали, что значит вычислить функцию по заданным правилам. Инженеры строили устройства для управления. Нейрофизиологи и психологи пытались описывать нервную активность и поведение. Философы спорили о природе мышления. Впоследствии эту разрознен-

ную работу начнут собирать под новым именем, но сначала участникам нужно было научиться видеть, что именно можно формализовать, а что пока остаётся только обещанием.

Формула не выбирает задачу

Когда мы говорим, что машина «рассуждает», часто воображаем, будто внутри неё находится весь процесс человеческого мышления, только ускоренный. Но формальное правило отвечает на более узкий вопрос: если входные сведения представлены именно так, а задача задана именно так, какой следующий шаг следует выполнить? Чтобы запустить процедуру, кто-то заранее решил, что считать входом, как обозначить элементы и что засчитывать за правильный результат.

Силлогизм может быть безупречным по форме и всё же опираться на неверную посылку. Машина может точно вычислить значение по таблице, а сама таблица может содержать ошибку. Релейная схема способна безотказно выполнить неверно заданное условие. Правило не проверяет себя на справедливость только потому, что записано символами. Формализация повышает прозрачность одного этапа рассуждения, но не автоматически всех решений, которые предшествовали ему.

Это напоминает разницу между вычислением и навигацией. Таблица может сообщить моряку, какое значение соответствует определённой позиции; прибор может измерить направление; алгоритм — выполнить расчёт. Но чтобы использовать результат, нужно понять, к какому маршруту он относится, насколько точны наблюдения и какова цена ошибки. В море обстоятельства меняются, и ответ, правильный для исходных данных, может оказаться бесполезным, если сами данные устарели. Вычисление отвечает на вопрос, который ему задали; оно не всегда замечает, что человек имел в виду другой вопрос.

Именно поэтому границы задачи так важны. В математическом доказательстве правила вывода помогают проверить цепочку от посылок к заключению. Они не решают, какие утверждения стоит доказывать и какие посылки принимать. В устройстве Бэббиджа перфокарты задавали операции, но не объясняли, зачем человеку понадобилась конкретная таблица. В схеме Шеннона реле меняли состояние, но не знали, что это состояние означает для оператора. На каждом уровне человек приносил контекст, который система могла не представлять.

Это не аргумент против машин и не доказательство, что любой человеческий ответ мудрее вычислительного. Человек тоже ошибается, путает причины и последствия, принимает желаемое за действительное. Смысл в другом: сравнивать человека и машину следует по конкретной работе, а не по расплывчатому образу «разума вообще». Машина может прекрасно выполнять формализованный расчёт; человек может лучше заметить, что ситуация вышла за границы расчётной модели. В иной задаче сильные стороны могут поменяться местами.

В будущем разработчики попытаются делать системы, способные самостоятельно выбирать шаги и работать с неполными данными. Но и тогда выбор цели, набор наблюдений и критерии ошибки не упадут с неба. Чем менее очевидны правила, тем важнее спросить, кто их создал и что останется незамеченным. Этот вопрос появляется задолго до того, как машине дали имя.

Когда поведение задают вручную, человеческие предположения обычно видны в правилах: кто-то записал условие и связал его с действием. Когда система настраивается на примерах, предположения могут скрываться в том, какие примеры собрали, какие случаи сочли ошибками и какой результат поощряли. Способ их размещения меняется; потребность в выборе остаётся. Даже слово «точность» требует уточнения: точность для какой группы, в каких условиях и по сравнению с какой альтернативой?

Поэтому формализация — не холодная оболочка вокруг готового решения. Это способ сделать одни решения явными, а другие оставить за рамкой модели. Хорошая инженерия помогает эту рамку увидеть и проверить. Плохой пересказ создаёт впечатление, будто машину можно спросить о мире напрямую, без посредников и допущений. История ранних вычислите-

лей учит обратному: между вопросом и ответом всегда есть устройство, процедура и человек, выбравший, что считать ответом.

Тьюринг меняет вопрос

В 1950 году статья Тьюринга в философском журнале *Mind* начиналась с простого вопроса: «Могут ли машины мыслить?» Почти сразу Тьюринг объяснял, почему спор о точном смысле слов «машина» и «мыслить» способен отложить разговор на неопределённый срок. Вместо этого он предлагал заменить исходный вопрос более конкретной игрой. Чтобы оценить этот ход, важно помнить, что его знаменитый «тест» в пересказах часто не совпадает с тем, что он описал.

В исходной игре участвуют три человека: мужчина, женщина и задающий вопросы, который находится в другой комнате и общается с ними письменно. Задача спрашивающего — определить, кто из отвечающих мужчина, а кто женщина. Затем Тьюринг предлагает заменить одного участника машиной и спросить, что произойдёт с игрой. Это не просто «поговорить с ботом и посмотреть, кажется ли он умным». Машину проверяют в сценарии, где ответы должны помочь судье различить участников; сама конструкция статьи разворачивается вокруг того, насколько поведение в разговоре может служить свидетельством мышления.

Тьюринг выбрал поведение не потому, что доказал: внутреннего опыта не существует или он не важен. Напротив, он столкнулся с проблемой знания о чужом сознании. Мы не заглядываем напрямую в чужую голову, чтобы удостовериться, что там есть переживания. В обычной жизни мы строим выводы из слов, поступков и общего контекста. Тьюринг спрашивал, почему к машине мы должны предъявлять другой тип доказательства, если претензия на мышление проявляется в её ответах и действиях. Но из этого не следует, что убедительная реплика сама по себе доказывает человеческое понимание. Тьюринг предложил игру как способ обсуждать задачу, а не сертификат сознания.

В статье он последовательно разбирал возражения. Одни апеллировали к религии и человеческой исключительности, другие — к математическим пределам вычисления, эмоциям, творчеству или неспособности машины совершить ошибку. Тьюринг не отвечал: «Компьютер будет умным, просто поверьте». Он рассматривал, какие аргументы зависят от представления о внутреннем опыте, какие — от ограничений вычислительных моделей и почему некоторые возражения переносят разговор обратно к неопределённому слову «мыслить». Это эссе, а не рекламный план готового продукта.

Игра, в которой судят по ответам

Почему Тьюринг сделал общение письменным? Если судья слышит голос или видит собеседника, физические особенности сразу меняют игру. Текст сужает доступные признаки и оставляет поведение в разговоре. Но даже такой обмен не сводится к бесконечной болтовне о погоде. Тьюринг обсуждал вопросы, с помощью которых можно проверять умение отвечать и рассуждать в разных контекстах, а сама статья рассматривает память, вычислительные возможности, обучение и ряд возражений. «Тест Тьюринга» — позднейшее краткое название, которое часто затмевает его более широкую попытку поставить вопрос о машинном мышлении в форме наблюдаемой игры.

Важна и роль самого судьи. Он не получает отчёт о том, как устроен отвечающий: ему доступны сообщения и правила игры. Значит, результат нельзя отделить от условий проверки. Вопросы могут быть удачными или поверхностными, судья — осторожным или доверчивым, а короткая беседа — слишком узкой, чтобы раскрыть нужную способность. Этот пример помогает читать любую оценку ИИ: прежде чем удивляться результату, полезно спросить, кто проверял систему, на каких примерах и что именно считалось успехом.

Результат игры зависит не только от машины. На него влияют ожидания судьи, правила обмена, продолжительность и критерий успеха. Если поставить слишком лёгкие вопросы, система может выглядеть убедительно, не имея способности, которую мы хотим исследовать. Если выбрать узкий набор задач, можно проверить его качественно, но не делать вывод обо всём мышлении. Если же потребовать от человека универсального знания и безошибочности, сравнение перестанет быть честным. Любой тест становится содержательным лишь после того, как мы объяснили, что он измеряет, а что оставляет за кадром.

Этот урок актуален и сегодня. Когда продукт отвечает на вопросы с интонацией собеседника, интерфейс вызывает привычку приписывать словам намерение и понимание. Пользователь видит цельный текст, а не механику производства следующего слова, поисковый запрос, систему модерации или историю обучения. В 1950 году Тьюринг уже отделял наблюдаемое поведение от недоступного напрямую внутреннего состояния. Но это не означает, что он объявил поведение исчерпывающим доказательством сознания. Он дал спору форму, в которой можно задавать более конкретные вопросы.

В современной культуре игру превратили в бинарный ярлык: либо машина обманула человека и «разумна», либо не обманула и «глупа». Такой вывод слишком велик для измерения. Человек может принять за разум поверхностно убедительный стиль, а система может проявлять полезные способности и при этом не иметь того, что мы подразумеваем под человеческим опытом. Тест поведения отвечает на вопрос о том, что увидел судья в ограниченной ситуации. Остальное требует других критериев и отдельной аргументации.

Учиться вместо того, чтобы заранее знать всё

Тьюринг обсуждал и другой способ приблизиться к машинному интеллекту. Возможно, не следует проектировать систему сразу как взрослого знатока, в котором заранее закреплён каждый факт и каждое правило. Можно создать исходный механизм и дать ему учиться. Тьюринг называл эту гипотетическую систему «детской машиной» и сравнивал её развитие с воспитанием. Это было предложение направления, а не спецификация устройства с чертежами, набором данных и планом обучения.

Метафора меняла инженерную задачу. Если машина должна заранее содержать все знания, разработчику придётся определить их состав и способ записи. Если машина учится, важными становятся начальное устройство, среда, примеры, обратная связь и критерии успеха. Возникает новый тип зависимости от человека: кто решает, какие примеры система увидит? Кто исправляет её ошибки? Какие способности поощряются? Когда прекращать обучение? Обучение переносит часть человеческого выбора внутрь процесса, но не устраняет этот выбор.

Тьюринг рассматривал аналогии с образованием и предполагал, что систему можно развивать через опыт и корректировку. Он не утверждал, что машина должна пережить детство так же, как человек. Это различие важно. Человеческий ребёнок учится через тело, взаимодействие, социальные отношения, язык и последствия действий. Для вычислительной системы «пример» — это форма данных, заданная её конструкцией. Даже если внешне результат напоминает рост навыка, механизмы могут быть радикально разными.

Впоследствии идея обучаемой системы станет одной из центральных линий ИИ. Сначала ей будут сопутствовать ограничения вычислительной мощности и трудность подобрать задачи, на которых обучение можно наблюдать. Но сам вопрос останется: нужно ли подробно описывать каждое правило вручную или можно задать процедуру, которая изменит внутреннее состояние системы на примерах? В XXI веке ответом станут не просто данные, а массивы данных, инфраструктура хранения и обработки, методы оптимизации, люди, оценивающие результаты, и компании, превращающие обучение в продукт. В 1950 году до этой системы далеко; однако направление уже различимо.

Называя машину «ребёнком», легко вызвать ощущение, будто интеллект возникнет естественно, если только не мешать развитию. В инженерной практике это опасная иллюзия. Обучающую среду кто-то строит. Цели и сигналы ошибки кто-то выбирает. Данные отражают мир не полностью и могут закреплять его перекосы. Поэтому обучаемая машина не становится нейтральной машиной. Ответственность не исчезает вместе с ручным программированием — она меняет адрес.

Нить, которая тянется дальше

К концу этой главы перед нами не одна машина, а несколько разных способов поставить вопрос. Аристотель отделял форму вывода от его содержания. Луллий строил схему, которая помогала комбинировать понятия внутри богословского проекта. Лейбниц мечтал о знаках и правилах, способных сделать часть рассуждений исчислимой. Бэббидж проектировал механизм, который выполнял бы заданные операции над числами; Лавлейс размышляла, как одна и та же машина могла бы обращаться с символами, если их отношения удастся формализовать. Буль и Шеннон связали логические правила с электрическими переключателями. Винер описал контуры управления и обратной связи. Тьюринг предложил формальную модель вычисления, математические пределы процедур и способ поставить вопрос о поведении машины.

Это не прямая линия, на которой каждый следующий исследователь добавляет недостающую деталь и в конце получает современную нейросеть. Большинство этих людей занимались разными проблемами; их идеи не всегда влияли друг на друга напрямую. Историк не должен склеивать их биографии в одну историю неизбежного прогресса. Но между этими работами можно увидеть общий сдвиг: мысль и действие начинают описывать через отдельные элементы, правила и переходы. Как только часть задачи становится достаточно ясной, возникает вопрос, можно ли передать её исполнителю — человеку, механизму или электрической цепи.

Этот сдвиг не отвечает на вопрос, что такое разум. Он делает этот вопрос труднее и продуктивнее одновременно. Если машина вычисляет быстрее человека, чем мы обязаны ей за это? Если она выполняет программу, нужно ли говорить, что она понимает программу? Если её ответы выглядят осмысленно, какую дополнительную проверку мы требуем и почему? Если система учится на примерах, кто отвечает за то, что было выбрано как пример и как ошибка? Каждое техническое достижение будет возвращать нас к этим вопросам, только уже с работающим устройством перед глазами.

Есть соблазн начать историю ИИ в тот день, когда появилось само название. Тогда до нового термина остаётся несколько страниц, и дорога выглядит аккуратно. Но история названия — не то же самое, что история идей, методов и условий, которые сделали его убедительным. К 1950-м годам существовали логика, теории вычисления, первые электронные компьютеры, исследования нейронной активности и практический опыт автоматизации. Ни один из этих элементов ещё не означал «искусственный интеллект» в современном смысле. Они создали почву для проекта, который затем попытается объединить их под общей формулировкой.

Алан Тьюринг не увидел этого проекта. Он умер в 1954 году, за два года до летней исследовательской встречи в Дартмуте. Но статья 1950 года оставила для будущей дисциплины неудобное наследство: недостаточно спросить, похожа ли машина на человека; нужно уточнять, какие способности проверяются, какие условия поставлены и что именно позволяет заключить результат. Это хорошее правило для читателя XXI века, когда убедительный ответ часто приходит быстрее, чем успевает появиться вопрос о том, как он был получен.

Следующая глава начинается с документа, в котором исследователи предложили название для новой области и сформулировали амбициозную программу. Это будет не момент, когда машина внезапно научилась думать. Это будет момент, когда разрозненные надежды получили общий заголовок, лето, список задач и людей, готовых попробовать их решить.

Глава 2. Лето, которому дали имя

В заявке на исследовательский проект есть странное соседство. На первых страницах четверо учёных предлагают изучить, как машинам научиться пользоваться языком, строить понятия, решать задачи, прежде считавшиеся человеческими, и даже улучшать собственную работу. В финансовом расчёте того же документа стоят куда более земные строки: зарплата, билеты на поезд, аренда жилья, секретарская помощь, размножение материалов и непредвиденные расходы. Итоговая сумма — 13 500 долларов.

Так выглядело предложение, которое должно было изменить название целой области. Не в форме триумфального манифеста, а как заявка на два месяца совместной работы летом 1956 года. 31 августа 1955 года Джон Маккарти, Марвин Минский, Натан Рочестер и Клод Шеннон подписали документ с заголовком «Предложение о летнем исследовательском проекте по искусственному интеллекту». Внутри лежала не готовая теория и не план построить мыслящую машину. Там была ставка: если достаточно точно описать отдельные стороны обучения и интеллекта, возможно, машина сможет их воспроизвести.

С этого места принято отсчитывать рождение искусственного интеллекта как исследовательской области. Удобная формула слишком гладкая. К тому времени уже существовали математические теории вычисления, электронные компьютеры, модели нейронных сетей и эксперименты с программами. Первая глава книги прошла по некоторым из этих дорог. Дартмут не создал их задним числом. Он сделал другое: впервые включил выражение «искусственный интеллект» в название коллективного исследовательского проекта и предложил собраться вокруг вопросов, которые до того не образовывали единой дисциплины.

Это различие — между рождением идеи и появлением общего имени — и есть сюжет главы. Название обещало масштаб, а документ показывал, насколько мало его авторы могли заранее гарантировать. Проект стал важным не потому, что участники за лето решили проблему разума. Он дал им возможность считать эту проблему общей научной работой — и тем самым открыл спор о том, что именно они собираются создавать.

Восемнадцать страниц и один большой глагол

Оригинальная машинопись включала титульную страницу и семнадцать страниц текста. Основная часть предложения занимала лишь несколько страниц; за ней шли описания опыта и научных интересов самих организаторов. Позднейшие пересказы часто цитируют одну центральную фразу — всякий аспект обучения или любую другую черту интеллекта можно, по предположению авторов, описать настолько точно, чтобы машина могла её имитировать. Но в документе важен не только этот тезис. Важны слова вокруг него: «предполагаем», «попытаемся», «считаем возможным».

Авторы не объявляли, что интеллект уже разложен на детали. Они предлагали принять гипотезу, чтобы проверить её исследовательской работой. Между этими формулировками и позднейшим образом самоуверенных инженеров, обещавших построить разум за лето, — существенная разница. У заявки был смелый масштаб, но её авторы признавали, что описания многих задач ещё не были разработаны, а примеры решений предстояло искать.

В документе обозначены амбиции, которые сегодня звучат почти как оглавление целой истории вычислений: сделать так, чтобы машины пользовались языком, формировали абстракции и понятия, решали задачи, которые люди обычно выполняют сами, и улучшали собственную работу. При этом авторы не ограничились философской декларацией. Они пытались перевести большую тему в рабочие вопросы: как программировать компьютеры, можно ли представить мышление как манипуляцию словами и правилами, как нейронные сети формируют понятия, как оценивать сложность вычисления и откуда может появиться оригинальность.

На этих страницах постоянно встречаются два регистра. Первый — исследовательский: определение задачи, вычислительный механизм, пример, проверяемый результат. Второй — почти пророческий: интеллект, творчество, самоулучшение. Сосуществование этих регистров не было случайным дефектом. Чтобы получить финансирование и собрать людей, организаторам нужно было показать, что вопрос одновременно велик и достаточно конкретен для работы. Если описать только ближайшую техническую задачу, проект выглядел бы частным. Если говорить только о человеческом разуме, он оставался бы философским обещанием. Дартмутская заявка пыталась удержать оба края.

Это помогло ей быть сильной и одновременно сделало уязвимой. Широкое утверждение привлекало внимание к множеству возможных исследований, но под одним заголовком оказывались очень разные проблемы. «Использовать язык» — не одна операция. «Формировать понятия» может означать создание внутренних категорий, поиск общих черт в данных или запись нового правила. «Улучшать себя» может быть изменением кода, настройкой параметров, накоплением опыта или выбором более эффективной стратегии. В 1955 году документ не давал единого определения для всех этих обещаний. Он предлагал создать место, где люди могли бы начать его искать.

Четверо подписантов, четыре рабочих стола

Под предложением стоят четыре имени и четыре учреждения: Маккарти представлял Дартмутский колледж, Минский — Гарвард, Рочестер — IBM, Шеннон — Bell Telephone Laboratories. Для читателя это может выглядеть как случайная строка биографий. На деле она показывает устройство амбиции: область хотели собрать между математикой, моделированием нервной системы, вычислительной промышленностью и теорией информации.

Джон Маккарти был математиком, заинтересованным в логике и программировании. Его влекла возможность описывать рассуждение так, чтобы им мог пользоваться компьютер; однако в собственном приложении к проекту он писал и о среде, в которой обучающаяся система действует. Для него машину нельзя было оценить, не спросив, в каком мире она получает сигналы и к каким последствиям приводит её действие. Он хотел изучать не только внутреннюю схему вычисления, но и отношения между системой и средой. В приложении к заявке он предлагал начинать с простой, ограниченной среды и постепенно усложнять её. Машина должна была получать сигналы, пробовать действия и замечать, как меняется окружающая ситуация. Если у неё есть только вход и выход, но нет способа связать действие с его последствиями, она будет перебирать варианты слишком медленно. Маккарти надеялся, что система сможет строить внутреннюю модель среды: сначала проверять возможный ход мысленно, а затем действовать. Это была программа исследований, а не готовый чертёж машины, но она добавляла к вопросу о рассуждении важную часть — связь между вычислителем и миром, на который он пытается влиять. В отдельном приложении Маккарти рассматривал естественный язык не только как словарь команд. Английский, полагал он, позволяет кратко формулировать аргументы, говорить о собственном ходе работы и строить догадки. Формальные языки того времени хорошо справлялись с вычислениями, заданными заранее, но хуже подходили для таких открытых операций. Поэтому Маккарти хотел описать язык, в котором компьютер мог бы говорить об объектах и событиях, проверять предположения и возвращаться к собственной задаче. Это был план исследований, а не утверждение, что повседневную речь можно просто исполнить как программу.

Марвин Минский подходил к вопросу через нейронные модели и теорию обучения. Ещё до встречи он работал над машиной, имитирующей обучение нервной сети, а в проектном документе рассматривал, как можно исследовать интеллектуальное поведение математически. Его интерес не сводился к утверждению, что мозг — это просто компьютер. Он проверял, какие функции способны выполнять упрощённые модели нейронов и что можно узнать, если перенести эту гипотезу в вычислительный эксперимент.

Натан Рочестер представлял IBM — организацию, где вычислительная техника уже была большим инженерным и промышленным проектом. Он участвовал в разработке IBM 701, одного из ранних больших компьютеров компании, и занимался тем, как заставить вычислительные машины выполнять задачи, прежде поручавшиеся человеку. Его присутствие связывало исследовательскую мечту с реальным оборудованием: сколько памяти доступно, как устроено программирование, какие задачи машина способна выполнять в срок и за какие ресурсы.

Клод Шеннон пришёл из Bell Telephone Laboratories, где сформулировал математическую теорию информации и применял логику к проектированию переключательных схем. Но в его интересах уже были не только каналы связи. Он предлагал программы для игры в шахматы и строил небольшие релейные устройства, демонстрировавшие элементы поиска и управления. Шеннон умел смотреть на информацию как на объект математического анализа, не смешивая её с человеческим смыслом сообщения. Для нового проекта это был важный ресурс и важное ограничение: измерить передачу сигнала — не то же самое, что объяснить понимание.

В приложениях к заявке каждый из четырёх организаторов описывал собственные направления работы. Это не была общая теория, которую они просто разделили на четыре части. Сам документ показывает различие интересов: формальные языки и окружение, нейронные модели, вычислительные машины и алгоритмы, теория информации и логические схемы. Их объединяла не договорённость о том, что такое интеллект, а решение работать под одной исследовательской вывеской, хотя бы на время.

У такого союза была практическая ценность. Математик мог увидеть в инженере того, кто превратит идею в работающую программу; специалист по компьютерам — получить задачу, выходящую за пределы арифметики; исследователь мозга — проверять гипотезы вычислением; теоретик информации — предложить язык для точности и сложности. Но союз не снимал разногласий. Он создавал условия, в которых разногласия можно было обсуждать как части одной проблемы.

Список задач, который оказался шире одной науки

Заявка не предлагала участникам сначала договориться, что такое интеллект, а потом уже работать. Она поступала практичнее: взять отдельные способности, найти способ их описать и посмотреть, удаётся ли поручить описание машине. Этот разбор на задачи сделал проект выполнимым. Одновременно он создал риск: если части исследуются отдельно, можно забыть, что они должны быть связаны в человеке.

Самый прямой путь шёл через автоматические вычислители. Если машину можно научить выполнять работу, рассуждали авторы, её действия можно представить как программу для калькулятора. Тут же появлялась оговорка: компьютеров середины 1950-х могло не хватить по скорости и памяти для моделирования сложных функций мозга, однако главным препятствием подписанты считали недостаток подходящих программ. Теперь это утверждение читается как очень смелая оценка инженерной задачи. Но в нём был и исследовательский вызов: прежде чем объявлять предел вычислительных устройств, нужно выяснить, умеем ли мы вообще описать нужную работу.

Программа при таком взгляде становилась центральным посредником между задачей и машиной. Электронное устройство могло быстро выполнять операции, но без подходящего описания оно оставалось мощным калькулятором, которому неизвестно, что считать. В то время идея уже проверялась на конкретных проектах: кто-то писал шахматные программы, кто-то исследовал сетевые модели, кто-то пытался формализовать доказательство. Дартмутская заявка не утверждала, будто язык программирования или объём памяти не имеют значения. Она ставила вопрос о другом барьере — о способности людей формулировать программы для сложных интеллектуальных задач.

Язык занимал в проекте особое место. Можно ли заставить компьютер работать со словами, а не только с числами? Один из возможных взглядов состоял в том, что значительная часть мысли — операции с языковыми единицами по правилам рассуждения и догадки. Обобщение тогда можно описать как введение нового обозначения и отношений, связывающих его с прежними выражениями. Авторы, однако, прямо отмечали, что эта идея пока не была разработана достаточно точно и не получила убедительных примеров. Это признание принципиально: в заявке язык был не доказанным методом, а областью, в которой предстояло придумать сам способ формализации.

Почему язык вообще казался привлекательным ключом? Слова позволяют коротко описывать классы объектов, события и отношения. Человек может сказать «положи книгу на стол», не перечисляя каждый пиксель поверхности и не задавая все возможные положения предмета. Чтобы программа справилась с такой инструкцией, ей нужно представить объекты, действия и отношения в форме, пригодной для обработки. Но обыденный язык не похож на таблицу, где у каждого знака ровно одно значение. Слова зависят от контекста, пропущенные сведения восстанавливаются из ситуации, а люди меняют тему, не объявляя об этом. Дартмутская заявка замечала потенциал языка, но ещё не обладала механизмом, который позволил бы безопасно перенести всё это в программу.

Рядом стояло исследование нейронных сетей. В документе спрашивали, как расположить гипотетические нейроны, чтобы из сети возникали понятия. Подписанты ссылались на более раннюю теоретическую и экспериментальную работу нескольких исследовательских групп и признавали, что есть частичные результаты, но нужна дальнейшая теория. Это не была попытка единодушно объявить нейронную сеть единственным путём к искусственному разуму. Скорее, её рассматривали как один из способов изучать, как простые элементы могут совместно формировать более сложную функцию.

Соседство языка и нейронных сетей сразу показывает, почему встреча не была собранием единомышленников с одинаковыми ответами. Один подход мог пытаться явно представить знания в символах и правилах; другой — выяснить, как сложное поведение рождается из сети более простых элементов; третий — исследовать поиск в пространстве вариантов. Они отличались не только техникой, но и тем, что считали объяснением. Для одних важно было, чтобы система следовала понятным правилам. Для других — чтобы она демонстрировала способность к обучению и обобщению. Общая вывеска дала возможность спорить об этом вместе, не разрешая спор заранее.

Авторы отдельно поднимали проблему вычислительной эффективности. Для задачи с формально проверяемым ответом можно перебирать варианты один за другим. В принципе такой поиск способен найти правильный результат, если вариантов конечное число и ответ среди них есть. Но пространство вариантов может расти так быстро, что поиск перестаёт быть практически полезным. Тогда мало знать, что решение существует: нужно оценить, насколько дорог путь к нему и как находить короткие или перспективные маршруты. Подписанты связывали это с теорией сложности вычисления — вопросом не только «может ли машина решить задачу?», но и «какие ресурсы потребует решение?».

Это был редкий для столь широкой заявки жест точности. Разговор об искусственном интеллекте легко превращается в спор о человеческой способности вообще. Теория сложности возвращала его к измеримому: можно описать задачу и сравнить методы решения по затратам. Однако даже на этом уровне возникал разрыв между формальным и практическим вопросами. Задачу, для которой можно проверить ответ, ещё требуется записать так, чтобы система знала, что перебирать; а сам перебор может быть непосильным. Уже в 1955 году было понятно, что вычислительная мощность не равна разумной стратегии.

Шахматы делают этот разрыв наглядным. Если программа рассматривает каждый возможный ход, а затем каждый ответ соперника, число вариантов быстро растёт с каждым дополнительным шагом поиска. Даже несколько вариантов на каждом ходе дают огромное дерево продолжений. Можно увеличить скорость компьютера, но это не отменяет вопроса, какие ветви стоит изучить первыми и когда дальнейший перебор уже не изменит выбор. Поэтому в задаче важны не только вычислительные ресурсы, но и стратегия сокращения поиска. Именно здесь инженерное ограничение становится частью интеллектуального вопроса: система должна не просто считать быстро, а решать, что считать.

Идея самоулучшения и творчества поднимала планку ещё выше. Если машина сможет менять собственные правила, улучшать способы решения и предлагать ходы, которые не были буквально прописаны программистом, где проходит граница между заранее заданным алгоритмом и неожиданным поведением? В предложении рассматривались схемы самосовершенствования, а различие между творческой и компетентной, но шаблонной работой связывалось с догадкой и некоторой долей случайности. Авторы не выдавали это за установленную теорию творчества. Напротив, называли такую догадку неполной и указывали, что случайность должна быть каким-то образом направлена.

Эта оговорка важна и сегодня. Если случайная инструкция появляется в произвольном месте программы, это скорее ломает вычисление, чем превращает его в творчество. Непредсказуемость сама по себе не является воображением. Нужно, чтобы система могла оценивать последствия, сохранять удачные варианты и ограничивать поиск. В 1955 году это пока обозначалось как вопрос, достойный исследования, а не как готовый рецепт. Но уже здесь видно, насколько рано ИИ взял на себя слово «творчество» и насколько осторожно, по крайней мере в отдельных местах самой заявки, авторы относились к объяснению этого слова.

Зачем Маккарти понадобилось новое название

До 1955 года люди уже исследовали устройства, которым сегодня мы без сомнений приписали бы отношение к ИИ. Работали с логическими схемами, автоматами, вычислительными алгоритмами, сетями нервных элементов, управлением и обучением. Но эти направления жили в разных интеллектуальных контекстах. В одних случаях обсуждали связь мозга и машины; в других — связь вычисления и логики; в третьих — управление и обратную связь. Между ними не было единого словаря и единого плана исследований.

Маккарти позже объяснял, что хотел закрепить за проектом отдельное направление. Его разочаровывало, что в сборнике *Automata Studies*, готовившемся вместе с Шенноном, лишь немногие работы прямо занимались тем, как заставить машины вести себя разумно. Сборник вырос из приглашений к исследователям, которые Маккарти и Шеннон разослали ещё в 1952 году; книга вышла в 1956-м. В ней автомат изучали прежде всего как формальную систему с заданными состояниями и переходами. Для Маккарти проблема была в фокусе: ему хотелось поставить в центр поведение, которое обычно называют разумным. Новый термин должен был не перечеркнуть теорию автоматов, а привлечь внимание к недостающему вопросу. Название «искусственный интеллект» он выбрал, чтобы обозначить приоритет и не дать основной теме раствориться среди исследований автоматов. Это воспоминание написано спустя полвека, и его следует читать именно как позднее объяснение одного из организаторов, а не как полный протокол мотивов всех четырёх. Но формулировка Маккарти полезна: речь шла не только о словах, но и о том, какой вопрос займёт центр внимания.

Название позволяло собрать под ним разные методы, не объявляя заранее победителя. «Теория автоматов» могла звучать как исследование формальных устройств. «Кибернетика» связывала разговор с управлением и обратной связью. «Машинное обучение» выделяло бы способность менять поведение на примерах. «Искусственный интеллект» было шире этих названий и привлекало к себе вопрос о функциях, которые люди связывают с разумом. Такая широта помогала приглашать математиков, инженеров и исследователей мозга на одну встречу.

Но широкий термин не был нейтральным контейнером. Назвать область — значит выбрать, с чем её сравнивать. Если речь идёт о «вычислении», в центре могут оказаться правила, процедуры и ресурсы. Если об «интеллекте», внимание смещается к человеческим способностям и к вопросу, насколько система на них похожа. Одно слово связывает эксперименты, которые технически сильно различаются, общей обещающей перспективой. Оно может открыть новые двери, но также заставляет людей ожидать от частного результата ответа на слишком общий вопрос.

Это не уникальная странность ИИ. Новые научные области часто создаются через названия, конференции, журналы, лаборатории и программы обучения. Вначале нет одной окончательной теории, которая объясняет, что именно объединяет все проекты. Есть исследователи, которые договариваются обсуждать свои работы в общей рамке, обмениваются методами и спорят о границах этой рамки. Если идея потом получает поддержку и воспроизводится в новых группах, название начинает казаться естественным — будто оно просто обнаружило область, которая всегда там лежала.

У выражения «искусственный интеллект» был дополнительный эффект: оно вызывало интерес за пределами лаборатории. «Автоматический вычислитель» описывал машину; «имитация человеческих функций» — исследовательскую цель; «искусственный интеллект» обещал разговор о способности мыслить. В одном коротком названии скрывалась гипотеза, что человеческие способности можно частично перенести в техническую систему. Такой заголовок было легче запомнить и обсуждать, чем перечень разных проектов. Но его заметность неизбежно опережала точность определения.

По позднему объяснению Маккарти, слово помогло «прибить флаг» к мачте. Метафора военная, но суть организационная: указать, где проходит территория проекта и какие исследования следует считать его частью. Это не значит, что авторы внезапно присвоили все похожие работы или что остальные исследователи приняли название без сопротивления. Речь о более скромном, но важном результате: новая рамка позволила обращаться к финансирующим организациям и приглашённым участникам с конкретным предложением о совместной работе.

Позднее историки будут оценивать, было ли Дартмутское событие началом новой области или прежде всего моментом её институционализации. Для самих организаторов эти формулировки не обязаны были исключать друг друга. До встречи существовали идеи и эксперименты, а после неё люди могли называть свою совместную область исследованием искусственного интеллекта, собирать проекты вокруг этого названия и передавать его следующим поколениям. Рождение области — не рождение всех её идей. Это устройство места, где они могут встретиться.

У проекта были деньги, сроки и список приглашённых

Летнее исследование требовало больше, чем хорошей идеи и свободной аудитории. Авторы попросили Фонд Рокфеллера выделить 13 500 долларов на двухмесячную работу группы из десяти человек. В смете шесть исследователей университетского уровня получали по 1 200 долларов, двум аспирантам отводили по 700. Ещё 2 400 долларов закладывали на дорогу и жильё, 850 — на секретарскую и организационную работу, 600 — на поездки краткосрочных гостей, 550 — на непредвиденные расходы. Эти цифры показывают масштаб замысла: организаторам нужно было не просто оплатить билеты, а создать условия, в которых разные специалисты могли бы на время отложить часть обычной работы.

Фонд не должен был платить всем одинаково. Организаторы ожидали, что сотрудники IBM и Bell Telephone Laboratories будут получать зарплату от своих учреждений; преподавателям Дартмута и Гарварда требовалась поддержка гранта. Аспирантам предлагали отдельные стипендии. Такой расчёт сводил за одним столом людей с разными работодателями, должностями и финансовыми условиями. Научное сотрудничество начиналось с вопроса, кто оплатит время каждого участника. Это было особенно важно для приглашённых из промышленных лабораторий: их присутствие зависело не только от интереса к теме, но и от того, мог ли работодатель освободить их от обычных обязанностей. Грант не покупал целиком лето каждого специалиста, а собирал части времени и поддержки из нескольких источников. Уже поэтому расписание в заявке было скорее координационным обещанием, чем гарантией, что десять исследователей одновременно окажутся доступны группе.

В строках о жильё и секретарской работе заметна повседневная сторона большой идеи. Некоторым исследователям пришлось бы одновременно оплачивать жильё в Дартмуте и квартиру дома. Кто-то должен был переписываться с приглашёнными, размножать предварительные материалы и согласовывать поездки. Без этой работы программа оставалась бы заявкой. Смета поэтому рассказывает не только о стоимости проекта, но и о его устройстве: исследователи должны были приехать, получить время для работы и иметь возможность обсуждать промежуточные результаты.

План включал регулярные семинары, индивидуальные занятия и встречи в небольших неформальных группах. До приезда потенциальным участникам собирались разослать предложение и спросить, над какой задачей они хотели бы работать; участникам предлагали заранее обменяться прежними работами и новыми идеями. Авторы надеялись собрать десять человек, чьи методы и интересы дополняли бы друг друга. Приглашения предполагалось направлять по оценке организаторов — исходя из того, какой вклад конкретный исследователь мог внести. От приглашённых ждали предварительной работы: прислать тему, поделиться уже написанными материалами и обозначить задачу, которую они хотели бы атаковать летом. Так организаторы надеялись не тратить первые недели на знакомство с интересами друг друга. Предложение строило мост от общей гипотезы к конкретным исследованиям ещё до того, как группа приезжала в Дартмут. Предварительный обмен был важен ещё и потому, что разные специалисты могли использовать одно слово для разных вещей. Если один исследователь говорил об обучении, он мог иметь в виду изменение параметров сети; другой — добавление фактов и правил в программу; третий — способность находить более короткое решение. Заранее присланные работы должны были дать группе общий материал для разговора и помочь быстрее перейти от терминов к конкретным методам. Это не гарантировало согласия: участники могли прочитать одну работу и увидеть в ней разные проблемы. Но спор хотя бы начинался с общего примера, а не с догадки о том, что собеседник имеет в виду.

Поэтому список людей вокруг проекта нельзя автоматически читать как список тех, кто провёл в Дартмуте всё лето. В нём соседствовали организаторы, потенциальные участ-

ники и люди, которым могла быть интересна новая тема. Приглашение зависело от личных обстоятельств; участие — от расписания, поддержки учреждения и собственных исследований. Вопрос «кто входил в область?» здесь ещё только формировался вместе с вопросом «кого позвали обсуждать её».

Позднее Маккарти вспоминал, что Фонд Рокфеллера выделил лишь половину запрошенной суммы. Это свидетельство организатора, записанное спустя десятилетия, а не полный финансовый отчёт; оно не позволяет пересчитать каждую статью расходов. Но оно объясняет, почему лето не могло в точности повторить проект на бумаге. Сокращение средств влияло на то, сколько времени и поддержки могли получить участники, а личные исследовательские планы в любом случае не исчезали вместе с билетом до Нью-Гэмпшира.

Уже в смете видна главная трудность: собраться вместе было возможно, но общая тема сама по себе не делала работу общей. Чтобы представить разницу между замыслом и состоявшейся встречей, нужно перейти от бюджета и плана к тому, кто действительно приехал, над чем работал и что Маккарти смог запомнить спустя полвека.

Лето 1956 года не исполнило план целиком

На бумаге участники должны были провести два месяца в Дартмуте, совместно работать над искусственным интеллектом и добиться значительного продвижения. В воспоминаниях Маккарти встреча выглядела иначе: люди приезжали в разные даты и оставались на разное время. У каждого была собственная исследовательская программа, и организаторам не удалось полностью переключить участников на одну общую задачу. Сокращённое финансирование тоже сыграло роль. Позже сам Маккарти отмечал, что первоначальный план не сработал так, как задумывался.

В позднем воспоминании Маккарти заметно и отсутствие некоторых учёных, которых он считал возможными важными собеседниками: Алан Тьюринг умер в 1954 году, а Джон фон Нейман к лету 1956-го был уже тяжело болен. Это наблюдение не превращает встречу в список приглашённых, которые «должны были» приехать. Оно напоминает, что область не создавалась в комнате, где собрались все главные авторы её предистории.

Эта разница между планом и фактическим событием важна не для того, чтобы поймать организаторов на преувеличении. Напротив: она объясняет, почему миф о едином съезде учёных вводит в заблуждение. Дартмутский проект был не конференцией, где в один день все участники слушали один доклад, а затем подписали общий манифест. Это была попытка устроить исследовательскую работу на протяжении лета. Люди появлялись, обсуждали свои темы, представляли программы и продолжали проекты, которые уже начались в других местах.

В позднем рассказе Маккарти одним из самых заметных эпизодов стало короткое появление Аллена Ньюэлла и Герберта Саймона: они приехали всего на несколько дней и представили Logic Theorist, программу, уже находившуюся в разработке. Её показ связывал общий тезис заявки с конкретной работой: компьютер мог искать доказательство в формальной системе, где результат допускает проверку. Механизм программы и то, что именно она доказала, станут предметом следующей главы.

Здесь нужно быть точными с хронологией. Logic Theorist не родилась в Дартмуте и не была создана в ответ на встречу. Ньюэлл, Саймон и программист Клифф Шоу разрабатывали её до летнего проекта; ручную симуляцию одной из теорем провели ещё в декабре 1955 года. Презентация в Дартмуте сделала работу видимой участникам, но не была моментом создания алгоритма. Это важная разница между лабораторным проектом, публичной демонстрацией и институциональной историей области.

Другие эпизоды Маккарти вспоминал через собственный интерес. Он рассказывал о шахматной программе IBM, которую представлял Александр Бернштейн, и о своей реакции на поиск в игре. Маккарти предложил использовать альфа-бета-отсечение: если по уже просмотренной части вариантов ясно, что продолжение не может оказаться лучше найденного, программу можно не заставлять перебирать его до конца. Это не отменяло поиска, но помогало направить ограниченное машинное время на перспективные линии. Бернштейн, по воспоминанию Маккарти, не принял совет. Важен не спор о том, кто оказался прав, а то, что программа уже ставила практический вопрос о цене поиска: как выбирать ходы, когда число возможных продолжений растёт слишком быстро? Идея, предложенная у доски, превращала общее слово «интеллект» в обсуждение конкретного алгоритмического решения.

Маккарти также описывал геометрический доказатель, над которым работал Минский, и признавал, что его собственные идеи о представлении здравого смысла математической логикой ещё не созрели для доклада. Эти детали показывают встречу не как безупречную выставку готовых открытий, а как рабочую среду, где одни идеи демонстрировали, другие только формировались, а участники расходились в том, что стоило делать дальше.

Сам Маккарти оговаривался, что помнит прежде всего события, связанные с его собственными научными интересами, и не претендует на исчерпывающий отчёт. Это редкая и важная оговорка в истории, которую часто рассказывают как ровную линию из нескольких знаменитых имён. Один из главных организаторов прямо признаёт предел своего рассказа. Поэтому его свидетельство ценно как память участника, но его нельзя превращать в полный список событий и людей. Для этого нужны архивные записи, письма, заметки других исследователей и документы самого проекта. Маккарти прямо называл работы, которые его рассказ почти не затрагивал: Раймонд Соломонофф занимался алгоритмической информацией, а Эдвард Мур — автоматами. Их имена важны здесь не как ещё одна полка в пантеоне, а как проверка самой легенды: даже память одного из организаторов не охватывает все линии работы, присутствовавшие летом.

Не было и торжественного итогового отчёта, который свёл бы все летние обсуждения к общей теории. В первоначальном документе предполагалось, что заинтересованным людям будут рассылать отчёт о проекте; позднее Маккарти отметил, что такого отчёта не появилось. Ожидаемый формальный продукт не возник. Но отсутствие отчёта не означает, что встреча ничего не дала. Его последствия распространялись через участников и их дальнейшие проекты: программы развивались в университетах и лабораториях, идеи переходили к новым сотрудникам, студенты становились исследователями, а выражение «искусственный интеллект» закреплялось в профессиональном употреблении.

Так и возникла особенность Дартмута: историческое значение встречи оказалось больше, чем сумма событий, которые можно перечислить за те два месяца. Она не дала ответа на центральную гипотезу, но помогла создать сообщество, в котором её можно было обсуждать и проверять. Название пережило краткое лето не потому, что участники нашли разум в машине, а потому, что оно стало полезным способом обозначать работы, которые до того разносились по разным направлениям.

Три программы — три способа проверить успех

Общая тема не означала, что работы участников можно было оценить по одной шкале. Маккарти вспоминал несколько конкретных проектов, и их соседство полезно само по себе: оно показывает, что под новым названием встречались задачи с разными правилами доказательства результата. Программа, которая ищет шахматный ход, геометрический доказатель и система формальной логики могут выглядеть как «машины, решающие задачи», но от каждой требуется особая работа.

У Logic Theorist был наиболее строгий критерий: если программа выводила теорему по принятым правилам, корректность цепочки можно было проверить. Это не отвечало на вопрос, понимает ли машина доказательство или рассуждает как человек. Зато позволяло точно установить, что именно она сделала: получила формальные выражения, применила допустимые преобразования и пришла к результату, который выдерживает проверку внутри системы логики. Для раннего машинного исследования это было важным преимуществом: демонстрация опиралась на прозрачный стандарт правильности.

Шахматная программа Александра Бернстайна работала в другом режиме. В игре нет одного доказательства, подтверждающего лучший ход во всех позициях; решения зависят от того, какую позицию программа сумела оценить и насколько глубоко просмотрела возможные продолжения. Здесь вопрос о «правильности» включает стратегию, ограничения поиска и сравнение с человеком или другими способами игры. Успешный ход может выглядеть убедительно, однако он не доказывает, что система понимает шахматы во всех смыслах этого слова. Он показывает, что заданная процедура способна искать в определённом пространстве вариантов. Более того, оценка хода не заканчивается на том, что программа его предложила: соперник отвечает и меняет положение на доске, поэтому решение приходится рассматривать как часть последовательности действий. Это делает шахматы показательным испытанием поиска и планирования, но не даёт той однозначности, что есть у проверки формального доказательства. Результат зависит от позиции, ограничений вычисления, выбранной стратегии и того, насколько сильный ответ способен найти другой игрок.

Геометрические задачи Минского давали третий пример. Чтобы доказать утверждение о фигурах, системе нужно представить объекты и их отношения так, чтобы правила поиска могли с ними работать. Её успех зависит не только от вычислений, но и от того, как поставлена задача, какие отношения доступны и каким путём можно получить доказательство. Такая программа сближалась с формальной логикой, но требовала другого описания исходного материала и другого способа организовать поиск. В воспоминании Маккарти идея Минского была особенно наглядна: геометрический доказатель должен был сокращать перебор, проверяя прежде всего утверждения, верные для данной диаграммы. Рисунок помогал отобрать перспективные шаги, но сам по себе ещё не заменял доказательство. В этом и состояла исследовательская задача: использовать интуицию, которую подсказывает схема, и при этом понять, какая часть вывода выдерживает строгую проверку.

Эти примеры не следует принимать за исчерпывающий список работ лета и тем более за сравнение, в котором один подход победил остальные. Маккарти сам ограничивал свои воспоминания интересами, которые были ближе ему. Их ценность здесь в другом: они помогают увидеть разницу между общим вопросом и отдельным техническим критерием. Во всех случаях люди проверяли, можно ли поручить вычислительной системе часть задачи, связанной с рассуждением или решением. Но доказательство теоремы, выбор шахматного хода и поиск геометрического вывода предъявляли разные требования к представлению знаний, алгоритму и оценке результата.

Отсюда следовала и практическая трудность: нельзя было объявить одну демонстрацию эталоном для остальных. В логике ответ проверяется относительно формальных правил; в геометрии важны и корректность вывода, и то, как система представляет фигуры; в шахматах качество приходится оценивать в ходе игры и относительно соперника. Успех в одном случае не давал готового способа измерить успех в другом. Даже слово «решить» меняло значение от задачи к задаче.

Не совпадал и смысл сравнения с человеком. Машина могла перебирать варианты иначе, чем шахматист, и всё же выбирать хороший ход; могла получить доказательство, не проходя через те же мысленные шаги, которые доступны ученику. Это не делало результат фиктивным. Это требовало точнее формулировать, что доказывает результат: что вычислительная процедура выполнила определённую задачу, а не что она обладает всеми человеческими способностями, которые мы связываем с этой работой.

С этого момента новая область приобретала сразу два свойства. У неё был общий предмет разговора — возможность выполнять на машине то, что связывают с интеллектом, — и не было единого метода, которому все должны были следовать. Это разнообразие не означало, что поле не сложилось. Оно означало, что его границы и критерии предстояло строить на конкретных задачах. Вместе эти различия удерживают масштаб вывода в границах доказательств. Успех в одной задаче не нужно обесценивать только потому, что система не похожа на человека; однако его нельзя незаметно превратить в свидетельство универсального разума. Это напряжение и останется одной из главных тем молодой области.

В этих различиях уже видна одна из причин, по которой общая область не могла опираться на один тест интеллекта. Формальная корректность доказывает, что последовательность шагов допустима в заданной системе; хорошая игра показывает, что алгоритм справляется в соперничающей среде; геометрическое доказательство зависит от удачного представления объектов и отношений. Каждый результат имеет значение, если ясно, к какому утверждению он относится. Но ни один из них не подменяет остальные и не даёт универсального измерителя способности мыслить.

Один сезон не мог вместить все эти вопросы

Здесь возникает соблазн объяснить расхождение между предложением и состоявшимся летом одним простым препятствием: не хватило денег, поэтому участники не достигли обещанного. Но такая версия слишком удобна. Деньги были реальным ограничением, однако сама программа объединяла задачи разного масштаба и разного типа. Чтобы продвинуться в теории нейронных сетей, требовалась одна последовательность рассуждений; чтобы написать шахматную программу — другая; чтобы определить, что означает работа со словами, — третья. Они могли влиять друг на друга, но не складывались в единый эксперимент с одним заранее известным результатом.

Два месяца — достаточно, чтобы провести интенсивную встречу, представить незавершённую работу, найти неожиданный вопрос и установить связи. Но этого срока мало, чтобы построить общую дисциплину с нуля и одновременно проверить десятки широких гипотез. Любая серьёзная исследовательская линия требует собственного времени: определить задачу, выбрать язык описания, написать и отладить программу, проверить её на случаях, которые не были подобраны специально ради удачного результата, а затем выяснить, что именно показал эксперимент. Если несколько линий используют разные понятия и методы, их нельзя просто ускорить, усадив участников в одной комнате.

Предложение, однако, не было наивным расписанием, где авторы ожидали решить всё перечисленное за лето. Оно прямо говорило о значительном продвижении по одной или нескольким проблемам. Это ограниченное ожидание иногда теряется в пересказах, которые соединяют широкую вступительную гипотезу с легендой о мгновенном появлении ИИ. В документе нет обещания, что участники построят универсальную систему. Есть намерение создать исследовательскую среду и проверить, насколько далеко можно продвинуться в отдельных частях вопроса.

Сама встреча показала, насколько трудно было согласовать такие разные части даже на уровне повседневной работы. Участники не приехали с одной общей задачей, которую можно было разделить на подзадачи и собрать в конце. Они принесли собственные исследования, уже начатые в иных институтах и командах. Совместное обсуждение могло помочь им увидеть связь, но не отменяло необходимости продолжать отдельную работу. Сообщество и лаборатория — разные формы организации: первое может возникнуть через разговор и общий термин; второму нужны постоянный персонал, ресурсы, оборудование, управление и годы накопленной практики.

Поэтому несоответствие первоначального замысла фактическому лету стоит понимать не как провал, который нужно скрыть, и не как чудо, которое нужно преувеличить. Это была граница между проектом на бумаге и новым полем, которому предстояло расти за пределами одного сезона. Дартмут дал исследователям общий вопрос и короткое окно, чтобы встретиться. Он не мог заменить собой годы работы в лабораториях, в которых разные ответы должны были создаваться, сравниваться и иногда терпеть неудачу.

Это также объясняет, почему отсутствие единого результата не обесценивает встречу. Некоторые исторические события важны как прорыв: появляется устройство или доказательство, после которого прежнее положение дел уже невозможно восстановить. Другие события важны как координация: они дают людям возможность продолжать разрозненные работы, пользуясь общим языком и понимая, что спорят о соседних вопросах. Дартмут принадлежит прежде всего ко второму типу. Если измерять его только количеством завершённых открытий за два месяца, результат будет скромным. Если смотреть на то, как после встречи стало возможно говорить о широкой области машинного интеллекта, значение выглядит иначе.

Такой взгляд помогает сохранить сразу две правды. Авторы выдвинули гипотезу смелого масштаба; конкретные результаты встречи были ограниченными и неоднородными. Позднейшая история не делает первое доказанным, а второе — несущественным. Между ними лежит настоящая работа: перевод общего имени в проекты, формулирование задач, сравнение методов, создание программ и постоянная проверка того, можно ли переносить успех из одной узкой способности на другую.

Гипотеза не была обещанием готового разума

В знаменитой формулировке предложения легко услышать больше, чем в ней написано. Если черту интеллекта можно точно описать, машина сможет её симулировать — такова исследовательская гипотеза. Но документ не утверждает, что интеллект уже полностью понят, что машина должна стать копией мозга или что участники за два месяца построят универсальный разум. Напротив, авторы рассчитывают на заметное продвижение по одной или нескольким задачам из большого набора. Риторический масштаб гипотезы шире, чем ожидаемый результат встречи.

Глагол «симулировать» позволяет удержать эту разницу. В вычислительной технике симуляция означает, что одна система воспроизводит некоторые свойства или поведение другой системы согласно выбранной модели. Симулятор полёта может помогать исследовать управление самолётом, не становясь самолётом. Компьютерная модель нервной сети может проверить гипотезу о передаче сигнала, не превращаясь в живой мозг. Для авторов Дартмутского проекта имитация интеллекта была инженерным направлением: взять способность, описать её достаточно точно и выяснить, может ли машина дать сопоставимый результат.

Но слово оставляет открытым главный критерий. Что считать «тем же самым»? Достаточно ли, чтобы машина выдала правильный ответ? Нужно ли, чтобы она пришла к нему тем же способом, что человек? Имеет ли значение, способна ли система объяснить промежуточные шаги, переносить навык в незнакомую среду или только повторяет успех в ограниченной задаче? Разные ответы приведут к разным проектам и к разным заявлениям о прогрессе. Дартмутская заявка не снимала эту неоднозначность — она создавала рабочее пространство, где её можно было исследовать.

В этом смысле документ одновременно был программным заявлением и приглашением к уточнению. Исследователи не собирались сначала найти полное определение интеллекта, а потом открыть лабораторию. Они разбивали широкое понятие на задачи, которые можно было формализовать и проверять. Такая стратегия не просто упрощает объяснение. Она помогает построить эксперимент: задать вход, описать допустимую процедуру, определить, какой результат засчитывается, а затем сравнить его с альтернативой.

Проблема начинается тогда, когда локальный успех превращают в общее доказательство. Программа доказывает теорему — значит ли это, что она рассуждает как математик? Система строит классификацию объектов — значит ли это, что она понимает категорию? Алгоритм предлагает удачный ход — значит ли это, что он обладает интуицией? Чтобы ответить, недостаточно показать впечатляющий результат. Нужно знать, какие именно условия система удовлетворяла, какие способы решения использовала и насколько далеко можно переносить вывод.

В Дартмутской заявке уже сосуществовали более осторожные и более смелые версии вопроса. Теория сложности вычислений требовала оценивать, сколько ресурсов нужно для решения формально заданной задачи. Исследование языка и творчества предполагало, что некоторые человеческие способности получится представить через знаки, правила, предположения и направленный поиск. Одни пункты были ближе к понятным математическим процедурам, другие опирались на гипотезы, для которых ещё не было удовлетворительной формализации. Общий термин объединял их раньше, чем они стали одинаково проверяемыми.

Это нельзя считать просто недостатком старого документа. В 1955 году люди только начинали выяснять, какие аспекты можно описать, а какие пока нельзя. Нельзя требовать от исследовательской программы готовой теории до того, как началась работа. Но можно спросить, что именно в ней известно, что предполагается и какую часть будущего авторы оставляют открытой. В этой заявке есть оба вида текста: точные пункты о вычислении и памяти, а рядом —

большие слова «интеллект», «самоулучшение» и «творчество». Читая её, важно не позволять второму набору отменить осторожность первого.

Как превратить слово «интеллект» в исследовательскую задачу

В заявке слово «интеллект» звучало как единое целое, но план работы был устроен иначе. Авторы перечисляли не один загадочный предмет, который предстояло обнаружить, а множество действий: распознавать закономерности, усваивать новое, использовать язык, строить понятия, решать задачи и менять собственное поведение. Такой ход был одновременно смелым и практичным. Если никто не мог сразу ответить, что такое разум вообще, можно было начать с вопроса, который уже допускает проверку: способна ли машина выполнить конкретную операцию, которую обычно связывают с мышлением?

Эта стратегия не возникла на пустом месте. Цифровой компьютер был универсальным вычислительным устройством в том смысле, что его поведение можно было задавать программой. Но универсальность аппарата ещё ничего не говорила о том, какие программы удастся написать и насколько убедительно они будут вести себя в сложной задаче. Исследовательский вопрос смешался с железа на описание действия: как представить задачу в форме, с которой машина может работать; какие правила нужны; сколько вычислений они потребуют; как понять, что полученный ответ годится.

Именно поэтому в проекте соседствовали темы, которые сегодня могут показаться принадлежащими разным дисциплинам. Обсуждение нейронных сетей обращалось к устройству нервной системы и к вопросу, можно ли построить систему из множества простых взаимосвязанных элементов. Работа с языком требовала понять, как слова обозначают предметы и отношения и как инструкции переходят в последовательность формальных операций. Теория вычислений задавала пределы: как измерять сложность задачи, когда машина сможет её решить и какой ценой. Самообучение добавляло ещё один уровень: может ли программа менять свои правила или улучшать действия, опираясь на результаты предыдущих попыток.

За каждой из этих тем стоял свой способ увидеть проблему. Сеть могла быть моделью того, как множество простых элементов совместно создаёт сложное поведение. Символическая программа могла представлять отношения между объектами и преобразовывать их по явным правилам. Теоретический анализ мог показать, что задача выполнима в принципе, но практически слишком дорога. Эти подходы не обязательно исключали друг друга, однако и не складывались автоматически в единую теорию. Летом 1956 года слово «искусственный интеллект» служило скорее общим адресом, чем обозначением согласованной научной школы.

В этом была полезная сторона. Если исследователи не обязаны были принять одну модель разума, они могли принести на встречу разные идеи и сравнить, какие части проблемы каждая из них проясняет. Но тут же обнаруживался риск: под общим названием можно было объединить проекты, которые лишь поверхностно похожи. Машина, доказывающая теорему, и машина, распознающая образ, обе демонстрируют некоторую способность, которую человек считает интеллектуальной. Но успех в одной задаче сам по себе не объясняет другую, а тем более не доказывает, что обе основаны на одном и том же механизме.

Вопрос о критерии успеха поэтому был не технической мелочью, а частью самой науки. Если программа решала логическую задачу, достаточно ли было правильного ответа? Нужно ли было, чтобы она пришла к нему тем же способом, что человек? Если программа использовала язык, должна ли была она понимать значение слов или только правильно преобразовывать символы? А если машина приспосабливалась к новым примерам, как отличить обучение от заранее прописанного набора ответов? Предложение ставило такие проблемы, но не обещало решить их до начала работы. Они были частью той области, которую ещё предстояло построить.

Разбивка на отдельные способности давала исследователям способ начать работу до того, как будет разрешён спор о природе мышления. Это было важно в 1955 году: вычислительные машины уже существовали, первые методы и проекты можно было проверить, но ни язык описания, ни правила сравнения результатов ещё не установились. Нужно было не только написать программы. Требовалось выработать привычку формулировать задачу так, чтобы разные исследователи могли понять, что именно проверяется и где проходит граница между демонстрацией и объяснением.

Цена такого подхода стала заметна позже и остаётся заметной. Хороший результат в узкой задаче может выглядеть как признак общего интеллекта, хотя на деле он опирается на ограничения, тщательно подобранные условия и труд человека, подготовившего данные или представление задачи. С другой стороны, требование сначала доказать универсальную теорию разума могло бы остановить любой эксперимент. Дартмутский план выбрал движение от частных задач к большому вопросу. Это позволило начать, но оставило открытым, как связывать отдельные успехи в общее объяснение.

Поэтому широкий список в заявке не был ни готовой картой дисциплины, ни случайным перечнем модных слов. Он показывал, что участники пытались определить пространство возможных исследований: какие функции можно поручить машине, какими методами к ним подступаться и какие ограничения нужно учитывать. В дальнейшем часть этих линий разойдётся, часть столкнётся, а некоторые будут объединяться снова. Но в 1955 году первым достижением было уже то, что их можно было обсуждать под одной формулировкой — не потому, что ответ найден, а потому, что вопрос стал общим.

Слово «симулировать» оставляло выбор за исследователем

Употреблённый в заявке глагол не требовал, чтобы машина была устроена как человеческий мозг. Он позволял изучать поведение через модель, то есть строить одну систему так, чтобы она воспроизводила выбранные свойства другой. Но всякое моделирование начинается с выбора: какие свойства считать существенными, какие детали опустить и какой результат считать совпадением. Для задач, где ответ можно проверить формально, границу провести легче. Для языка, творчества и обучения — гораздо труднее.

Это превращает формулировку 1955 года не только в технический тезис, но и в вопрос о стандарте доказательства. Если программа получает тот же ответ, что человек, этого может быть достаточно для одной инженерной цели и недостаточно для объяснения мышления. Если она действует по иным правилам, нужно решить, сравниваем ли мы только результат или также путь к нему. Если программа справляется лишь в заранее ограниченной среде, результат остаётся настоящим, но его нельзя автоматически переносить на все обстоятельства, где человек использует ту же способность.

Подобные вопросы нельзя решить одним определением слова «интеллект». Они требуют подробного описания конкретной задачи. Что система получает на вход? Какие операции ей доступны? Как проверить ответ? Как устроены примеры, на которых её оценивали? В каких новых ситуациях она продолжает работать, а где ломается? В этом смысле первая заявка предложила гораздо больше, чем оптимистичную фразу о симуляции: она открыла пространство для спора о том, как отличать точную демонстрацию от общего вывода о разуме.

Что именно началось в Дартмуте

Слово «основание» обычно вызывает образ торжественного начала: собрались люди, решили создать дисциплину, сформулировали программу и ушли с планом будущего. В истории искусственного интеллекта всё было менее аккуратно. Предложение 1955 года заранее назвало проект, перечислило вопросы, предложило участников и попросило ресурсы. Летом 1956 года реальность оказалась частичной версией этого замысла. Участники работали не одинаково долго и не обязательно совместно. Единого отчёта не появилось. И всё же имя пережило конкретную встречу.

Чтобы понять почему, полезно разделить три события, которые часто сливаются в одно. Первое — формулирование замысла: возможность исследовать машинный интеллект как набор достаточно точно описываемых задач. Второе — организационная попытка: собрать людей, которые уже работали над различными аспектами этой проблемы, в одном месте и на определённое время. Третье — долгосрочное признание: использование словосочетания «искусственный интеллект» как названия области, к которой могли относить новые проекты. Первые два события занимают месяцы вокруг 1955–1956 годов; третье происходит постепенно и не заканчивается вместе с летом.

Именно третье событие делает историю важной для книги. В науке название — не просто вывеска. Оно позволяет искать собеседников, описывать проекты в заявках и публикациях, объяснять студенту, к какому кругу вопросов относится работа. Оно может объединить исследования с разными методами и тем самым создать возможность для обмена. В то же время оно влияет на то, что считается центральным вопросом, за какие результаты исследователи получают внимание и какие обещания слышит внешняя аудитория. Поэтому название может быть одновременно полезным и опасным: оно даёт области видимость, но также создаёт ожидание, что под ним скрывается единый, уже понятный предмет.

В 1955 году такой единой теории не было. Авторы предложения не показали, что машинное обучение, работа с языком, доказательство теорем, нейронные сети, самоулучшение и творческие действия являются вариантами одного механизма. Они собрали эти направления под общей гипотезой и предложили, что их можно исследовать в рамках одного проекта. Это была постановка программы, не её завершение. Разница между этими двумя вещами важна и сегодня: заявить область вопросов — не то же самое, что решить их; создать демонстрацию в узкой задаче — не то же самое, что объяснить интеллект во всей его широте.

Сама широта объясняет, почему обещание оказалось одновременно плодотворным и неустойчивым. Если слишком рано сузить вопрос, можно принять один метод за весь разум. Если оставить его предельно широким, почти любой вычислительный проект может объявить себя частью искусственного интеллекта, а успех одной задачи легко представить как успех всей программы. Позже исследователям снова и снова приходилось договариваться о границах: что считать интеллектом, чему научилась система, как проверить её поведение и насколько результат зависит от условий эксперимента. Эти споры не были случайными недоразумениями вокруг изначально ясной науки. Они продолжали вопросы, оставленные открытыми в самом первом плане.

Рассказ о Дартмуте часто ставит в центр четыре фамилии на титульном листе. Это верно, но недостаточно. За документом стояли также те, кто организовывал встречу, искал деньги, координировал приглашения и помогал превратить свободную идею в расписание, список задач и рабочее место. За дальнейшей историей — программисты и помощники, без чьего труда идеи не стали бы выполняемым кодом; студенты и сотрудники лабораторий, которые проверяли гипотезы и исправляли программы; те, чьи исследовательские подходы не вошли в каноническую версию легенды. Даже в этой короткой истории полезно помнить, что научное

название создают несколько подписей, а научную работу — гораздо более широкая сеть людей и условий.

Новая область не появилась потому, что все будущие специалисты согласились с одной исходной идеей. Она появилась, потому что стало возможным говорить о разнородных проектах как о соседних, находить общие вопросы и спорить о границах между ними. В этом смысле Дартмутская встреча была не столько рождением машины, сколько попыткой устроить пространство, в котором вопрос о машине и интеллекте мог стать общей работой. Это менее мифический и более человеческий сюжет: вместо внезапного открытия — документ, бюджет, неполное финансирование, разные сроки приезда, программы, уже начатые до встречи, и название, которое постепенно приобрело силу.

У этой главы поэтому нет вывода «всё началось именно тогда» в буквальном смысле. Идеи вычисления, автоматизации, логики, обучения и моделирования возникли задолго до 1956 года. Первая глава привела нас от давних способов переложить умственную работу на инструменты к моменту, когда электронная машина стала обещать новый масштаб. Дартмутское предложение попыталось дать этому масштабу имя и программу, но не готовый ответ.

Наиболее наглядным примером машинного рассуждения, связанным с летом 1956 года, была программа *Logic Theorist*, которую создавали Аллен Ньюэлл, Герберт Саймон и Клифф Шоу. Она не возникла из общего решения участников Дартмута и не доказывала, что машина умеет мыслить так же, как человек. Но она умела совершать конкретную, проверяемую работу: искать доказательства логических теорем. Следующая глава перейдёт от широкого исследовательского обещания к устройству этой программы, людям, которые её создавали, и границам того, что она смогла показать. Это и будет проверкой дартмутской гипотезы в её самой конкретной форме: не окончательным ответом о природе разума, а точным вопросом о работе, которую машина действительно может выполнить.

Глава 3. Логика учится искать

В 1956 году одна программа получила задачу из книги, написанной людьми, которые сами считались вершиной математической логики. Она должна была построить доказательство: цепочку рассуждений, где каждый шаг разрешён правилами. Её успех не закрыл философский спор о машинном мышлении. Он дал более точный результат: компьютер нашёл проверяемый путь к теореме.

Программу называли *Logic Theorist* — «теоретик логики». В ранних публикациях встречается и название *Logic Theory Machine*, «машина логической теории». Её создавали Аллен Ньюэлл, Герберт Саймон и Джон Клиффорд Шоу, которого обычно называли Клиффом. Это была одна из первых программ, специально построенных для решения задачи, которую человек назвал бы интеллектуальной. Важно не превращать это первенство в безоговорочную легенду: до неё существовали вычислительные процедуры для математических задач, а сама программа работала только в узком формальном мире. Но именно *Logic Theorist* соединила программу, поиск доказательства и исследование человеческого решения задач в один наглядный опыт.

Предыдущая глава завершилась рассказом о *Logic Theorist*. Команда из Питтсбурга и Санта-Моники начала разрабатывать программу до встречи в Дартмуте; летняя демонстрация сделала проект заметным. Теперь можно рассмотреть его устройство. История отделяет две темы: компьютер действительно смог решить конкретную задачу, а вопрос о том, мыслит ли он как человек, остался открытым.

Задача, у которой можно проверить каждый шаг

В повседневной жизни мы редко получаем задачу в виде аккуратного набора правил. Нужно понять, что имел в виду собеседник, вспомнить контекст, решить, важна ли мелочь, и иногда спросить, чего именно от нас хотят. Доказательство в формальной логике устроено иначе. Исходные выражения заданы, правила вывода перечислены, а правильность результата можно проверить, не читая мыслей автора.

Команда выбрала фрагмент логики высказываний — формальную систему, где простые высказывания соединяются словами вроде «и», «или», «не» и «если..., то...». В символической записи отдельные высказывания обозначаются буквами, например p и q . Формула «если p , то q » говорит об отношении между ними, не сообщая, о каких именно предметах или событиях идёт речь. Для программы такая запись удобна: она работает со структурой выражения и его местом в доказательстве, а не с человеческим смыслом каждого предложения.

В этом заключалось и преимущество, и ограничение опыта. Когда Logic Theorist манипулировала формулой, ей не требовалось знать, означает ли p «на улице идёт дождь» или «число делится на два». Система была выбрана так, чтобы смысл можно было отложить в сторону и сосредоточиться на правильности переходов. Если машина из принятых посылок и правил выводила заключение, человек мог проверить цепочку по пунктам.

Основным источником задач служила *Principia Mathematica* Альфреда Норта Уайтхеда и Бертрана Рассела. Это многотомный труд начала XX века, в котором авторы стремились построить значительную часть математики на точных логических основаниях. Для Logic Theorist команда использовала теоремы из второй главы первого тома — участок, подходивший для исследования формального вывода. Программа не читала книгу так, как её читает студент. Ей задавали формализованные аксиомы, ранее полученные теоремы и цель, которую требовалось доказать.

Слово «теорема» здесь означает выражение, для которого уже построена допустимая цепочка вывода. Аксиома — исходное утверждение, которое система принимает без доказательства; правила вывода разрешают получать из уже принятых выражений новые. Это похоже на настольную игру: правила определяют, какие ходы законны, а партия состоит из последовательности ходов. Но сравнение с игрой имеет предел. В доказательстве не всякая законная последовательность ведёт к нужному результату. Задача — найти именно ту цепочку, которая заканчивается заданной формулой.

И здесь начинался настоящий поиск. Готовое доказательство можно проверять по шагам; самостоятельный поиск создаёт множество вариантов. Программа должна была выбирать подходящую аксиому, подстановку или преобразование, связывать цель с уже известными теоремами и определять, какой из допустимых шагов продолжит путь. Корректность отдельного хода не подсказывала, какой из тысяч возможных ходов приблизит её к результату.

Узкая формальная задача снимала часть неопределённости, но не снимала трудность выбора. Именно это делало её подходящей для исследования: можно было точно установить, что программа получила и что выдала, а затем разбирать, как она прошла от одного к другому.

От шахматной доски к логическим формулам

Logic Theorist не была исходным планом. Ньюэлл и Саймон пришли к ней через другую задачу: шахматы. Игра привлекала их как пример деятельности, которую легко назвать интеллектуальной и при этом можно точно описать. Правила известны, позиция видна на доске, а ход соперника меняет ситуацию по определённым законам. Однако количество продолжений быстро росло. Если рассматривать ответы на ответы, дерево вариантов разветвлялось так быстро, что полный перебор оказывался непрактичным даже для машины.

В разработке обсуждали и геометрические задачи. У них тоже были достоинства: фигуры, отношения между точками и линиями, возможность доказывать утверждения. Но геометрия приносила трудный вопрос о диаграммах. Рисунок помогает человеку увидеть, что может быть верным, однако внешний вид фигуры способен подсказывать свойства, которых в условии на самом деле нет. Если компьютер использует схему, нужно понять, какие сведения она действительно кодирует и что лишь кажется очевидным глазу.

Позднее Саймон вспоминал переход от шахмат через геометрию к логике как результат технических и случайных обстоятельств. В его рассказе книга *Principia Mathematica* оказалась под рукой. Это воспоминание не стоит пересказывать как легенду о внезапном озарении: выбор подходящей задачи обычно складывается из интереса исследователей, доступных материалов и ограничений оборудования. Но эпизод передаёт важный принцип. Они искали не обязательно самую грандиозную интеллектуальную задачу, а ту, где можно было отделить работу механизма от множества других проблем.

У доказательства были ясные правила и проверяемый критерий: либо конечная цепочка соответствует аксиомам и правилам вывода, либо нет. В шахматах оценка качества зависела от позиции, соперника и глубины поиска. В геометрии нужно было договориться о том, какие свойства фигуры представлены программе. Формальная логика уменьшала число неизвестных вокруг эксперимента. Она не делала задачу лёгкой; она делала ошибку заметной.

Это не означало, что математика считалась единственным настоящим видом мышления. Скорее, команда начинала с того участка, где можно было проверить идею информационной обработки на сложной задаче. Если программа докажет теорему, это не покажет, что она понимает язык, распознаёт обстановку или принимает решения в организации. Зато станет видно, можно ли описать часть рассуждения последовательностью операций и заставить компьютер самостоятельно выбрать между альтернативными путями.

Выбор также позволял соединить интересы Ньюэлла и Саймона. Они не ставили целью построить самый быстрый в мире автоматический доказатель. Их занимало, может ли программа воспроизводить некоторые процессы решения задач, которые наблюдаются у людей. Теорема была испытательным стендом: достаточно трудным, чтобы потребовать выбора, и достаточно формальным, чтобы процесс можно было разбирать.

Есть и важная разница между «машина получила правильный ответ» и «машина нашла доказательство». Для обычного вычисления результат может быть одним числом; проверка часто состоит в повторном вычислении. В формальной логике программа должна была выдать последовательность выражений, а каждый переход в ней должен был оправдываться правилом. Это делало работу видимой и проверяемой для людей, не обязанных верить авторскому описанию программы на слово.

Впрочем, читателю не нужно принимать выбор формальной логики за доказательство того, что все человеческое мышление устроено так же. Это была гипотеза исследователей, которую предстояло испытывать, а не установленный вывод. Чтобы понять её масштаб, нужно сначала увидеть, кто именно построил программу — и почему её невозможно приписать одному человеку.

Три участника, три разных вида работы

Аллен Ньюэлл работал в RAND Corporation и занимался исследованием человеческого решения задач и моделированием процессов обработки информации. К моменту Logic Theorist он уже участвовал в проектах, где компьютерные модели использовали для разбора того, как люди принимают решения в сложной обстановке. Его роль соединяла психологический вопрос — какие операции совершает решающий задачу человек — с инженерным: можно ли точно описать эти операции программой.

Герберт Саймон пришёл к этому исследованию не из традиционной математической логики. Он изучал политологию и административные организации, преподавал в Carnegie Institute of Technology и интересовался тем, как люди принимают решения при ограниченном времени и неполной информации. Позднее эту линию он назовёт ограниченной рациональностью: реальные люди не перебирают все возможности и не знают всего, что нужно для безошибочного выбора. Они ищут решение, которое достаточно хорошо работает при доступных ресурсах. В Logic Theorist Саймон видел способ исследовать такую стратегию на задаче, где её можно наблюдать с большой точностью.

Джон Клиффорд «Клифф» Шоу был программистом и инженером в RAND. Он работал с вычислительными машинами, участвовал в разработке JOHNNIAC и создавал программные средства, необходимые для обработки структурированных символических данных. В рассказах, где два учёных придумали алгоритм, а затем «просто запрограммировали» его, исчезает центральная часть работы. Шоу помогал превратить описание поиска в инструкции, которые могла выполнить конкретная машина. Кроме того, команда разрабатывала собственный язык, чтобы такие программы вообще можно было записывать удобным способом.

Их сотрудничество было распределённым. Ньюэлл и Шоу находились в RAND в Санта-Монике; Саймон работал в Питтсбурге и сотрудничал с RAND. Это не была единая группа, ежедневно сидевшая у одной машины. Концепции, психологические наблюдения и программирование пересекались через поездки, переписку, встречи и работу над общим проектом. Расстояние и разные профессиональные задачи могли замедлять взаимодействие, но одновременно заставляли участников точно объяснять, что именно они предлагают.

Обстановка RAND тоже имела значение. После войны организация выполняла крупные исследовательские задачи, в том числе для военных ведомств, и располагала вычислительной техникой, доступной для экспериментальной работы. Это не значит, что Logic Theorist создавали как боевое устройство или что доказательство теорем имело прямое военное назначение. Точнее сказать, исследования выросли внутри организации, где существовали машины, технические специалисты и ресурсы для экспериментов. Без этой инфраструктуры интеллектуальная схема могла бы остаться на бумаге.

Нельзя свести распределение труда к простому разделению «идея — код». Ньюэлл занимался и программной стороной, Саймон — не только психологической интерпретацией, Шоу участвовал в разработке языка и логики представления задач. Но их вклады различались, и эта разница важна. У программы есть научный вопрос, формальная схема, реализация, отладка и проверка на машине. Если убрать любой из этих слоёв, останется не действующая система, а замысел о ней.

Трио позже стали называть NSS — по фамилиям Newell, Shaw и Simon. Такое сокращение полезно для истории, но не должно создавать впечатление, что все решения принимались одновременно и одинаково. Программа стала возможна благодаря сочетанию навыков: анализировать решение задачи, формулировать его в виде процедур и заставить компьютер выполнять эти процедуры в заданных ограничениях. История Logic Theorist начинается не с гения у доски, а с распределённой команды, которой требовалось согласовать мысль, язык и машину.

Сначала алгоритм исполнили вручную

Один из самых выразительных эпизодов предшествует работе программы на компьютере. 15 декабря 1955 года команда провела ручную симуляцию Logic Theorist: люди последовательно исполняли описанные операции и проверяли, к чему приводит предложенный поиск. По позднему свидетельству Ньюэлла, это было первое успешное испытание процедуры на одной из теорем *Principia Mathematica*. Электронный компьютер ещё не выполнял всю программу в её окончательном виде.

«Ручная симуляция» звучит как театральная постановка, но здесь это был метод проверки алгоритма. Участники брали текущее выражение, применяли предусмотренные операции, записывали возникающие варианты и решали, какой путь исследовать дальше. Бумага, карточки, таблицы и инструкции временно заменяли память и команды машины. В этом смысле люди изображали вычислительный процесс, а не разыгрывали персонажей, которые будто бы стали компьютерами.

Ручной этап экономил дорогую работу над программой для JOHNNIAC: ошибка в самом замысле могла стоить больше, чем ошибка в коде. В середине 1950-х компьютер был общим и ограниченным ресурсом; программу для символьных задач ещё нужно было спроектировать, закодировать и отладить. Если процедура на бумаге не могла продвинуться даже при последовательном выполнении шагов, переписывание её на машинном языке не спасло бы идею. Ручной прогон позволял проверить, имеет ли поиск шанс завершиться и где он начинает блуждать.

Такой тест не доказывал, что программа будет эффективной на компьютере. Люди могли удержать в уме то, что не поместилось бы в машинную память; могли незаметно подправить ход или истолковать правило гибче, чем буквальная инструкция. Поэтому ручная симуляция была промежуточным свидетельством: схема поиска на конкретном примере оказалась исполнимой, но предстояло перенести её в точную программу и столкнуться с ограничениями JOHNNIAC.

Этот эпизод также помогает отделить программу от оборудования. Алгоритм — описание шагов и способов выбора — можно исполнять разными средствами. Человек может пройти по инструкции, компьютер — выполнить её электронными операциями. Но физический носитель влияет на скорость, память и удобство проверки. Алгоритм не тождествен железу, однако его практическая судьба от железа зависит.

У ручного этапа была и научная ценность. Команда могла наблюдать за поиском, отмечать, где возникает тупик, и пересматривать процедуру. Если исследователи одновременно хотели понять человеческое решение задач, они могли сравнивать поведение программы с тем, как человек решает аналогичную задачу. Это не превращало один удачный ручной прогон в модель человеческого мышления. Но оно позволяло перейти от смутного разговора о «рассуждении» к наблюдаемым операциям: выбору выражения, применению правила, возврату к прежнему узлу поиска.

Появление программы поэтому не следует датировать одним щелчком выключателя. Сначала была исследовательская идея, затем схема, ручная симуляция, язык программирования, реализация и запуск на машине. Эти этапы частично перекрывались. В позднем воспоминании Ньюэлла рабочая программа датируется летом 1956 года, тогда как рукопись и первые описания относятся к тому же году. Дартмутская встреча совпала с периодом, когда работу уже можно было показывать и обсуждать, но не была местом, где участники впервые придумали Logic Theorist.

Прежде чем машина могла искать, ей нужен был язык

Рассказывая об искусственном интеллекте, легко перейти от идеи к алгоритму, словно между ними нет промежуточных ступеней. В реальном проекте Logic Theorist эти ступени занимали месяцы. Чтобы программа могла работать с формулами, требовалось не только определить, какие правила вывода разрешены, но и придумать, как хранить выражения, находить их части, заменять одну структуру другой и сохранять список открытых вариантов поиска.

Обычный числовой расчёт часто можно представить как последовательность операций над значениями: сложить два числа, умножить результат, записать его в память. Формула устроена вложенно. Например, в выражении «если p , то (q или не- r)» одна операция применяется ко всему выражению, другая — только к его части, а отрицание относится к отдельной букве. Программе нужно знать не только символы по порядку, но и то, как они собраны друг в друга. Иначе она может случайно заменить кусок формулы, который выглядит похоже, но играет другую роль.

Ньюэлл, Шоу и Саймон разрабатывали Information Processing Language — IPL, «язык обработки информации». Это был не язык общения с компьютером в нынешнем бытовом смысле и не английские команды, которые машина якобы понимала. Это программная система для описания операций над символическими структурами. Важной её особенностью были списки — последовательности элементов, которые можно связывать и изменять. Вместо того чтобы хранить формулу как плоскую строку, программа могла представить её составные части так, чтобы к ним было удобно обращаться.

Представим полку с карточками, на каждой из которых записан один элемент, а на карточке есть указатель на следующую. Такая модель помогает понять список: можно пройти по его элементам, вставить новую часть, связать два фрагмента или отделить один участок. Аналогия не описывает все детали IPL и тем более не заменяет техническую документацию. Она объясняет, почему список оказался удобен для выражений, где элементы вложены и отношения между ними важнее простого соседства знаков.

Это решение имело цену. Списки и указатели требовали памяти и дополнительных действий. Программисту приходилось следить за тем, куда ведут связи и как новая структура соотносится со старой. Но именно такая гибкость делала возможной работу с символьными объектами. Язык программирования здесь не был декоративным инструментом, выбранным после того, как теория уже готова. Он влиял на то, какие процедуры можно было описать и сколько усилий потребует их исполнение.

Система IPL развивалась вместе с Logic Theorist, а не возникла как универсальный готовый инструмент. Команде понадобились операции, подходящие для её задач, и на их разработку уходили силы, которые иначе можно было бы потратить на доказатель. Это знакомая закономерность: новый класс задач часто требует новых средств выражения. Если язык умеет удобно складывать числа, но неудобно представлять вложенные формулы, разработчик сначала строит недостающий слой, а уже потом может сосредоточиться на самом рассуждении.

Эта работа особенно важна для роли Клиффа Шоу. Он знал машину, её инструкции и то, как собрать пригодный для исполнения язык. Саймон и Ньюэлл формулировали процедуры и исследовательские цели, но их реализация зависела от того, можно ли сделать из символической идеи конкретное представление в памяти. Вклад Шоу нельзя свести к тому, что он «перевёл» готовый алгоритм в код: сам перевод требовал решений о структуре данных, организации операций и поведении программы.

IPL показывает, почему ранняя история искусственного интеллекта — одновременно история алгоритмов и история инфраструктуры. Когда новая идея требует непривычного способа представления данных, программистам приходится создавать средства, с которыми потом

смогут работать и другие исследователи. Этот труд обычно исчезает, как только инструмент становится привычным. Но без него мысль так и осталась бы записью в блокноте.

Машина в комнате RAND

Logic Theorist запускалась на JOHNNIAC — компьютере, созданном в RAND Corporation по проекту класса машин, восходившему к разработкам Института перспективных исследований. Название отдавало дань Джону фон Нейману, чьи идеи повлияли на устройство компьютера. Но JOHNNIAC не был универсальной абстракцией, в которой программа существует сама по себе. Это была конкретная машина, построенная и обслуживаемая людьми в конкретном исследовательском центре в Санта-Монике.

Мы привыкли думать о программе как о файле, который можно переслать и открыть почти на любом устройстве. В 1956 году перенос программы между компьютерами не был лёгким действием. Разные машины имели свои инструкции и ограничения; язык высокого уровня, который скрывает особенности оборудования, только зарождался. JOHNNIAC работал с электронными компонентами, но вычислительный процесс по-прежнему был тесно связан с машинным кодом, устройствами ввода-вывода и доступом к ограниченной памяти.

Это меняло повседневный масштаб разработки. Участники не могли бесконечно запускать эксперимент на личном компьютере, мгновенно менять строку и повторять попытку. Чтобы узнать, где система ошиблась, нужно было подготовить программу, подать ей данные, получить печать или другой результат и разобраться, связано ли поведение с логикой поиска, ошибкой кодирования или особенностью машины. Программирование было не только написанием инструкций, но и организацией доступа к вычислительному ресурсу.

В таких обстоятельствах тезис «машина умеет доказывать теоремы» означал сразу несколько вещей. Во-первых, исследователи сумели формально описать задачу. Во-вторых, нашли способ выразить процедуру в программном языке. В-третьих, машина смогла выполнить её в имеющихся аппаратных ограничениях. Наконец, её результат можно было прочитать и проверить. Если не различать эти уровни, легко приписать успех одному изобретению, хотя он возник на стыке логики, психологии, программирования и инженерии.

Есть соблазн представить оборудование просто препятствием для большого замысла: не хватало памяти, машина была медленной, и всё же исследователи победили. Но ограничения не только мешали. Они заставляли искать способы не тратить время на полный перебор, выбирать представление выражений, контролировать объём поиска. Это не значит, что ограниченный компьютер автоматически порождает интеллект. Скорее, исследователи были вынуждены сделать вопрос управления вычислениями явным.

Здесь проявляется материальная сторона научной идеи. Набор символов не перемещается по невидимой доске сам собой. Его нужно закодировать, разместить в памяти, обработать и вывести так, чтобы человек мог оценить результат. Если программа должна моделировать рассуждение, сначала нужно, чтобы машина надёжно выполнила инструкции. Между формулой на странице и доказательством на распечатке лежала работа Шоу, инженеров RAND и всех тех, кто поддерживал машину в рабочем состоянии.

Поэтому выражение «Logic Theorist работала на JOHNNIAC» не просто добавляет марку компьютера в историю. Оно напоминает, что первая демонстрация машинного рассуждения происходила в среде, где вычисления были осязаемым ресурсом. Результат не был бесплатным и мгновенным. Его приходилось строить через программную и аппаратную инфраструктуру, которой почти не существовало до того, как она понадобилась самой задаче.

Знать правила — не значит знать, какой ход сделать

Если доказательство формально корректно, его можно проверить. Но написать программу, которая проверяет уже готовую цепочку, и написать программу, которая сама её находит, — разные задачи. Проверяющая процедура движется по одному предложенному маршруту. Поисковая должна решать, какой из множества разрешённых маршрутов стоит исследовать, когда продолжать, когда вернуться назад и как понять, что текущая ветвь бесперспективна.

Удобно представить не один лабиринт, а развилку, за которой стоят новые развилки. Каждый допустимый вывод создаёт потенциальное продолжение. Даже если правило известно, его можно применить к разным выражениям, разными способами подставить формулу вместо переменной и получить разные результаты. Все полученные выражения могут быть законными, но большинство из них не ведёт к теореме, которую требуется доказать. Глубина поиска быстро увеличивает число возможных путей.

Полный перебор выглядит надёжно: пробовать всё, пока цель не появится. Если пространство конечно и процедура перечисляет каждый вариант, такой метод в принципе может найти доказательство, если оно там есть. Но «в принципе» не оплачивает вычислительное время. Если число вариантов растёт очень быстро, программа может не дойти до нужного шага за разумный срок. Поиск, который гарантирует полноту, может оказаться бесполезным на практике.

Ньюэлл и Саймон хотели понять, чем эффективный поиск отличается от слепого перечисления. Здесь и появляются эвристики — правила выбора, которые помогают оценить, какие варианты стоит рассмотреть первыми. Греческое слово «эвристика» связано с нахождением, но в компьютерной программе эвристика не волшебная подсказка и не доказательство правильности маршрута. Это практическая стратегия: она повышает шанс быстро продвинуться, но может увести не туда или пропустить решение.

Представьте, что вы ищете выход из незнакомого здания. Можно по очереди открыть каждую дверь, вне зависимости от того, куда она ведёт. Можно предпочесть коридор, на котором видны указатели, или проверить двери рядом с лестницей, потому что они чаще ведут к выходу. Второй подход использует опыт и предположения. Он экономит время, но указатель может быть устаревшим, а лестница — привести в подвал. Эвристика работает похожим образом: направляет внимание, но не гарантирует, что выбранная ветвь окажется удачной.

Для Logic Theorist это имело и психологическое значение. Исследователи связывали эвристический поиск со стратегиями решения задач у людей. Математик не применяет все возможные правила к каждой формуле: он замечает знакомые связи, выделяет подцели и сводит неизвестное к уже известному. Команда хотела описать такие приёмы настолько точно, чтобы выполнить их на компьютере и сравнить поведение программы с человеческими решениями.

В первой публикации об исследовании Ньюэлл, Шоу и Саймон подчёркивали интерес не к методу, гарантирующему решение любой ценой, а к эвристикам, которые могут справиться с задачей при ограниченных вычислительных средствах. Это важный выбор. Они не просто хотели получить ответ. Они хотели исследовать процесс, который производит ответ, потому что этот процесс мог рассказать что-то о человеческом решении задач.

Однако сходство на уровне общей стратегии ещё не устанавливает, что программа и человек думают одинаково. Если оба используют частичные цели, это не значит, что у них одни и те же внутренние переживания или знания. Чтобы модель была полезна, нужно указывать, какие именно действия она объясняет и какие наблюдения могли бы её опровергнуть. Logic Theorist давала исследователям объект для такого анализа: последовательность формальных шагов можно было сохранить, изучить и сопоставить с человеческими решениями.

В этом проекте эвристика стала признанием двух ограничений одновременно. Во-первых, пространство поиска слишком велико для механического перебора. Во-вторых, у исследователя нет универсального способа заранее знать, какой ход приведёт к успеху. Эвристика сокращает пространство за счёт предположений. Она делает поиск осуществимым и одновременно открывает дверь для ошибок. Скорость и риск здесь не противоположны: они рождаются из одного решения — не проверять всё.

Как выглядит один доказанный результат

Чтобы понять работу Logic Theorist, полезно отойти от больших слов и рассмотреть пример из публикации команды 1957 года. Исследователи просили программу доказать формулу, записанную как теорема 2.01 в соответствующем наборе из *Principia Mathematica*. Затем они показывали несколько шагов, в которых используются аксиома, подстановка и замена логических связей.

Правило подстановки позволяет заменить переменную в формуле другим выражением, соблюдая условие, что замена выполняется последовательно везде, где переменная встречается. Если исходная формула говорит о p , можно вместо p подставить составное выражение вроде « q или r » и получить другой экземпляр той же схемы. Правило замены позволяет переписать связку через её определение: например, импликация «если p , то q » выражается через отрицание и «или». Правило отделения, или *modus ponens*, говорит: если уже доказано p и доказано «если p , то q », можно заключить q .

Для читателя названия правил могут звучать как экзаменационный билет, но их роль проста. Каждое правило ограничивает, какие новые строки разрешено добавить к доказательству. Программа не могла просто написать убедительно выглядящую формулу и объявить дело закрытым. Она должна была связать цель с исходными аксиомами через разрешённые переходы.

В примере с теоремой 2.01 машина брала одну из аксиом, подставляла вместо переменной нужный символ, переписывала связку и применяла правило отделения. В отчёте говорится, что этот небольшой вывод был получен примерно за десять секунд вычислительного времени. Цифра зависит от конкретного запуска и машины того времени; её смысл не в сравнении скорости с современным процессором. Она показывает, что после задания аксиом и правил программа сама выдала цепочку, которую можно сверить.

Пример важен также потому, что демонстрирует разницу между проверкой и поиском. Когда читатель видит готовую последовательность, каждый шаг кажется почти очевидным. Но машине нужно было решить, какую аксиому взять, какую замену попробовать и когда применить правило. В отчёте печатается результат, но работа программы заключалась не в переписывании уже предложенного человеком решения. Она порождала варианты и оценивала их по процедуре поиска.

Более сложный пример показывает, как ситуация меняется с глубиной. В публикации авторы описывали, как Logic Theorist доказала теорему 2.45 примерно за двенадцать минут, используя набор из 38 ранее полученных результатов, доступных к этому моменту. Здесь поиск уже зависел не только от нескольких исходных аксиом: накопленный запас теорем расширял возможности программы. Каждое ранее доказанное утверждение становилось новым инструментом, хотя использование большого набора также усложняло выбор.

Есть и более важная деталь: машина не всегда находила доказательство. Для теоремы 2.31, приведённой в том же исследовании, программа работала около 23 минут, а затем сообщала, что исчерпала доступный ей поиск и не смогла доказать утверждение. Это не было доказательством того, что теорема неверна или что доказательства не существует. Это означало только, что данная процедура, с данным управлением поиска и при доступных ресурсах, не нашла путь.

Такой отказ — не примечание мелким шрифтом. Он помогает понять, что именно демонстрировала Logic Theorist. Машина могла находить проверяемые доказательства, но её успех зависел от представления задачи, набора доступных теорем, выбранных эвристик и пределов вычисления. Если показывать только найденные решения, легко принять программу за более универсальную, чем она была. Включённая в раннюю публикацию неудача задаёт более чест-

ную картину: программа была экспериментальной системой, чьи возможности и слабые места исследователи пытались измерить.

И наконец, доказательство остаётся доказательством только в рамках выбранной формальной системы. Человек проверяет не то, «поняла ли» машина смысл всей математики, а то, следует ли итоговая формула из исходных посылок по допустимым правилам. Это не умаляет достижение. Это точно называет его.

Британский музей и цена полного перебора

В статье 1957 года Ньюэлл, Шоу и Саймон предложили мысленный эксперимент, чтобы показать, почему доказательства нельзя искать простым перечислением. Они назвали его «алгоритмом Британского музея»: представьте, что все возможные последовательности логических выражений уже где-то собраны, а задача состоит в том, чтобы найти среди них цепочку, заканчивающуюся нужной теоремой и соответствующую правилам доказательства.

Название отсылает к образу огромной коллекции, в которой якобы можно отыскать всё, если достаточно долго искать. В этой статье авторы раскладывали решение задачи на две части. Генератор производит возможные ответы, а проверка определяет, является ли каждый из них решением. Для кодового замка генератор предлагает комбинации, а замок подтверждает или отвергает попытку. Для доказательства генератор строит последовательности формул, а формальные правила позволяют проверить, является ли последовательность доказательством нужной теоремы. Такое разделение помогало оценить каждую часть отдельно: сколько стоит породить очередной вариант, насколько быстро его можно проверить и как изменить порядок перебора.

Это различие многое объясняет в истории Logic Theorist. Проверить готовую цепочку можно сравнительно механически: убедиться, что начальные строки относятся к аксиомам или прежним теоремам, а остальные получены допустимыми переходами. Труднее придумать генератор, который не тратит всё время на бесполезные последовательности. Программа должна была производить варианты и фильтровать их, но её эвристики отвечали за то, какие из них появятся раньше и какие подцели будут поставлены. В такой схеме корректность доказательства и эффективность поиска — разные качества. Система могла быть совершенно надёжной в проверке и всё же долго не находить решение.

Когда авторы сравнивали подходы, они потому обсуждали не только число теорем, но и стоимость пути к ним. Можно гарантировать, что решение рано или поздно появится, и одновременно получить такой счёт вычислительного времени, при котором гарантия теряет практический смысл. Можно искать выборочно и решить задачу быстро, рискуя пропустить доказательство. Эта рамка — кандидат, проверка и цена поиска — помогала обсуждать поведение программы точнее, чем слово «умная». Такой алгоритм был бы надёжен в принципе: он перебирал бы варианты один за другим, проверял бы каждый и когда-нибудь нашёл бы доказательство, если оно существует. Но скорость делала гарантию пустой для практики. Авторы оценивали, что один из более сложных примеров мог занять при таком переборе тысячи лет вычисления, а расширение поиска на весь набор задач *Principia Mathematica* требовало бы колоссального времени. Это были грубые оценки по тогдашним методам и машинам, а не измерение современной вычислительной сложности, но мысль ясна: возможность перечислить всё ещё не означает, что ответ будет найден вовремя.

Для сравнения они предложили простую задачу — открыть кодовый замок. Можно перебирать все комбинации, проверяя каждую, пока замок не откроется. Если код конечен и комбинации исчерпываются, метод гарантирует успех. Но если комбинаций очень много, охранять ценности при помощи замка с таким кодом всё ещё имеет смысл: знать принцип перебора — не то же самое, что успеть его завершить. Этот пример не был описанием самой Logic Theorist. Он показывал разницу между алгоритмом, гарантирующим ответ, и эвристикой, которая может найти ответ быстрее, но не обещает, что справится во всех случаях.

Команда не считала, что эвристика автоматически лучше. Для некоторых задач надёжный алгоритм — правильный выбор, даже если он требует времени. Но когда полный поиск непосилен, приходится принимать риск: программа может не найти решение, хотя оно существует. Работа становится похожа на повседневное решение задач: ограниченный исследова-

тель выбирает, какой вариант кажется более перспективным, и не может одновременно проверить всё.

В этом и заключалась научная ставка проекта. Можно было изучать не только правильные правила логики, но и правила, которые управляют вниманием: как сузить поиск, какие подцели поставить, когда признать, что путь не работает. Поиск доказательства становился моделью решения задачи — не потому, что доказательство и всякое человеческое мышление одно и то же, а потому, что здесь ход работы можно было формально описать и затем наблюдать.

В наши дни слово «алгоритм» часто используется как синоним любой автоматической процедуры. В статье 1957 года авторы аккуратно разделяли гарантированный поиск и эвристический поиск. Различие полезно и сейчас. Алгоритм в их узком смысле обещает найти решение, если оно есть, но может быть слишком медленным. Эвристика ограничивает размах поиска и может решить задачу вовремя, но способна потерпеть неудачу. Умение выбирать между гарантией и практической осуществимостью само становится частью инженерного решения.

Четыре способа искать дорогу к теореме

Logic Theorist не была одной универсальной догадкой. В описании 1957 года авторы разбирали несколько методов, которые могли по-разному сократить поиск. Для широкой аудитории важнее понять их логику, чем запомнить названия.

Первый путь — подстановка. Если аксиома задаёт общую схему, в неё можно подставлять конкретные выражения и получать новые утверждения. Это похоже на шаблон договора: сам бланк общий, но когда на места переменных вписывают имена и условия, появляется конкретный экземпляр. Подстановка не означает произвольной замены любого символа. Нужно соблюдать структуру формулы, чтобы вывод оставался корректным. Такой способ способен сразу производить новые теоремы, но число возможных подстановок быстро растёт.

Второй путь — отделение. Если известны утверждение А и условная формула «если А, то В», можно вывести В. Это знакомое рассуждение: если поезд отправится в 10:00, он успеет к пересадке; поезд отправится в 10:00; значит, пересадка возможна. В формальной системе примеры выражаются символами, но логическая связь та же. При поиске доказательства программа могла рассмотреть цель В и искать такое уже известное утверждение, которое говорит «А влечёт В». Тогда А превращается в подцель: сначала надо доказать её, чтобы получить В.

Третий и четвёртый методы в статье — цепочки вперёд и назад по импликациям. Если задача требует доказать, что А влечёт С, можно искать промежуточное утверждение В: найти уже известное «А влечёт В», а затем проверить, можно ли показать «В влечёт С». Или двигаться с другого конца: искать «В влечёт С» и сводить задачу к «А влечёт В». Такие переходы напоминают сбор моста с обоих берегов. Когда встречные участки соединяются, появляется доказательство. Если выбранные опоры не совпадают, приходится искать другую промежуточную точку.

Ни один метод не «понимает» теорему. Каждый задаёт способ породить возможное продолжение или подзадачу. Результат зависит от того, какие выражения уже известны, какие подстановки разрешены и как организована очередь поиска. Команда могла запускать методы по очереди, а один метод — работать над подзадачами, созданными другим. В этом смысле программа была не просто коллекцией логических правил, а управляющей системой, которая определяла вычислительное время между возможными направлениями.

Здесь особенно заметно различие между поисковым механизмом и критерием проверки. Механизм предлагает кандидатов; проверка определяет, допустим ли переход и достигнута ли цель. Можно менять очередность, стратегии и эвристики, не меняя математические правила корректности. Если программа нашла доказательство, оно не становилось правильным лишь потому, что эвристика считала путь перспективным. Каждое звено всё равно должно было пройти проверку.

Эти методы дают и более точное объяснение фразы «программа рассуждала». Logic Theorist не получала от человека готовое доказательство, которое лишь оформляла. Она строила варианты на основе формальных операций и пыталась связать цель с аксиомами и известными теоремами. Но пространство вариантов, правила перехода и сама цель были заданы заранее людьми. Программа самостоятельно выбирала следующий вычислительный шаг внутри созданной исследователями рамки. Именно это, а не таинственное появление мысли, было содержанием демонстрации.

Когда компьютер пропускает решение, а находит другое

Самым памятным эпизодом Logic Theorist часто называют теорему 2.85. По рассказам участников и последующим историческим исследованиям, программа построила доказательство, которое заметно отличалось от опубликованного в *Principia Mathematica* и оказалось короче или проще по структуре. Точность пересказа здесь важна: не нужно говорить, будто машина «обнаружила ошибку» в фундаментальной книге или превзошла человеческий разум вообще. Она нашла другую формально допустимую цепочку для конкретного результата в заданной системе.

Особенность состояла в том, что программу не писали специально ради создания более красивых доказательств. Исследователи задавали ей правила поиска, а не эстетический критерий «сделай элегантнее, чем Рассел». Новая цепочка появилась как побочный результат процедуры, которую строили для поиска доказательств и исследования эвристик. Иногда нужный эффект возникает без того, чтобы кто-то отдельно программировал его как цель. Программа не имела вкуса к краткости; она исследовала маршруты, и один из них оказался компактнее известного.

Этот результат помогает отделить механическую корректность от человеческой оценки качества. Корректность можно формализовать: каждый переход разрешён. Краткость тоже можно измерить, например числом шагов. А вот «элегантность» включает суждение математиков о том, почему доказательство хорошо устроено и что оно проясняет. Работа программы не отменяла человеческую оценку, но заставляла её обратить внимание на результат, которого раньше не было в привычном наборе решений.

Герберт Саймон отправил Бертранию Расселу описание работы. Рассел ответил письмом от 2 ноября 1956 года, что рад узнать: *Principia Mathematica* теперь можно выполнять с помощью машины, и готов поверить, что дедуктивную логику в принципе можно поручить механизации. Для Рассела это был не просто любопытный трюк с компьютером. Он десятилетиями работал над формальными основаниями математики, и теперь один из методов, описанных в книге, исполнялся машиной.

Ответ не означает, что Рассел признал всякую человеческую мысль сводимой к вычислению. Речь шла о дедуктивной логике — формальном фрагменте, который сама книга стремилась описывать правилами. Важен масштаб: то, что философы и математики годами обсуждали на бумаге, стало процедурой, чьи шаги генерировала машина и проверяли люди.

Для публики и профессионального сообщества это был эффектный сюжет. Машина не просто повторила арифметическое вычисление быстрее человека. Она нашла символический путь от аксиом к теореме. Но самый ценный итог был не в соревновании «кто умнее». Программа показывала, что компьютер способен участвовать в работе с формальными доказательствами, а исследователь может изучать эту работу как последовательность операций. Возможность доказать теорему становилась наблюдаемым экспериментом.

Результат относится к поиску в ограниченной формальной системе. Logic Theorist могла найти путь, который разработчики заранее не записали как ответ, — значимое свойство программы. Это показывает, что процедура способна производить новый и проверяемый результат, полезный людям. Оценка красоты, неожиданности и математической важности по-прежнему оставалась за человеком.

Программа была и моделью человеческого решения

Для Ньюэлла и Саймона Logic Theorist была не только автоматическим доказателем. Они рассматривали её как модель сложной обработки информации: система получает задачу, хранит представление о ней, выбирает операцию, оценивает результат и продолжает поиск. Если человеческое решение задач состоит хотя бы частично из таких операций, их можно попытаться формализовать и проверить на машине.

Такой взгляд отличался от идеи, что психологию следует изучать только по внешней реакции человека. Команда хотела говорить о процессах между условием и ответом: какие подцели человек строит, какие ходы рассматривает, как замечает знакомую структуру. Компьютер помогал сделать эти гипотезы явными. Вместо фразы «человек каким-то образом нашёл решение» исследователь мог записать предполагаемую процедуру, реализовать её и посмотреть, выдаёт ли она ответы в наблюдаемых условиях.

Здесь легко запутаться в значении слова «модель». Модель может быть похожа на объект в одном отношении и сильно отличаться в остальных. Карта метро точно показывает связи между станциями, но не передаёт длину туннелей или запах платформы. Logic Theorist могла моделировать поиск по подзадачам и использование эвристик, не моделируя человеческие чувства, биографическую память или опыт математического образования. Для научной модели это нормально, если исследователи ясно называют, какую часть поведения она объясняет.

В статье 1957 года авторы сами ограничили вывод: в ней они разбирали поведение программы на формальной логической задаче, а последствия для психологической теории человеческого мышления оставляли для дальнейшей работы. Эта осторожность часто исчезает в поздних пересказах, где программа объявляется прямой копией человеческого разума. На деле создатели использовали машину как инструмент исследования, а не утверждали, что нашли завершённую теорию психики.

Дальнейшие исследования команды включали наблюдение за тем, как люди решают задачи. Одним из методов стало «мышление вслух»: участникам предлагали произносить то, что приходит им в голову во время решения, а исследователи записывали ход работы и кодировали его. Такой метод требовал отдельной проверки. Разговор человека вслух может влиять на решение; слова не открывают автоматически все скрытые процессы; люди могут не замечать собственных привычек. Но полученные протоколы помогали сравнивать конкретные последовательности действий человека и программы.

Это сравнение не предполагало, что компьютер — эталон, а человек — несовершенная копия. Иногда программа успешно решала формальную задачу, но использовала стратегию, которую люди не применяли. Иногда человеческие участники находили способы, отсутствовавшие в программе. Такие различия были исследовательским материалом: они показывали, какие части модели согласуются с наблюдениями, а какие требуют пересмотра.

Здесь находится один из важных результатов Logic Theorist, который легко потерять за рекордом «первая программа доказала теорему». Компьютер дал способ проверять теории о мышлении с помощью работающих систем. Программа стала не только инструментом для решения задач, но и лабораторным объектом, чьё поведение можно было менять, сравнивать и объяснять. Для психологии и ранней когнитивной науки это был новый тип эксперимента.

Но модель и человек оставались разными вещами. У программы были правила, структуры данных и вычислительное время; у человека — тело, опыт и сложная способность ориентироваться в ситуации. Даже если некоторый ход поиска совпадал, это не доказывало совпадение внутреннего мира. Logic Theorist показала, что имитация определённой деятельности может быть научно полезной без заявления, будто машина приобрела сознание.

Что именно показали 38 доказательств

Позднейшие краткие истории часто сводят успех Logic Theorist к одной цифре: программа доказала 38 из первых 52 теорем, предъявленных ей из второй главы *Principia Mathematica*. Эта цифра опирается на исторические свидетельства и стала удобным способом передать масштаб результата. Но без пояснения она звучит так, будто машине дали произвольную книгу, а она прочитала и поняла её. В действительности человеческая подготовка была значительной.

Исследователи выбрали формальную систему, задали символическое представление, подготовили аксиомы и правила вывода, формализовали теоремы и написали процедуры поиска. Человек определил, какие фрагменты книги окажутся входными данными и что будет считаться корректным доказательством. Программа не сама выбрала *Principia Mathematica* в библиотеке, не извлекла из страниц математическую структуру и не сформулировала вопрос, который надо исследовать.

Это не обесценивает результат. Научные инструменты обычно требуют, чтобы люди подготовили эксперимент. Телескоп не выбирает астрономическую гипотезу; микроскоп не решает, какие клетки сравнивать. Важен точный вклад системы в процедуру: после того как цель и формальные правила были заданы, Logic Theorist искала цепочки, а не просто воспроизводила заранее введённые доказательства. Проверяемый шаг за пределами ручной подготовки и был центральной демонстрацией.

Число 38 также не означает, что остальные четырнадцать теорем были неверными или недоказуемыми. Как показывает пример теоремы 2.31, программа могла исчерпать свой поиск, не найдя доказательства. Её эвристики были неполными: они направляли поиск к многообещающим вариантам, но не гарантировали, что в каждом случае путь будет найден. Кроме того, результат зависел от того, какие ранее выведенные теоремы система могла использовать. Накопленные доказательства расширяли её арсенал.

Понятие «первый успешный ИИ» поэтому полезно только как историческая метка с оговорками. У Logic Theorist была конкретная задача, конкретный формализм и конкретный набор ресурсов. Она не умела читать произвольную математическую статью, самостоятельно выбирать аксиоматику, доказывать каждое утверждение или объяснять его человеку обычным языком. Она демонстрировала возможность машинного поиска доказательств и моделирования некоторых стратегий решения задач. Это много — именно потому, что мы не приписываем ей больше.

Демонстрация также не зависела от того, победила ли машина человека в прямом соревновании. Её целью не был турнир, где один математик решает задачи, а компьютер должен получить более высокий счёт. Исследователи тестировали гипотезу о том, что процесс решения можно представить как программу и исследовать опытным путём. Успех состоял в том, что система действительно производила доказательства, а исследователи могли анализировать и улучшать процедуру.

С этой точки зрения, ограничения программы были не провалом замысла, а частью результата. Она помогла увидеть, какие элементы формальной задачи легче автоматизировать, как быстро растёт поиск, почему нужны эвристики и что приходится задавать вручную. Если бы Logic Theorist доказала все теоремы без единой ошибки, это всё равно не решило бы вопрос о мышлении в общем. А её реальные неудачи позволили поставить более содержательные вопросы о границах подхода.

Сильная демонстрация в очень узком мире

У формальной логики есть необычное преимущество: мир уже подготовлен к вычислению. Теоремы записаны в символах, правила перечислены, цель известна. Программа не должна видеть страницу, переводить человеческую речь в формулы или решать, какие детали ситуации важны. Большую часть трудной работы по постановке задачи выполняли люди до запуска системы.

Рассмотрим бытовую инструкцию: «Положи чашку на стол, который не шатается». Чтобы выполнить её, человек должен определить, какая чашка имеется в виду, найти стол, понять, что считается шатанием, оценить расстояние, поднять предмет и заметить, если стол занят. В формальной системе все объекты и отношения уже записаны в допустимом языке. Вход не приходит в виде потока неясных слов и сенсорных сигналов; он появляется как тщательно подготовленная формула.

Поэтому способность доказывать теоремы не переносится автоматически на открытые задачи. В доказательстве заранее известен набор разрешённых операций. В реальной жизни часто сначала надо понять, какие операции вообще уместны. В логике программа проверяет структуру символов. В разговоре те же слова могут менять смысл из-за контекста, интонации, общего прошлого и неполноты фразы. Формальный успех не отвечает на вопрос, способна ли машина построить правильное представление сама.

Эта граница не уникальна для Logic Theorist. Почти всякая сильная программа сначала получает задачу в более аккуратной форме, чем ту, с которой сталкивается человек. Для автоматизации важна не только сила алгоритма, но и труд по преобразованию реальности в формат, где алгоритм может действовать. Если не учитывать эту подготовку, система кажется автономнее, чем она есть.

Но узость задачи одновременно была преимуществом эксперимента. Когда программа ошибалась, исследователи могли отследить ход поиска и понять, связано ли затруднение с применением правил, выбором подцели или ограниченным ресурсом. В открытой среде причин сбоев было бы слишком много: неверно распознанный объект, неполный контекст, двусмысленное указание или неудачная последовательность действий. Сначала нужно было получить работающий результат в мире, который возможно контролировать.

Так Logic Theorist воплощала характерный обмен. Чем точнее ограничена задача, тем убедительнее можно проверить способность машины; чем шире и ближе задача к повседневной человеческой деятельности, тем меньше заранее известных правил и тем труднее установить, что именно программа поняла или сделала. Это не дилемма с одним правильным ответом. Исследователям нужны были и чистые эксперименты, и способы постепенно расширять их границы.

Позднее этот метод назовут символическим подходом: знания и задачи в нём представлены символами, связями и явными правилами. Logic Theorist была особенно наглядным примером, поскольку работала с формулами и операциями над ними. За пределами формальной логики обнаружится следующая трудность: повседневная деятельность опирается на огромное количество фактов, правил и исключений, которые сложно перечислить заранее. В 1956 году команда показывала, что программа может искать в пространстве формальных рассуждений, а не только выполнять числовые вычисления.

После доказательства остаётся вопрос о разуме

Logic Theorist поставила старый спор о машинном разуме на новую основу. Одни могли счесть корректное доказательство признаком рассуждения; другие — указать, что программа манипулирует символами без человеческого понимания их значения. Результат не обязывал принять одну философскую позицию.

И это нормально. Термин «интеллект» охватывает разные вещи: способность решать задачи, знание, адаптацию, понимание, сознательный опыт. Logic Theorist свидетельствовала об одной способности — поиске формальных доказательств. Она не давала оснований считать, что система обладает всеми свойствами, которые человек связывает с математиком. Чтобы утверждение было честным, его нужно соразмерять с задачей, на которой получено доказательство.

Фраза «программа понимала логику» требует уточнения. Она хранила формулы, выполняла правила и находила проверяемые выводы. Назвать такую работу пониманием или вычислением — вопрос о том, какие признаки мы считаем достаточными для понимания. Технический отчёт не может решить философскую проблему одним словом.

Сила ранней программы как научного события состояла в переходе от общего спора к конкретным наблюдениям. Система перебирала варианты, строила последовательности вывода, находила одни теоремы, не справлялась с другими и порой предлагала путь, неожиданный для разработчиков. Обсуждение машинного мышления теперь могло опираться на поведение работающей системы.

Есть риск качнуться в любую крайность. Если объявить программу просто механическим трюком, исчезает реальный результат: она действительно производила доказуемые цепочки, и эти операции можно было исследовать. Если объявить её математиком в полном человеческом смысле, исчезают ограничения: система зависела от подготовленной формальной среды, заданных правил и написанной людьми процедуры. Точнее удерживать обе части: способность была реальной, а общий вывод о разуме — открытым.

Такой способ разговора и сегодня важен. Когда алгоритм выполняет полезную работу, можно признать эту способность, не приписывая ему сознание. И можно исследовать механизм, не отказываясь от больших вопросов. Logic Theorist не заставила философов закрыть спор. Она сделала его предметным и дала инженерам новый объект для экспериментов.

От программы к исследовательской программе

В 1956 году Ньюэлл и Саймон представили Logic Theorist как «сложную систему обработки информации». Заголовок подчёркивал не мистику разума, а инженерное устройство: программа получает формальные выражения, проводит над ними операции и строит последовательность, которая может стать доказательством. Публикация в журнале по теории информации сообщала специалистам, что произошло, но сама по себе не делала систему общепринятым объяснением мышления.

В 1957 году Ньюэлл, Шоу и Саймон описали серию экспериментов с программой и рассмотрели эвристики как предмет исследования. Теперь читатель видел не только общее описание машины, но и примеры её поведения: какие теоремы она находила, где терпела неудачу, что означает гарантия решения и почему простой перебор становится неприемлемым. Это важное изменение жанра. Идея уже не оставалась частным замыслом трёх исследователей: она превращалась в программу опытов, которую можно обсуждать, критиковать и расширять.

В Дартмуте проект получил аудиторию, но не там впервые возник. Ньюэлл и Саймон уже вели работу, а команда располагала и программой, и первыми результатами. Презентация на летней встрече показала участникам, что формулировка предложения 1955 года может означать конкретную программу. Вместо общего обещания «изучать интеллект» можно было указать на работающую систему, рассказать, как она ищет доказательство, и предъявить результат для проверки.

Демонстрация не была исчерпывающим доказательством того, что машинный интеллект создан. Участники Дартмута увидели один исследовательский подход. Его значение — в следующем шаге: выяснить, какие ещё задачи можно представить через эвристический поиск и можно ли распространить программный метод за пределы доказательства теорем.

Команда вскоре двинулась в этом направлении. General Problem Solver — «универсальный решатель задач» — пытался работать не только с теоремами, но и с различными проблемами, используя более общие стратегии вроде анализа различия между текущим состоянием и желаемой целью. Это не означает, что программа стала универсальной в буквальном смысле или могла решать любую человеческую задачу. Название выражало амбицию искать более переносимый способ постановки подцелей и выбора действий. Logic Theorist стала для этой линии ступенью: проект помог выработать язык, идеи и исследовательский метод.

Тот переход был одновременно логичным и рискованным. Успех на одной задаче подталкивает расширить подход, но правила, удобные в формальной логике, не обязательно работают в повседневном мире. Чем шире задача, тем больше неясностей появляется до начала поиска: что считать исходным состоянием, как представить цель, какие действия доступны, чем измерять продвижение. Программа, созданная для небольшой системы формул, не могла просто взять с собой все правила и ожидать, что мир будет столь же аккуратно устроен.

Для молодой области это была продуктивная проблема. Logic Theorist демонстрировала, что программу можно изучать по внутреннему процессу, а не только по ответу. Ошибки и успехи становились данными, которые могли изменить следующую версию. То, что компьютер быстро доказывал некоторые формулы и не справлялся с другими, позволяло ставить более точные вопросы о представлениях, эвристиках, памяти и вычислительных затратах.

Машина вывела формулу. Учёные должны были вывести границу

Сила Logic Theorist не в том, что программа завершила философский спор о разуме. Её сила в том, что она заставила спор учитывать реальную работу машины. Компьютер выполнял процедуру поиска, выдавал доказательство, терпел неудачу в конкретных условиях, а люди могли проверить его результат и разобрать путь.

Это достижение имеет точные границы. Программа не выбирала собственную математическую проблему, не читала естественный язык *Principia Mathematica* и не оценивала, какие теоремы важны для науки. Люди формализовали вход, подготовили правила и разработали механизм. Внутри этой рамки программа делала больше, чем печатала заранее заданный ответ: она строила последовательность вывода и иногда находила неожиданный маршрут. Оба утверждения верны одновременно.

Эта двойственность и делает раннюю программу интересной сегодня. С одной стороны, она показывает, как много может дать точное представление задачи: символы, операции и критерии проверки создают пространство, где компьютер способен выполнять работу, прежде считавшуюся исключительно человеческой. С другой — напоминает о скрытом труде, который предшествует вычислению. Нужно решить, какие сведения превращаются в символы, какие отношения кодируются, какие правила доступны и какие результаты будут засчитываться.

Логика удобна для автоматизации благодаря явным правилам. В задачах с неформализованными знаниями и меняющимися условиями машине труднее выбрать представление. Logic Theorist обозначила эту границу: дальнейшие исследования будут проверять, задаёт ли человек правила системе или та способна выделять закономерности из примеров.

Следующая глава перейдёт к перцептрону, который предлагал иной путь. Исследователи изучали сеть простых вычислительных элементов, меняющих параметры по примерам, вместо программирования каждого шага доказательства вручную. Перцептрон не решал те же задачи, что Logic Theorist; он проверял, способна ли машина научиться различать объекты, корректируя поведение по ошибкам.

За этим переходом скрывается не простая смена старого подхода на новый. Logic Theorist показала силу явных правил и управляемого поиска. Перцептрон исследовал обучение и настройку по данным. Одна линия пыталась точно записать, что системе нужно делать; другая — выяснить, можно ли получить нужное поведение из множества простых изменений. В следующих главах эти направления будут пересекаться, спорить и менять свои роли.

Дартмутское название объединило их прежде, чем появилась общая теория, объясняющая, как они связаны. Logic Theorist стала одним из первых убедительных ответов на вопрос, может ли компьютер выполнить работу, связанную с рассуждением. Она не сказала, что такое интеллект целиком. Но после неё исследователям уже не нужно было обсуждать машинное рассуждение только как далёкую возможность: перед ними были программа, её шаги, её успехи и её пределы.

Глава 4. Нейрон без мистики

В июле 1958 года в зале сидели журналисты, перед ними стояла машина размером с комнату, а рядом лежала стопка перфокарт. Фрэнк Розенблатт показывал, как система различает, с какой стороны карточки напечатана отметка. После примерно пятидесяти предъявлений она стала чаще выбирать правильный ответ.

У демонстрации было два названия, и между ними — важная разница. Перцептрон запускался на IBM 704, большом компьютере, который обрабатывал перфокарты. Специальную машину, Mark I Perceptron, ещё только создавали. Позднейшие рассказы нередко слили программную демонстрацию и отдельный аппарат в один эффектный эпизод: будто журналисты уже видели работающий электронный глаз, самостоятельно научившийся читать. В действительности событие было скромнее и интереснее. Розенблатт показал не искусственного человека в шкафу, а вычислительный способ, который менял параметры под действием опыта. Mark I стал следующим шагом, материальным воплощением той же исследовательской программы.

Различие важно. Оно позволяет увидеть, что именно произошло в 1958 году — и не требовать от этого события невозможного. Перцептрон не понимал карточки, не узнавал предметы в человеческом смысле и не обладал собственными целями. Он решал задачу классификации: относил заданный вход к одному из классов. Новизна состояла в способе получить правило разделения. Вместо того чтобы вручную прописать ответ для каждого варианта, исследователь давал системе примеры, а та корректировала числовые связи между входом и ответом.

За этим скромным изменением скрывалась большая ставка. А что, если часть правил не нужно записывать по одному? Можно ли построить машину, которая находит их по опыту?

От доказательства к обучению

Предыдущая глава закончилась программой Logic Theorist. Она искала доказательство в пространстве формальных правил: брала утверждение, применяла разрешённые преобразования и проверяла, ведут ли они к нужной цели. Такой подход был мощным, но требовал, чтобы человек заранее задал язык задачи и правила перехода между выражениями.

Перцептрон предлагал другой способ действовать. Не перечислять машине все признаки, по которым левую отметку нужно отличить от правой, а показать набор примеров и позволить системе подобрать числовое разделение. Это не отменяло человеческого проектирования: люди по-прежнему задавали вход, классы и процедуру обучения. Но часть решения теперь находилась не в тексте программы, а в настройках, которые менялись во время работы.

Это различие звучит привычно, потому что сегодня мы часто встречаемся с обученными моделями. В 1950-х идея ещё не была повседневностью. Компьютер обычно воспринимали как исполнителя точной последовательности команд. Если нужно получить другой результат, меняют инструкцию. Розенблатт исследовал механизм, у которого программа задаёт процесс, а конкретные числовые параметры складываются из опыта.

Здесь полезно сразу договориться о словах. «Нейронная сеть» — не цифровая копия мозга, не сознание из проводов и не обязательно сложная система. В простейшем виде это набор вычислительных элементов, каждый из которых получает числовые сигналы, объединяет их и выдаёт результат. «Вес» — число, показывающее, насколько сильно один сигнал влияет на следующий. «Обучение» означает, что некоторые веса меняются по заданной процедуре, чтобы ответы на примеры становились точнее.

Само слово «нейрон» напоминает о биологии, но аналогия ограничена. Живые клетки передают сигналы в сложной химической и электрической среде; они меняют связи по множеству механизмов, работают во времени и встроены в тело. Перцептрон был математической моделью некоторых общих идей о сигнале, связи и адаптации. Его не следует принимать за реалистичный чертёж мозга. И всё же аналогия дала исследователям продуктивный вопрос: можно ли получить распознавание из взаимодействия множества простых элементов, не прописывая каждую реакцию отдельно?

Как простая модель выбирает

Представим плоскую карту и две группы точек. Одни обозначают карточки с меткой слева, другие — с меткой справа. Если превратить изображение в числа, у каждого примера появится набор входных значений. Самая простая модель придаёт каждому входу свой вес, складывает результаты и сравнивает сумму с порогом. Если сумма выше порога, она выбирает первый класс; если ниже — второй.

Геометрически это похоже на линию, которая разделяет карту на две области. Все точки по одну сторону получают одну метку, точки по другую — другую. В пространствах с большим числом входных признаков вместо линии возникает гиперплоскость — граница, которую трудно нарисовать, но легко задать формулой. Эта граница и есть краткий ответ модели на вопрос «как различать классы?».

Сначала веса могут быть выбраны случайно или заданы исследователем. Затем система получает пример и выдаёт ответ. Если ответ неверный, правило обучения меняет веса, двигая границу так, чтобы следующий похожий пример получил правильный класс. В простом учебном варианте можно вообразить, что каждому входу начисляют небольшой числовой «голос»: сигнал, связанный с нужным ответом, усиливают, а ошибочно поддержавший выбор ослабляют. Это лишь наглядная версия принципа; конкретные ранние перцептроны и разные алгоритмы обучения не сводятся к одной современной формуле.

Важный момент — система не получает готовое объяснение «почему слева». Она получает указание, какой ответ требовался для предъявленного примера, и корректирует числовую конфигурацию. После серии примеров веса могут кодировать закономерность, которую инженер не записывал в виде обычного правила. Зато по одному лишь набору весов не всегда легко пересказать эту закономерность человеческим языком.

У такого обучения есть естественная граница. Если два класса действительно можно разделить подходящей прямой или гиперплоскостью, однослойный перцептрон способен найти такую границу при определённых условиях и достаточно хороших данных. Если правильное решение требует более хитрого разделения, один слой бессилён. Проблема не в том, что машине не хватило старания. Ей просто дали геометрическую форму, которая не может выразить нужное правило.

Розенблатт интересовался не только распознаванием отдельных карточек. В его работах «перцептрон» обозначал целое семейство моделей и машин, призванных изучать связь восприятия, памяти и действия. Поэтому полезно разделять три вещи: идею адаптивных сетей, математическое описание в его статьях и конкретное устройство Mark I. Они связаны, но не тождественны. Когда газетный заголовок обещает «машину, которая учится», читатель может представить единое устройство с одной понятной схемой. В лаборатории же были отчёты, варианты сетей, программы и аппаратные эксперименты.

Психолог в авиационной лаборатории

Фрэнк Розенблатт пришёл к перцептрону через психологию. Он учился в Корнеллском университете, окончил его в 1950 году, а докторскую степень по психологии получил в 1956-м. Его интересовали восприятие, обучение и память — то есть вопросы, которыми занимаются психологи, физиологи и философы, а не только инженеры.

В Корнелле Розенблатт работал с вычислительным анализом данных и ещё до перцептрона разрабатывал Electronic Profile Analyzing Computer — систему, предназначенную для статистического анализа психологических профилей. Это не была ранняя нейросеть в сегодняшнем смысле. Но этот опыт показывает, что компьютер для него был не просто быстрым калькулятором. Он искал способы формализовать сложные явления, которые нельзя было свести к одной арифметической операции.

Затем его работа продолжилась в Корнеллской авиационной лаборатории в Буффало. В названии учреждения слышится отрасль, далёкая от психологии, но послевоенная наука регулярно соединяла исследования мозга с задачами обороны. Военные и государственные организации вкладывали средства в радары, навигацию, обработку сигналов и интерпретацию изображений. Специалистам приходилось быстро разбирать множество визуальных данных; естественным становился вопрос, нельзя ли часть распознавания поручить вычислительной машине.

Это институциональное окружение объясняет, почему абстрактный вопрос об обучении получил бюджет, инженеров и возможность построить оборудование. Office of Naval Research (Управление военно-морских исследований США) и Rome Air Development Center финансировали работу над Mark I. Но неверно превращать этот факт в сюжет, будто военные заказали готового автономного разведчика или машина сразу имела прикладную боевую функцию. Исследовательская программа изучала модели восприятия и памяти и возможность их аппаратной реализации; потенциальные применения лежали впереди и не совпадали с тем, что устройство реально умело в лаборатории.

В 1957 году Розенблатт выпустил технический отчёт «Перцептрон: воспринимающий и распознающий автомат». Через год в журнале *Psychological Review* вышла более теоретическая статья, где он описывал перцептрон как вероятностную модель для хранения и организации информации в мозге. Это были разные типы текстов. Отчёт обсуждал техническую осуществимость и проектную программу; статья связывала формальный аппарат с вопросами психологии. В обоих случаях он пытался не доказать, что мозг уже разгадан, а построить модель, на которой можно проверять гипотезы о том, как простые связи могут поддерживать обучение.

Розенблатт был необычным посредником между лабораторией и публикой. Он мог говорить о математике, психологии и будущих применениях. Такое положение помогает понять ранний оптимизм: он видел в перцептроне не только один алгоритм классификации, но и направление исследований. Одновременно широкая амбиция делала особенно важным различие между тем, что аппарат демонстрировал, и тем, что ещё предстояло выяснить.

IBM 704 и стопка карточек

Вернёмся в июль 1958 года. Журналисты собрались на демонстрацию, организованную при участии Office of Naval Research. Перед ними был не современный экран и не маленький настольный компьютер, а IBM 704 — машина весом в несколько тонн, занимавшая большое помещение. Данные поступали на перфокартах, где отверстия кодировали информацию для компьютера.

Задача была зрительно простой: различать карточки, на которых отметка стояла слева или справа. Класс определялся положением небольшого элемента, а не богатым содержанием картинки. После примерно пятидесяти учебных примеров программа стала выполнять это различие. Публика увидела, как компьютер изменяет свою работу на основании примеров, вместо того чтобы механически выполнять один заранее написанный ответ для каждой карточки.

В газетной истории здесь возникло расхождение между скоростью заголовка и скоростью постройки. Позднейшая колонка *The New Yorker* специально уточняла: продемонстрированная система пока была смоделирована на IBM 704; специализированный Mark I ещё не был готов. Физический перцептрон создавался примерно в то же время, и музейная датировка Smithsonian указывает 1958 год. Но это не один и тот же объект. Программный эксперимент доказал, что процедуру можно выполнить на существовавшем компьютере; отдельный Mark I показывал, что её можно воплотить в специальной аппаратуре.

Такое разделение не делает демонстрацию менее значимой. Напротив, оно даёт хороший урок о том, как появляются технологии. Сначала идею проверяют на доступной платформе; параллельно строят устройство, которое должно соответствовать новой концепции. Однако фотогеничный образ — «электронный мозг с глазом, который учится» — легко вытесняет невидимую разницу между программой, вычислительным оборудованием и обучаемым правилом.

В лаборатории карточку подбирали под задачу. Исследователь решал, какие визуальные признаки попадут на вход, как закодировать примеры, сколько их предъявить, какой ответ считать правильным и когда закончить обучение. Отметка слева или справа оказалась удачной демонстрационной задачей именно потому, что её можно объяснить и проверить без длинного рассказа. Задача была простой, результат наблюдаемым, а изменение параметров — достаточно наглядным для газетного отчёта.

Но способность выучить обучающую подборку ещё не равна универсальному зрению. Любая система может запомнить примеры, если их мало и они однообразны. Более серьёзная проверка — получит ли она правильный ответ на новые карточки, которые отличаются освещением, размером, наклоном или фоном. Как меняется точность, когда этих случайных изменений много? Как узнать, научилась ли машина общему признаку, а не особенностям конкретного набора? Это вопросы об обобщении, и они сопровождают машинное обучение с первых лет.

Слово «учится» потому требует уточнения. В опыте 1958 года оно обозначало настоящий технический процесс: числовые связи корректировались по примерам. Оно не означало, что машина сама придумала задачу, распознала её смысл или перенесла навык на всё разнообразие мира. Научный результат был уже того газетного будущего, которое люди могли вообразить, но гораздо существеннее простой заранее прописанной сортировки. У машины появился способ менять параметры по обратной связи.

Маршрут сигнала внутри Mark I

У физического Mark I Perceptron была более сложная схема, чем одна формула на доске. Система связывала три вида элементов: сенсорные (S), ассоциативные (A) и ответные (R). Такой язык напоминает нейрофизиологию, но обозначает конкретные части модели. Сенсорные элементы получали вход; ассоциативные объединяли сигналы; ответные формировали один из вариантов вывода.

Представить маршрут можно как три этапа. Светочувствительные элементы превращали рисунок в электрические сигналы. Сигналы проходили к ассоциативным элементам, где некоторые комбинации могли активироваться сильнее других. Затем система передавала результат на выход, который показывал выбранный класс. Это не была камера, которая сохраняла целостную фотографию и затем «вспоминала» её по-человечески. Изображение превращалось в набор ограниченных сигналов, а ответ зависел от пути и силы связей.

О музейном аппарате часто пишут, что он имел сетку из 400 светочувствительных элементов — своего рода входную сетчатку 20 на 20. Такая сетка хорошо показывает, насколько грубым был зрительный вход, если сравнивать его с человеческим зрением. Но количество сенсоров само по себе не говорит, что машина распознавала. Разрешение изображения, схема соединений, настройка, задача и способ обучения вместе определяли, что система могла сделать.

Здесь особенно легко ошибиться, вообразив, что «все провода обучались». В Mark I не каждая связь свободно меняла свою силу. Архитектура включала фиксированные соединения от S-элементов к A-элементам и регулируемые связи между A и R. Части схемы задавались конструкцией, а часть весов могла изменяться во время обучения. Именно поэтому не стоит без оговорок переносить на исторический аппарат компактную формулу современного однослойного классификатора, где все параметры описаны одним и тем же набором символов. Mark I был устройством определённого типа, а «перцептрон» — более широким понятием.

Внешний вид машины тоже не был пустой декорацией. Устройство с реальными датчиками, проводами и переключателями позволяло исследователям изучать, какие части системы можно реализовать электронно, как физическая архитектура влияет на вычисление и какие формы обучения доступны. Аппаратная реализация задавала ограничения на скорость, число соединений, способ задания обратной связи и стабильность компонентов. В эпоху, когда вычисления стоили дорого, конструкция была частью научной гипотезы.

При этом физический размер мог производить впечатление, несоразмерное возможностям. Человек видел шкафы, лампы, сетку, похожую на глаз, и невольно заполнял пробелы воображением. В действительности внутри не сидел маленький наблюдатель. Электроника принимала сигнал, комбинировала его по заданной структуре и выдавала ответ. Это не уменьшает достижения инженеров. Просто оно помогает оценить его точно: необычным был не искусственный субъект, а экспериментальная машина для проверки того, можно ли изменить распознавание через опыт.

Что значило обучаться без примеров от всего мира

Принцип обучения на примерах сегодня связывают с огромными наборами данных. Mark I работал в совершенно иной среде. Изображение приходилось подготовить, образцы были ограничены, а сама вычислительная техника несравнима с нынешней. Вопрос состоял не в том, можно ли загрузить в машину миллионы фотографий, а в том, как связать несколько сотен входных сигналов с правильным выбором и затем скорректировать ограниченное число весов.

Это не было «обучением без человека». Люди выбирали, каким будет пример, где провести границу между классами, как представить данные и какие соединения аппаратно доступны. В обучении с учителем, как позже назовут распространённый подход, система получает пример вместе с меткой — указанием правильного ответа. Если на карточке ожидается «левая отметка», исследователь должен сообщить это системе. Машина меняет параметры, но критерий верного ответа пришёл от человека.

В простом современном объяснении это похоже на настройку компаса по серии маршрутов. Если прибор ошибочно уводит направо, мы сравниваем результат с контрольным направлением и слегка подкручиваем настройки. После многих проверок прибор становится полезнее для выбранного типа маршрутов. Однако аналогия перестаёт работать, если представить, что он самостоятельно понял географию или решил, куда стоит идти. Он подстроился под цель и примеры, которые задали люди.

И ещё одно различие: хорошо подстроиться под виденные карточки — не то же самое, что правильно работать на незнакомых. Если все учебные карточки одинаковы, модель может уловить случайный признак — например, царапину в углу, — а не расположение отметки. Тогда в лаборатории кажется, что она «выучила» задачу, но новое освещение рушит результат. Чтобы проверить обобщение, нужны новые примеры и честное разделение между материалом обучения и материалом проверки. В 1958 году терминология ещё складывалась, но сама проблема была уже здесь.

Позднейшие рассказы иногда противопоставляют «обучение» и «программирование», как будто с перцептроном программист исчез из уравнения. Точнее сказать, человеческая работа переместилась. Инженер меньше пишет индивидуальные инструкции для каждого случая и больше проектирует архитектуру, данные, процедуру корректировки и критерии оценки. Ошибки тоже становятся иными: если правила явные, можно искать неверную строку кода; если поведение зависит от множества весов, приходится исследовать данные и обучение.

Для своего времени это был важный поворот. Розенблатт не снял с людей ответственность за постановку задачи. Он предложил способ, при котором часть внутреннего решения могла возникнуть из примеров. Машина перестала быть только исполнителем списка инструкций; она стала системой, чья будущая реакция зависела и от того, как её обучали.

Мозг в заголовке

Пресса быстро подхватила демонстрацию. Заголовок *The New York Times* объявлял о новом устройстве военно-морских сил, которое «учится, как оно действует». В подобных материалах можно было встретить предсказания о чтении, распознавании, речи, исследовании космоса и системах, способных самостоятельно адаптироваться. Некоторые формулировки принадлежали не журналистам, а самому Розенблатту, который говорил о возможном будущем с заметным энтузиазмом.

Это не значит, что он обманывал публику. Он действительно считал перцептрон важным шагом к адаптивным системам и хотел исследовать, насколько далеко можно развить эту идею. Но публичный язык почти всегда уплотняет перспективу. В лаборатории фраза «мы показали процедуру обучения для ограниченной задачи» превращается в «машина учится». В следующем пересказе исчезает ограниченная задача. Ещё шаг — и способность различать карточки звучит почти как понимание изображения вообще.

Такой сдвиг работает в обе стороны. Большие обещания помогают привлечь внимание и средства, а скромный технический результат может показаться публике слишком малым, если он описан без будущего. Однако если ожидания растут быстрее результатов, неизбежно наступает разочарование. Общественная дискуссия тогда спорит не с реальным перцептроном, а с обещанным машинным разумом.

Подобная динамика повторялась во многих технологических эпохах, но глава о перцептроне не должна превращаться в нравоучение про «хайп». В 1958 году был настоящий научный результат: компьютерная процедура могла корректировать числовые связи на основе примеров. Было и реальное теоретическое предложение исследовать обучение и восприятие математически. Были ожидания, часть которых превосходила доказанное. Все три слоя сосуществовали, а не сменяли друг друга аккуратными эпохами.

Полезный вопрос для читателя — не «правда ли, что машина училась?», а «какой именно навык она приобрела, на каких данных и каким способом?». Это вопрос, который пригодится и спустя десятилетия, когда системы будут выглядеть гораздо убедительнее. Чем человечнее звучит ответ машины, тем важнее понимать, на чём он основан.

Тобермори, который должен был услышать

Розенблатт и его коллеги не ограничивали идею только изображениями. Один проект носил имя Тобермори — говорящего кота из рассказа Саки, английского писателя Гектора Хью Манро. В рассказе животное начинает говорить с людьми и обнаруживает, что человеческая учтивость плохо выдерживает честный разговор. В лаборатории имя относилось к проекту распознавания речи.

Тобермори не был бытовым голосовым помощником, не отвечал на вопросы и не понимал разговор. Задача распознавания речи — определить, какие звуковые образцы соответствуют каким категориям или словам. Это уже иной вход: не световые элементы и пространственный рисунок, а меняющийся во времени сигнал, в котором влияют голос, темп, произношение и шум. Подход, основанный на обучаемых связях, мог быть привлекательным именно потому, что заранее выписать все варианты произношения было трудно.

Название проекта иногда рассказывает о научной группе больше, чем серьёзная терминология. Оно соединяло техническую задачу с литературной шуткой, но в этой шутке был риск: в реальности программа не получала ни остроты, ни независимости книжного кота. Она должна была классифицировать акустический материал. Между голосом, который можно услышать, и смыслом, который слушатель ему приписывает, лежит длинная цепочка вычислений и человеческих ожиданий.

Группа Розенблатта была смешанной. В работе участвовали инженеры, программисты и молодые исследователи; позже сотрудники вспоминали, как помещения лаборатории постепенно заполнялись специальными машинами и экспериментальным оборудованием. Среди историй, которые сохранила Cornell Chronicle, — программист Билл Айснер, которому приходилось превращать замысел в работающий код, и Террелл Кокен, которого Розенблатт привлёк к работе подростком, в шестнадцать лет. Исследовательская программа существовала не только в статьях знаменитого руководителя. Её собирали люди, которые настраивали аппаратуру, готовили данные, отлаживали программы и проверяли, что именно происходит после очередной итерации.

В этом есть человеческий масштаб истории, который теряется в рассказе о «машине, учившейся сама». За каждым набором примеров стояли чьи-то часы подготовки. За каждым результатом — чьё-то решение, что считать ошибкой. За красивой демонстрацией на сцене — программисты и техники, которые должны были сделать так, чтобы система вообще запустилась. Автоматизация меняет распределение труда, но редко устраняет труд.

Есть ещё один любопытный эпизод: для исследований речи и восприятия Розенблатт использовал наблюдательные и экспериментальные пространства, связанные с собственным домом и обсерваторией. Эта деталь напоминает, как далеко академическая работа могла выходить за пределы стандартной лаборатории. Но важнее не романтический образ учёного, работающего под звёздами, а то, что изучение интеллекта тогда ещё не было разделено на современные дисциплинарные коробки. Психологические гипотезы, электронные схемы, анализ речи и теория обучения могли обсуждаться одной исследовательской группой.

Где проходит предел одной границы

Любая история о перцептроне рано или поздно приходит к XOR — исключаящему ИЛИ. Эта логическая операция выдаёт «да», если два входа различаются: при сочетании 0 и 1 результат равен 1, при 1 и 0 — тоже 1, а при 0 и 0 или 1 и 1 — 0.

Представим четыре угла квадратной карты. На двух противоположных углах поставим точки класса «да», на двух других — класса «нет». Попробуйте провести одну прямую так, чтобы все «да» оказались по одну сторону, а все «нет» — по другую. Не получится. Прямая неизбежно захватит точку неправильного класса или оставит правильную по ту сторону. Тут не требуется сложная математика: достаточно нарисовать точки и проверить варианты.

Таков предел однослойного перцептрона: он может построить линейную границу, а XOR требует нелинейного разделения. Значит ли это, что нейронные сети в целом не способны вычислить XOR? Нет. Сеть с дополнительным слоем может сначала создать новые промежуточные признаки, а затем разделить классы. Простейший пример можно построить даже вручную. Но остаётся инженерный вопрос: как системе научиться подходящим внутренним признакам, а не получить их готовыми от человека?

XOR полезен именно как простая наглядная модель архитектурного ограничения. Он показывает, что на результат влияет не только объём примеров, но и форма вычислительной системы. Если задать задачу, которую выбранная модель не может выразить, повторение примеров не поможет. Эта мысль актуальна и в современном машинном обучении: не всякий алгоритм одинаково подходит для любой задачи.

Проблема возникает, когда учебный пример становится целой историей. Из того, что простейший перцептрон не может вычислить XOR, иногда выводят, будто нейронные сети были математически опровергнуты. Это чрезмерное обобщение. Ограничение касалось однослойной модели и других свойств изучаемых архитектур. Оно не доказывало, что никакие многослойные сети не способны вычислять сложные функции. Оно делало более острым следующий вопрос: если дополнительные слои нужны, как их эффективно обучать?

Минский, Пейперт и математическая критика

В 1969 году Марвин Минский и Сеймур Пейперт опубликовали книгу *Perceptrons: An Introduction to Computational Geometry*. Её часто описывают как гвоздь, забитый в крышку нейронных сетей. Это сильный образ, но плохое резюме содержания.

Книга была математическим исследованием возможностей определённых классов перцептронов. Авторы анализировали, какие задачи система способна представлять и какие ограничения возникают из её структуры. В центре были не только простые логические примеры, но и более общие свойства распознавания конфигураций, геометрии, локальных связей и способов построения вычислений. Это был подробный аргумент о том, что можно и чего нельзя получить из конкретного семейства моделей при заданных архитектурных условиях.

Один из полезных результатов такой критики — заставить сторонников метода сформулировать границы обещаний. Если однослойная система не может различать определённые структуры, это важный факт для теории и практики. Математический контраргумент не становится несправедливым от того, что его позднее могли пересказать упрощённо.

В то же время влияние книги нельзя отделить от научной среды. Минский был выдающимся исследователем, чьи идеи и институциональный вес привлекали внимание. В конце 1950-х и 1960-х символический ИИ укреплялся в лабораториях и получал ресурсы; нейросетевые направления конкурировали с ним за внимание, деньги и молодые кадры. Историки науки изучали, как математические результаты были интерпретированы внутри этой борьбы. Распространённое прочтение «сложные сети бесполезны» не совпадает с буквальным утверждением, что авторы математически доказали невозможность любого обучения в многослойной сети.

В 1988 году, когда вышло расширенное издание, сами авторы обсуждали, что изменилось с момента первой публикации. Уже сам факт продолжения спора показывает: книжный миф о едином вердикте скрывает более длинную дискуссию. Математика книги была реальной; её воздействие зависело также от того, какие исследования считались перспективными и кому доверяли финансирующие организации.

Розенблатт и Минский знали друг друга ещё со школы — Bronx High School of Science. Позднее они спорили публично о потенциале нейронного подхода. Розенблатт называл Минского «лояльной оппозицией» — человеком, чья критика была постоянной частью дискуссии. Эта фраза звучит живее, чем легенда о личных врагах, но всё равно не даёт полного доступа к их отношениям. У нас есть публичная история научного спора, а не право сочинять частные мотивы.

Соперничество можно описать точнее, если спросить, что каждая сторона считала продуктивным объяснением интеллекта. Символический подход строил явные представления, правила, планы и логические операции. Нейросетевой подход исследовал, как навыки могут возникать из связей и адаптации. Вопрос был не в том, кто из учёных любил машины больше. Нужно ли сначала описать интеллектуальную задачу через символы, или часть её структуры можно извлечь из примеров? Эти две программы предлагали разные ответы и разные виды доказательств.

Минский и Пейперт не были единственной причиной охлаждения интереса к нейросетям; сам Розенблатт не был обладателем решения, которое просто ждало денег. Ограничения вычислительной техники, трудность построения и обучения более сложных сетей, доступность конкретных задач, смена приоритетов финансирования и успехи альтернативных подходов действовали одновременно. Когда идею принимают или откладывают, редко существует один документ, в котором написано: «с этого дня направление закрывается». Чаще решение принимают многие лаборатории, грантовые комиссии, руководители проектов и молодые исследователи, выбирающие тему диссертации.

Почему простота работала против обещаний

У перцептрона была двойная репутация. С одной стороны, он демонстрировал новое отношение к программе: параметры могли меняться на примерах. С другой — самые наглядные задачи были слишком узки, а математические ограничения слишком понятны, чтобы не задавать следующий вопрос. Если система распознаёт отметку на карточке, научилась ли она видеть? Если она различает речь, поняла ли она слова? Если она подстраивает веса, приобрела ли она общее знание?

На каждый вопрос приходилось отвечать отдельно. Нет единственной шкалы, по которой способность классифицировать простые входы превращается в человеческое восприятие. Набор задач мог расширяться, но вместе с ним росли требования к сенсорам, архитектуре, памяти, данным и вычислениям. В 1950-х доступная аппаратура не могла предоставить ресурсы, которые современные исследователи считают обычными.

При этом было бы ошибкой свести неудачу к «не хватило компьютеров». Даже с большими вычислениями неподходящая структура останется ограниченной. И наоборот, мощная теория без подходящего оборудования может годами оставаться исследовательским обещанием. Для полезной системы требуется совпадение нескольких условий: выразительная архитектура, метод обучения, примеры, вычислительные ресурсы и задача, на которой результат имеет ценность. Перцептрон показал лишь часть этой комбинации.

Символический ИИ тогда мог выглядеть практичнее. Если эксперт умеет объяснить, почему неисправна деталь или как доказать утверждение, можно попросить его сформулировать правило. Это обещало прозрачность: программист мог открыть список знаний и проследить путь решения. Нейросеть же предлагала подстраивать параметры, а значит, часть рабочего знания оказывалась распределена в числовой конфигурации. Для исследователя 1960-х это не обязательно было преимуществом. Если результат неверен, не всегда понятно, что именно нужно исправить.

Но и у явных правил обнаружился свой предел. Человек часто умеет распознавать лицо, акцент или тонкую неисправность, но плохо формулирует полный перечень признаков, которыми пользуется. Экспертное знание не всегда доступно в форме коротких инструкций. Спор методов в итоге был не просто между «объяснимым» и «необъяснимым» ИИ, а между разными трудностями: как извлечь правила из головы человека и как извлечь структуру из примеров, не потеряв возможность проверить результат.

От нервной клетки к вычислительному элементу

Розенблатт не начал с пустого листа. К середине XX века уже существовали разные попытки описать нервную систему языком математики и электрических цепей. В 1943 году Уоррен Мак-Каллок и Уолтер Питтс предложили формальную модель нейрона: элемент получал сигналы и срабатывал, если определённое условие выполнялось. Их работа показала, как из простых бинарных элементов можно составлять логические операции. Это была важная интеллектуальная предпосылка, но не готовый перцептрон и не полный план мозга.

Здесь легко незаметно подменить историю. Когда сегодня говорят «нейросеть», может показаться, будто учёные просто поочерёдно копировали биологию всё точнее. На деле они выбирали отдельные свойства нервной системы, которые можно было выразить математически и проверить на доступной технике. У Мак-Каллока и Питтса акцент был на логическом вычислении. Розенблатта интересовала адаптация: что происходит, если связи между элементами меняются на опыте?

Переход от логического нейрона к обучаемому элементу менял сам тип вопроса. Если входы активируют нейрон по заранее заданному правилу, можно строить сложные логические схемы. Если же сила связей может перестраиваться, система получает шанс выработать разные реакции без того, чтобы программист выписал их все. В обоих случаях элементы искусственные, но исследовательские ставки отличаются: в первом случае важно вычисление по фиксированной схеме; во втором — возможность изменять схему по данным.

Научный язык середины века был полон слов вроде «мозг», «память», «обучение» и «организация». Они помогали объединить психологов, инженеров и математиков, но легко обещали больше, чем содержали формулы. В статье Розенблатта нейронная сеть описывалась как вероятностная модель хранения и организации информации. Слово «вероятностная» здесь существенно: речь шла о поведении системы в классе возможных входов и связей, а не о том, что каждый отдельный провод случайно угадывает ответ.

Называя систему мозгоподобной, нужно каждый раз уточнять: в каком именно отношении? У Mark I были датчики, соединённые с ассоциативными и ответными элементами. Это напоминает общий маршрут от ощущения к ответу. Но оно не означает, что в машине воспроизводились зрительная кора, память, внимание или субъективное ощущение видеть. Сходство задаёт исследовательскую аналогию; оно не подтверждает тождество механизма.

Подобные оговорки не лишают нейронный подход его научного значения. Модель мозга может быть полезна, даже если она сильно упрощена: благодаря упрощению можно спросить, какой минимальный набор правил объясняет наблюдаемое поведение. Перцептрон был не фотографией мозга, а лабораторным стендом для вопроса об адаптации. Его ценность измеряется тем, что на нём можно было формулировать гипотезы о поведении обучаемой системы и затем смотреть, что происходит.

Что именно меняется после ошибки

Чтобы понять, почему исследователей занимали «веса», представим два сигнала на входе. Один — пиксель или датчик, реагирующий на положение отметки в левой части карточки; другой реагирует на правую. Если система слишком часто голосует за класс «левая отметка», можно ослабить влияние входов, которые сопровождали ошибочные решения, и усилить те, которые указывали на другой ответ. Простая модель не знает, что отметка «находится слева». Она получает набор чисел и меняет коэффициенты, чтобы итоговая комбинация лучше совпала с заданной меткой.

Вес — не маленький кусок знания с подписью «левый край». Он лишь числовой множитель. Если конкретный вход обычно поддерживает правильный ответ, его вес может сделать влияние сильнее. Но значение каждого коэффициента зависит от остальных входов и от представления задачи. Перемешай кодирование, добавь новые признаки или измени способ нормализации — и те же числа будут означать другое.

Это важное отличие от привычной программы на правилах. В правилах логика может быть написана словами вроде «если А, то В». У обучаемого классификатора поведение возникает из числовой конфигурации. Порой она компактна и понятна; порой даже разработчику трудно объяснить вклад каждого параметра. У раннего перцептрона число параметров было небольшим по нынешним меркам, но уже сама идея распределённого представления ставила вопрос: можно ли сказать, какое знание хранится в какой связи?

Строго говоря, для корректного обучения недостаточно реагировать на ошибку как на факт. Нужно знать, в какую сторону менять веса и насколько сильно. Нужно также решить, когда остановиться и как проверить, что уменьшение числа ошибок не произошло только на уже виденных образцах. В простейших случаях геометрия даёт ясную процедуру: неверный пример подталкивает границу в нужном направлении. Но для сложных задач обучение превращается в инженерную дисциплину, где важны темп обновления, последовательность примеров, структура сети и точность обратной связи.

Поэтому лозунг «покажем машине примеры» лишь начало. Какие примеры? В каком порядке? Сколько раз? Кто поставит метки? Какой уровень ошибки допустим? Нужно ли считать два почти одинаковых образца одним классом? Не перепутала ли модель основной признак с фоном? На все эти вопросы отвечают люди и проект системы. Машина корректирует параметры внутри заданных рамок, но рамки определяют, какой вид ответа вообще возможен.

Ещё одна ловушка — считать, что обученная модель хранит точную копию каждого примера. Иногда модель действительно переобучается и запоминает детали тренировочных данных; в других случаях найденная граница позволяет переносить навык на новые образцы. Нельзя заранее сказать, что делает конкретная система, не проверив её на новых данных. Перцептрон тем самым рано поставил проблему, ставшую центральной для машинного обучения: как отличить обобщение от простого подражания обучающим примерам?

Демонстрация как эксперимент, а не чудо

Публичный показ обычно стремится быть коротким. Исследователь хочет, чтобы зрители увидели изменение до и после обучения; журналисту нужен понятный сюжет; организатору важно не тратить время на технические сбои. Поэтому демонстрация выбирает задачу, в которой результат можно быстро оценить. Отметка слева или справа подходила для этого лучше, чем распознавание незнакомой рукописи или разговор на естественном языке.

Но лабораторный результат нельзя понять по сценическому показу одному. Нужно знать, как были подготовлены карточки, что именно кодировалось, какие ответы считались правильными, сколько было учебных предъявлений и как проверяли незнакомые примеры. Популярный пересказ «после пятидесяти карточек научилась» звучит завершённо, хотя без определения обучающего набора не сообщает читателю, что модель обобщила. Одна и та же цифра может означать сильный результат для одной задачи и почти ничего для другой.

Такие ограничения не были секретом, известным лишь критикам. Розенблатт публиковал теоретические работы и рассматривал перцептрон как программу исследований, а не как готовую машину общего назначения. Но публичная аудитория часто не видела разницы между концептуальной перспективой и предъявленным результатом. Заголовок «компьютер учится» понятнее, чем описание условий эксперимента, и потому получает больше внимания.

Сходная проблема знакома исследователям сегодня. Демонстрация удобного примера может доказать, что система работает хотя бы в этом режиме; она не даёт автоматически измерения надёжности, устойчивости к новым условиям или поведения на исключениях. Необходимо различать «может сделать один раз», «может повторять на похожих примерах», «обобщает в заранее определённой области» и «способна справляться с открытым миром». Эти уровни иногда склеивают одним глаголом «понимает», но технически это разные утверждения.

Для исследователя публичная демонстрация была всё равно полезна. Она давала конкретный объект разговора: не абстрактное «может ли машина учиться?», а вычислительный процесс, который можно наблюдать и воспроизводить. Журналисты могли преувеличивать перспективу, но сама возможность переучить систему на примерах не была рекламной фантазией. Результат ограничен — и потому проверяем. Наука часто начинается именно с подобной узкой постановки, а не с решения самой широкой версии задачи.

Особенно ценно, что перцептрон легко объяснить через противопоставление ручному перечислению. Представьте, что нужно различать карточки с отметкой на одном из двух краёв. Можно написать отдельную инструкцию для каждого положения, размера и небольшого сдвига. Но число вариантов быстро растёт. Обучаемая система вместо этого меняет коэффициенты по примерам. Это не значит, что она автоматически справится с любым изменением изображения; значит только, что вместо списка случаев используется параметризованное правило, способное подстраиваться.

Этот переход — от ручного правила к настраиваемому механизму — и есть исторический масштаб эксперимента. Карточки сами по себе не впечатляют; впечатляет изменение способа задавать компьютерную работу. Вопрос «как запрограммировать каждый случай?» получает конкурента: «как определить архитектуру и научить её по образцам?» Эти вопросы останутся рядом в последующих десятилетиях, иногда соединяясь в одной системе.

Военные деньги и гражданская наука

Контекст финансирования важен, если не сводить его к единственной причине проекта. После Второй мировой войны государственные ведомства США расширяли поддержку фундаментальных и прикладных исследований. Военные системы генерировали большие объёмы информации, а интерпретация сигналов и изображений была очевидной технической трудностью. Поэтому государственные организации поддерживали разработки, которые могли расширить возможности обработки и анализа данных.

Корнеллская авиационная лаборатория находилась в подходящей среде для подобных задач. Она занималась инженерными исследованиями, где теория должна была встретиться с аппаратурой. Для перцептрона это было практически важно: психологическая идея требовала датчиков, электронных соединений и вычислительной инфраструктуры. Плата за оборудование, работа техников и время на настройку не появляются из одной интеллектуальной смелости. Их обеспечивает организация, способная финансировать долгую работу без немедленного потребительского продукта.

Office of Naval Research и Rome Air Development Center были среди организаций, связанных с поддержкой проекта. Но из наличия военного финансирования нельзя выводить конкретное военное применение там, где источники подтверждают лишь исследование модели восприятия и памяти. В истории науки слова «оборонное финансирование» иногда становятся коротким объяснением всего, хотя внутри грантовой программы могли соседствовать фундаментальные вопросы, инженерные прототипы и далекие прикладные ожидания.

И у финансирующего ведомства, и у лаборатории был свой язык ценности. Для науки важен вопрос: чему учит эксперимент? Для учреждения — какая задача может получить практическое применение и оправдать расходы? Для прессы — что из этого поймёт читатель сегодня? Один и тот же перцептрон можно было представить как математическую модель мозга, обучаемый распознаватель или аппарат с потенциалом военного значения. Это не три разных машины, а три разных рамки, каждая из которых выделяет одно измерение проекта и затеняет другие.

Поддержка давала возможность строить систему, но не гарантировала бесконечного терпения. Финансирование зависит от приоритетов, доказанных результатов, новых задач и институциональной конкуренции. Когда направление перестаёт соответствовать ожиданиям, ресурсы переходят к другой программе — не обязательно по злой воле и не обязательно после одного решающего отчёта. Научное поле живёт в экономике внимания: аспиранты идут туда, где видят перспективу, руководители формируют группы, ведомства выбирают проекты, а заметные публикации влияют на следующий выбор.

Именно поэтому поздний спад нельзя рассказать как месть Минского или как простое наказание Розенблатту за громкие обещания. Математическая критика имела значение; вычислительные трудности имели значение; конкуренция подходов и меняющийся климат финансирования тоже. Свести сложный переход к одной причине — значит сделать рассказ проще, а историю менее точной.

Почему правила снова казались лучшим ответом

Когда обучаемые модели столкнулись с пределами, символьные программы не выглядели устаревшим заблуждением. Они решали задачи, где знания можно было представить явно: логические выводы, шахматные ходы, математические выражения, планирование. Если эксперт способен сформулировать условие, компьютер может применить его точно и многократно. В такой системе легче указать, какой шаг привёл к ответу, и легче исправить неверное правило.

Это не означает, что символьные методы победили потому, что нейросети ошиблись. Разные программы исследований преуспевали на разных типах задач. Перцептрон был привлекателен там, где признаки трудно перечислить вручную и можно собирать примеры. Символьный подход был удобен там, где структуры и отношения можно описать через формальные правила. В реальных интеллектуальных задачах часто встречаются оба вида сложности: есть регулярности, которые лучше извлекать из данных, и ограничения, которые люди могут ясно сформулировать.

Ретроспективно легко представить, что учёные могли просто сложить методы вместе. Но исследовательская среда не была супермаркетом с готовыми компонентами. Архитектура нейросети, обучение многослойной структуры, быстрые вычисления и доступные массивы данных ещё не были собраны в практическую комбинацию. А символьные программы уже показывали результаты на задачах, которые можно было описать и проверять. Выбор направления зависел от того, какие вопросы казались решаемыми на существующей технике.

Был и психологический контраст. Правила можно было «прочитать» и обсудить, веса нужно было исследовать статистически. Для учреждения, которому важно понимать, почему система дала ответ, явный код казался более управляемым. Для машинного обучения оставался трудный вопрос: как убедиться, что модель опирается на нужные признаки и будет вести себя устойчиво за пределами лабораторного набора? Этот вопрос не исчез с ростом вычислительных мощностей — он по-прежнему сопровождает обучаемые системы.

В следующей главе появится необычный пример именно символьного подхода. ELIZA не будет искать веса по примерам и не будет копировать работу мозга. Её ответы будут сформированы людьми заранее: набором ключевых слов, шаблонов и операций над текстом. Это покажет, что программа может выглядеть внимательно даже без механизма перцептрона. И тогда спор о вычислении перейдёт в спор о том, что люди готовы считать пониманием.

Восприятие как задача, а не метафора

Для психолога слово «восприятие» означало не только способность назвать предмет. Это был вопрос о том, как организм организует неоднородный сигнал и выделяет в нём устойчивое. Один и тот же предмет меняет вид при повороте, удалении, разном освещении; один и тот же контур может быть представлен с пропусками или шумом. Человек обычно не замечает, сколько вариантов он мысленно объединяет в категорию. Компьютер же получает конкретный массив значений и должен работать с тем представлением, которое ему передали.

У перцептрона это различие было особенно явным. Фотодатчик регистрирует интенсивность света в своей точке. Несколько сотен датчиков могут дать грубую карту: где светлее, где темнее. Но сам сигнал ещё не является предметом, буквой или намерением человека. Чтобы система различала категории, кто-то должен решить, какие изображения составляют примеры одной категории и какое соотношение между ними полезно считать существенным.

Это можно показать на двух написанных от руки буквах. Человеку очевидно, что слегка наклонённые варианты «А» всё ещё одна буква. Однако для сетки светочувствительных ячеек сдвиг линии на несколько ячеек меняет конкретные сигналы. Чтобы машина сохраняла категорию при изменениях, она должна иметь подходящую архитектуру, обучающие примеры или оба условия сразу. Если показать только один образец, система может выучить его расположение, но не правило, которое человек считает «буквой А».

Здесь кроется разница между низкоуровневым сигналом и семантическим смыслом. Перцептрон обрабатывает сигналы, которые закодировали люди, и выдает класс, который назначили люди. Это может быть полезная классификация. Но нельзя перескочить от неё к утверждению, что машина видит тот же объект и приписывает ему тот же смысл, что наблюдатель. Даже когда ответ системы совпадает с человеческим, путь к этому ответу может быть совершенно иным.

Розенблатт выбрал зрительное восприятие как область, где такой вопрос можно было материализовать. Свет можно измерить датчиком, изображение можно разбить на элементы, ответ можно проверить. Это сильная исследовательская стратегия: начать с достаточно определённого входа и формализуемого результата, а затем спрашивать, какие принципы масштабируются на более сложные задачи. Но зрительная «простота» обманчива. Даже маленькая сетка чувствительна к тому, как расположены примеры, а разнообразие реального мира почти сразу раздувает число возможных сочетаний.

Из-за этого Mark I полезно представлять не как робота с глазами, который смотрел на всё подряд, а как экспериментальную систему, получавшую заранее подготовленный материал. Такое описание менее кинематографично, зато отвечает на научный вопрос. Когда речь идёт о машинном распознавании, всегда следует спросить: что было сенсором, что считалось объектом, как формировались категории и на каких случаях проверяли вывод?

Обещание и масштаб реального результата

В публикациях и газетных текстах о перцептроне сталкивались три разных масштаба времени. На коротком горизонте были уже существующие эксперименты: входные данные, корректировка связей и проверяемая классификация. На среднем — построение Mark I, испытания дополнительных моделей и переход от плоской карточки к более сложным сигналам. На дальнем — возможность машин, которые воспринимают мир, адаптируются и выполняют задачи, требующие сейчас человеческого участия. При пересказе эти горизонты часто превращались в одну фразу о машине будущего.

Полезно держать их отдельно, потому что каждое обещание требует своей проверки. Машина могла быть обучаемой, но обучаемость сама по себе не гарантирует широкого набора навыков. Успешная классификация на ограниченной задаче не гарантирует надёжности на новых условиях. И даже высокая точность не доказывает, что система использует человеческое объяснение категории. Это не придирки постфактум; это разные уровни технического утверждения.

Розенблатт сам говорил о перспективах, выходящих за пределы показанных карточек. В эпоху, когда компьютеры только становились объектом массового внимания, именно такие высказывания делали новую идею доступной. Журналист мог пересказать её без диаграмм; читатель видел связь с собственным опытом — ребёнок учится отличать формы, человек узнаёт букву после нескольких примеров. Сравнение помогало понять задачу, но могло спрятать различие в масштабе и устройстве обучения.

Например, младенец не получает набор размеченных карточек от преподавателя и не начинает с сенсорной сетки, построенной конструктором. Его обучение связано с движением тела, звуком, вниманием, социальным взаимодействием и миллионами повторений в непрерывном мире. Перцептрон получал искусственно подготовленный сигнал в лабораторной процедуре. Сравнение «как ребёнок» помогает сформулировать цель, но не является доказательством сходства процесса.

При этом широкие аналогии сыграли конструктивную роль. Они позволяли поставить задачу, которую инженерное сообщество иначе могло не считать вычислительной. Если память и распознавание можно описать как связи, то возможно ли построить электронную систему с похожими принципами? Если обучение меняет соединения, можно ли реализовать это в аппаратуре? Ответ был не в том, что человека заменили машиной, а в том, что некоторые элементы обучения можно было перенести в вычислительную процедуру.

Это и есть осторожная оценка ранней амбиции: Розенблатт увидел направление, которое оказалось важным, но не мог заранее знать весь набор условий, без которых оно не станет масштабируемой практикой. Иногда исторический вклад состоит не в том, чтобы предсказать готовый продукт, а в том, чтобы дать будущим исследователям точную проблему и работающий первый инструмент. Перцептрон был в этом смысле началом разговора, а не его окончательным ответом.

Чему учит карточка

Карточка с меткой была одновременно входом и маленьким договором между человеком и машиной. Человек сообщал: вот изображение, а вот категория, к которой оно относится. Система подстраивалась под этот договор. Если исследователь менял правило маркировки, менялся и смысл «правильного» ответа. Внешне машина всё так же сортировала карточки, но задача, которую она решала, определялась человеческой разметкой.

Это кажется очевидным, пока не перенести идею на более сложные примеры. Когда в наборе есть изображения предметов, кто решает, что считать чашкой: сосуд с ручкой, стакан без ручки, чашку с трещиной, чашку в рекламе? Когда два специалиста расходятся в категории, модель получает не естественную истину, а след конкретного соглашения или институциональной практики. Перцептрон 1958 года не сталкивался с современными спорами об огромных наборах данных, но уже демонстрировал базовое устройство обучаемой классификации: чтобы система «училась», люди должны были превратить желаемый ответ в метку, понятную процедуре.

Есть также разница между числом примеров и разнообразием опыта. Пятьдесят карточек могут содержать пятьдесят почти одинаковых случаев. Это много предъявлений и мало нового. Десять тщательно подобранных вариантов могут проверить больше различий, чем сотни повторов одного шаблона. Поэтому вопрос «сколько примеров нужно?» нельзя решить одной цифрой: важны их вариативность, репрезентативность и соотношение с будущими случаями.

В 1950-х эти вопросы обсуждались не под современными названиями практики данных, но они не были чужды исследовательской логике. Чтобы сказать, чему научилась система, необходимо описать, чему её учили. Чтобы сравнить варианты, нужно менять условие контролируемо: один набор входов, другой набор весов, одинаковый критерий проверки. Машинное обучение с самого начала было не только вычислительным методом, но и способом поставить эксперимент.

Тут появляется человеческая цена, которую легко упустить в рассказе о «самообучающейся машине». Чем больше система зависит от примеров, тем важнее работа по их подготовке и проверке. В ранней лаборатории это могли быть карточки, датчики и часы ручной настройки. В более поздних проектах масштаб вырос, но зависимость не исчезла. Автоматизация классификации не автоматизирует решение о том, какие категории важны, какие ошибки допустимы и кто будет отвечать за последствия неверного ответа.

Поэтому перцептрон полезно помнить не только как предка современных моделей, но и как ранний урок о разделении труда. Машина рассчитывает границу; люди определяют пространство, где граница имеет смысл. Такой вывод не отменяет технического достижения. Он точнее описывает его: обучение позволило алгоритму менять параметры и находить различия, но цель, критерии и наблюдаемый мир оставались человечески сконструированы.

Розенблатт и жизнь исследования

Фрэнк Розенблатт погиб в 1971 году, в возрасте сорока трёх лет. Его смерть иногда превращают в слишком удобный финал: пионер умер, не дождавшись нейросетевого возрождения. Но такое описание делает его биографию прологом к будущей славе, а не самостоятельной научной работой.

Он не построил уменьшенную версию современных больших моделей, не предсказал точный облик глубокого обучения и не решил проблему интеллекта. Он предложил важную линию исследования, создал математические модели, руководил экспериментами и помог публично сформулировать вопрос об обучении связей. Многие его надежды остались гипотезами; часть идей сохранила значение потому, что последующие исследователи вернулись к ним в совершенно других условиях.

И сам перцептрон не исчез в один день. Нейросетевые эксперименты продолжались, иногда под другими названиями и в отдельных исследовательских группах. Параллельно росли статистические методы распознавания, развивались символические системы, алгоритмы поиска и экспертные базы знаний. История не была пустой паузой между Mark I и возрождением нейронных сетей. Она была конкурентной средой, в которой разные подходы получали разные задачи и бюджеты.

Поэтому связь с современными моделями правильнее проводить не через буквальную преемственность корпуса машины, а через принцип. В обучаемых системах параметры меняются по данным и обратной связи. Современные модели работают на иных архитектурах, иных объёмах данных и вычислений, учатся на гораздо более сложных задачах. Между ними и Mark I нет прямой линии, где каждый шаг гарантированно ведёт к нынешним результатам. Есть идея, к которой разные поколения возвращались, меняя и теорию, и оборудование.

От лабораторного времени к историческому

Лаборатория работала в другом ритме, чем нынешняя индустрия программного обеспечения. Устройство нельзя было обновить удалённо; изменить вход значило подготовить новые материалы, подключить датчики и проверить цепи. Даже если общая идея казалась простой, испытание требовало коллективной ручной работы. Это ещё одна причина не путать математическую краткость с инженерной лёгкостью. Несколько строк формулы могут описывать систему, на изготовление и настройку которой уходили месяцы.

В то же время аппарат не был концом этой истории. Модель можно было запускать на IBM 704; отдельная версия воплощалась в Mark I; другие исследователи могли проверять теорию на собственных схемах и задачах. У перцептрона не было единственной формы, и его значение не зависело исключительно от того, насколько хорошо работал один музейный экземпляр. Идея расходилась через статьи, отчёты, доклады и критику. В этом отношении она была одновременно машиной, математической программой и предметом спора.

Когда позднее нейросети вернулись в центр внимания, они не просто включили старый аппарат и продолжили прерванный эксперимент. Поменялись сами средства: цифровые вычисления, методы обучения, объём данных и структура моделей. Историческая связь проходит через исследовательский вопрос — как система может извлекать регулярности из примеров, — а не через прямую передачу готового решения от одного поколения к другому.

Такой тип преемственности менее эффектен, чем легенда о пророке, опередившем время, зато точнее. Розенблатт не знал, каким станет глубокое обучение; позднейшие инженеры не получили от него чертёж современных сетей. Между ними лежит работа многих поколений, иногда продолжавших, иногда критикуя, а иногда заново формулируя похожую проблему. Перцептрон оставил не устройство, которое нужно было лишь дождаться, а направление, для которого ещё предстояло собрать теорию, технику и данные.

После карточки

Mark I Perceptron теперь хранится в Национальном музее американской истории Smithsonian. Музейный экспонат напоминает, что ранний ИИ был физическим: фоточувствительные элементы, схемы, переключатели, шкафы и люди, которые должны были подключить всё это так, чтобы эксперимент состоялся. Увидеть аппарат — значит заметить разницу между образом «искусственного мозга» и инженерной работой по созданию конкретной обучаемой системы.

Путь от июльской демонстрации 1958 года до современных моделей не был прямым и не был неизбежным. Одни обещания перцептрона оказались слишком широкими; некоторые ограничения были математически доказаны для конкретных моделей; практические инструменты для обучения более сложных сетей сложились позднее. Но остался вопрос, который не уместился в спор о газетных заголовках: может ли система приобрести полезное поведение, если ей показывать примеры и корректировать ошибки, а не выписывать полный список правил заранее?

На этот вопрос можно было ответить не только нейронной сетью. В следующей истории появится программа, устроенная почти противоположным образом. Она не будет менять сеть связей после каждой реплики: человек заранее задаст ключевые слова, шаблоны и правила обработки фраз. Но в разговоре с ней пользователи увидят не только программу, но и собственную готовность приписывать машине больше понимания, чем в ней заложено. Её будут звать ELIZA.

Глава 5. Программа, которая умела слушать

Однажды секретарь Джозефа Вейценбаума попросила его выйти из комнаты, чтобы она могла продолжить разговор с программой наедине. Вейценбаум сам создал эту программу. Секретарь знала, как она устроена и над чем он работал. И всё же ей захотелось приватности.

Историю пересказывают так часто, что она почти превратилась в притчу: компьютер обманул человека, человек поверил в искусственный разум. Но в этом пересказе слишком много удобства. Мы знаем об эпизоде прежде всего со слов самого Вейценбаума, который описал его позднее. Мы не знаем, что именно думала его коллега и насколько буквально она воспринимала собеседника. Возможно, ей просто было неловко говорить о личном при авторе программы. Возможно, разговор увлёк её, даже если она понимала, что перед ней компьютер. Чужие мысли нельзя восстановить по одной реплике.

Зато можно точно сказать, что эпизод задел самого Вейценбаума. Он написал программу, которая не понимала человеческий язык в обычном смысле, и увидел, как легко форма разговора создаёт ощущение собеседника. ELIZA не обладала сочувствием, памятью о жизни пользователя или знаниями психотерапевта. Она искала ключевые слова и перестраивала фразы по правилам. Но человек, сидевший перед терминалом, приносил в диалог всё, чего программе не доставало: собственную историю, контекст, ожидания и способность связывать вопросы с переживаниями.

В этом была не история о глупом пользователе и коварной машине. Вейценбаум знал её механизм лучше всех. История была о силе разговорной формы — и о том, что происходит, когда привычный социальный жест, вопрос или повторение слов получает неожиданный источник.

ELIZA — это программа, DOCTOR — роль

В январе 1966 года Вейценбаум опубликовал описание ELIZA в журнале *Communications of the ACM*. Полное название статьи обещало не психологическое чудо, а «программу для изучения коммуникации на естественном языке между человеком и машиной». Само название ELIZA он взял у Элизы Дулитл — героини пьесы Бернарда Шоу «Пигмалион», которую широкая публика знала также по мюзиклу «Моя прекрасная леди». В сюжете Элиза учится говорить на языке, который открывает перед ней новые социальные возможности. Компьютерная программа тоже работала с речью, но сравнение заканчивается здесь: ELIZA не проходила обучение и не меняла собственную манеру говорить.

Важно и другое различие. ELIZA была не одной заранее готовой личностью. Это был механизм для обработки текстового диалога. Он мог работать с разными наборами правил — сценариями, которые определяли, какие слова замечать и как отвечать. Самым известным стал DOCTOR: сценарий изображал психотерапевта на первой беседе с пациентом.

Сегодня мы легко называем ELIZA чат-ботом. В середине 1960-х такое слово ещё не было привычной категорией, а сам Вейценбаум описывал более широкую задачу: исследовать, как человек и компьютер могут обмениваться фразами на естественном языке. Поэтому точнее считать ELIZA ранней разговорной программой, а DOCTOR — одним, самым знаменитым способом использовать её механизм. Когда позже вспоминают «психотерапевта ELIZA», это короткое имя объединяет программу, правила и роль в одно целое. Так и возникала её популярная личность.

Эта конструкция уже знакома каждому, кто запускал приложение с несколькими режимами: общий механизм остаётся тем же, меняется набор инструкций. Для ELIZA сценарий был не декорацией поверх персонажа. Он содержал основную часть разговорной компетенции. Интерпретатор знал, как анализировать и преобразовывать текст; сценарий подсказывал, какие преобразования применить и в каком порядке. Если написать другой сценарий, можно было получить иное подобие разговора, не создавая систему заново.

Разделение механизма и сценария стало одним из важных технических решений проекта. В коде не нужно было жёстко закреплять все детали психотерапевтической роли. Вместо этого правила описывали ожидаемые образцы фраз, важность ключевых слов и способ собрать ответ. Вейценбаум мог продемонстрировать, как один программный механизм допускает разные сценарии взаимодействия. Даже до появления современных больших моделей здесь видна идея программного слоя: движок выполняет общие операции, а отдельная конфигурация задаёт поведение.

Но не следует переносить на 1966 год весь смысл нынешней «модели плюс системный промпт». Сходство ограничено разделением кода и настраиваемого содержимого. Современная языковая модель обучается на огромном количестве текстов и строит ответы иначе. Сценарий DOCTOR представлял собой набор человечески написанных правил. В нем не было скрытой статистической модели, самостоятельно выучившей психотерапевтический диалог.

Что программа делала с фразой

Вейценбаум описал механизм без мистики. ELIZA принимала строку текста, разбивала её на элементы, просматривала их в поисках ключевых слов и выбирала подходящее правило. Правило могло разделить фразу на части: одну часть распознать как знакомое начало, другую сохранить для будущей сборки. Затем программа перестраивала сохранённые фрагменты и помещала их в шаблон ответа.

Возьмём условный пример. Пользователь пишет: «Я постоянно ссорюсь с сестрой». Сценарий может обратить внимание на тему семьи и сформировать вопрос о сестре. Другой вариант — заметить оборот «я чувствую» и превратить утверждение в приглашение продолжить: что именно заставляет вас так чувствовать? Это не пример точного вывода о личности человека. Программа не выясняет, кто такая сестра, когда была ссора и чего хочет пользователь. Она берёт распознанный элемент и запускает правило, которому этот элемент соответствует.

При преобразовании фразы могли меняться местоимения и грамматические формы. В английском нужно было, например, превратить «I» в «you», «my» в «your», чтобы пересказанный фрагмент звучал как обращение терапевта к собеседнику. Такие преобразования были не простым поиском и заменой одного слова: часть фразы подчинялась правилам разложения, а результат собирался по подходящему шаблону. Но глубина анализа оставалась ограниченной языковыми образцами, перечисленными в сценарии.

Одни ключевые слова имели больший приоритет, чем другие. Если в длинной реплике встретишься тема, которой DOCTOR придавал высокую важность, программа могла использовать соответствующий набор реакций. Если подходящего правила не находилось, у неё были общие ответы и способы вернуть разговор к новой формулировке. В опубликованном алгоритме были также механизмы для отложенных фрагментов: фраза могла не исчезнуть сразу, а стать материалом для последующего ответа, когда более подходящее правило не срабатывало.

Поэтому ELIZA не просто случайным образом хваталась за первое слово. В ней была структура выбора, и у сценария существовали заранее определённые связи между типами входа и типами ответа. Но эта структура не превращалась в полноценную модель смысла. Программа могла отреагировать на слово «мать» как на тему разговора о семье, не зная, жив ли родитель, о ком именно идёт речь и что слово значит для человека в конкретной ситуации.

Можно представить механизм как театральную труппу. Один участник читает реплику, отмечает знакомые слова и выбирает нужную сцену из папки сценария. Другой переставляет фрагменты фразы, подставляет подходящую формулировку и выдаёт её собеседнику. Это сравнение полезно, пока мы помним границу: внутри ELIZA не было актёра, который понимал роль. Были код и текстовые правила, подготовленные Вейценбаумом.

Такой механизм мог создавать связные локальные переходы. Если программа спрашивала о семье после упоминания сестры, пользователь мог решить, что она заинтересовалась именно его отношениями с сестрой. В техническом смысле ответ был выбран потому, что сработало правило. В человеческом смысле связь между репликами возникала совместно: машина давала зацепку, а собеседник достраивал контекст.

Почему психотерапевт был удачной ролью

Выбор DOCTOR был не случайной попыткой сделать машину модной. Вейценбаум хотел получить сценарий, который мог бы поддерживать разговор, не требуя от системы подробной энциклопедии о реальном мире. Для этого подходила модель недирективного психотерапевтического интервью, связанная с работой Карла Роджерса.

В недирективной беседе специалист не обязан сразу предлагать диагноз и готовый совет. Он может возвращать человеку его слова, просить пояснить, что тот имеет в виду, задавать открытые вопросы и помогать ему самому развернуть рассказ. Пациент говорит большую часть времени. Терапевт слушает, уточняет и не подменяет рассказчика собственными объяснениями. Это упрощённое описание подхода, но для устройства диалога важна именно его форма.

Такая беседа предъявляла к программе более сильные требования, чем, скажем, роль преподавателя математики или адвоката. Преподаватель должен знать материал и обнаруживать ошибки в решении. Адвокат должен учитывать обстоятельства дела и нормы права. От терапевтического персонажа в начальном интервью можно было ожидать вопросов и внимательного отражения слов пользователя. Роль позволяла дольше обходиться без знаний о внешнем мире и не требовала быстро сообщать проверяемые факты.

Вейценбаум позднее прямо объяснял, что выбрал такую форму потому, что это один из немногих жанров парного разговора, где один участник может занимать позицию человека, почти ничего не знающего о реальном мире. Неведение здесь не разрушало роль сразу: психотерапевт часто спрашивал, а не рассказывал. То, что в других ситуациях выглядело бы уклонением от ответа, могло показаться естественным приглашением продолжить.

Эта роль подходила и по ритму. Пользователь делал основную содержательную работу. Он приносил личную проблему, описывал отношения, связывал настоящее с прошлым. Машине достаточно было время от времени направить внимание на знакомое слово или вернуть фразу в вопросительной форме. Граница между «система отвечает» и «пользователь сам продолжает разговор» становилась неочевидной.

Важно не говорить, будто программа действительно стала психотерапевтом или намеренно эксплуатировала уязвимость пользователя. Нет свидетельств, что такова была цель Вейценбаума. Его статья описывала способ изучения общения человека с машиной, а DOCTOR представлял ограниченную языковую роль. Именно удобство этой роли для эксперимента помогло сделать её убедительной и одновременно создало риск, что аудитория воспримет пародию на один стиль беседы как подобие помощи.

Профессиональная помощь — это не только набор вопросов. Она включает подготовку, ответственность, этические обязанности, понимание ситуации и способность реагировать на последствия слов. Перцептрон классифицировал примеры, ELIZA запускала сценарий. Ни один из этих процессов не заменяет отношений, в которых один человек отвечает перед другим. Нельзя незаметно подменить «может поддержать обмен репликами» утверждением «может лечить».

Интерфейс был бумажным и общим

Позднее люди вспоминали ELIZA как текстовое окно, в котором программа мгновенно отвечала на сообщение. Оригинальный опыт был материальнее. В середине 1960-х рядом не было смартфона или персонального ноутбука. ELIZA работала в среде MIT на большой вычислительной системе; пользователи взаимодействовали с ней через терминалы и печатающие устройства. Время машины распределялось между несколькими участниками, подключёнными к общей системе.

Сеанс мог идти через печатный терминал: пользователь вводил строку, компьютер выполнял команды, ответ появлялся на бумаге. В отличие от нынешнего чата, реплика не растворялась в подсветке экрана. Перед человеком оставалась физическая последовательность обмена. Скорость ответа зависела не только от самой программы, но и от устройства и доступности вычислительных ресурсов. Это была встреча с машиной, которая занимала комнату, даже если собеседник видел лишь клавиатуру и лист бумаги.

Одновременно система разделения времени создавала новое ощущение прямого доступа. До этого компьютер часто был ресурсом организации, к которому пользователь обращался через оператора, очередь задач или специалиста. Терминал позволял печатать команды самому и получать ответ без видимого посредника. Компьютер оставался общим, но взаимодействие с ним становилось индивидуальным. Человек сидел один перед печатающей машиной и читал ответ, обращённый к нему.

Эта непосредственность была частью эффекта. Терминал ничего не изображал человеческого лица, но ритм «вопрос — ответ» уже был знаком. Отсутствие оператора не доказывало, что на другом конце есть личность; оно просто делало обмен менее похожим на работу с удалённой вычислительной службой. Машина переставала выглядеть как абстрактный ресурс центра обработки данных и начинала занимать социальную позицию собеседника.

Техническая среда также определяла, что именно сохранялось. Некоторые версии диалога печатали взаимодействие, и его можно было перечитывать; в других условиях разговор воспринимался как последовательность текущих ответов. Сегодня интерфейс привычно включает историю, аватар, анимацию набора, уведомления и постоянную доступность. Вейценбаум не создавал всего этого. Но ELIZA уже показала, что канал передачи участвует в восприятии не меньше, чем алгоритм.

Здесь есть любопытный контраст с символическими программами предыдущих глав. Logic Theorist выдавал доказательство, которое можно было разбирать как формальную цепочку. Перцептрон демонстрировал изменение поведения после примеров. ELIZA работала в устройстве, которое само по себе приглашало к социальному прочтению: человек печатал личную фразу, получал вопрос и отвечал снова. Вычислительная процедура стала частью маленькой сцены.

ELIZA среди ранних языковых программ

Когда Вейценбаум начал работать над ELIZA, компьютерные исследователи уже пытались заставить машины обрабатывать английский. В лабораторной среде MIT можно было встретить программы, решавшие задачи в ограниченных языковых мирах: например, преобразовывавшие текстовую задачу по алгебре в формальные выражения и вычисления. В таких проектах знания часто были тесно связаны с самой программой. Если добавить новую предметную область, приходилось менять и представление задачи, и код.

Вейценбаум хотел исследовать взаимодействие другого типа. Пользователь не только передавал машине готовую задачу, но и мог ввести собственную фразу; программа затем задавала вопрос, который приглашал уточнить её. Ему было важно, можно ли сделать механизм более общим, чем одна жёстко заданная задача. ELIZA не стала общим решением проблемы языка, но разделение сценария и интерпретатора позволяло хотя бы исследовать разные формы текстового поведения в одной программной среде.

Здесь слово «естественный» легко обманывает. Естественный язык — это английский или русский, которыми люди пользуются без формального курса грамматики; но программа не получает значение фразы просто оттого, что принимает слова. На естественном языке можно выразить почти что угодно, пропустить половину контекста, использовать иронию, намёк, двусмысленность или вспомнить событие, о котором система не знает. Вейценбаум мог обрабатывать текст на английском, не решив проблему того, как компьютер связывает слова с общим миром.

Это различие заметно в самом названии статьи. Она говорит о коммуникации между человеком и машиной, а не о полноценном понимании. Коммуникация может состоять из обмена знаками, ответов и ожиданий; понимание подразумевает более сильное утверждение о том, что система делает со смыслом. Если программа отвечает на «я чувствую себя одиноко» вопросом об одиночестве, обмен состоялся. Но из одного этого не следует, что машина представляет, что такое одиночество, как оно переживается или что сейчас нужно человеку.

В лаборатории и в газетах один и тот же проект мог получить разные формулировки. Для автора это эксперимент с языковым интерфейсом и набором процедур. Для пользователя — разговор с компьютером. Для журналиста — машина, которая умеет беседовать. Каждая версия отбирает правду, но последняя легко становится самой громкой. Именно так разговорная способность начала жить отдельно от вопроса, какие операции стояли за конкретной репликой.

ELIZA была особенно важна не потому, что превосходила другие программы во всех задачах обработки языка. Она сместила внимание на человеческую сторону интерфейса. Исследователь мог смотреть на ключевые слова и правила, а пользователь — на вопрос, который словно обращался именно к нему. Впервые техническое устройство стало известным благодаря тому, как люди читали его поведение, а не благодаря объёму математической задачи, которую оно решало.

Правило под микроскопом

Рассмотрим условный текст: «Мой отец часто сердится на меня». Это не стенограмма из статьи Вейценбаума, а схема, которая показывает логику DOCTOR. Программа может встретить слова, связанные с родителем, и присвоить теме более высокий приоритет, чем нейтральному слову вроде «часто». Затем она может выбрать образец, который сохраняет часть фразы, а reassembly-правило превращает её в вопрос о том, почему отец сердится. Если родительская тема не подходит или уже исчерпаны варианты, сценарий может воспользоваться другой зацепкой или общей репликой.

Для человека такая последовательность выглядит как небольшое умозаключение: он рассказал про отца, собеседник заметил семейный конфликт и спросил, что за ним стоит. Для программы это может быть цепочка операций: найти ключ, выбрать соответствующий шаблон, выделить фрагмент текста, изменить местоимения, подставить фрагмент в формулу ответа. Одна и та же внешняя реплика допускает два описания. Первое описывает человеческий смысл разговора; второе — алгоритм, который выдал текст.

Правила разложения давали сценарию некоторую гибкость. Вход не обязательно должен был в точности совпасть с одной фразой целиком. Шаблон мог распознать знакомый фрагмент и сохранить остальное. Например, слово или выражение задавало ключ, а часть после него становилась переменной частью ответа. Программа могла использовать это содержание, не анализируя его как самостоятельное предложение с полноценными синтаксическими и семантическими отношениями.

Сборка ответа могла производиться по нескольким заготовленным вариантам. Если одна и та же тема каждый раз вызывала одну и ту же фразу, пользователь быстро заметил бы повтор. Наличие нескольких шаблонов позволяло слегка менять реакцию. Но разнообразие заготовок — не то же самое, что генерация бесконечного числа новых мыслей. Менялась формулировка; механизм оставался внутри сценария.

Порядок выбора тоже был значим. В сообщении «Мать считает, что я должен уехать, но я не хочу» программа могла обнаружить несколько ключевых слов. То, какое из них получало приоритет, влияло на ответ. Высший приоритет не означал, что ELIZA поняла, что мать давит на сына; он означал, что сценарий устроен так, чтобы один тип реакции выбирался раньше другого. Человеческий разработчик задавал это распределение внимания.

В опубликованной версии есть механизм памяти в узком вычислительном смысле. Если главный ключ не удавалось обработать привычным способом, программа могла временно сохранять часть текста и вернуться к нему позже. Это не биографическая память и не накопленный опыт о пользователе. Система не вспоминала встречу недельной давности и не строила модель отношений. Она хранила фрагменты, чтобы использовать их по правилам текущего сценария.

Такое различие полезно потому, что слова «память», «внимание» и «знание» встречаются и в описании программы, и в разговоре о человеческом сознании. В компьютере «память» может означать область, где временно хранится строка; у человека — воспоминание о событии с его эмоциональной и социальной связью. Термин общий, явление разное. Чтобы не приписывать машине лишнего, приходится уточнять технический смысл каждого такого слова.

В типичном случае ELIZA могла отвечать уместно. Но при неоднозначном сообщении ключ мог увести разговор не туда; если слово было незнакомым, включался общий ответ; если пользователь возвращался к той же теме, шаблон мог повториться. Иногда хватало одной настойчивой проверки, чтобы заставить сценарий ходить по кругу или терять связность. Этот провал не был случайной аварией, как и удачная реплика не доказывала понимание. Оба результата следовали из одного устройства.

Что делал пользователь

Человек, разговаривавший с DOCTOR, выполнял часть работы, которую многие тогда могли не замечать. Он помнил свои предыдущие фразы. Он знал, почему сказал «мать», а не «родитель». Он связывал новый вопрос со своей историей, даже если программа просто выбрала ближайший шаблон. Он мог догадаться, какой ответ машина ожидает, и написать следующую реплику так, чтобы продолжить игру.

Психологи, изучающие общение, говорят об общих предпосылках: чтобы беседа работала, участники обычно считают, что слова относятся к какой-то общей ситуации. В человеческом разговоре часть этих предпосылок действительно разделяется. Собеседник знает, что такое семья, конфликт, страх; он способен соотнести рассказ с интонацией и обстоятельствами. ELIZA разделяла с пользователем только ограниченный текстовый сигнал и правила сценария. Остальное собеседник мог представить как общее знание, потому что так устроено обычное общение.

Это не значит, что пользователь ошибался в каждом случае. Человек мог понимать ограничения и всё равно получать пользу от самого процесса. Дневник тоже не понимает содержание, но письмо в дневник помогает разложить мысли. Специалист иногда задаёт простые вопросы, потому что формулировка вопроса важнее его новизны. Разговор с программой мог выполнять роль внешнего повода сказать вслух то, что человек уже знал, но ещё не собрал в слова.

Отсюда важно различать по меньшей мере четыре вещи: веру в сознательность машины; временное принятие её социальной роли; полезность общения для пользователя; и готовность профессионала доверить системе ответственность за лечение. Можно быть уверенным, что программа не человек, и всё равно отвечать ей искренне. Можно считать разговор интересным, не считать его терапией. И наоборот: кто-то может переоценить технические возможности, даже не испытывая глубокого эмоционального контакта.

Легенда о секретаре часто схлопывает эти различия в один вывод: если она просила приватности, значит поверила, что перед ней настоящий психотерапевт. Но исходная история такого не доказывает. Даже если пользователь знает, что программа механическая, ему может хотеться остаться наедине с личным текстом. Приватность — нормальное человеческое желание, а не тест на то, считает ли человек компьютер живым.

В этом смысле знаменитый эпизод рассказывает как минимум о двух видах интимности. Первая — ощущение, что машина отвечает лично. Вторая — собственная потребность человека в пространстве, где можно говорить без чужого взгляда. Компьютер может не понимать содержание, но интерфейс всё равно становится местом, куда пользователь приносит то, что не хочет произносить перед людьми. Такая возможность одновременно полезна и уязвима: кому-то легче говорить, но система и её создатели получают доступ к личным словам.

Психотерапия — больше, чем набор вопросов

Образ психотерапевта был удачным для эксперимента, но в реальной практике недирективность не означает отсутствие профессиональных знаний. Карл Роджерс делал акцент на уважении к опыту клиента, эмпатическом слушании и создании условий, где человек может исследовать собственные чувства. Вопросы и перефразирование — заметная часть такой работы, но не всё содержание отношений.

Живой специалист не просто обнаруживает эмоционально окрашенное слово. Он следит за тем, как меняется рассказ, замечает противоречия, слышит темп и интонацию, принимает во внимание молчание, прошлый опыт и реакцию клиента на вопрос. Он несёт ответственность за границы отношений и способен пересмотреть подход, если замечает, что вмешательство приносит вред. Даже «внимательное отражение» требует суждения: одно и то же повторение может быть поддержкой, механическим эхом или раздражающим уходом от ответа.

DOCTOR воспроизводил внешний приём — возврат слов и открытый вопрос — в ситуации, где пользователь выполнял содержательную часть разговора. Это было достаточно, чтобы проиллюстрировать технику. Но одна техника не равна профессии. Аналогично, скрипт, который задаёт вопросы о симптомах, не становится врачом, а словарь юридических терминов не превращает программу в адвоката. Профессиональная роль включает больше, чем текстовый стиль.

Это различие важно и потому, что сама терапевтическая форма может усиливать доверие. Человек рассказывает личное, получает вопрос, раскрывает больше. Если программа не умеет распознавать риск и не знает, что делать с тяжёлым признанием, её привычная реплика может быть опасно неуместной. Удобная роль способна сгладить границу между исследовательской демонстрацией и обещанием помощи.

Вейценбаум не создавал клиническую систему и не проводил испытания эффективности психотерапии. Если позднее кто-то видел в DOCTOR заготовку для автоматического консультирования, это уже другое утверждение, требующее других доказательств. Возможно ли в принципе автоматизировать отдельные поддерживающие функции? Это вопрос исследовательский. Работала ли ELIZA как терапия? Её статья такого не доказывала.

Пользователю следует сообщать не только о возможностях инструмента, но и о том, каким видом поддержки он не является. Эта мысль не требует современных нормативных правил, чтобы быть понятной. Если человек считает беседу лечением, а разработчик называет её демонстрацией, перед нами не просто разница терминов. Это несовпадение ожиданий, которое может повлиять на решение человека искать или не искать помощь у другого человека.

Имя, голос и старые социальные роли

Имя Элизы приходит из «Пигмалиона». В пьесе Шоу профессор фонетики Генри Хиггинс берётся изменить речь цветочницы Элизы Дулитл. Новая манера говорить меняет то, как её воспринимает общество; за уроками произношения стоят вопросы класса, достоинства и права определять, кто считается образованным. Название компьютерной программы не делает её продолжением этого сюжета, но для современного читателя оно добавляет культурный оттенок: голос и речь служат не только передачей информации, но и пропуском в социальную роль.

Вейценбаум взял имя у литературной героини, которая осваивает новый способ говорить. Саму систему он не обучал речи и не давал ей историю становления. ELIZA не проходила путь Дулитл. Название работало как каламбур и подсказка: перед нами программа, которая преобразует человеческий язык, но не становится человеком за счёт правильной формулировки.

Сценарий DOCTOR тоже был не нейтральным. Он выбирал голос профессиональной заботы — человека, который задаёт вопросы и не торопится с ответом. Пользователь видел не таблицу правил, а институциональную роль, уже знакомую по жизни и культуре. Так же, как униформа может создать ожидание компетентности ещё до разговора, сценарий задавал рамку, в которой вопрос машины звучал иначе, чем тот же вопрос от случайной программы.

В современном разговоре о голосовых и текстовых помощниках иногда кажется, что достаточно убрать аватар, назвать программу «инструментом» и всё недоразумение исчезнет. ELIZA напоминает: ожидания рождаются не только из имени и дизайна. Их создают чередование реплик, личная формулировка, память о предыдущих словах, ощущение доступности и место, в котором идёт разговор. Убрать один визуальный знак полезно, но социальный смысл взаимодействия состоит из множества сигналов.

Это не довод против дружелюбного интерфейса. Это довод не путать дружелюбие с отношениями, а роль — с ответственностью. Название и манера речи могут помочь человеку пользоваться инструментом. Но они не создают за разработчиками право обещать профессиональную помощь, понимание или эмоциональную взаимность.

Не один диалог, а множество версий

Слово ELIZA часто создаёт впечатление, что существовала одна неизменная программа с одной личностью. На деле развитие шло через несколько версий и сценариев. В статье 1966 года Вейценбаум представил механизм и DOCTOR, приложив сам сценарий. В последующей работе он обсуждал расширения: более сложную обработку, дополнительные сценарии и хранение результатов вычислений. Некоторые версии можно было использовать для задач, заметно отличавшихся от психотерапевтической беседы.

Это важно для истории программного обеспечения. Современный цифровой продукт тоже редко существует в единственной форме. Есть версии, экспериментальные ветви, разные интерфейсы, наборы данных и конфигурации. Когда позднейшие пользователи запускали «ELIZA», они могли взаимодействовать с копией, переписанной на другом языке, с изменённым сценарием и другой скоростью ответа. Историческое имя оставалось тем же, но опыт не всегда был идентичным.

Популярная память любит стабильного персонажа: вот программа, вот её знаменитый разговор. Исследовательская реальность была более подвижной. Статья описывала конкретный вариант и демонстрационный материал, но код мог меняться. Сценарии расширялись, программный механизм развивался. Позднее энтузиасты переносили ELIZA на персональные компьютеры и переписывали на доступных языках. В этих версиях сохранялась узнаваемая идея правил, но сам пользовательский опыт зависел от реализации.

Такое разнообразие делает особенно важным вопрос об источниках. Если современный сайт показывает ELIZA, это не значит, что перед нами точная копия программы 1966 года. Возможно, это реконструкция принципа, современный сценарий или стилизация. Чтобы утверждать, что именно видел исторический пользователь, нужно смотреть на опубликованную статью, доступный код и архивные материалы, а не полагаться на любую программу с тем же именем.

Исследователи и архивисты в последние годы восстанавливали ранний код по сохранившимся распечаткам и материалам архива Вейценбаума. Работа с такими объектами помогает увидеть не просто «простейший чат-бот», но реальную структуру программы, созданной на языках и вычислительных системах эпохи. Восстановление кода не делает ELIZA умнее задним числом; оно даёт более точное представление о том, из каких процедур складывался разговор.

Понятие «сценарий» здесь особенно продуктивно. Оно позволяет заметить, что поведение программы задавалось не только логикой интерпретатора, но и написанным для неё текстом. Человек, создававший сценарий, выбирал слова, которым программа придавала вес, ответы, которые она могла повторять, и темы, к которым направляла пользователя. Техническая архитектура отделяла эти слои, но не устраняла авторство. В ELIZA слышался не безличный голос машины, а набор человеческих решений, оформленный в исполняемом виде.

Почему люди продолжали говорить

Вейценбаум описывал реакцию на ELIZA как удивительную. Его секретарь — самый известный эпизод, но не единственный повод задуматься. Люди могли вовлекаться в разговор, хотя ответы программы были шаблонными и часто уклончивыми. Некоторые пользователи просили поговорить с системой без наблюдателей. Вейценбаум позднее отмечал, что довольно короткого знакомства с программой порой было достаточно, чтобы обычные люди начали приписывать ей больше понимания, чем могло подтвердить её устройство.

Историки и исследователи человеко-компьютерного взаимодействия часто называют подобную склонность «эффектом ELIZA». Термин обычно означает, что человек достраивает за системой внутреннее понимание или намерение, опираясь на небольшой набор внешних признаков. Но если остановиться на фразе «люди легко обманываются», получится не объяснение, а упрёк.

Разговор всегда распределяет работу между участниками. Если один человек говорит: «Мне было страшно», а другой отвечает: «Что именно вас напугало?», второй не обязательно пересказывает факты или демонстрирует экспертное знание. Он приглашает первого продолжить. В этой ситуации смысл беседы строится из обеих реплик, но основная информация исходит от рассказчика. DOCTOR мог использовать такой ритм, хотя сам ничего не знал о личной истории пользователя.

Пользователь мог одновременно понимать, что разговаривает с программой, и всё же участвовать в игре. Знание механизма не отключает привычки восприятия. Мы можем растрогаться выдуманной историей, зная, что она придумана; можем ругаться на персонажа видеоигры, не считая его живым; можем отвечать голосовому помощнику вежливо, хотя понимаем, что перед нами интерфейс. Участие не всегда равно вере.

Взаимодействие с ELIZA могло быть и экспериментом над самим собой. Человек проверял программу, сравнивал ответы с ожидаемыми, пытался сломать сценарий. Другой мог использовать беседу как способ сформулировать собственную мысль: вопрос запускал новый кусок рассказа. Третий мог на время принять роль собеседника и посмотреть, куда заведёт обмен. Одно и то же поведение внешне — продолжение диалога — не доказывает одну и ту же внутреннюю причину.

Поэтому реакцию пользователей лучше не описывать как массовую ошибку. Интерес к разговору не означает, что человек всерьёз решил, будто компьютер обладает человеческим сознанием. Люди могут играть, исследовать, проверять границы, использовать удобный повод проговорить мысль или просто получать удовольствие от необычного контакта. ELIZA давала достаточно структуры, чтобы разговор ощущался отзывчивым, и достаточно пустоты, чтобы смысл поступал от человека.

И всё же риск неверного прочтения реален. Если учреждение видит в удачной демонстрации доказательство, что машина может лечить, судить или принимать решения за человека, игра превращается в ставку с последствиями. Вейценбаума тревожил не факт, что пользователи разговаривают с программой. Его тревожило, что технические специалисты и некоторые профессионалы могли делать далеко идущие выводы о том, что такая программа способна выполнять.

От эксперимента к обещанию терапии

В своей статье Вейценбаум осторожно описывал DOCTOR как пародию на разговорный стиль недирективного психотерапевта. Это слово — «пародия» — полезно, хотя может звучать обесценивающе. Речь не о том, что терапия смешна, а о том, что программа воспроизводит ограниченный внешний шаблон взаимодействия, не обладая теми знаниями и ответственностью, которые стоят за работой специалиста.

Позднее Вейценбаум был встревожен сообщениями, что часть психологов и психиатров видела в программе потенциал для почти автоматизированной психотерапии. Здесь важно различать источники: он фиксировал собственное восприятие реакции профессионального сообщества; это не означает, что все психотерапевты поддерживали идею или что существовал клинический консенсус. Но сама возможность, что специалисты могли счесть простой механический сценарий началом автоматической помощи, произвела на автора сильное впечатление.

Чтобы оценить разницу, представим ситуацию, когда человек сообщает о намерении причинить себе вред. Роль настоящего специалиста — не просто подобрать подходящий вопрос. Нужно оценить ситуацию, заметить риск, отвечать с учётом контекста, понимать пределы компетенции и знать, как действовать дальше. ELIZA не могла надёжно различать такие обстоятельства. Она могла отреагировать на знакомое слово и выдать очередную реплику; сама форма разговора не превращала её в безопасного помощника.

В этом нет аргумента против любого цифрового инструмента для поддержки человека. Есть аргумент за точное описание функции и границ. Если программа помогает записать симптомы, найти нужную информацию или сформулировать вопрос врачу, её полезность можно оценивать на этих задачах. Если её называют терапевтом, пользователь вправе ожидать другого уровня ответственности. Название интерфейса меняет риск, даже когда код не изменился.

ELIZA не была продуктом здравоохранения, но её история показывает, как быстро прототипу могут приписать роль, на которую он не рассчитан. Сначала программа демонстрирует, что текстовый обмен технически возможен. Затем окружающие видят социальную ценность. Потом возникает соблазн перенести эту ценность на более серьёзную сферу, не проверив, какие способности для неё необходимы. Между «разговор поддерживается» и «человеку оказана помощь» лежит разрыв, который интерфейс способен скрыть.

Эта ошибка особенно вероятна, когда роль выбрана удачно. Если программа говорит как врач, психолог или юрист, человек может ожидать профессиональной компетентности даже без прямого обещания. Вейценбаум позднее настаивал, что некоторые решения не следует передавать компьютеру, даже если его можно запрограммировать выполнить формальную процедуру. Его возражение касалось не вычислительной скорости. Оно касалось вопроса, кто вправе судить и кто несёт ответственность за решение.

Джозеф Вейценбаум: инженер, ставший критиком

К моменту создания ELIZA Вейценбаум был не философом, внезапно заглянувшим в компьютерную лабораторию, а инженером и программистом с опытом реальных систем. Он родился в Берлине в еврейской семье; в середине 1930-х, после прихода нацистов к власти, семья бежала из Германии и поселилась в Соединённых Штатах. Он занимался аналоговыми вычислениями, участвовал в создании цифровых машин, работал над банковскими вычислительными системами и системами обработки списков. Позже он преподавал в Массачусетском технологическом институте.

Этот путь важен, потому что критика Вейценбаума выросла внутри компьютерной практики. Он знал, как формальные процедуры автоматизируют работу, какие задачи удобно передать машине и как инженерные решения становятся частью учреждения. В его карьере были и программы обработки языка, и системы, способные влиять на повседневный труд. Он не отрицал компьютеры и не призывал отказаться от вычислений. Его вопрос был о том, где вычислимое заканчивается и где начинается человеческое суждение, ответственность и ценность.

Опыт банковской автоматизации давал ему дополнительную оптику. Компьютер может ускорить расчёт и изменить штатную работу; он не становится из-за этого носителем цели учреждения. Люди выбирают, какие операции считать, какие процессы перестроить и кому поручить последствия. Техническая эффективность может сопровождаться социальной переменой. Поэтому Вейценбаум замечал не только чудо программы, но и организационную власть, которая возникает, когда машине доверяют статус объективного арбитра.

ELIZA усилила этот интерес. Вейценбаум увидел, что люди могут приписывать машинному выводу человеческие качества. Он не сделал из этого вывода, что любой пользователь наивен или что компьютеры обречены на вред. Скорее он начал задавать вопросы о границах автоматизации: какие задачи разумно поручать машине, а какие включают выбор ценностей и последствия для живых людей? Если компьютер быстро выдаёт результат, это ещё не делает его ответственным за то, что произошло.

В книге *Computer Power and Human Reason* («Вычислительная мощь и человеческий разум», 1976) он развил эту позицию. Название подчеркивало различие между вычислительной мощью и человеческим рассуждением. Вейценбаум критиковал веру, что достаточно увеличить мощность компьютера и ловкость программиста, чтобы система смогла справиться с любой человеческой задачей. Некоторые решения можно формализовать, но вопрос о том, следует ли их формализовать и передавать машине, остаётся политическим и этическим.

Его позицию иногда упрощают до образа человека, который создал ELIZA, ужаснулся и «отвернулся от ИИ». Это неверно. Он оставался учёным и не считал все компьютерные технологии бесполезными. Критиковал он претензию компьютерной логики на универсальность и готовность общества делегировать машине решения, которые требуют человеческого понимания контекста. Скепсис был частью его опыта специалиста, а не отказом от науки.

Кто решает, что можно автоматизировать

История ELIZA помогает отделить несколько вопросов, которые часто сливаются в один. Может ли программа обработать текст? Да. Может ли она подобрать ответ, который покажется уместным в знакомом сценарии? Да. Понимает ли она переживание пользователя так, как его может понять человек? Нет оснований утверждать это по одной только связности разговора. Должна ли организация поручать ей терапевтическую или моральную ответственность? Это уже не технический вопрос о качестве совпадения шаблонов.

Одна из опасностей — считать формальное решение нейтральным. Но кто-то выбирает, какой вопрос должна решать система, какие данные считать правильными и какой порог ошибки приемлем. Если программа выдаёт оценку, решение о применении этой оценки остаётся за учреждением. Фраза «так показал компьютер» может звучать как снятие ответственности, хотя за ней стоит человеческий выбор процедуры.

Вейценбаум различал вычисление и суждение. Компьютер способен выполнять формальную процедуру быстрее и стабильнее человека. Но человеческое решение нередко включает обстоятельства, которые невозможно полностью задать списком заранее. Судья, врач, педагог или психотерапевт не просто выбирает действие из таблицы. Он работает с историей человека, неопределённостью, конфликтующими ценностями и последствиями ошибки.

Это не значит, что люди всегда судят лучше или что профессиональные решения непогрешимы. Человеческие институты тоже несправедливы, предвзяты и ошибаются. Но передача решения машине не уничтожает проблему; она может сделать её менее видимой. Тогда разработчик отвечает за модель, организация — за её применение, а пользователь сталкивается с результатом, который выглядит как факт. Вейценбаум обращал внимание именно на эту цепочку.

В следующих главах книга вернётся к автоматизированным оценкам, системам рекомендаций и решениям, которые меняют доступ людей к работе, деньгам и услугам. Здесь достаточно запомнить ранний урок: способность системы выдать уверенный ответ не отвечает на вопрос, кто вправе считать этот ответ основанием для действия. Голос и роль могут сделать вычисление похожим на суждение; это не одно и то же.

Почему ELIZA не была ранней языковой моделью

Сравнение с сегодняшними чат-ботами возникает естественно. Пользователь пишет текст, система отвечает, разговор продолжается. На поверхности сходство сильное. Но оно не должно стирать техническую пропасть между программами.

ELIZA не обучалась на больших корпусах текстов. Она не предсказывала следующее слово по статистическим закономерностям, найденным в миллиардах примеров. Её ответы собирались по человечески написанным правилам, ключевым словам и шаблонам. Если разговор уходил за пределы этих правил, программа могла повторяться, выдавать неуместную реакцию или просить пользователя сформулировать фразу иначе.

Современные языковые модели обучаются на намного более сложной задаче и способны создавать формулировки, которые никто не записывал заранее как готовые ответы. Они могут учитывать длинный контекст, комбинировать сведения и менять стиль. Это фундаментальное техническое различие. Называть их «тем же самым, только побольше» — значит перепрыгнуть через десятилетия алгоритмической и инженерной работы.

Однако ELIZA остаётся важной не потому, что предсказала нынешнюю архитектуру. Она обнаружила устойчивое человеческое обстоятельство: разговор сам по себе служит сигналом социального присутствия. Мы склонны интерпретировать вопрос как интерес, связную реакцию — как внимание, а воспоминание о наших словах — как знание о нас. Современная система может заслуживать большего доверия в конкретных задачах, чем ELIZA; но сходство интерфейса всё равно заставляет нас проверять, чем именно подтверждается наше впечатление.

Нельзя использовать ELIZA как отговорку, будто все языковые системы «всего лишь шаблоны». Это было бы фактически неверно и аналитически лениво. Но также нельзя принимать беглую речь за доказательство сознания. Здесь работают разные уровни: качество вычисления, способность переносить закономерности, наличие внутреннего опыта и социальная реакция человека. Один из уроков ELIZA — не смешивать их.

Личный разговор и публичное обещание

Люди, знакомившиеся с ELIZA, могли видеть в ней разные вещи. Для одного это был эксперимент по обработке естественного языка. Для другого — неожиданная игрушка. Для третьего — инструмент, который позволял проговорить тему без страха немедленной оценки. Для четвёртого — доказательство, что разговор можно частично автоматизировать. Нельзя вывести одну мотивацию из всех случаев.

У программы была особая асимметрия. Человек знал о себе много; система — почти ничего. Но человек мог принять общий вопрос за реакцию на личную историю, потому что сам сохранял эту историю в памяти. Если собеседник повторял слово «семья», пользователь связывал его с предыдущими событиями. Для программы это мог быть сработавший пункт словаря; для человека — нить разговора.

В этом взаимодействии нет ничего необычного для человеческого общения. Мы постоянно интерпретируем слова в контексте и заполняем паузы. Отличие было в источнике реплики. ELIZA произносила её по алгоритму, но социальная работа всё равно происходила в сознании пользователя. Машина создавала повод; человек превращал его в разговор.

Сложность возникала, когда индивидуальный опыт становился публичным доказательством. «Моя знакомая говорила с программой» превращалось в «люди доверяют ей». «Она продолжала отвечать» превращалось в «она понимает». А «если достаточно усовершенствовать» — в «можно заменить специалиста». Каждый шаг расширял утверждение, не добавляя новых данных о способности программы.

Именно поэтому Вейценбаум настаивал на техническом разборе. Если показать, какие правила стоят за ответом, магия не исчезает из человеческого опыта — но становится понятнее. Пользователь по-прежнему может чувствовать вовлечённость. Просто он знает, что именно делает программа и чего из этого не следует выводить. Инженерное объяснение не отменяет психологический эффект; оно мешает превратить впечатление в необоснованное обещание.

Не экзамен на человечность

Из-за того, что ELIZA отвечала в диалоге, её иногда сближают с тестом Тьюринга. Сходство есть на самом общем уровне: в обоих случаях человеческая речь становится способом оценивать взаимодействие с машиной. Но публикацию Вейценбаума не следует пересказывать как эксперимент, который доказал, что программа успешно выдала себя за человека.

Тест Тьюринга в статье 1950 года описывался через игру в имитацию с определёнными ролями и условиями, а не как всякий случай, когда компьютер печатает ответ. ELIZA же была инструментом для изучения коммуникации между человеком и машиной. Её интересовали работа сценария и реакция на правила, а не победа в слепом сравнении с человеческим собеседником. Если пользователь включался в диалог, это было свидетельством интересного взаимодействия, но не автоматическим подтверждением человеческого интеллекта.

Разница между «могу продолжать разговор» и «не могу определить, кто на другом конце» огромна. Пользователь мог знать, что система механическая, а автор мог открыто описать правила. Это не отменяло впечатления от беседы, но меняло смысл результата. Вопрос был не в том, скрыла ли ELIZA свою природу, а в том, почему даже видимая вычислительная процедура могла занимать социальное место собеседника.

Этот вопрос намного конкретнее спора о том, можно ли вообще назвать машину разумной. Он приглашает разбирать отдельные свидетельства: что программа обработала, как ответил человек, что именно пользователь приписал системе и при каких условиях. Именно такое разделение делает историю ELIZA ценной для широкой аудитории: читателю не требуется сначала решить, что такое сознание, чтобы понять, почему простой разговорный механизм произвёл большое впечатление.

Разговор становится частью машины

До ELIZA искусственный интеллект часто обсуждали через способность решать задачи, которые считались интеллектуальными: доказывать теоремы, искать ходы в шахматах, классифицировать образы. Здесь же внимание сместилось к взаимодействию. Важно было не только, что вычисляет программа, но и как человек читает её ответ и продолжает общение.

Это изменение шире истории одного чат-бота. Любой интерфейс предлагает пользователю модель того, что происходит. Кнопка с надписью «посоветовать» обещает больше, чем кнопка «показать статистику». Персонаж с именем и голосом создаёт иное отношение, чем безличная форма. Память о прошлой беседе может восприниматься как внимание, хотя технически это сохранённая запись. Дизайн не меняет вычислительный механизм напрямую, но меняет ожидания, по которым пользователь оценивает его работу.

DOCTOR был особенно удачно встроен в роль, допускавшую неопределённость. Вопросы возвращали разговор к человеку. Повторы могли звучать как внимательное слушание. Паузы давали время сформулировать мысль. У такой формы были реальные коммуникативные свойства, хотя она не основывалась на сопереживании или понимании личности. Этот парадокс и сделал программу интересной не только программистам, но и психологам, философам и специалистам по взаимодействию человека с компьютером.

Из этого не следует, что пользовательский опыт — обман, а техническая схема — единственная истина. Опыт тоже реален: человеку могло быть легче написать программе то, что трудно произнести знакомому. Но причины пользы и пределы должны быть названы точно. Записная книжка может помогать осмыслить событие, не зная о нём; программа может поддерживать саморефлексию, не став терапевтом. Полезность не доказывает человеческую природу инструмента, а отсутствие сознания не означает нулевую ценность.

Это различие пригодится в будущей истории искусственного интеллекта. Если спорить только о том, «настоящий» ли интеллект у машины, можно пропустить вопросы о её реальном влиянии: какие решения она ускоряет, какие привычки формирует, кому облегчает работу и кого оставляет без человеческого собеседника. ELIZA не знала, что меняет социальные ожидания. Но люди, создававшие и использовавшие её, уже могли обсуждать этот эффект.

У терминала не было человеческого лица

В ранних воспоминаниях об ELIZA акцент обычно падает на ответы программы, но сам терминал был не менее важной частью опыта. Пользователь не видел мигающего аватара и не слышал человеческий голос. Он печатал на устройстве, которое выводило строки на бумагу. Это оставляло заметную дистанцию: перед ним была вычислительная машина, а не персонаж на экране. И всё же ритм обмена репликами достаточно быстро создавал место для воображаемого собеседника.

ELIZA работала на системе с разделением времени, благодаря которой один большой компьютер мог обслуживать разных пользователей. Это отличалось от пакетного режима, где человек отдавал задание и ждал результат, иногда не взаимодействуя с машиной по ходу вычислений. В интерактивном режиме пользователь мог ввести фразу, получить реакцию, уточнить мысль и продолжить. С точки зрения архитектуры это был обмен с общим ресурсом; с точки зрения сидящего за терминалом — отдельная беседа.

Ирония в том, что ощущение «приватного разговора» возникало в инфраструктуре, которая по устройству была коллективной. Мэйнфрейм находился в вычислительном центре, программа работала в общей среде, а терминалы подключались к ресурсу учреждения. У пользователя не было оснований считать этот канал исповедальней с гарантированной тайной. Тем не менее персональный ритм вопроса и ответа мог восприниматься интимно. Интерфейс сузил мир до печатной строки перед человеком, хотя сама вычислительная система оставалась частью организации.

Сегодня такая деталь напоминает, что интерфейс не только помогает взаимодействовать с алгоритмом. Он управляет ожиданием о том, кому видны слова, кто хранит историю и насколько личным является место общения. Вейценбаум в 1966 году создавал эксперимент, а не сервис с обещанием конфиденциальности. Пользователь мог довериться разговору психологически, хотя техническая среда не была устроена как защищённая консультация.

Какой успех можно было считать успехом

Для одной аудитории ELIZA была любопытной программой, которая умеет поддерживать текстовый обмен. Для другой — исследованием того, насколько разговор можно свести к формальным преобразованиям. Для третьей — началом автоматической психологической помощи. Эти прочтения неравноценны, но они могли появиться из одного и того же опыта.

Если задача состояла в демонстрации обработки текста, достаточно показать, что программа находит заданные конструкции и генерирует ответ. Если задача — поддержать связный диалог, нужно проверять уже больше: как часто программа повторяется, может ли работать с неожиданной формулировкой, помнит ли тему, как долго пользователь ощущает связность. Если речь о психологической помощи, критерии становятся ещё строже: безопасность, уместность, последствия и сравнение с профессиональной поддержкой. Одно испытание не закрывает все эти уровни.

В этом отношении ELIZA полезнее не как ранняя победа машины над человеческим разговором, а как пример того, насколько по-разному может пониматься слово «работает». Система могла выполнить техническое преобразование правильно и при этом ответить странно. Могла поддержать беседу несколько ходов и затем сорваться в повтор. Могла оказаться полезной человеку как повод подумать, но не иметь ни терапевтической компетентности, ни доказанной клинической эффективности. Смешение результатов превращает демонстрацию в обещание.

Подобная путаница возникает, когда публичное описание заменяет измерение. «Люди продолжали разговаривать» говорит о поведении пользователей. «Программа обработала ключевое слово» говорит о работе кода. «Человек почувствовал, что его выслушали» говорит о субъективном опыте. Это связанные, но разные наблюдения. Чтобы понять систему, нужно сохранять их отдельными, даже если история делает их частью одного яркого эпизода.

Вейценбаум сам помог разделить эти уровни, подробно описав работу ELIZA. Его техническое изложение было частью аргумента: если показать правила простыми словами, читатель сможет оценить, что программа делает, а что ей приписывает наблюдатель. Он не считал, что техническое объяснение отменит ощущение разговора. Он хотел, чтобы это ощущение не путали с доказательством общего понимания.

Личный текст не всегда требует веры

Мы можем представить близкую сцену: человек печатает о семейной ссоре, программа отвечает общим вопросом, и пользователь продолжает рассказ. Может быть, он уверен, что за ответом нет человека. Может быть, ему просто легче написать, чем объяснить всё знакомому. Может быть, он проверяет, насколько далеко продвинется программа. Продолжение разговора само по себе не показывает, какая из этих причин сработала.

Это важно, потому что человеческое участие в диалоге часто принимают за доказательство доверчивости. Но людям не нужно верить в сознание системы, чтобы разговаривать с ней. Мы вежливы с навигацией, кричим на неисправный принтер, разговариваем с домашним животным и пишем в дневник. В каждом случае социальное поведение может включаться по-разному. Машина способна стать адресатом фразы, не став человеком.

Вместе с тем личный текст остаётся личным. Если пользователь раскрывается, разработчик и организация должны думать, что сохраняется, кому доступно и каково последствие неверного ответа. Это современное расширение урока ELIZA, а не утверждение, что программа 1966 года собирала такие данные или имела современные сетевые функции. Исторический опыт просто делает вопрос видимым: интерфейс может вызвать откровенность раньше, чем пользователь разберётся в технических и институциональных условиях.

Поэтому суть эффекта ELIZA не в том, что «нормальные люди верят машинам». Более аккуратно сказать: форма диалога способна активировать привычки общения даже тогда, когда обе стороны знают, что одна из них — программа. Это наблюдение полезно для дизайна и оценки систем. Не нужно презирать пользователей за участие; нужно понимать, какие сигналы интерфейс подаёт, какие ожидания вызывает и кто несёт ответственность, если эти ожидания не совпадают с реальными возможностями.

От впечатления — к ответственности

Через десять лет после публикации ELIZA Вейценбаум выпустил книгу о вычислительной мощи и человеческом разуме. Изобретатель разговорной программы оказался одним из наиболее заметных критиков чрезмерного доверия к компьютерам. В массовом пересказе это выглядит как раскаяние: создал искусственного собеседника, понял опасность и отвернулся от своего изобретения. Реальная позиция была содержательнее. Он не отрицал техническую ценность машин. Он спрашивал, кто и на каком основании расширяет их власть в человеческих делах.

Вейценбаум считал, что общество слишком легко принимает удобную формализацию за полноценное понимание. Если аспект жизни можно представить числами и правилами, организация может решить, что его можно вычислить и тем самым решить. Но формальная модель всегда выделяет одни свойства и оставляет другие вне рамки. Выбор рамки сам по себе ценностный. Компьютер затем действует внутри неё, иногда с большой скоростью и последовательностью.

В этой логике история ELIZA — не только психологический курьёз. Программа давала человеку ощущение слушателя; учреждение могло бы увидеть в ней более дешёвый способ обеспечить консультацию. Пользователь видел реакцию; разработчик — механизм. Именно между этими перспективами и появляется вопрос ответственности. Кто должен объяснить ограничения? Кто принимает решение о применении? Кто отвечает, если удобный разговор заменил нужную помощь?

Вейценбаум не мог ответить на каждый вопрос будущей индустрии, и ему не следует приписывать предвидение всех сегодняшних систем. Но он рано сформулировал конфликт, который никуда не исчез: способность автоматизировать операцию не доказывает, что её надо автоматизировать. Скорость и повторяемость ценны, но не заменяют моральное суждение. Данные и правила могут улучшить работу, но не решают сами, какие цели достойны преследования.

В этом смысле он продолжил разговор, начатый самой ELIZA. Программа показывала, как пользователь приписывает значение ответу; критика Вейценбаума показывала, как общество приписывает значение вычислительной процедуре. В обоих случаях человек склонен заполнять пустоты: в первом — образ собеседника, во втором — образ объективной машины, которая якобы принимает решение вместо нас.

Программа умела отвечать. Слушать — другое дело

ELIZA не была тайным психологом в терминале. Она не знала пользователя, не понимала его историю и не могла нести ответственность за последствия разговора. Она принимала английский текст, находила ключевые слова, применяла правила разложения и преобразования, а затем выдавала ответ из подготовленного сценария. Всё это можно было объяснить достаточно простым языком — и всё равно разговор с ней мог ощущаться личным.

В этом сочетании механизма и впечатления и заключается сила главы. Машина ничего не чувствовала, но заставила людей обсуждать чувства. Она не понимала чужие слова так, как их понимает человек, но показала, что часть разговора может поддерживаться за счёт участия самого собеседника. Она не стала терапевтом, однако заставила инженеров спросить, где проходит граница между языковым поведением и человеческой ответственностью.

После перцептрона ELIZA предложила почти противоположную модель. Перцептрон корректировал числовые связи по примерам. DOCTOR следовал правилам, написанным человеком заранее. Но обе истории вывели на похожий вопрос: что именно мы наблюдаем, когда программа выдаёт убедительный результат? В одном случае это классификация, в другом — разговор. Ни там, ни там один впечатляющий эпизод не позволяет автоматически заключить, что машина приобрела весь набор способностей, который мы связываем с человеком.

При этом сама ELIZA не вызвала кризис финансирования и не стала причиной первой зимы ИИ. Она была одним заметным эпизодом в более широком поле ожиданий: программа могла впечатлить публику, не решив задачу общего понимания языка. Её история помогает увидеть, как разрыв между показом и обещанием увеличивается; историю того, как разные проекты и финансирующие учреждения реагировали на этот разрыв, предстоит рассмотреть отдельно.

Дальше тема выйдет из лаборатории и упрётся в ожидания, бюджет и масштаб задач. Теорема могла быть доказана, образ — классифицирован, фраза — обработана. Но когда исследовательская демонстрация превращалась в обещание системы общего назначения, оставался вопрос о расстоянии между аккуратной задачей и реальным миром. Именно это расстояние станет одним из главных героев следующей главы — о первой зиме искусственного интеллекта, которую нельзя объяснить одной причиной.

Глава 6. Зима, которую нельзя объяснить одной причиной

Слово «зима» обещает ясную картину. Ещё вчера в лабораториях говорили о машинах, которые будут переводить, видеть и рассуждать; сегодня якобы иссякли деньги, погас свет и исследователи разошлись пережидать холод. У этой истории есть удобный жанр: сперва все поверили в будущее, затем несколько критиков показали ошибку, финансирование исчезло, а искусственный интеллект на годы пропал.

Так не было.

Компьютеры продолжали работать. Статьи выходили. Исследователи строили программы, роботов и новые способы описывать знания. Менялись гранты и критерии оценки; часть направлений теряла поддержку, часть выживала, часть переименовывала себя или доказывала пользу на более узких задачах. Даже слово «зима» появилось как позднее обобщение многих процессов, а не как официальная дата, в которую весь мир отменил исследования.

Охлаждение, однако, было реальным. К началу 1970-х финансирующие организации и исследователи всё чаще спрашивали, где заканчиваются впечатляющие демонстрации и начинаются результаты, которые можно получить в рабочей среде. Отчёты ALPAC в США и Лайтхилла в Великобритании стали заметными эпизодами этого спора. Они не были взаимозаменяемы и не действовали в одинаковых странах; каждый оказался частью собственной институциональной истории. Между ними лежали технические ограничения, невыполненные обещания, разные представления о том, для чего нужна наука, и реальные решения о распределении денег.

Чтобы понять первую зиму, нужно не искать одного виновника, а посмотреть, как надежда превратилась в обязательство отчитаться.

Демонстрация становится обещанием

Молодая технология получает в лаборатории особую привилегию. Ей разрешают работать в небольшом, тщательно подготовленном мире, потому что даже узкий успех может открыть новый способ мышления. Программа доказывает одну теорему; машина переводит несколько предложений; робот переставляет блоки в комнате, где ничего неожиданного не происходит. В этот момент задача не в том, чтобы сразу построить универсальную систему. Важно показать, что компьютер вообще способен выполнить такой тип действия.

Но у демонстрации быстро появляется вторая жизнь. Вопрос «может ли машина сделать это при заданных условиях?» сменяется вопросом «когда она будет делать это везде?». Если переводит короткий текст — когда переведёт статью? Если видит заранее выбранную фигуру — когда разберётся с обычной фотографией? Если задаёт вопрос по шаблону — понимает ли язык? Если робот перемещает предмет на чистом столе — сможет ли он работать в комнате, где вещи лежат не по инструкции?

Каждый следующий вопрос расширяет задачу. Иногда расширяет слишком быстро.

Исследователь может показать, что метод работает на нескольких примерах. Заказчик ждёт надёжного инструмента. Журналист описывает будущее, которое обещает технология. Руководитель гранта должен решить, за что платит общество. Эти ожидания не всегда одинаковы, но они часто сходятся в одной фразе: «Это уже почти готово». Затем выясняется, что между работающим опытом и системой, пригодной для широкого применения, лежит новый слой инженерии.

В 1950-е и 1960-е многие направления искусственного интеллекта росли в атмосфере высоких ожиданий. После войны вычислительные машины расширяли возможности науки и обороны; государственные ведомства поддерживали исследования языка, распознавания сигналов и автоматизации. Холодная война придавала некоторым задачам срочность: понимание зарубежных публикаций, обработка информации, связь и принятие решений. Если компьютер мог ускорить доступ к знанию, это имело практический смысл.

Сама срочность делала обещания привлекательнее. Машинный перевод казался не философской мечтой, а способом быстрее читать научные и технические материалы на русском языке. Машинное зрение обещало помочь разбирать изображения. Автоматизация и робототехника могли освободить людей от определённых операций. Но практическая ценность зависела от качества, скорости, стоимости и количества человеческой работы, нужной после машины. Компьютерная демонстрация отвечала лишь на часть этих вопросов.

Когда вложения росли, рос и спрос на сравнения. Заказчику было мало узнать, что программа выдала текст. Он хотел понимать, быстрее ли это, точнее ли, дешевле ли обычного процесса. Если систему приходится долго настраивать, исправлять вручную и ограничивать узкой предметной областью, её нельзя оценивать только по впечатлению от демонстрации. Это не атака на фундаментальную науку; это различие между исследовательским результатом и обещанием готового продукта.

Первая зима начиналась не с момента, когда машина перестала удивлять. Она начиналась там, где окружающие перестали считать удивление достаточным отчётом о пользе.

Переводчик в центре государственной задачи

Машинный перевод стал одним из самых заметных ранних испытаний. В январе 1954 года университет Джорджтауна и IBM провели публичную демонстрацию перевода с русского на английский. Это был небольшой эксперимент с ограниченным словарём и специально подготовленными предложениями. Его сила заключалась не в том, что машина уже могла переводить любую книгу, а в другом: вычислительная система смогла продемонстрировать саму возможность автоматического перехода от языка к языку.

Пресса услышала возможность масштабирования. Если компьютер переводит простые предложения, возможно, он скоро поможет разбирать большие объёмы советской научной литературы. Для государства это была понятная ставка. Человеческих переводчиков было не так легко нанять в количестве, нужном для всего потока документов; значительная часть материалов могла быть полезна специалистам, даже если перевод не был литературным. Черновой перевод, который передаёт общий смысл, мог бы сэкономить время.

Но перевод — не подстановка одного слова вместо другого. Слова меняют значение в зависимости от контекста, порядок и грамматика различаются, устойчивые выражения нельзя всегда понять буквально, а технические термины требуют предметного знания. Система может выдать грамматически узнаваемый текст и при этом перепутать отношение между объектами. Ошибка в одном слове способна изменить смысл всего предложения. Человеческий переводчик не только знает словарь: он использует контекст, культуру и профессиональную практику.

В середине 1960-х американские ведомства попросили более трезво оценить состояние направления. Для этого был создан Automatic Language Processing Advisory Committee — Комитет по автоматической обработке языков, известный по аббревиатуре ALPAC. Его возглавил Джон Пирс, инженер и исследователь Bell Labs; в работу вошли специалисты разных областей, в том числе лингвисты и исследователи машинного перевода. Перед комиссией стоял не общий вопрос «может ли компьютер когда-нибудь понимать язык?». Её интересовало практическое состояние перевода для американских нужд и оправданность вложений в системы, которые обещали полезный результат в обозримый срок.

Именно поэтому отчёт 1966 года обсуждал не только алгоритмы. Комиссия рассматривала объём и качество доступных переводов, сроки, стоимость, потребности читателей, работу человеческих переводчиков и процессы постредактирования — исправления машинного текста человеком. Если после перевода специалисту всё равно приходится тщательно сверять и переписывать каждое предложение, вопрос не в том, «сработал» ли компьютер вообще. Вопрос — стало ли всей организации лучше.

В этом месте часто возникает карикатура: будто люди из ALPAC не поняли технологию и просто заявили, что машинам запрещено переводить. Но комиссия занималась оценкой системы в конкретных условиях. Её выводы можно критиковать, и позднейшие исследователи критиковали их; однако прежде нужно понять, что именно сравнивалось. Техническая возможность преобразовать текст не тождественна практической замене переводчика.

Перевод — это процесс, а не кнопка

Публичная демонстрация автоматического перевода обычно показывает экранный момент: предложение вошло, предложение вышло. Но заказчику важен весь процесс до и после этого момента. Нужно подготовить словарь, выбрать тип текстов, провести вход через машину, проверить сомнительные места, исправить результат, передать читателю. Если исходник сжатый и технический, одна ошибка может превратить «часть устройства удерживает давление» в противоположное утверждение. Если переводчик не знает предмета, даже грамматически правильная фраза способна ввести в заблуждение.

Постредактирование добавляет ещё одну задачу измерения. Если специалист исправляет машинный текст, нельзя считать лишь секунды вычисления. Нужно учесть, сколько времени он тратит на исправления и сверку с оригиналом. Иногда черновик ускоряет работу; иногда редактору легче начать с нуля; иногда машинная версия полезна для быстрого понимания, но не годится для публикации. Ответ зависит от жанра, языка, точности, ставки ошибки и навыков человека.

Представим две задачи. В одной нужно быстро понять, о чём научная статья, чтобы решить, стоит ли читать её целиком. Черновой перевод может быть полезен даже с неловкими формулировками. В другой нужно перевести инструкцию, по которой техник будет ремонтировать оборудование. Тут цена неверного значения намного выше, а человеческая проверка обязательна. Одна и та же технология может быть удачной в первом процессе и непригодной во втором.

ALPAC оценивал перевод именно как часть рабочего конвейера. В его материалах обсуждались потребность в переводах, скорость получения, качество, стоимость и роль человеческого редактирования. Такой взгляд мог разочаровать команду, чья задача состояла в том, чтобы показать вычислительную возможность. Но именно он был необходим государственным заказчикам, которые хотели знать, экономит ли компьютер время и деньги для реального читателя.

Отчёт также обращал внимание на то, что больше переведённых страниц не автоматически означает больше пользы. Если в огромном массиве материалов лишь небольшая доля действительно нужна, ценность может заключаться в поиске, аннотировании или выборе текстов, а не в переводе всего подряд. Это меняло постановку задачи: иногда труднее не преобразовать каждое предложение, а понять, какой текст вообще стоит преобразовывать.

Для общества здесь видна человеческая ставка. Машинный перевод обещает сократить потребность в ручной работе; одновременно люди остаются теми, кто готовит словари, маркирует ошибки, проверяет тексты и несёт ответственность за опубликованный смысл. Автоматизация может убрать часть операций и добавить другие. Если измерять только компьютерное время, труд человека исчезает из расчёта — а вместе с ним исчезает и честная оценка результата.

Что ALPAC написал — и чего не писал

Отчёт ALPAC оценивал тогдашний машинный перевод как недостаточно качественный и неэффективный по сравнению с человеческим переводом. В нём говорилось об отсутствии немедленной или предсказуемой перспективы полезного машинного перевода в том виде, которого ожидали финансирующие организации. Это было суровое заключение о состоянии работ и ближайшей отдаче, а не доказательство вечной невозможности перевода компьютером.

Разница принципиальна. Отчёт не говорил, что языковые исследования бесполезны. Он призывал продолжать работу по вычислительной лингвистике, улучшать методы оценки переводов, поддерживать средства, помогающие людям переводить, и разбираться в реальной потребности читателей. В рекомендациях были словарные и редакторские инструменты, организация переводческого процесса и оценка качества. Комиссия не предлагала просто закрыть любую работу с языком.

Однако в последующие годы финансирование практических систем машинного перевода в США действительно сократилось, а сам отчёт стал символом разочарования в направлении. Вот где возникает расхождение между документом и его последствиями. Текст отчёта может содержать более сложные рекомендации; учреждение, которое распределяет бюджеты, может услышать прежде всего: «обещанный результат не пришёл». Последующая память может ещё сильнее упростить историю: «ALPAC остановил машинный перевод».

Такой эффект характерен для оценки технологий. Комитет исследует конкретную программу и её пользу, но результаты оценки начинают воспринимать как суждение обо всей области. В ведомстве ищут решение, которое можно принять на основании отчёта; исследователь видит риск для долгой научной работы; историк позже пытается отделить первоначальный текст от решений, принятых после публикации.

ALPAC сравнивал системы не только с идеальным переводчиком, которого не существует, но и с реальной профессиональной работой и потребностями заказчиков. Для него было важно, сколько времени уходит на постредактирование, как читатели используют переводы и даёт ли система достаточный выигрыш. Такая проверка могла быть слишком узкой для оценки фундаментальной ценности языкового исследования, но достаточно уместной для вопроса об инвестициях в конкретный продукт.

Государство могло сделать два разных вывода. Первый: текущие машинные переводчики не обеспечивают обещанную экономию и качество. Второй: поэтому следует отказаться от всех связанных исследований. Первый вывод поддерживался отчетом гораздо лучше, чем второй. Но в реальном бюджете эти различия иногда стираются. Финансирование не всегда устроено так, чтобы ясно отделять фундаментальную работу от прикладных обещаний, особенно когда обе части продаются под одним названием.

У истории была и географическая сторона. Снижение практического финансирования в США не означало, что работа исчезла по всему миру. Исследования продолжались в других странах и организациях; в Канаде, Европе и Советском Союзе существовали свои задачи и программы. Позже появились работающие системы для ограниченных предметных областей. Понятие «зима машинного перевода» точнее, чем фраза «машины перестали переводить»: оно описывает охлаждение ресурсов и ожиданий в отдельных контекстах, а не исчезновение научной проблемы.

ALPAC стал сильным предупреждением для широкой области, потому что поднял неудобный вопрос: кто решает, какой результат считается достаточным? Переводчик может ценить помощь, даже если она не заменяет его работу. Заказчик может требовать полной автоматизации. Исследователь может считать ценным новый метод, даже если коммерческая польза отложена. Один отчёт редко способен одновременно оценить все эти виды ценности.

Невозможность, неудача и неподходящая цена

Когда система не достигает обещанного, люди обычно смешивают три разных вывода. Первый: конкретная программа оказалась слабой. Второй: используемый метод плохо справляется с этим типом данных или масштабом задачи. Третий: сама задача невозможна для машин. Первый и второй выводы могут быть основаны на опыте. Третий требует намного более сильного доказательства.

В 1966 году слабость машинного перевода не означала, что компьютер никогда не сможет переводить тексты с практической пользой. Но позднейший успех статистических или нейронных методов не делает раннюю критику несправедливой. Если система того времени была медленнее, дороже и менее точной, чем альтернативный процесс, это было важным фактом о той системе. Будущее не может задним числом сделать её более полезной.

Так же важно не путать «невозможно» и «не готово». Некоторые проблемы действительно требуют новых идей; другие становятся решаемыми, когда появляется больше вычислительной мощности, данных или качественно иной алгоритм; третьи остаются дорогими из-за нужды в человеческом контроле. В середине XX века невозможно было знать, какой из этих случаев перед исследователями. Поэтому честный отчёт должен говорить о том, что можно измерить сейчас, и ясно обозначать, что остаётся гипотезой.

Технологическое обещание часто содержит срок, но не всегда содержит план проверки. «Когда-нибудь компьютер сможет переводить всё» — практически бессодержательное обещание, если не сказано, какой язык, какая предметная область, какой уровень ошибок, сколько времени и при чём контроле. ALPAC показал цену отсутствия этих условий. Комитет спрашивал не только о том, существует ли демонстрация, но и о затратах всего процесса.

Это не означает, что вся фундаментальная наука должна оправдываться немедленной окупаемостью. Ядерная физика, теория информации и многие математические результаты могли долго не иметь массового применения. Но практическая программа, которую финансируют именно потому, что обещана скорое решение, не может бесконечно просить оценивать её как чистое исследование. Здесь проблема часто не в науке, а в двусмысленности заявления: публике обещают продукт, а при провале защищают фундаментальную идею.

Идеи могут быть ценны и без продукта. Продукт может быть полезным, даже если он не реализует все теоретические амбиции. Но нельзя подменять одно другим в разговоре о результатах. Когда задача узкая, это нужно говорить. Когда результат экспериментальный, его не следует рекламировать как готовое решение. Когда обещан масштаб, нужно объяснить, что позволит системе его достигнуть.

У каждого результата свой метр

Финансирующая организация часто просит один ответ: достиг ли проект цели? Но в молодой области сама цель может иметь несколько уровней. Программа может доказать, что операция вычислима; получить устойчивый результат на тщательно заданном наборе; продемонстрировать пользу в одной организации; стать экономически выгодной; наконец, приблизиться к общему интеллекту. Успех на одном уровне не автоматически отвечает на следующий.

Это видно на переводе. Машина могла показать, что определённые конструкции можно преобразовать автоматически. Затем нужно было проверить качество на тексте, который не готовили специально. Потом — оценить, помогает ли черновик переводчику. И лишь затем — спросить, сможет ли система заменить часть профессионального процесса. Если публичный рассказ перепрыгивал через эти ступени, появлялось разочарование; если отчёт считал ценным только конечный продукт, он мог недооценить промежуточные научные результаты.

Та же проблема существовала в робототехнике. Один удачный захват предмета доказывает, что определённая конфигурация сработала. Стабильная серия захватов — уже лучшее свидетельство надёжности. Работа в незнакомой комнате — более сильное испытание переноса. Способность действовать без заранее подготовленной среды — ещё более высокая планка. Слова «робот умеет брать предметы» не говорят, на какой именно ступени находится система.

Полезный отчёт должен поэтому называть измеряемый уровень, а не только итоговый ярлык. «Система работает» слишком расплывчато. «Система правильно обрабатывает заданный набор примеров при ручной проверке результата» — длиннее, зато сообщает границы вывода. Такая точность могла бы снизить завышенные ожидания и одновременно позволить обществу ценить ранний прогресс, не принимая его за готовый продукт.

В этом смысле спор 1970-х касался не только будущего ИИ, но и языка оценки. Кто выбирает метр? Какой этап считать достаточным, чтобы продолжить грант? Когда узкое улучшение оправдывает расширение проекта? Эти вопросы не имели универсального ответа, но их отсутствие делало разочарования почти неизбежными.

Комбинаторный взрыв: не один враг, а свойство масштаба

В 1970-е обсуждение ограничений ИИ часто приводило к выражению «комбинаторный взрыв». Термин звучит как предупреждение о катастрофе, хотя описывает простую арифметику. Если в каждом шаге у программы есть несколько вариантов выбора, число возможных цепочек быстро растёт с длиной задачи. Из двадцати действий можно составить множество последовательностей; если каждое действие открывает новые состояния, перебор очень быстро становится невозможным.

Представим робота, которому нужно забрать книгу со стола. В лаборатории стол пуст, книга лежит в известной точке, путь свободен, рука заранее настроена. Программа выбирает заданный порядок: подъехать, схватить, поднять, переместить. Теперь добавим реальную комнату. На столе чашка; книга частично закрыта бумагой; свет падает сбоку; ручка робота движется с погрешностью; человек ставит сумку на пути. Система должна распознать, какие предметы перед ней, какие важны, что можно двигать, что нельзя, как избежать столкновения и что считать успешно выполненной задачей.

Каждая новая возможность добавляет ветвления. Можно выбрать разные цели, разные предметы, разные способы захвата, разные маршруты. Программа, которая перебирала варианты в маленьком мире, может оказаться беспомощной в комнате, где человек даже не замечает половины условий. Он использует знания о том, что такое стол, книга, край, хрупкий предмет, препятствие. Эти знания не появляются автоматически из списка возможных движений.

Комбинаторный взрыв был не открытием 1970-х и не приговором искусственному интеллекту. Исследователи давно знали, что полный перебор растёт слишком быстро. Разные методы пытались уменьшить пространство поиска с помощью эвристик: правил, которые помогают выбирать перспективные варианты и игнорировать маловероятные или бессмысленные. Вопрос заключался в том, удастся ли сформулировать достаточно хорошую эвристику и сколько знаний придётся заранее вложить в систему.

Ещё один вопрос: как система узнает, что попала в новый случай? Если программа отбрасывает вариант, который кажется ей бесперспективным, а тот на самом деле ведёт к решению, эвристика может отсечь нужный путь. Если не отбрасывать почти ничего, поиск разбухает. Умение выбирать — не бесплатное сокращение вычислений; оно само требует знания задачи и критериев.

Слово «взрыв» иногда звучало так, будто достаточно добавить компьютеру больше мощности — и трудность исчезнет. Но рост вычислительной скорости не всегда успевает за ростом вариантов. Это похоже на увеличение скорости автомобиля, когда число развилок удваивается после каждого километра: быстрее ехать полезно, но карта всё равно становится сложнее. Новые ресурсы решают часть проблем и одновременно позволяют ставить более трудные задачи.

Лайтхилл сделал комбинаторный взрыв центральным объяснением разочарований, но не изобрёл сам аргумент и не был единственным, кто его обсуждал. Его отчёт придал этой трудности особый вес для финансирующего совета. Для искусственного интеллекта он стал вопросом о том, есть ли у поля общий способ справляться с ростом сложности или существуют только локальные алгоритмы, применимые к отдельным задачам.

Британия искала собственный масштаб

К началу 1970-х британские университетские группы пытались понять, какое место занимать в международной вычислительной науке. В Эдинбурге работали над машинной логикой, восприятием и робототехникой; в других университетах развивались лаборатории экспериментальной биологии, машинного зрения и автоматизации. У этих групп были разные научные вопросы, но государственному совету нужно было решать, что именно из множества заявок является частью «искусственного интеллекта» и как сравнить просьбы о поддержке.

У области не было единого согласованного определения. Кто-то видел её как производство автоматических помощников, кто-то — как теорию интеллекта, кто-то — как математическое моделирование нервной системы. Одна и та же лаборатория могла работать одновременно на нескольких стыках. Из-за этого счётчик «сколько сделано ИИ?» не имел очевидной единицы измерения.

Такой административный вопрос легко принять за бюрократическую помеху. На деле он определял рамку продолжения работы. Если совет считал робототехнику прикладной, он мог требовать показать полезный аппарат. Если считал её инструментом для когнитивной теории, результатом могли быть знание о планировании или обработке сигналов. Если воспринимал её как обещание общего машинного интеллекта, планка результата уходила в будущее.

Лайтхиллу поручили превратить неоднородную область в обзор, пригодный для решений. Это не означало, что сам совет уже решил закрывать лаборатории. Но растущее число заявок означало, что привычное доверие больше не было достаточным: нужно было объяснить, какие направления существуют, чего уже достигли и почему следующие инвестиции оправданы.

Чистый мир и настоящая комната

Многие ранние программы проверяли в так называемом «мире блоков»: несколько предметов, плоская поверхность, ограниченный набор действий. Это был полезный лабораторный мир. Исследователь мог изолировать проблему планирования: какие действия нужно выполнить, чтобы переместить блок А из-под блока В, не нарушая заданных правил? Если программа находила допустимую последовательность, это показывало, что поиск и представление состояния можно автоматизировать.

Но реальная комната добавляет неопределённость ещё до первого действия. Предмет может быть частично закрыт; его контур может совпадать с фоном; поверхность — быть скользкой; освещение — менять оттенок; рука робота — иметь погрешность. Система должна решить, что именно видит, а затем выяснить, какое действие безопасно. Если план рассчитан на состояние «блок стоит здесь», а камера ошиблась на несколько сантиметров, дальнейшие шаги могут последовательно закрепить ошибку.

У человека такой сбой часто остаётся незаметным. Мы слегка меняем хват, переставляем ноги, замечаем, что предмет тяжелее, чем выглядит. Каждый небольшой коррективный акт использует знания, приобретённые в обычной жизни. Компьютерная программа, напротив, должна иметь процедуру для каждого случая, который может нарушить план. Можно добавить новые правила, но это увеличивает пространство возможных взаимодействий и число исключений, которые тоже нужно учитывать.

Так становится понятно, почему победа в лабораторном сценарии не была бессмысленной и одновременно не решала задачу робота для дома. Упрощённый мир позволял увидеть, какую часть поведения уже можно формализовать. Реальная среда показывала, сколько ещё скрытых предпосылок остаётся незадаанными. Ошибка состояла не в том, чтобы начать с маленького мира; ошибка возникала, когда маленький мир выдавали за близкий эквивалент большого.

Лайтхилл видел именно этот разрыв в робототехнике. Даже различить предметы разной формы на достаточно чистом фоне требовало значительных вычислений; зрительно-моторная координация добавляла инженерные сложности, которые не исчезали от простого увеличения скорости компьютера. Его описание было пессимистичным, но сама сложность задачи подтверждалась опытом лабораторий. Спорным оставалось то, что из этих трудностей следовало для долгосрочной ценности исследований.

Лайтхилл получает поручение

В Великобритании к началу 1970-х складывалась своя история охлаждения. Исследования искусственного интеллекта были сосредоточены в нескольких университетских группах, особенно заметных в Эдинбурге, Кембридже и других центрах. Учёные работали над робототехникой, машинным восприятием, моделями мозга и общими теориями решения задач. Это была небольшая область, но она уже обращалась к Science Research Council за заметной поддержкой.

Совет получал всё больше заявок на исследования, которые относили к ИИ, и хотел иметь общую оценку поля. Он попросил математика Джеймса Лайтхилла подготовить обзор. Лайтхилл был известным специалистом по прикладной математике, не исследователем ИИ. Этот факт позже стал предметом критики. Однако Совет намеренно искал внешнего оценщика, который мог бы рассмотреть заявки с позиции человека вне сообщества.

В самом докладе Лайтхилл честно описал условия работы. Он изучил литературу и разговаривал с исследователями примерно два месяца; назвал текст личной оценкой, а не исчерпывающим техническим обзором. Это уточнение важно. Документ не претендовал на окончательный приговор всей области. Он был инструментом для совета, которому нужно было решать, как распределять ресурсы.

В 1972 году Лайтхилл завершил свой обзор; в 1973 году Science Research Council опубликовал его в сборнике вместе с комментариями учёных. Заглавный текст назывался «Искусственный интеллект: общий обзор». Публичный формат позволял не только прочитать оценку внешнего эксперта, но и увидеть ответы людей, чьи программы и лаборатории оценивались. Таким образом доклад стал и административным документом, и площадкой для открытого спора.

Лайтхилл пытался не просто перечислить удачные и неудачные программы. Он спрашивал, существует ли у ИИ достаточная научная связность, чтобы оправдать отдельную область исследований, и как оценивать движение от локальных задач к более общим формам интеллекта. Его критика относилась к большой дистанции между обещанными целями и достигнутыми результатами. Роботы, которым приписывали зрение и здравый смысл, работали в специально подготовленных условиях; перенос в обычный мир оставался чрезвычайно трудным.

Представитель финансирующего совета спрашивал: если в заявке говорится об общем интеллекте, по каким промежуточным результатам оценить прогресс? Исследователь мог ответить, что теория развивается и измерить её пользу заранее невозможно. Это не обязательно означало, что один из них не понимает науку. Они решали разные задачи: один распределял ограниченные средства между конкурирующими предложениями, другой искал объяснительный принцип, который не обязан сразу стать продуктом.

Лайтхилл был не злодеем, который пришёл закрыть лаборатории. И исследователи не были безответственными мечтателями, которые просто просили денег на роботов из фантастики. Разногласие касалось того, как связать фундаментальный поиск с практическими целями и когда считать обещание достаточно зрелым, чтобы продлевать финансирование.

Буквы А, В и С

Лайтхилл разделил исследовательское поле на три категории. «А» — продвинутая автоматизация, то есть применение вычислений к задачам с практической целью. «С» — модели центральной нервной системы, где компьютеры помогали исследовать биологические и психологические механизмы. «В» — создание роботов и более общих систем, которые должны были связать методы автоматизации с моделями познания и поведения.

Такая схема делала поле наглядным для человека, который должен оценить, что именно финансировать. Но она была не нейтральной картой. В категории А находились более прикладные задачи, а в С — работы, опиравшиеся на биологические вопросы. Самая смелая претензия на собственно искусственный интеллект концентрировалась в В: программа должна была получить общее поведение, которое не сводится к одной автоматизируемой операции или к одной модели мозга.

Лайтхилл считал, что достижения А и С можно оценивать по их собственным целям, хотя и там прогресс оказывался скромнее некоторых ожиданий. Для В, которую он связывал с построением роботов, разрыв между целями и результатами был особенно велик. Роботы в лабораториях с трудом справлялись с задачами, которые человеку кажутся элементарными: видеть объекты, координировать движение, перемещаться по незнакомой среде. Отсюда он задавал вопрос, насколько разумно финансировать промежуточный проект, если он не демонстрирует ни быстрой прикладной ценности, ни ясного научного результата.

Критика исследователей была направлена не только на его скепсис, но и на сами категории. Некоторые возражали, что Лайтхилл помещает базовые исследования интеллекта в категорию «строительства роботов», словно робот — конечная цель, а не испытательное оборудование для теоретической работы. Другие настаивали, что исследования эвристик, представлений и программирования уже дают знания о том, как организовать сложное поведение, даже если робот не способен обслужить дом.

Вот где спор становился вопросом об определении поля. Лайтхилл видел отдельные исследовательские программы и спрашивал, образуют ли они одну связную дисциплину. Его оппоненты видели общую задачу — понять и моделировать интеллектуальное действие — и считали, что множество локальных экспериментов со временем соединятся в общую теорию. Если каждый оценивается по краткосрочной практической отдаче, такая теория может так и не появиться; если область защищает себя лишь будущим синтезом, совету трудно решить, когда она добилась прогресса.

Лайтхилл был особенно обеспокоен тем, что рост компьютеров сам по себе не решит проблему. Если число возможных состояний увеличивается гораздо быстрее, чем скорость обработки, просторный склад гипотез может превратиться в ловушку. Чтобы двигаться дальше, нужна не просто более мощная машина, а организация знания и стратегии поиска. В этом пункте критика касалась реальной технической трудности, хотя выводы о сроках и перспективах оставались спорными.

В докладе были также упомянуты очень широкие предсказания о машинах, способных превзойти человека в течение десятилетий. Лайтхилл сопоставлял их с тем, что исследователи уже умели делать. Это было сильное риторическое решение: читатель видел не только локальный результат, но и расстояние до заявленного будущего. В ответ участники сообщали, что такие предсказания не отражали взгляды большинства специалистов или не были планом их лабораторий.

Этот спор тоже нельзя свести к тому, что «учёные обещали сверхразум, а потом не смогли». Некоторые самые громкие прогнозы были публичными и реальными, но не каждый исследователь их разделял. Преувеличение ожиданий могло приходиться из заголовков, заявок,

интервью или из отдельных оптимистических оценок. Нельзя распределить ответственность за общий образ технологии по одному набору цитат. Нужно смотреть, кто говорил, в каком контексте и какие решения эти слова должны были убедить принять.

Ответы и публичная сцена

Сборник Science Research Council включал не только обзор Лайтхилла, но и ответы других участников. Среди тех, кто спорил с его формулировкой поля и оценкой результатов, были британские исследователи Дональд Мичи и Ричард Грегори; американский учёный Джон Маккарти также изложил возражения. В 1973 году дискуссия продолжилась в публичных дебатах в Королевском институте в Лондоне. Телевизионная камера фиксировала спор, который обычно происходит в грантовых отзывах, письмах и коридорах университетов.

Публичная сцена привлекала внимание именно потому, что за ней стоял распределительный вопрос. Разговор был не только о том, можно ли считать эвристику признаком общего интеллекта или правильна ли классификация «А/В/С». Он касался того, какие исследования получают ресурсы, каким лабораториям дадут возможность продолжать работу и какие обещания будут считаться обоснованными.

Мичи и другие исследователи возражали против того, чтобы судить общие методы только по видимому поведению роботов. В их интерпретации робототехника могла быть способом проверять теорию: машина — лабораторный прибор, на котором видно, как представления и планы взаимодействуют с миром. Если смотреть только на то, насколько полезен сам робот, можно не заметить теоретическую ценность эксперимента. Похожим образом аэродинамический туннель ценен не потому, что сам летит, а потому, что позволяет изучать полёт.

Это сильный аргумент, но он не отменяет вопроса совета: сколько стоит эксперимент и как понять, что теория действительно продвигается? Фундаментальная наука не обязана немедленно выпускать продукт, но и слово «теория» не может автоматически оправдать любой проект. Нужны критерии, и сами критерии становятся частью научного спора.

Лайтхилл отвечал за оценку от внешнего лица и практических перспектив; участники — за описание внутренней логики исследований. Их позиции не симметричны: одному поручили помочь совету принять бюджетные решения, другие защищали направления, в которых работали. Это не лишает ни одну сторону аргументов. Напротив, показывает, почему спор нельзя читать как чистую техническую дискуссию. У разных участников были разные обязанности и разные риски.

В публичном поле отчёт начал означать больше, чем его страницы. Для политиков он мог быть сигналом, что ожидания завышены. Для исследователей — сомнением в легитимности всей дисциплины. Для газет — историей о том, что машины разочаровали. Сборник с ответами усложняет эту картину, потому что фиксирует сам процесс несогласия. Это был не приговор, с которым никто не спорил, а полемика, где стороны пытались определить, что считается исследовательским прогрессом.

Американская переоценка шла своим путём

В США за военные исследования ИИ отвечали не те же учреждения, что распределяли университетские гранты в Великобритании. Одним из заметных источников поддержки была DARPA — ведомство, финансировавшее исследовательские программы с возможным значением для обороны. У таких программ был особый договор с фундаментальной наукой: заказчик мог дать исследователям свободу, пока верил, что долгий поиск в итоге создаст важные возможности.

По мере накопления разочарований эта свобода сужалась. После того как британский спор стал заметен в США, Министерство обороны организовало American Study Group для оценки программы ИИ. Исторический обзор National Research Council связывает эту проверку с изменением приоритетов DARPA: средства стали больше направлять на «миссионерские» исследования с более ясной связью с оборонными задачами и меньше — на неограниченный поиск общих методов. Это было не простое прекращение финансирования, а смена условий доступа к нему.

На практике различие могло быть жёстким. Группа, которая строила робота для общего исследования восприятия и движения, должна была объяснить, как её работа связана с конкретной задачей. Программа речи, планирования или машинного зрения с понятным заказчиком получала более убедительную заявку. При этом граница между фундаментальным и прикладным не всегда чистая: методы, созданные для одной задачи, могут позже оказаться полезны в другой. Но в момент составления бюджета будущее применение конкурировало с задачами, которые уже имели заказчика.

Эта система создавала не только давление, но и новые возможности. Оборонные проекты могли финансировать дорогостоящие вычисления, крупные команды и долгую инженерную работу. Прикладной фокус также заставлял исследователей измерять производительность на реальных условиях. Обратная сторона — риск, что программы, не способные быстро показать миссионерскую пользу, перестанут получать поддержку, даже если их научная ценность ещё не проявилась.

Поэтому американскую часть первой зимы лучше описывать как изменение режима финансирования, а не как «DARPA бросила ИИ». Отдельные программы продолжали существовать, но у исследовательских групп появлялся более узкий коридор, в котором нужно было обосновывать работу. Сильнее всего это било по направлениям, чья польза заключалась в общем понимании интеллекта, а не в готовой функции для определённого заказчика.

Лайтхилл не закрыл американские лаборатории

Часто можно встретить версию, будто отчёт Лайтхилла вызвал первую зиму искусственного интеллекта во всём мире. Это неверно по географии и по времени. Его мандат относился к британскому исследовательскому полю и решениям Science Research Council. В Соединённых Штатах финансирование и политика менялись по собственной траектории, хотя британская дискуссия повлияла на то, как американские ведомства обсуждали свои программы.

После доклада Министерство обороны США организовало American Study Group — группу оценки программы искусственного интеллекта, связанной с DARPA. Она задавала некоторые похожие вопросы о результатах и практической ценности. Исторический обзор National Research Council связывает эту переоценку с более сильным поворотом DARPA к целевым, связанным с конкретной миссией проектам и отдалением от неограниченных фундаментальных программ. Это не означало, что деньги на ИИ внезапно исчезли; скорее менялось то, какие темы и обещания были наиболее убедительны для заказчика.

В этом различии важна логика оборонного финансирования. Когда государство поддерживает фундаментальную работу, оно может рассчитывать на долгосрочный эффект, который заранее трудно определить. Когда заказчик требует миссионной направленности, проект должен демонстрировать связь с конкретной задачей. Такая связь не всегда означает немедленный продукт, но она меняет рамки заявки. Робот, который свободно исследует проблему навигации, может потерять приоритет перед системой, которая прямо решает задачу управления или распознавания для конкретного ведомства.

При этом «ориентированность на миссию» могла даже увеличить финансирование отдельных направлений и дать им большие прикладные бюджеты. Но программа, не укладывавшаяся в приоритет, могла пережить сокращение. Поэтому лучше говорить о перераспределении, изменении критериев и потере свободы для части исследовательских групп, а не о тотальном выключении ИИ в США.

В Великобритании пострадавшие области тоже нельзя описывать как исчезнувшие без следа. Поддержка отдельных центров и групп стала сложнее; исследователи могли искать прикладные обоснования, менять организационную рамку или продолжать работу на меньшем масштабе. Данные о расходах на ИИ и компьютерные исследования тогда не собирались в виде единой современной статистики, поэтому утверждения о точном общенациональном падении финансирования требуют осторожности. Истории отдельных лабораторий могут быть драматичными, но они не автоматически описывают всё поле.

Позднее британские парламентские исследователи отмечали, что часто повторяемое утверждение о прямом и всеобъемлющем эффекте Лайтхилла трудно подтвердить систематическими данными. Скепсис к области и сокращение поддержки существовали; но точный масштаб влияния доклада и его связь с каждым бюджетным решением остаются предметом исторического анализа. Эта оговорка не оправдывает отчёт и не отменяет последствий. Она помогает не путать символ события с точным измерением его эффекта.

У слова «зима» нет единственной даты

Историки и инженеры обычно помещают первую зиму где-то между серединой 1960-х и концом 1970-х, но границы зависят от того, о какой программе идёт речь. Для машинного перевода важной датой стал отчёт ALPAC 1966 года. Для британского ИИ — доклад Лайтхилла 1972 года и его публикация в 1973-м. Для некоторых американских лабораторий значимее были собственные обзоры DARPA и переход к более целевым программам. Всё это происходило не одновременно.

И само выражение «зима ИИ» стало ретроспективной рамкой. Оно объединяет этапы, которые участникам не обязательно казались одним историческим сезоном. Исследователь мог видеть сокращение средств в своей лаборатории, но одновременно наблюдать рост другой области. Студент мог считать, что его направление только начинается; чиновник — что обещания давно не оправдываются; журналист — что публика больше не интересуется темой.

Метафора делает историю доступной, но может создавать ложную паузу между двумя эпохами подъёма. На деле работа 1970-х повлияла на экспертные системы, робототехнику, программирование и методы представления знаний, которые позднее снова вышли на первый план. Идеи не ждут весны в герметичной банке. Они сохраняются через статьи, людей, учебные программы, код и исследовательские инструменты.

В случае машинного перевода география особенно важна. Американское охлаждение практических исследований не равнялось мировому прекращению работы. За пределами США продолжали развиваться проекты, в том числе направленные на более ограниченные типы текстов и задач. Систему, предназначенную для одной категории научных или погодных сообщений, легче оценить и довести до полезного режима, чем универсальный переводчик, который должен справляться с любым стилем.

Отсюда следует осторожное правило для исторического рассказа: указывать, для какого ведомства, страны, области и периода использован термин «зима». Иначе читатель воображает единый выключатель, хотя на деле перед нами карта с разными погодными условиями.

Зима была неодинаковой

Слово «первая зима ИИ» объединяет периоды и страны, которые не переживали одинакового холода. Машинный перевод после ALPAC, британские университетские группы после отчёта Лайтхилла, американские проекты DARPA и нейросетевые исследования после критики перцептрона — это разные истории. Они соприкасаются в общественном снижении ожиданий, но их нельзя свести к одному решению или общему календарю.

Даже внутри одной области разные лаборатории могли переживать спад по-разному. Университет с сильной поддержкой от нескольких источников мог продолжать программу, а небольшая группа, зависящая от одного гранта, должна была искать новую тему. Прикладная система могла сохранить средства потому, что обещала определённую функцию; теоретическая программа могла потерять их, потому что её ценность было сложнее объяснить за короткий срок. Для студента, который готовился к защите диссертации, смена приоритетов была не абстрактным изменением терминологии — она могла повлиять на доступ к вычислениям, руководству и будущей работе.

Не все исследования назывались «искусственным интеллектом» и до спада. Программирование, распознавание образов, планирование, когнитивное моделирование и автоматизация могли жить в разных дисциплинах. После того как термин стал ассоциироваться с завышенными ожиданиями, некоторые исследователи с большей осторожностью использовали его в названиях заявок. Это не обязательно скрывало работу; иногда конкретное обозначение точнее говорило, что именно изучается.

Ресурсы тоже не исчезали одинаково. Компьютерная наука в целом продолжала расти, а финансирование вычислительных систем могло переходить от открытого исследования к проектам с более ясной прикладной целью. Это означало не конец вычислительной науки, а изменение отношений между фундаментальным вопросом и государственным заказом. Работать над программой можно было, но всё труднее было просить деньги просто «на общий интеллект».

Исследования за пределами США и Великобритании также не остановились. Канада продолжала развивать машинный перевод в задачах с ясным доменом, например переводом погодных сводок; в Европе и Советском Союзе были свои коллективы и приоритеты. Позднее системы, ограниченные конкретным словарём и форматом, могли оказаться полезными там, где универсальный переводчик проваливался. Это показывает: сужение задачи могло быть не капитуляцией, а методом поиска полезной точки применения.

Если назвать все эти различия одним словом «зима», нужно помнить, что это метафора, а не единый режим погоды. Где-то инвестиции сокращались; где-то менялась их структура; где-то публичный интерес падал, а техническая работа продолжалась. Исторический ярлык помогает увидеть общий ритм, но может скрыть тех, кто продолжал работать внутри периода, который позднее описали как остановку.

Когда меняется грант, меняется и биография

У разговоров о финансировании есть соблазнительная безличность. «Поддержка области снизилась» звучит как изменение линии на графике. Но за ней стоят конкретные решения: продолжит ли студент исследование после защиты; получит ли лаборатория время на больший прототип; сможет ли технический сотрудник сохранить место; возьмётся ли молодой учёный за тему, которую комиссия может счесть рискованной.

Мы не должны придумывать, что каждый исследователь 1970-х чувствовал себя изгнанником или считал, что его карьеру уничтожили. Положение разных людей зависело от учреждения, источников финансирования и того, как их работу можно было описать. Но структурное следствие понятно: чем труднее получить средства на открытую программу, тем больше лаборатории вынуждены подтверждать ближайшую пользу или искать название, которое точнее совпадает с приоритетом гранта.

Прикладная формулировка могла помочь сохранить вычислительные ресурсы и исследовательскую группу. Она могла также сузить вопрос. Если лаборатория изучала общие методы поиска, но заявляла проект как оптимизацию одного процесса, ей приходилось удерживать одновременно научную амбицию и обещание конкретному заказчику. Иногда эта связь становилась плодотворной; иногда разрыв между ними накапливал новое разочарование.

Сокращение средств влияло и на невидимую инфраструктуру. Для работы ИИ нужны не только профессора и авторы статей. Нужны компьютеры, операторы, программисты, инженеры, специалисты по данным и техническое обслуживание. Когда проект заканчивается, может исчезнуть не только гипотеза, но и команда, которая научилась строить эксперимент. Восстановить такую способность позже бывает труднее, чем вновь опубликовать идею.

Это не аргумент против оценки результатов. Он показывает, что бюджетное решение имеет цену в обе стороны. Финансировать неподтверждённое обещание дорого; отказаться от программы, которая могла созреть при следующем технологическом условии, тоже дорого. Оценивать приходится не только вероятность успеха, но и то, как распределяется риск: кто платит за неудачу, кто теряет возможности и кто получает пользу, если метод работает.

Наука и отчётность: конфликт, который никуда не исчез

История первой зимы задаёт неприятный вопрос: кто имеет право требовать от исследователя практической пользы и в какой момент? Если от фундаментальной работы ждать немедленного дохода, часть открытий никогда не получит времени на развитие. Если обещание будущего автоматически оправдывает продление гранта, у исследовательской программы исчезает механизм подотчётности.

Лайтхилл считал тесную связь исследования с практической задачей особенно важной. Историк науки Джон Агар, анализирувавший отчёт, показал, что позиция Лайтхилла была связана с его устойчивым представлением о том, как должна выглядеть хорошая наука: она должна быть сопряжена с реальным решением задач. Кроме того, хотя доклад часто описывают как независимый, Science Research Council задавал ему предварительный контекст и направление. Это не означает, что текст был заказным приговором; значит, его независимость не следует понимать так, будто он возник вне институциональной задачи.

Исследователи, в свою очередь, могли утверждать, что хороший теоретический механизм ценен до того, как его применили в роботе или продукте. Они также были правы: инструмент иногда сначала меняет способ думать о задаче, а практическая польза приходит позже. Проблема начиналась, когда широкий лозунг «интеллект скоро станет общим» заменял объяснение того, что именно будет проверено за следующий грантовый период.

Оценка науки поэтому должна разделять горизонты. Ближайший: работает ли прототип при заданных условиях? Средний: переносится ли он на новые примеры и сколько стоит его использование? Долгий: меняет ли программа наше понимание задачи и какие новые методы из неё вырастают? Нельзя требовать, чтобы каждый фундаментальный проект дал продукт за год. Но нельзя и использовать далёкую перспективу как защиту от любой проверки.

У финансирующих организаций есть собственная проблема. Их решения видимы и могут быть политически спорными; неполученные результаты иногда проще объяснить «неудачным направлением», чем неопределённостью исследовательского процесса. Но отказ от проекта также не нейтрален: он определяет, какие идеи будут развиваться, кто сможет остаться в лаборатории и какое будущее станет доступным для проверки. Бюджетная экспертиза — часть истории науки, а не внешний шум вокруг чистых формул.

Эта проблема вернётся и в современные технологии. Компании и ведомства будут обещать полезные системы, показывать демонстрации, просить финансирование и объяснять ограничения. Цикл не обязательно плох: он может дисциплинировать ожидания и защищать общество от бесконечного финансирования пустых обещаний. Но оценка должна быть конкретной. Нельзя судить о переводе, роботе, теории познания и нейронной сети одной шкалой «готово / не готово».

Компьютеру не хватало не только скорости

Когда слышишь о ранних неудачах, легко решить, что все проблемы сводились к слабым компьютерам. Действительно, вычислительная мощность была ограниченной, память стоила дорого, а программы обрабатывали данные медленнее, чем хотелось бы. Но даже вообразив более быстрый процессор, мы не получаем автоматически программу, которая знает, на что обратить внимание.

Чтобы робот взял чашку, ему нужно отличить её от соседнего предмета, определить положение ручки, оценить, можно ли схватить её, и понять, не закрыта ли она другим объектом. Если рядом стоит другая посуда, система должна знать, что можно сдвинуть, а что может упасть. Каждый из этих шагов зависит от предыдущего, но также от ожиданий о физическом мире. Быстрый поиск не спасает, если программе неизвестно, какие варианты стоит рассматривать и какие условия важны.

Это была проблема представлений. Как описать ситуацию так, чтобы компьютеру было удобно рассуждать о ней? Какие факты нужно хранить? Как обновить описание, если объект сместился? Как не перечислять каждую возможную конфигурацию отдельно? Философы и инженеры могли говорить о «здоровом смысле», но для программы это означало множество конкретных взаимосвязанных ожиданий: предметы занимают пространство, закрывают друг друга, падают, ломаются, поддерживают вес, могут быть похожи, но выполнять разные функции.

Системы раннего ИИ успешно работали там, где описание мира было достаточно полным и устойчивым. Но реальные задачи были открытыми: пользователь мог назвать незнакомый объект, изменить цель, совершить неожиданное действие. У программы не всегда был способ определить, какое правило уместно. Если в систему добавить больше знаний, возникает новый вопрос: как проверить, что они согласуются и не порождают бесконечный перебор?

Поэтому «недостаточно мощности» — правильный, но неполный ответ. Нужны были новые способы представить знания, выбирать действия, учиться на опыте и сохранять устойчивость к незадаанным условиям. Каждый из этих шагов требовал собственных идей. Техника могла сделать систему быстрее; она не могла сама придумать для неё понятие чашки, хрупкости или разумного плана.

Под одной вывеской скрывались разные дисциплины

Слово «искусственный интеллект» объединяло программы, которые решали совсем разные задачи. Logic Theorist строил доказательства. Перцептрон менял веса по примерам. ELIZA выбирала языковые шаблоны. Машинный перевод пытался преобразовать текст между языками. Робототехника связывала зрение с физическим движением. Каждая система требовала своего вида данных, своей аппаратуры и своего критерия успеха.

Это разнообразие было источником силы: исследователи могли обмениваться идеями и переносить методы между задачами. Но оно усложняло оценку. Если программа переводит предложения, её можно сравнить с переводчиком по точности, скорости и стоимости. Если система доказывает теорему, можно проверить цепочку вывода. Если робот в лаборатории берёт блок, надо решить, считать ли успехом само действие, его надёжность или способность переносить навык на другую комнату. Если ELIZA поддерживает диалог, нужно отделить техническую связность от человеческого ощущения разговора.

Единой шкалы для всех этих результатов не было. Тот факт, что один тип программы провалил практическое ожидание, не обязательно говорит о другом типе исследований. Но название «ИИ» могло сделать так, что неудача одной системы воспринималась как отчёт о всей дисциплине. Именно поэтому ALPAC и Лайтхилл нельзя склеивать в единый приговор. Один оценивал прежде всего машинный перевод и связанные языковые процессы в американском контексте; другой пытался классифицировать широкую британскую область, включавшую автоматизацию, модели центральной нервной системы и робототехнику.

Критика также приходила от разных сторон. Заказчик спрашивал, решена ли его практическая задача. Инженер проверял, можно ли запустить систему вне лабораторного стенда. Учёный спорил, даёт ли эксперимент общую теорию. Философ мог сомневаться в самом отождествлении вычисления и мышления. Все говорили об ИИ, но измеряли разные вещи.

Это не значит, что дисциплина была искусственно создана одним названием и потому не имела общего ядра. Общим был интерес к системам, которые выполняют действия, обычно связываемые с человеческими способностями. Но способы исследовать эти действия расходились. В одних проектах главным было формализовать знания; в других — научить машину классифицировать; в третьих — связать восприятие и моторное управление. Слово «ИИ» соединяло их как зонтик, который не отменял разницы в погоде под ним.

Любая оценка описывает свою эпоху

Отчёт всегда измеряет систему теми средствами и по тем задачам, которые доступны в момент проверки. ALPAC не оценивал статистический перевод конца XX века или нейронные модели XXI века; Лайтхилл не мог проверить вычислительные ресурсы и подходы, появившиеся позднее. Их выводы нужно читать как ответы на тогдашние заявки, а не как всевременные предсказания.

Это не повод снисходительно улыбаться экспертам прошлого. Именно они должны были принимать решения при реально имевшихся данных. Но и мы не должны превращать их ограничения в закон природы. Полезная историческая оценка сохраняет два факта одновременно: системы тогда не справлялись с обещанным масштабом, а будущие поколения могли изменить саму задачу, методы и инфраструктуру.

Чего не произошло

Не произошло такого, что в начале 1970-х каждый исследователь остановил работу и признал искусственный интеллект невозможным. Исследовательские вопросы продолжались, иногда меняя имя и контекст. Отчёты не превратили математику в ложь, не запретили роботам двигаться и не закрыли компьютеры. Они изменили степень доверия к широким обещаниям и усложнили получение ресурсов для некоторых программ.

Не произошло и обратного: будто критики были неправы во всём, потому что спустя десятилетия появились мощные системы. Комбинаторная сложность, хрупкость восприятия и нехватка здравого смысла были реальными. Ограниченность тогдашних методов была реальной. Если сегодня машина переводит лучше, это произошло благодаря новым идеям, данным, вычислениям и инженерным системам, которых у команды ALPAC не было. Позднейшая победа не отменяет раннего диагноза о конкретном состоянии дел.

Не следует думать, что «зима» была наказанием за слишком смелую мысль. Наука движется не только теми, кто верит, что задача решаемая. Ей нужны критики, которые показывают пределы; организаторы, которые спрашивают о расходах; инженеры, которые исправляют слабости; и финансирующие структуры, способные поддерживать исследования, результат которых нельзя заказать точно по сроку. Ошибка начинается, когда один вид участника объявляют единственным владельцем истины.

Охлаждение изменило язык. Вместо «общего интеллекта» проще было заявить конкретную функцию; вместо системы, которая знает всё, — программу, которая помогает в одной предметной области. Иногда слово «ИИ» исчезало из вывески, но методы оставались. Распознавание, планирование и обработка языка могли продолжаться под именами компьютерной науки, статистики, инженерии или прикладного анализа.

Одновременно менялся характер публичного обещания. Хорошая демонстрация перестала автоматически означать, что коммерческий или государственный продукт уже на горизонте. Вопрос о масштабировании стал не второстепенной оговоркой, а центральной частью оценки. Что система умеет в чистой комнате? Что происходит, когда меняется фон, словарь, пользователь или цена ошибки? Сколько ручного труда остаётся после автоматизации? Эти вопросы звучали скучнее обещаний мыслящей машины, но были ближе к реальности.

В этом смысле первая зима была болезненной формой взросления области. Она не доказала, что проект искусственного интеллекта ошибочен. Она показала, что единая вывеска прикрывает множество разных задач, у каждой из которых собственная трудность, стоимость и мера успеха. Там, где раньше хватало обещания «создать интеллект», теперь нужно было показать, какой именно метод работает, при каких условиях и почему стоит продолжать.

Почему после зимы выбрали узкую задачу

После разочарования в универсальных обещаниях сильнее выглядела идея выбрать конкретную область и перестать просить от машины всё сразу. Если эксперт умеет диагностировать неисправность, можно попытаться записать его правила. Если геолог умеет отбирать вероятные месторождения по данным, можно формализовать часть его опыта. Если инженер знает, какие конфигурации системы допустимы, машина может применить эти знания быстро и последовательно.

Такой подход менял язык ценности. Вместо общего обещания человеческого разума появлялся узкий вопрос: сможет ли программа помочь специалисту в одной хорошо описанной задаче? Вместо обучения на примерах или большого перебора — явные факты и правила, которые передал человек. Вместо демонстрации «машина думает» — результат, который можно сравнить с работой конкретного эксперта.

Экспертная система не отменяла все старые проблемы. Человек часто не может полностью объяснить, как он распознаёт ситуацию; правила могут противоречить друг другу; база знаний устаревает; система может работать только в узкой области. Но обещание стало проверяемым. Можно было показать, на каких случаях программа помогает, где ошибается, как её обновлять и кто отвечает за правила.

Это не было поражением ИИ. Это была стратегия, которая соответствовала научной и экономической погоде. После зимы легче было финансировать систему, умеющую хорошо делать одно, чем программу, обещающую научиться делать всё. Рынок хотел полезности; исследователи получили более чёткие задачи; пользователи могли оценивать функцию, а не будущий разум.

Следующая глава начинается именно с этого поворота. Компьютеру дадут не сеть связей, которая меняется после примеров, и не открытый поиск универсального интеллекта. Ему дадут знания экспертов — инструкции, условия и правила, записанные в явном виде. Обещание станет уже. И впервые за долгое время именно узость покажется преимуществом.

Глава 7. Если эксперт — это правила

В конце первой зимы искусственный интеллект выглядел как обещание, которое слишком долго просили подождать. Машины умели решать отдельные задачи, но не приближались к тому универсальному разуму, о котором говорили на конференциях и в заявках на финансирование. Перевод не стал автоматическим, зрение не стало надёжным, а разговор с компьютером по-прежнему требовал либо узкого сценария, либо терпеливого собеседника.

У исследователей оставался менее эффектный, но куда более практичный вопрос: а что, если машине не нужно понимать всё?

Можно выбрать область, где люди уже умеют работать, найти задачу, которую можно очертить, и перенести в программу часть профессионального знания. Не научить компьютер быть врачом, а помочь ему разобрать несколько типов инфекций. Не сделать химика из микросхем, а сократить список молекулярных структур, которые специалисту предстоит проверить. Не поручить машине проектирование целого компьютера, а попросить проверить, совместимы ли детали в заказе.

Это была смена масштаба. Обещание «создать разум» уступало место обещанию «помочь специалисту в определённой работе». После нескольких лет громких ожиданий такой поворот казался почти скромностью. На деле он оказался началом одной из первых попыток сделать искусственный интеллект полезным за пределами университетской лаборатории.

Новая идея держалась на простом различии. Программа общего назначения должна сама разбираться, что важно в незнакомой ситуации. Узкая экспертная система получает предметные знания заранее: правила, ограничения, связи между признаками и выводами. Разработчики не просили её открыть всю логику химии или медицины. Они пытались дать ей достаточно сведений, чтобы пройти конкретный участок работы так, как его прошёл бы подготовленный специалист.

Слово «эксперт» здесь звучало убедительно и немного опасно. Оно могло создать впечатление, будто в компьютере поселился специалист. В действительности в программу попадали не человек и даже не целостное описание его опыта, а ограниченная модель выбранной задачи. Она могла быть сильной в своём коридоре и совершенно беспомощной у ближайшей двери.

Химическая задача, которую нельзя было решить перебором

История экспертных систем начинается не с больницы и не с бизнеса, а с вопроса о молекулах. В Стэнфорде середины 1960-х исследователи работали на пересечении информатики, химии и генетики. Важную роль в проекте сыграл Джошуа Ледерберг — генетик, лауреат Нобелевской премии, который интересовался тем, как компьютеры могли бы участвовать в научном открытии. Рядом с ним работали химик Карл Джерасси, исследователь искусственного интеллекта Эдвард Файгенбаум, программист Брюс Бьюкенен и другие участники университетской команды.

Их проект получил имя DENDRAL. Название было придумано для программы, которая должна была строить возможные структуры органических молекул по измерениям масс-спектрометра. Это звучит так, будто прибор выдавал загадку, а компьютер должен был назвать ответ. На практике путь был длиннее: прибор предоставлял данные о массе молекулы и о фрагментах, на которые она распадалась при измерении. Эти фрагменты были уликами, но не полным описанием строения.

Представьте, что вам дали несколько осколков разбитого предмета и попросили восстановить исходную вещь. Вдобавок часть осколков потеряна, некоторые похожи друг на друга, а у вас есть только ограниченное число способов проверить догадку. Масс-спектрометрия была гораздо точнее этой бытовой картины, но аналогия показывает главное: один набор измерений мог соответствовать нескольким возможным структурам. Химику нужно было использовать теорию и опыт, чтобы понять, какие гипотезы правдоподобны.

Компьютер мог быстро генерировать варианты. Но если запускать все комбинации связей подряд, число кандидатов быстро становилось слишком большим. Это знакомый комбинаторный взрыв: каждое новое решение — какой атом с каким соединить, какое кольцо построить, где поместить функциональную группу — умножает число вариантов. Скорость компьютера помогает, но не спасает от поиска по пространству, в котором слишком много лишнего.

Химик решал задачу иначе. Он знал, какие структуры несовместимы с известными свойствами вещества, какие фрагменты обычно образуются при распаде, что говорят химические связи и какие ограничения накладывает уже установленный состав. Он мог отсеять целые семейства кандидатов до того, как тратил время на подробную проверку. Это знание было не роскошным комментарием к вычислению, а условием, которое позволяло вычислению не захлебнуться в собственных возможностях.

DENDRAL строилась вокруг такого сотрудничества. Алгоритм порождал варианты, а химические знания и эвристики — практические правила, помогающие искать перспективные решения, — сужали множество гипотез. В ранних версиях важной частью работы были ограничения на возможное строение и способы разбиения молекулы. Позднее программы научились использовать более систематические сведения о том, как определённые химические структуры дают характерные фрагменты в спектре.

Здесь важно не вообразить современную универсальную нейросеть. DENDRAL не изучала любую науку из любых данных и не «понимала» химическую природу вещества в человеческом смысле. Её создатели выбирали способ представить задачу, снабжали программу химическими знаниями и проверяли, насколько разумно система сокращает путь от спектра к кандидатам. Сила возникала из тесного соответствия между моделью и конкретной областью.

Знание как способ не искать всё подряд

В раннем искусственном интеллекте часто надеялись, что достаточно найти общий механизм решения задач: перебор, поиск, логический вывод. DENDRAL продемонстрировала, почему одного общего механизма мало. Если программа не знает особенностей химии, она не понимает, какие ходы имеют смысл. Она может перебрать множество формально возможных структур, но формальная возможность ещё не делает их хорошими научными гипотезами.

В этом отношении экспертная система отличалась от обычного калькулятора. Калькулятор одинаково выполняет одну операцию независимо от того, зачем она нужна. Экспертная система должна была располагать некоторой картиной предметной области: знанием о допустимых объектах, типичных сочетаниях и последствиях. Это знание меняло сам поиск. Программа не просто делала вычисления быстрее, а пыталась выбирать, какие вычисления стоит делать.

Команда DENDRAL не записывала каждое действие химика в форме готового рецепта. Она пыталась разделить задачу на представление данных, генерацию гипотез, применение ограничений и оценку соответствия экспериментальным свидетельствам. В этом процессе участвовали химики, которые могли сказать, почему одна структура заслуживает внимания, а другая не выдерживает проверки. Их роль была не в том, чтобы выдать машине волшебную формулу, а в том, чтобы сделать предметные различия видимыми разработчикам.

Именно здесь появляется важная черта, которая потом определит судьбу экспертных систем: программу можно улучшить, добавив не абстрактно «больше интеллекта», а конкретные сведения о мире задачи. Это звучало обнадеживающе. Если система ошибается, возможно, ей недостаёт правила. Если она не видит важного исключения, добавим исключение. Если не умеет отличать два спектра, запишем нужное различие.

В лаборатории это обещание работало лучше, чем в лозунгах. DENDRAL стала исследовательской системой, способной помогать в анализе органических соединений. Её значение было не только в том, что она выдавала список вариантов. Проект показал, что большая доля эффективности узкой программы может происходить из специального знания, а не из универсально хитрого алгоритма. Эту мысль Файгенбаум позднее сформулировал как центральный тезис экспертных систем: глубокое знание ограниченной области нередко полезнее поверхностного знания обо всём.

Фраза звучит очевидно, пока не сравниваешь её с прежней мечтой об общем решателе. Универсальный механизм обещал сэкономить ручной труд: однажды придумать хорошую процедуру и применить её к разным задачам. Экспертная система предлагала обратный обмен. Механизм можно сделать сравнительно общим, но для каждой новой области нужно кропотливо собирать специальные знания. Универсальность переезжала из самой программы в надежду на повторное использование её каркаса.

DENDRAL также напоминала, что научное открытие не сводится к поиску ответа в каталоге. Система генерировала гипотезы, которые могли направить дальнейшее исследование, но эксперимент и химическая оценка сохраняли свою роль. Компьютер мог помочь решить, какой кандидат проверять следующим; он не превращал предположение в истину одним выводом на экране. Этот образ — машина как партнёр по сужению поиска — оказался долговечнее истории о машине, которая сама делает открытие.

Для людей за проектом это была не только интеллектуальная задача. Междисциплинарная команда должна была переводить слова между языками химии и программирования. Химику естественно говорить о структуре, спектральном сигнале и механизме распада. Программисту нужны объекты, переменные, ограничения и процедуры. Если одно и то же слово

для специалиста включает десятилетия опыта, а для машины становится условием в строке кода, перевод требует постоянной проверки.

Так DENDRAL стала одним из первых примеров того, что позднее назовут инженерией знаний: систематической работой по извлечению, формализации и представлению экспертных сведений в программе. Она не началась с готовой дисциплиной и чётким учебником. Практика возникала прямо внутри проекта. Команда обнаруживала, что трудность часто заключается не в том, как написать программу, а в том, что именно должно быть в ней написано.

Meta-DENDRAL: можно ли получить правило из примеров?

Следующий шаг проекта был особенно интересен, потому что ставил вопрос, который позднее станет центральным для машинного обучения. Если экспертные правила так полезны, обязательно ли человек должен формулировать каждое из них?

Программа Meta-DENDRAL исследовала возможность находить закономерности в данных масс-спектрометрии и предлагать правила, объясняющие поведение фрагментов молекул. Название связывало систему с DENDRAL, но задача у неё была другая. Одна программа применяла знания, чтобы анализировать новые случаи; другая пыталась вывести некоторые знания из набора наблюдений.

Представим, что в группе измерили спектры многих молекул и заметили, что определённые структурные элементы часто сопровождаются определёнными фрагментами. Исследователь может превратить эту повторяемость в правило: если в структуре присутствует такой-то мотив, при определённых условиях вероятен такой-то распад. Meta-DENDRAL пыталась автоматизировать часть пути от наблюдений к подобным обобщениям.

Но между регулярностью и знанием лежит проверка. Данные могут быть ограниченными, измерения — неоднозначными, а найденная закономерность — случайной или слишком грубой. Если правило плохо, его применение будет систематически давать плохие результаты. Поэтому человеческий эксперт оставался в контуре: оценивал предложенные закономерности, определял область их применения и решал, можно ли включить их в рабочую базу.

Здесь видна историческая непрерывность, которую легко потерять, рассказывая историю как смену мод. Экспертные системы не были исключительно ручными инструкциями, а обучение на данных не возникло внезапно после их исчезновения. Уже внутри символического ИИ люди пробовали объединять явное знание с извлечением закономерностей. Различие было в том, какая часть работы задавалась человеком, какие данные система могла видеть и что считалось приемлемым объяснением результата.

У Meta-DENDRAL оставался предметный каркас: система работала с представлением химических структур и массовых спектров, в заранее определённой задаче и с определённым классом правил. Это не было современным обучением на огромном разнородном корпусе. Но проект показывал, что правила могут быть не только записаны заранее, но и предложены в результате анализа примеров. А значит, вопрос о будущем был уже не «правила или данные», а «какую роль играет эксперт на разных этапах получения, проверки и применения знания».

Эта линия не сразу победила инженерную практику. Система, которая извлекала правила, всё равно требовала человеческого решения о том, что считать релевантным примером, как описывать данные и какие гипотезы принимать. Однако именно это сочетание позднее стало одной из центральных тем ИИ: какой труд можно передать машине, а какой переносится в подготовку данных, выбор задачи и проверку результата.

От молекул к инфекциям

В 1970-х Стэнфордская группа обратилась к задаче, где цена неправильного вывода ощущалась совсем иначе. MYCIN разрабатывали как консультационную систему для помощи в диагностике некоторых серьёзных инфекций и выборе противомикробной терапии. В центре внимания сначала были инфекции крови, включая бактериемию, а также менингит.

Это была не попытка создать цифрового врача. Система работала в форме структурированного диалога с врачом: запрашивала сведения о пациенте, предполагаемом месте инфекции, симптомах и результатах лабораторных исследований; затем использовала правила, чтобы сформировать гипотезы о возбудителе и предложить терапию. Диалог мог выглядеть как разговор на английском языке, но MYCIN не понимала свободную человеческую речь. Врач отвечал на заранее сформулированные вопросы, выбирая допустимые варианты или вводя структурированные данные.

Разница между этими двумя описаниями важна. «Компьютер поговорил с врачом и поставил диагноз» звучит как полномочия, которых у программы не было. «Программа задавала вопросы в заданной форме, обрабатывала ограниченные типы ответов и выдавала рекомендации в узкой медицинской области» точнее показывает устройство системы. MYCIN была впечатляющей не потому, что превратилась в человека, а потому, что демонстрировала, насколько далеко может зайти тщательно построенная база знаний в выбранной задаче.

Система была связана с медициной через конкретных людей и конкретные ограничения. Эдвард Шортлифф, врач и компьютерный исследователь, стал одной из ключевых фигур проекта. Брюс Бьюкенен и другие специалисты по ИИ занимались представлением знания и логикой вывода. Медицинские эксперты помогали формулировать сведения об инфекциях и лечении. Создание системы требовало постоянного сотрудничества: вопросы о микробиологии и антибиотиках переходили в вопросы о том, какие факты программа должна запросить, как представить неопределённость и в каком порядке применять правила.

Для врача рекомендации должны были иметь практический смысл. Подбор терапии зависел не от одного симптома, а от сочетания сведений: где вероятнее инфекция, какие микроорганизмы возможны, что показали исследования, какие антибиотики подходят и какие ограничения следует учитывать. Ошибка могла привести к плохому лечению, но и задержка тоже имела цену. Поэтому задача не сводилась к поиску «единственно правильного ответа» — программа должна была помочь рассуждать, когда доступные данные ещё неполны.

Система должна была задавать вопросы в зависимости от того, что уже известно. Если определённый результат мог изменить список вероятных возбудителей, стоило его запросить. Если нужной информации не было, MYCIN не должна была молча считать её отрицательной. Это один из самых важных практических вопросов при работе с неполными сведениями: отсутствие ответа не всегда является ответом «нет». Медицинские данные часто просто ещё не получены.

В этом смысле MYCIN была не пассивной таблицей соответствий. Она строила консультацию: выбирала, какие сведения нужны, применяла правила к известным фактам и формировала промежуточные выводы. Но эта активность всё равно происходила внутри спроектированной области. Система не могла самостоятельно решить, что история болезни неполна из-за неверно заданного вопроса, что пациент описывает ситуацию необычным образом или что врачу следует выйти за пределы её сценария, если для этого не было предусмотрено отдельного механизма.

Как работает правило, если показать его без жаргона

Чтобы понять устройство MYCIN и подобных систем, полезно на минуту забыть о компьютерах. Представьте папку с карточками. На одной карточке написано: если известны такие признаки и такие результаты, усиливается гипотеза о конкретном классе бактерий. На другой — если гипотеза достаточно поддержана и известны дополнительные обстоятельства, рассмотрим определённую группу препаратов. На третьей — при определённом результате лабораторного анализа измени оценку предыдущего предположения.

Каждая карточка — не полная медицинская инструкция. Это небольшой фрагмент знания, который система может применить к текущему случаю. В классической записи такое правило часто описывают конструкцией «ЕСЛИ условие, ТО действие или вывод». Например, в условном учебном примере: если у пациента обнаружен определённый тип инфекции и лабораторный анализ показывает такой-то признак, увеличить поддержку гипотезы о соответствующем возбудителе. Реальные клинические правила намного сложнее, и упрощённую формулу нельзя использовать как медицинский совет. Здесь важен принцип: правило связывает наблюдаемые факты с промежуточным заключением или следующим шагом.

Программа хранит факты о текущем случае отдельно от правил. Фактами могут быть ответы врача, результаты тестов и уже полученные промежуточные выводы. Правила содержат общие связи, которые разработчики и эксперты внесли в базу знаний. Между ними работает механизм вывода — часть программы, которая проверяет, какие правила подходят к уже известным сведениям, и определяет, что делать дальше.

Такое разделение напоминает библиотеку и читателя. Библиотека содержит справочные сведения, а читатель выбирает, какую страницу открыть для ответа на вопрос. В экспертной системе «читателем» становится алгоритм вывода. Сама библиотека правил не знает, как провести консультацию; механизм вывода не знает медицины без содержимого базы. Если поменять медицинскую базу на правила для настройки компьютеров, часть движка можно сохранить, но предметная компетентность будет совершенно другой.

В MYCIN широко применялся обратный вывод: система начинала с вопроса, который хотела разрешить, и искала правила, способные его поддержать. Допустим, нужно оценить, какой микроорганизм может объяснить инфекцию. Система находит правила, в выводе которых фигурирует такой микроорганизм, и проверяет их условия. Если условия неизвестны, она может запросить дополнительный факт. Вместо того чтобы применить каждое правило к каждому факту, она движется от целевого вопроса назад к нужным свидетельствам.

Это похоже на работу следователя, который не опрашивает всех жителей города без разбора, а сначала формулирует версию и выясняет, какие сведения могут её подтвердить или опровергнуть. Сравнение полезно, но ограничено: экспертная система не обладает человеческим суждением следователя. Она исследует только те версии и свидетельства, которые представлены в её правилах и запросах.

Набор правил обычно называют базой знаний, а механизм, который их применяет, — механизмом вывода. Термины важны, потому что отделение одного от другого позволило повторно использовать часть программного устройства. Но такое разделение не было абсолютным: правила могли зависеть от того, какие типы данных умеет хранить движок и как он задаёт вопросы. Архитектура и содержание влияли друг на друга.

Пример карточек также объясняет, почему «добавить ещё одно правило» не всегда просто. Новая карточка должна согласовываться с остальными. Что если два правила дают разные рекомендации? Как определить, какое из них применимо? Может ли один факт опровергать вывод, который прежде поддерживался другим? Нужны ли исключения? Нужно ли уточнить

формулировку вопроса? Эти проблемы быстро превращали красивую метафору из стопки карточек в инженерную систему с тысячами взаимосвязанных условий.

Неопределённость без притворства точности

Одной из особенностей MYCIN стали коэффициенты уверенности, или *certainty factors*. Разработчикам требовался способ выразить, что свидетельство поддерживает гипотезу в определённой степени, но не доказывает её окончательно. Медицинская консультация часто начинается до того, как собраны все анализы, и разные признаки могут говорить в пользу разных объяснений.

Коэффициент позволял системе учитывать степень поддержки вывода и комбинировать свидетельства по заданным правилам. Это делало её поведение гибче, чем простая логика «истина или ложь». Если один факт говорил за гипотезу, а другой ослаблял её, программе требовалось каким-то образом представить это напряжение.

Но слово «уверенность» легко принять за точную вероятность. Это не одно и то же. В MYCIN коэффициенты не были просто результатом статистической калибровки на большом массиве случаев и не означали, что число, например, 0,7 гарантирует правильность диагноза в семи случаях из десяти. Они представляли инженерную схему оценки свидетельств, встроенную в программу и её правила. Число помогало организовать рассуждение, но не снимало вопроса о том, как оно было получено и насколько уместно его толковать.

Это различие особенно полезно сейчас, когда любой десятичный показатель может выглядеть объективнее словесного «скорее всего». Число на экране не возникает само по себе. Кто-то выбрал шкалу, способ комбинирования и смысл, который приписывается результату. В MYCIN создатели старались сделать работу с неопределённостью явной, но модель неопределённости оставалась частью их конструкции, а не независимым фактом о природе болезни.

Медицинские специалисты могли видеть, что конкретная рекомендация опирается на не вполне определённые данные. Коэффициенты помогали упорядочить неопределённость, а не уничтожить её. Это было честнее, чем выдавать каждый вывод за логически неизбежный. И одновременно открывало новый вопрос: как оценивать качество числовых мер, если они созданы как практический инструмент, а не как статистические вероятности?

Вопрос не был уникальным для медицины. Любая экспертная система, работающая с неполными сведениями, должна решать, что делать с сомнением. Можно отложить вывод до получения всех данных — иногда слишком долго. Можно вывести ответ без оговорок — слишком смело. Можно позволить правилам работать с градациями поддержки — тогда нужно объяснить, откуда эти градации взялись и что они означают для пользователя.

MYCIN не закрыла этот спор. Она помогла сделать его инженерно видимым. Неопределённость перестала быть туманом вокруг программы и превратилась в часть её устройства, которую можно анализировать и обсуждать.

«Почему вы спрашиваете?» — и «почему вы так решили?»

Одной из сильных сторон MYCIN было умение показывать ход работы. Врач мог спросить, почему система задаёт определённый вопрос или на чём основана рекомендация. Программа могла представить цепочку правил и фактов, которые привели к промежуточному выводу. В её терминологии объяснения типа WHY отвечали, зачем нужен вопрос; объяснения типа HOW — как система получила заключение.

Это имело практическую ценность. В медицинской консультации вопрос от программы не должен был выглядеть произвольным. Если врачу непонятно, зачем система интересуется конкретным результатом, он мог запросить объяснение и решить, стоит ли продолжать. Если рекомендация казалась странной, можно было проверить, какие сведения её поддержали.

Для многих ранних исследователей эта возможность была принципиальной. Систему не требовалось принимать как загадочный оракул: её правила можно было прочитать, обсудить и исправить. Объяснение связывало выход с формальной структурой системы. Если известно, какие условия сработали, появляется шанс найти ошибочное условие или недостающее правило.

Однако прозрачность не равна истинности. Представьте, что в инструкции опечатка: человек, следующий инструкции, может очень подробно объяснить, почему поступил неверно. Так же и программа способна убедительно восстановить собственный путь от плохого правила к плохой рекомендации. Видимость причин помогает проверке, но сама по себе не исправляет исходную модель.

Более того, объяснение машины может быть понятным только в рамках её собственной картины задачи. Если в базе знаний отсутствует важный факт, система не сможет объяснить влияние того, о чём она ничего не знает. Если правило слишком широко, цепочка применения может выглядеть стройной, хотя условие на самом деле не годится для конкретной ситуации. Читабельный след вывода — средство контроля, но не гарантия полного понимания.

Здесь заключено одно из главных различий между экспертной системой и поздними моделями, чьи знания распределены по миллионам параметров. В экспертной системе человек мог буквально увидеть правило. Это облегчало локальный аудит и изменение. Но база правил становилась громоздкой, и цепочка объяснения не обязательно показывала все предпосылки, заложенные в выборе представления и границ системы.

У этого подхода была ещё одна сторона: объяснения влияли на доверие. Врач не обязан был верить программе только потому, что она выдала медицинский термин. Он мог спросить, почему этот вопрос нужен и как получена рекомендация. Такое право задавать вопросы переводило систему из роли тайного советчика в роль инструмента, который должен был предъявить основания. Это не устраняло проблему авторитета машины; оно хотя бы делало основания доступными для обсуждения.

Объяснимость MYCIN часто приводят как преимущество символических систем. Это справедливо, пока мы помним, что речь об объяснимости конструкции: программа могла показать, какие явные правила сработали. Она не обязательно раскрывала все причины, по которым создатели выбрали именно эти правила, почему не внесли другие или насколько полно модель отражала реальную медицинскую практику.

Проверка системы — не только сравнение ответов

Оценивать медицинскую программу оказалось сложнее, чем предъявить серию задач по математике и посчитать правильные ответы. Врачебные решения могут различаться между специалистами. Разные эксперты не всегда одинаково оценивают приемлемую терапию, особенно когда исходные сведения неполны. Как отделить ошибку программы от разногласия между людьми? И кого считать эталоном, если сами эксперты расходятся?

Исследователи MYCIN проводили оценки с участием специалистов по инфекционным заболеваниям. В одной известной работе рекомендации системы сопоставляли с оценками экспертов по клиническим случаям. В этих условиях терапевтические рекомендации MYCIN соответствовали стандартам приемлемой практики, принятым экспертами Стэнфорда, примерно в 90,9 процента случаев. Эта цифра звучит внушительно, но её нельзя превращать в универсальную формулу «MYCIN была права в 90,9 процента всех случаев». Она относится к определённой процедуре оценки, выбранным случаям и критерию приемлемости.

В другой, более узкой слепой оценке по десяти тестовым случаям менингита восемь независимых экспертов сравнивали варианты противомикробного лечения, предложенные MYCIN и врачами. Система получила приемлемую оценку в 65 процентах случаев; оценки схем, предложенных пятью врачами-специалистами, находились в диапазоне от 42,5 до 62,5 процента. Этот результат также требовал осторожного чтения: выборка была небольшой, оценивалась конкретная задача, а приемлемость лечения — не то же самое, что исход пациента в реальной клинической практике.

Но даже с этими оговорками сравнения имели значение. Система показывала, что формализованная база правил может удерживать в памяти большой набор специализированных сведений и выдавать рекомендации, которые медицинские эксперты сочтут приемлемыми в тщательно очерченных случаях. Она конкурировала не со всеми врачами во всех областях, а с экспертами, отвечавшими на определённые вопросы по представленным данным.

Именно так можно одновременно признать достижение и сохранить масштаб. Достижение: программа демонстрировала уровень, который заставлял профессионалов всерьёз оценивать её выводы. Ограничение: лабораторная оценка не доказывала, что система подходит для любой больницы, любого пациента или всей работы врача. Самым важным результатом мог оказаться не процент, а появление строгого вопроса о том, как вообще сравнивать машину с клиническим суждением.

В реальной практике нужны были бы новые доказательства. Систему пришлось бы проверять на других группах пациентов, учитывать качество и доступность лабораторных данных, наблюдать за её взаимодействием с врачами и оценивать последствия использования рекомендаций. Тестовый набор случаев не мог ответить на все вопросы о безопасности, удобстве, ответственности и влиянии на лечение.

То, что MYCIN не стала повседневной клинической системой, не следует объяснять одной эффектной причиной. Результат на стенде и работающая больничная служба — разные вещи. Нужно было интегрировать данные из медицинских систем, учесть реальные рабочие процессы, обеспечить обновление сведений о диагностике и препаратах, определить ответственность за ошибки, обучить пользователей и проверить, что система не создаёт новых рисков. Даже рекомендации, которые врачи считают хорошими, не обязательно можно безболезненно встроить в клинику.

Отсутствие широкого внедрения не доказывает, что MYCIN была бесполезна, как и успешная оценка не доказывает, что её следовало немедленно использовать на пациентах. Проект дал исследователям технический и методологический опыт, а клинические тесты показали

трудность измерения машинного суждения. Это были разные результаты, которые не нужно сводить к простой истории о провале или триумфе.

Оболочка без медицинского содержимого

Разработчики заметили, что не всё в MYCIN относилось к медицине. Вопросы и ответы, применение правил, механизм работы с неопределённостью, объяснения и общая организация консультации могли быть полезны и в других областях. Если отделить этот общий механизм от конкретной базы медицинских знаний, появится оболочка экспертной системы — программный каркас, который можно наполнить правилами для новой задачи.

Так появилась EMYCIN, расширенная версия и обобщение подходов, использованных в MYCIN. Её можно было использовать как основу для новых консультационных систем. Идея похожа на кухонную форму: сама форма позволяет выпекать разные блюда, но не содержит ингредиентов и не знает, что именно вы хотите приготовить. Оболочка обеспечивала структуру программы; новая область требовала своей базы знаний, своих вопросов и своих правил.

Это было важно для коммерческой надежды на экспертные системы. Если каждую программу нужно строить с нуля, рынок останется набором дорогих единичных проектов. Если часть механизма можно переиспользовать, стоимость следующей системы может снизиться. Вендор мог предложить инструменты, а организация — внести собственные знания.

Но оболочка решала только часть проблемы. Наличие двигателя не означало, что кто-то уже знает, как сформулировать знания для него. Чтобы построить систему для банка, лаборатории или инженерной службы, нужно было определить подходящую задачу, собрать опыт специалистов, проверить правила, предусмотреть исключения и связать программу с данными организации. Технически общая основа ускоряла работу, но содержание оставалось дорогим и специфичным.

Здесь зарождалась новая профессиональная роль — инженер знаний. Он или она не обязательно были лучшими специалистами в области задачи и не всегда были программистами движка. Их работа состояла в том, чтобы связывать предметных экспертов с формальной моделью: задавать вопросы, переводить ответы в правила, проверять поведение системы на примерах и находить пробелы.

Само название подсказывало, что возник новый вид инженерии. Но оно могло скрыть более неловкую реальность: инженер знаний часто должен был извлечь из человека то, чего сам человек не умел изложить в виде готового списка. Работа начиналась там, где эксперт отвечал: «Я просто вижу, что этот вариант неверен». Такой ответ полезен в профессии, но бесполезен как правило, пока не удастся понять, на какие признаки опирается это «вижу».

Что знает эксперт, когда не может объяснить

В учебном примере профессионал будто бы располагает стопкой готовых инструкций: «если увидел А, сделай Б». В реальной работе знания не всегда хранятся в голове в таком виде. Опытный специалист может быстро замечать сочетание деталей и реагировать раньше, чем успевает проговорить рассуждение. Он узнаёт знакомый случай, чувствует, что конфигурация «не складывается», или задаёт уточняющий вопрос, потому что определённое несоответствие насторожило его.

Это не обязательно мистическая интуиция. Часто за ней стоят годы практики, внимание к признакам, которые новичок не замечает, и множество маленьких усвоенных различий. Но эксперт может не помнить, когда именно научился так действовать, и не иметь слов для каждого шага. Задача интервьюера — не только записать ответ, но и помочь сделать скрытые различия наблюдаемыми.

Инженер знаний мог попросить эксперта разобрать реальный пример: что он увидел первым, какая информация изменила оценку, на каком основании был отброшен вариант? Затем нужно было проверить реконструкцию на других случаях. Если специалист говорит, что всегда поступает одним образом, но на примерах меняет решение при определённом условии, значит, первоначальное правило было слишком грубым. Если два эксперта действуют по-разному, требуется понять, это разные школы, разные обстоятельства или разные уровни допустимого риска.

Поэтому получение знаний редко выглядело как один разговор, после которого инженер возвращался к компьютеру с полным описанием профессии. Это был цикл: интервью, формализация, тестирование, разбор противоречий, новое интервью, уточнение. Система помогала вскрыть места, где экспертное описание было неполным. Вопрос «что вы делаете?» не всегда давал полезный ответ; иногда важнее было спросить, при каких обстоятельствах специалист изменил бы решение.

Новый специалист по ИИ нередко оказывался посредником между двумя мирами. Эксперт говорил на языке медицинских или инженерных случаев; программист — на языке структур данных и вычислений. Инженер знаний должен был услышать, какие различия имеют значение, а затем помочь представить их так, чтобы программа могла применить. Это требовало уважения к предметной области, но также и способности спорить: правило, которое звучит разумно, всё равно нужно проверить на случаях, включая неудобные.

Эксперт при этом не был непогрешимым источником истины. У разных профессионалов могли быть разные предпочтения; привычка одного специалиста не обязательно являлась общепринятым правилом. На практике решение могло зависеть от доступного оборудования, местной политики и готовности рисковать. Если инженер знаний превращал всё услышанное в универсальное условие, программа закрепляла индивидуальную практику в форме объективного алгоритма.

Так возникало тонкое политическое последствие: экспертная система заставляла решать, чьё знание и чьи нормы попадут в программу. Кто считается экспертом? Кто выбирает примеры? Чьи исключения войдут в базу, а чьи останутся за рамками? Систему могли представить как нейтральную автоматизацию, хотя её правила отражали выборы конкретных людей и организаций.

Разработчики не обязательно считали, что человеческое знание можно полностью перенести в программу. Часто достаточно было закодировать ту часть, которая повторялась, подавалась проверке и приносила пользу. Но фраза «мы автоматизировали опыт специалиста» звучала шире реального результата. На деле автоматизировался выбранный фрагмент работы, переведённый в выбранную формальную схему.

Когда исключение рождает новое исключение

Даже хорошее правило быстро сталкивалось с жизнью. Допустим, эксперт говорит: «Если условия А и Б выполняются, обычно выбирают действие В». Слово «обычно» сразу задаёт вопросы. Всегда ли выполняется это условие? Что если присутствует признак Г? Что делать при конфликте с другим правилом? Как система узнает, что данных недостаточно? А если изменилась технология, препарат или корпоративная политика?

Первое исключение можно добавить отдельным правилом. Второе — тоже. Но новые правила взаимодействуют с прежними, и количество возможных сочетаний растёт. Правило, которое исправляет один случай, может случайно менять вывод для другого. База знаний перестаёт быть аккуратным списком и становится системой зависимостей, где небольшое изменение способно повлиять на несколько частей поведения.

В ручной инструкции человек часто использует фоновые знания, чтобы понять, что правило здесь не подходит. Он замечает, что ситуация необычна, что термин употреблён в другом смысле или что неизвестное условие нельзя автоматически трактовать как отсутствие проблемы. Программе нужно явно задать, как распознавать такие ситуации. Если этого не сделать, система может применить общее правило слишком уверенно именно там, где эксперт предпочёл бы остановиться и попросить помощи.

Это и есть одна из форм хрупкости экспертной системы. В хорошо описанной области программа может быть очень сильной; за её границей она не обязательно понимает, что вышла за пределы допустимого. Достаточно, чтобы входные данные были похожи на знакомые и какое-то правило формально совпало, — и система продолжит вывод. У неё может не быть общего чувства странности, которым человек сигнализирует себе: «здесь что-то не так».

Хрупкость не означала, что системы постоянно выдавали бессмыслицу. Она означала, что граница между знакомым случаем и незнакомым приходилась на разработчиков: они должны были предусмотреть, как программа обнаружит отсутствие знания. Чем больше исключений добавлялось, тем труднее становилось поддерживать ясную картину этой границы.

Можно было запрограммировать отказ: если нет достаточной информации, система просит дополнительные данные или передаёт случай специалисту. Но отказ тоже требовал критерия. Сколько свидетельств достаточно? Как сообщить пользователю, что программа не знает ответа? Как не превратить систему, которая отказывается во всех сложных случаях, в бесполезную? Автоматическое «не уверен» — это не просто фраза, а отдельная часть инженерного замысла.

В отличие от человека, который может использовать широкий опыт, экспертная система обычно не знала, что поставленная задача оказывается частью более крупного процесса. MYCIN могла рассуждать о бактериях и лечении в пределах своей модели, но это не означало, что она знала о больнице, семье пациента, страховании, истории предыдущих ошибок или практической доступности препарата. Некоторые из этих сведений могли быть важны в конкретном решении, но не входили в её «мир».

Именно здесь узость оказывалась одновременно преимуществом и риском. Ограниченная область позволяла добиться высокого качества. Та же граница оставляла вне модели всё, что разработчики не сочли нужным описать. Хорошо работающее правило не гарантировало, что программа понимает, когда его не следует использовать.

XCON: когда правила попали в производственный процесс

Если MYCIN показала, насколько серьёзной может быть экспертная система в лабораторной консультации, то проект R1/XCON показал, что правила могут войти в работу большой компании. Digital Equipment Corporation, или DEC, продавала компьютеры семейства VAX. Заказчик мог выбирать множество компонентов, которые нужно было соединить в работающую конфигурацию. Одних заказов не доставало необходимых частей; другие включали несовместимые сочетания.

Это была не головоломка, которую покупатель решал ради интереса. Ошибка в заказе создавала производственные задержки и дополнительные расходы. Специалисты, называвшиеся техническими редакторами, проверяли заказы и помогали определить, как собрать систему. Их работа стала узким местом: число вариантов росло, детали и модели менялись, а проверка каждой конфигурации отнимала время.

В конце 1970-х Джон Макдермотт из Университета Карнеги — Меллона начал разрабатывать систему R1 для помощи в конфигурации компьютерных заказов. Имя XCON закрепилось за системой в DEC и расшифровывалось как Expert CONfigurer. Задача была хорошо ограничена: на входе — описание заказанной машины и набор доступных деталей; на выходе — проверенная конфигурация с размещением компонентов по стойкам. Требования были конкретными, а неправильный результат можно было обнаружить до сборки.

Смысл задачи понятен и без знания компьютерной архитектуры. Представьте заказ кухни, где нужно подобрать шкафы, столешницу, бытовую технику, соединения и точки питания. Нельзя просто сложить всё в список: детали должны помещаться, соединяться и работать вместе. Но у серверов и мини-компьютеров количество вариантов и технических ограничений было гораздо выше. Сотрудники DEC знали эти ограничения благодаря опыту; XCON пыталась сделать часть этой проверки повторяемой.

R1/XCON использовала правила продукции: если обнаружены определённые элементы заказа и известны их ограничения, выполнить действие — добавить нужный компонент, проверить совместимость или разместить устройство в подходящем месте. Механизм вывода многократно сопоставлял условия правил с текущим описанием заказа. Когда система находила совпадение, она могла добавить новые факты или изменить конфигурацию, после чего правила проверялись снова.

Некоторые правила напоминали инструкцию сборщику: если в заказе есть такое устройство, убедись, что предусмотрены нужные модули или соединения; если компонент занимает определённое место, не размещай рядом несовместимую деталь; если выбранная комбинация требует дополнительных элементов, включи их в конфигурацию. Конкретные правила были привязаны к продуктам DEC и их особенностям, а не к абстрактному понятию «компьютер».

Такой способ рассуждения называют продукционной системой: набор условий приводит к действиям или новым заключениям. В отличие от процедуры, где каждый шаг заранее выстроен программистом в одном длинном алгоритме, правила могут быть организованы как множество отдельных фрагментов. Это облегчает добавление знаний и позволяет описывать область в терминах её специалистов. Но независимость фрагментов условна: система должна решать, какие правила проверять и что делать при конкурирующих совпадениях.

Именно практика выдавала, насколько трудно было создать полезную систему. Универсальную программу конфигурации можно было придумать на бумаге, но для реальных заказов требовалось знать номенклатуру DEC, несовместимости, типы корпусов, ограничения производственного процесса и внутренние соглашения. Макдермотту и университетской команде приходилось тесно взаимодействовать с сотрудниками компании. Они не могли придумать всю базу правил, сидя в кампусе.

Первоначальную версию XCON доставили в DEC в 1980 году. В описании проекта начала 1980-х речь шла примерно о 750 правилах в ранней системе. Но переданной версии было недостаточно, чтобы просто подключить её и считать задачу закрытой. Правила требовали расширения и уточнения, а сама компания должна была научиться их поддерживать. В дальнейшем, по опубликованным описаниям опыта DEC, система выросла примерно до тысячи правил, а затем до нескольких тысяч. Цифры менялись вместе с версиями и практикой, поэтому полезны прежде всего как показатель масштаба, а не как одна неизменная мера.

Когда XCON стала частью производственного процесса, она уже не была университетским экспериментом. Заказ нужно было обработать вовремя, база знаний — поддерживать, а изменения продуктов — отражать в правилах. DEC сформировала группу, отвечавшую за систему. Часть сотрудников компании не имела опыта разработки экспертных систем; им предстояло освоить специализированную среду и понять, как устроено поведение программы. Передача технологии требовала не только программного кода, но и людей, которые могли бы работать с его знаниями.

Наиболее убедительным доказательством успеха был не лозунг о том, что компьютер превзошёл человека, а изменение конкретного процесса. Система помогала проверять и конфигурировать заказы, которые прежде требовали значительного ручного труда. Компания могла рассматривать её с точки зрения пропускной способности, качества и стоимости обработки. Для менеджеров это был гораздо более осязаемый аргумент, чем общий разговор о машинном разуме.

Однако оценивать XCON тоже нужно осторожно. В популярном пересказе успех часто сводят к одной внушительной сумме сэкономленных долларов. Такие суммы зависят от периода, способа подсчёта и того, что сравнивают: затраты на ручную проверку, стоимость исправленных ошибок, рост объёма продаж или расходы на содержание самой системы. Лучше не превращать оценочные корпоративные цифры в независимую научную меру. Главное подтверждение ценности — система была встроена в реальный процесс и постоянно использовалась для конфигурации заказов.

Её коммерческий эффект помог убедить организации, что символический ИИ способен решать задачи, где есть вполне земная метрика. Не нужно было объяснять, понимает ли программа компьютер по-человечески. Можно было спросить, правильно ли она собирает конфигурации, сколько заказов проходит через процесс и сколько исправлений нужно после её работы. Такой вопрос оказался удобен бизнесу — и он изменил ожидания от искусственного интеллекта.

В компании обнаружилось второе производство — производство правил

XCON помогла обнаружить неожиданный факт: одной автоматизации недостаточно, если организация не умеет поддерживать знания, на которых она работает. В 1984 году инженер DEC Джудит Бачант описала, как компания училась сопровождать систему после передачи из университета. Ранний XCON требовал исправлений и расширения; сотрудникам приходилось осваивать язык продукционной системы и налаживать контакты с экспертами, чтобы поддерживать базу.

Машина не освобождала компанию от необходимости иметь специалистов. Она меняла состав работы. Часть технических редакторов получила инструмент для более быстрой обработки заказов, но сама база знаний требовала новых ролей: людей, которые понимали систему, знали предметную область, тестировали изменения и координировали их с разработкой продукции.

Когда DEC выпускала новые варианты оборудования, правила совместимости нужно было пересматривать. Старый компонент мог исчезнуть; новая модель — потребовать иных соединений; производственный процесс — поменяться. Иногда изменение касалось только одного участка. Но если оно влияло на базовые допущения, приходилось проверять, не сломало ли новое правило сотни других случаев.

Так система становилась похожа на инфраструктуру, а не на разовый проект. В начале программисты и менеджеры могли представить работу как «внести экспертные знания в программу». После внедрения организация обнаруживала непрерывный цикл: производственный процесс меняется, поэтому нужно обновить базу; база обновлена, поэтому нужно проверить её на существующих примерах; тесты прошли, поэтому изменение можно включить; затем нужно наблюдать за реальной работой и исправлять новые расхождения.

У цифровой системы появлялась редакционная служба. Кто-то решал, какое знание актуально; кто-то проверял, какие правила конфликтуют; кто-то выяснял, почему программа дала неожиданный результат. В этом смысле экспертная система превращала опыт организации в программный актив, но одновременно делала этот актив зависимым от постоянной человеческой работы.

Это было не случайной неудачей XCON, а общим свойством знания. Профессиональная область не стоит на месте. Правило «правильно» только относительно конкретной версии процесса, продукта и представлений о допустимом результате. Извлечь правило однажды и считать его вечным — всё равно что написать техническую инструкцию и никогда не обновлять её после изменения устройства.

В системах для бизнеса обслуживание часто оказывалось самым дорогим этапом. Нужно было содержать людей, которые умели менять базу; интегрировать систему с остальным программным обеспечением; проверять обновления; документировать версии и объяснять сотрудникам, какие решения программа вправе принимать. Даже если первая версия быстро принесла пользу, её дальнейшая цена зависела от того, насколько устойчиво организация могла поддерживать специализированную компетенцию.

Публикация Бачант и Макдермотта о «жизни в окопах» XCON рассказывала именно о передаче технологии и многолетнем сопровождении, а не только о моменте первого успеха. Эта сторона истории хуже помещалась в газетный заголовок. «Компьютер конфигурирует компьютеры» — хорошая новость. «Чтобы это продолжалось, организации нужно создать команду по сопровождению базы знаний и обучать её работе со специализированными инструментами» — менее эффектно, но гораздо полезнее для тех, кто собирался повторить успех.

Когда язык правил становится отдельным рынком

К началу 1980-х экспертные системы начали привлекать внимание за пределами университетов. Компании увидели возможность автоматизировать узкие задачи, которые прежде выполнялись специалистами и были описаны в процедурах. Появились специализированные программные оболочки, консультационные проекты и продукты для корпоративных клиентов. Вокруг них формировался рынок, обещавший сделать знания организации доступнее и сократить зависимость от редких экспертов.

Для аппаратного обеспечения это тоже открывало новые возможности. Экспертные системы часто создавали в языках и средах, удобных для символической обработки, включая Lisp. На рынке появились Lisp-машины — компьютеры и рабочие станции, оптимизированные для таких программ и языков. Они стоили дорого и предлагались как профессиональная инфраструктура для разработки ИИ. Здесь снова возникал круг взаимных обещаний: сложные программы требуют специальных машин, а наличие специальных машин должно сделать сложные программы практически полезными.

Но коммерциализация изменила характер разговора. В академической лаборатории можно было годами изучать вопрос, на который пока нет практического ответа. Клиент спрашивал, что именно система будет делать, сколько будет стоить, кто обновит её знания и что произойдёт при ошибке. Перспективы рынка росли быстрее, чем подготовка организаций к созданию и обслуживанию экспертных систем.

Некоторые компании пытались не просто приобрести продукт, а найти случай, который можно автоматизировать с помощью правил. Это была ошибка выбора. То, что специалист делает ежедневно, не обязательно хорошо формализуется. Задача может требовать наблюдения за физическим миром, свободного общения, обмена опытом между коллегами или принятия решений с учётом множества нефиксированных обстоятельств. Даже если эксперт способен выполнить работу, это не означает, что он может описать её набором правил.

Потому первые убедительные области имели ряд полезных свойств: задача была ограничена, исходные данные можно было представить в программе, критерии результата были понятны, а ошибок можно было избегать формальной проверкой. Масс-спектрометрия и конфигурация оборудования хорошо иллюстрировали эту структуру. Клиническая помощь требовала более аккуратных мер и контроля, но тоже могла быть построена как консультация с чёткими вопросами. Чем дальше задача отходила от этих условий, тем дороже становилось её формализовать.

Бизнесу нравилась возможность упаковать редкое знание и передать его в отдел, где раньше не хватало специалиста. Такая цель могла быть разумной, например при обучении новых сотрудников или проверке стандартных случаев. Но если организация надеялась полностью заменить эксперта одной базой правил, она рисковала автоматизировать только видимую часть работы, оставив сложные и необычные случаи без надёжной поддержки.

Тогда система становилась не столько цифровым сотрудником, сколько новым участником организационной структуры. Кто-то должен был решить, когда рекомендация обязательна, кто может её оспорить, кто видит причину и кто отвечает за обновление знания. Технический проект создавал управленческие вопросы, которые не решались одной удачной архитектурой.

Узкая система и большая тень за её пределами

Экспертная система могла быть сильной именно потому, что заранее ограничивала, о чём будет рассуждать. В медицинской базе MYCIN находились сведения о бактериях, симптомах, тестах, местах инфекции и терапии. Это помогало программе работать в конкретной задаче. Но в её модели почти не было множества вещей, которые человек считал бы частью медицинского мира: организации больницы, жизненных обстоятельств, последовательности лечения, взаимодействия пациента с системой здравоохранения.

Философ Джон Маккарти использовал подобные ограничения, чтобы показать, как специализированная компетентность отличается от здравого смысла. Программа может хорошо рассуждать о выбранном классе объектов, но не обладать самоочевидными для человека знаниями о пациентах, больницах, действиях и последствиях. MYCIN могла вести управляемую консультацию, не понимая свободный разговор и не представляя полную человеческую ситуацию. Её полезность зависела от того, что у пользователя есть собственное здравое суждение и понимание границ программы.

Это не опровержение MYCIN, а точное описание разделения труда. Врач предоставлял структурированные факты, понимал, в каких обстоятельствах применять систему, и мог оценить совет. Система выполняла специализированное рассуждение. Риск начинался, если забыть об этом распределении и начать воспринимать рекомендации как автономное решение.

В 1984 году Маккарти прямо писал, что экспертные системы могут впечатляюще работать в специализированной области и всё же быть хрупкими за её пределами. Они трудно расширялись на случаи, не предусмотренные разработчиками, и обычно не распознавали собственных ограничений. В этой критике заключён мост к следующей главе: за узкой компетентностью скрывалась проблема здравого смысла, то есть способности использовать фоновые знания, которыми люди обычно даже не считают нужным делиться.

Обычный человек знает, что если чашку отпустить в воздухе, она упадёт. Он предполагает, что дверь и комната продолжают существовать, даже когда на них не смотрят. Он понимает, что одно событие меняет часть мира, а не стирает всё остальное. Большую часть подобных ожиданий люди не формулируют. Экспертная система получает лишь те куски мира, которые её проектировщики успели выразить. Чтобы не выходить за границу, ей нужны дополнительные знания о том, что может произойти, что должно оставаться неизменным и когда ситуация перестала быть обычной.

Так правило «если эксперт не может описать опыт, нужны примеры» оказалось лишь частью ответа. Данные могли помочь в тех задачах, где есть много повторяемых случаев и можно определить, какой ответ правильный. Но за примерами тоже стоял человек, который выбирал их, размечал и объяснял, в каком мире они имеют смысл. А универсального набора примеров на все будущие обстоятельства не существовало.

Экспертные системы не провалились от того, что им не доставало одного волшебного правила. Они помогали там, где задачу можно было очертить, знания — добыть, входы — представить, а результат — проверить. За пределами этих условий их сильная сторона становилась уязвимостью. Нужна была либо ещё более тщательно собранная база знаний, либо другой способ получать и обновлять поведение, либо способность программы признавать неизвестность.

Проверка правил становится работой исследователя

Есть ещё одна часть этой истории, которую легко не заметить, если смотреть только на готовую рекомендацию: экспертную систему нужно проверять не только как программу, но и как формализованное знание. Если она выдала неожиданный результат, ошибка может скрываться в коде механизма вывода, в самом правиле, в том, как эксперт описал случай, или в том, как данные были представлены. Эти причины требуют разного ремонта.

Для этого разработчики собирали тестовые случаи, включая примеры, на которых система должна была дать известный результат. Затем они проверяли промежуточные выводы, задавали экспертам вопрос, почему правило сработало, и выясняли, не пропустила ли программа важное условие. Так тестирование становилось продолжением сбора знаний. Тестовый пример был не просто отметкой «верно» или «неверно», а поводом уточнить границу между двумя случаями.

Можно представить правило, которое правильно срабатывает на десяти примерах. Одиннадцатый случай показывает, что оно использует признак слишком широко. Разработчики добавляют условие; после этого двенадцатый пример перестаёт проходить, потому что условие перекрывает другое правило. Такая цепочка не означает, что подход бессмыслен. Она показывает, что формализовать компетентность — итеративная работа, где каждое улучшение должно быть проверено на соседних участках знания.

Особенно опасна «подгонка» базы только под известные примеры. Если правило становится всё точнее на небольшой подборке, но теряет устойчивость на новом случае, система запомнила особенности тестов, а не достаточно общее знание. Поэтому нужны были независимые проверки, разбор ошибок и обратная связь от людей, которые знают реальные условия работы. И даже после этого остаётся вопрос, насколько разнообразны проверенные случаи и где проходит граница применения системы.

В этом отношении экспертные системы предвосхитили современные споры о проверке ИИ. Разработчики уже тогда должны были спросить: на каких случаях измеряется качество, кто выбрал эталон, что значит приемлемая ошибка и каким образом обнаруживается выход за пределы задачи? Удобный показатель помогает сравнивать версии, но он не заменяет объяснения того, что именно система умеет. Без такой ясности цифра успеха может выглядеть надёжнее, чем сам продукт.

Что осталось после обещания заменить специалиста

Из сегодняшнего дня легко увидеть в тысячах вручную заданных правил архаичную предысторию современных моделей. Такое чтение теряет важные достижения. Экспертные системы создали практику инженерии знаний, показали ценность предметной компетенции, помогли выстроить отдельные методы вывода и заставили исследователей серьёзно обсуждать объяснение машинного результата. DENDRAL применяла знания к научным гипотезам; MYCIN исследовала консультационное медицинское рассуждение; XCON стала частью повседневной работы компании.

Они также изменили вопрос, который задавали разработчики. В ранних разговорах об ИИ часто звучало: «Как построить программу, которая сможет думать?» Экспертные системы предлагали спросить: «В каком конкретном деле человек принимает решение, какие сведения ему для этого нужны и как можно проверить, что компьютер помогает?» Вопрос стал более приземлённым и более продуктивным.

Узкая постановка задачи позволила получить результаты, которых было трудно добиться от универсальных обещаний. Но ограниченность не была случайным недочётом, который можно исправить несколькими дополнительными правилами. Она была частью модели. Чтобы расширить область, нужно было знать, какие новые факты значимы, как меняются правила и что делать с ситуацией, для которой в базе нет подходящего решения.

Проекты показали и цену человеческого знания. Оно не лежало в голове эксперта в готовом виде, ожидая загрузки в компьютер. Его приходилось извлекать через интервью и примеры, согласовывать между специалистами, переводить на формальный язык и сопровождать вместе с изменением самой области. «Автоматизация» не убирала людей из процесса. Она перераспределяла их усилия: одни помогали системе работать, другие поддерживали правила, третьи проверяли результат.

Успехи создали рынок, а рынок создал новые ожидания. Клиенты хотели, чтобы экспертная система была не только демонстрацией на нескольких случаях, но и устойчивым продуктом. Им требовалось, чтобы она работала с реальными данными, переживала изменения, объясняла ошибки и вписывалась в организацию. Обещание вышло из лаборатории, где допустимо остановиться после хорошей демонстрации, и встретилось с компаниями, которым нужно было платить за дальнейшее обслуживание.

И в этом столкновении было много ценного. XCON доказывала, что узкая программа может стать частью производства. MYCIN показывала, как машина может разбирать сложное медицинское рассуждение и предъявлять путь вывода. DENDRAL соединяла вычисления с научным опытом. Ни один из этих проектов не требовал верить, что компьютер стал человеком. Достаточно было заметить, что он научился выполнять часть человеческой работы в условиях, для которых его подготовили.

Но за каждым работающим правилом оставался вопрос: что делать с тем, чего эксперт не успел объяснить? Если область меняется, кто перепишет инструкцию? Если случай выходит за пределы базы, узнает ли программа об этом? А если требуемое знание проще показать на тысячах примеров, чем извлечь в форме правил?

К концу 1980-х интерес к экспертным системам столкнётся с новой волной разочарования. Системы были полезны, но их разработка и обслуживание оказывались дороже и сложнее, чем обещали оптимистичные презентации. Сложность корпоративного рынка, ожидания покупателей, специализированное оборудование и затраты на поддержку сошлись в отдельном кризисе. Это будет не простое повторение первой зимы: замёрзнут уже не только научные гранты, но и бизнес-модель.

Однако следующий поворот начнётся не с полного отказа от правил. Он начнётся с нового смещения внимания: вместо того чтобы описывать каждый шаг специалиста, исследователи попробуют учить программу на примерах. Данные обещали снять часть нагрузки с инженера знаний, но не могли объяснить, что значит правильный пример и почему знакомая закономерность должна сработать в новом случае.

Прежде чем эта ставка станет центральной, остаётся рассмотреть задачу, которую правила не смогли охватить. Человек постоянно использует фоновое знание о предметах, действиях, времени, причинах и других людях. Оно кажется таким простым, что его почти никогда не формулируют. Машине же нужно было сообщить всё это явно — или надеяться, что она каким-то образом догадается сама.

Так специализация привела к вопросу о мире за пределами специализации. Следующая глава начинается не с того, чего машина знает, а с того, что человек знает, даже не замечая этого.

Глава 8. Здравый смысл в коробке

Поставить красный куб на синий — простая задача. Человек обычно справляется с ней, не открывая учебник по механике и не выясняя, одинаковы ли у собеседника и у него самого понятия «верх» и «низ». Мы видим предметы, понимаем, что они занимают место, и ожидаем, что один предмет может поддерживать другой. Если куб слишком тяжёлый или стол шатается, мы замечаем это как часть ситуации. Если рука занята, задача требует дополнительного шага. Большую часть этих знаний никто не произносит.

Именно молчаливые знания и стали одной из главных трудностей искусственного интеллекта. Компьютер можно научить распознавать слова, применять правила и планировать последовательность действий. Но когда он встречается фразу «поставь это туда», ему нужно понять, что означает «это», где находится «туда», можно ли взять объект, не разрушит ли действие что-то ещё и что должно остаться неизменным после перемещения.

Взрослый человек отвечает на такие вопросы почти автоматически. Для программы каждый из них означает представление мира, правила вывода и иногда дополнительный вопрос пользователю. Слово «очевидно» не экономит вычислений. Оно только прячет работу, которую человеческий опыт давно проделал за нас.

В 1960-х и 1970-х эта проблема стала особенно заметна в исследованиях языка, планирования и робототехники. Компьютеры уже могли впечатляюще работать в маленьких, тщательно заданных мирах. Но попытка выйти из такого мира быстро приводила к вопросу: сколько всего нужно знать, чтобы машина не просто отвечала на предложение, а понимала его в обстоятельствах, которые человек считает обычными?

До комнаты блоков: советчик, которому давали сведения

Идея научить программу здравому смыслу появилась задолго до SHRDLU. В статье «Programs with Common Sense», представленной Джоном Маккарти в 1958 году и опубликованной в сборнике 1959 года, он описал гипотетическую программу, которую назвал Advice Taker — «принимающий советы». Это не была готовая система, которую можно было запустить на компьютере. Маккарти предложил направление: программа должна принимать сведения о мире в формальном языке, выводить из них следствия и выбирать действия, соответствующие поставленной цели.

Разница с обычной программой была принципиальной. Во многих программах знания о задаче скрыты в последовательности команд, написанной программистом. Если поведение нужно изменить, приходится менять саму программу. Advice Taker должен был получать новые факты и правила как утверждения о мире. Пользователь мог сообщить, что дверь заперта, что за ней находится комната и что цель — добраться до кухни; дальше система должна была рассуждать, какие действия допустимы.

Такой замысел связывал общий интеллект с тем, что программа знает, а не только с тем, какие вычислительные процедуры выполняет. Система должна была делать «достаточно широкий класс непосредственных выводов» из того, что ей сообщили и что она уже знает. На человеческом языке это напоминает способность понять простое объяснение и применить его к новой задаче. На машинном языке это требует формального представления фактов, логики вывода, описания действий и правил выбора.

В этой ранней идее уже заключена будущая проблема здравого смысла. Если системе сообщают, что дверь заперта, ей нужно знать, что запертую дверь нельзя открыть обычным способом. Если её просят пройти в другую комнату, она должна понимать, что такое перемещение, каким образом действие меняет положение человека и что при этом остаётся прежним. Нельзя просто добавить факт «дверь заперта» и ожидать, что всё остальное возникнет автоматически.

Advice Taker также показывает, что история здравого смысла не началась как реакция на одну неудачу машинного перевода или нейросетей. Уже в символическом ИИ было ясно: вычислительной системе мало знать алгоритм. Ей нужно представление о том, в каком мире она действует, и механизм для рассуждения о последствиях. Маккарти сформулировал задачу, а последующие проекты исследовали отдельные способы её решения — от формальной логики до микромиров и больших баз знаний.

Между гипотетическим советчиком и SHRDLU есть важное различие. Advice Taker должен был рассуждать о широком круге фактов и действий, но оставался программным предложением. SHRDLU стала работающей системой, которая решала реальную исследовательскую задачу, только в очень ограниченной области. В этом смысле она не исполнила мечту о здравом смысле, а превратила часть мечты в проверяемый эксперимент: можно ли связать язык, знания и действия так, чтобы машина поддерживала последовательный диалог?

SHRDLU: разговор в комнате с цветными блоками

В конце 1960-х аспирант Массачусетского технологического института Терри Виноград начал работать над программой, которая могла бы взаимодействовать с виртуальным миром блоков через английский язык. Проект описан в его диссертации и техническом отчёте MIT 1971 года под названием *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. Программа получила имя SHRDLU.

Пользователь мог вводить команды и вопросы: попросить передвинуть предмет, уточнить положение фигур, спросить, что лежит на чём, или попросить объяснить, почему система выполнила действие. Программа разбирала грамматику предложения, связывала слова с объектами мира, строила план и меняла состояние виртуальной сцены. Затем она могла ответить на вопросы о получившемся расположении.

Для зрителя это было необычайно убедительно. Компьютер не просто повторял готовые фразы. Между текстом и ответом находился мир, в котором существовали красные и синие блоки, пирамиды, столы и действия вроде поднять, поставить или переместить. Если команда не могла быть выполнена сразу, программе требовалось определить препятствие и найти допустимую последовательность действий. Если вопрос ссылался на объект из предыдущей реплики, система использовала контекст диалога, чтобы понять, о чём речь.

В самой работе Виноград подчёркивал, что система использовала синтаксическую структуру, сведения о контексте и общие знания о своём мире. Эта комбинация была важнее одного удачного языкового трюка. SHRDLU должна была понять, какие слова относятся к каким объектам, что пользователь хочет сделать и как изменить описание мира, чтобы выполнить команду.

Возьмём простой пример. Если пользователь попросит переместить блок, который в данный момент накрыт другим предметом, система не может просто выдать команду «поднять». Сначала надо определить, что мешает. Затем освободить нужный объект, выполнить перемещение и учесть новое положение. Для этого нужны одновременно язык, планирование и модель физического мира. Ошибка в любом слое ломает всю цепочку: программа может неверно разобрать фразу, выбрать не тот объект или построить невозможный план.

Именно поэтому SHRDLU исторически интересна не только как демонстрация понимания языка. Она соединяла несколько задач, которые часто изучались отдельно. Языковой анализ связывался с тем, что система знала о предметах; знание влияло на план; действие меняло модель сцены; новая модель становилась контекстом для следующего вопроса. Пользователь получал целостный ответ, потому что несколько компонентов были согласованы вокруг одной ограниченной области.

Но граница этой области была очень чёткой. В мире программы находились только те предметы, свойства и действия, которые разработчики включили в описание. Слова «блок», «пирамида» и «на столе» имели конкретные значения. «Пирамида» не могла внезапно оказаться зданием, финансовой схемой или метафорой иерархии. Система не должна была разбираться в людях, погоде, расписаниях, собственности или политике. Любой смысл за пределами этого словаря оставался за пределами мира SHRDLU.

В этом нет научного обмана. Микромир — полезный способ изучать сложную проблему: исследователь контролирует объекты и условия, поэтому может выяснить, как программа разбирает фразы, планирует действия и отслеживает контекст. Лабораторный мир работает как модельный стенд для двигателя. Если двигатель на нём работает, это настоящее достижение. Но модельный стенд не доказывает, что автомобиль уже готов к любой дороге.

SHRDLU показывала не то, что компьютер почти научился понимать любой разговор. Она показывала, что сочетание языка с моделью мира может дать значительное поведение в

пределах хорошо описанной ситуации. Именно это ограничение позволяет оценить результат честно: система была сильной там, где её знания и словарь совпадали с задачей.

Что скрывалось за простой командой

Снаружи SHRDLU выглядела как собеседник, который просто понял просьбу. Внутри команда проходила через несколько слоёв обработки. Программа анализировала предложение, строила его синтаксическую структуру, определяла значение слов в своём словаре, сопоставляла их с объектами и отношениями в блоковом мире, а затем передавала полученную цель планировщику. Если пользователь спрашивал о последствиях действия, система обращалась к представлению сцены и истории диалога.

Для обычного читателя важен не список компонентов, а то, что каждый слой менял смысл следующего. Глагол «поставь» требовал понимать, какой предмет перемещается и где он должен оказаться. Предлог «на» обозначал отношение опоры, а не просто направление движения. Местоимение могло ссылаться на объект, упомянутый раньше. Команда «убери блок» означала действие, которое нужно выполнить в мире, а не текст, который достаточно пересказать другими словами.

Планировщик раскладывал задачу на действия и проверял условия, при которых они возможны. Чтобы поставить один блок на другой, нужно было освободить нужную поверхность и убедиться, что верхний предмет можно поднять. Если он накрыт третьим предметом, сначала придётся переставить его. Такой план требует причинной модели: действие меняет положение объекта, а изменившееся положение влияет на то, что можно сделать дальше.

Именно связь между процедурой и описанием мира отразилась в названии диссертации Винограда. В его системе грамматические знания не были только статической таблицей правил; процедуры помогали обрабатывать фразы и соотносить их с действиями. Мир тоже не был пассивной картинкой. Его состояние обновлялось, когда программа выполняла команду, а новые вопросы опирались на обновлённое состояние.

Подобная интеграция была сильной и одновременно хрупкой. Если команда содержала незнакомое слово, внеплановый тип объекта или действие, которого нет среди допустимых операций, программа не могла компенсировать пробел «интуицией». Человек может догадаться, что «убери вот эту штуку» значит переместить предмет подальше, даже если не знает его точного названия. SHRDLU нуждалась в том, чтобы нужный объект и действие были представлены в её модели.

Ограниченность позволила Винограду проверить, как язык может управлять планом и как план возвращается в язык в виде объяснения. Это куда интереснее, чем просто сказать, что в системе был маленький словарь. Область была небольшой, но внутри неё программа должна была соединить грамматику, объекты, причины, действия и разговорный контекст. Именно это сделало демонстрацию сильной и породило вопрос, можно ли расширить такой способ работы на более богатую среду.

Почему маленький мир так легко принять за большой

Причина привлекательности SHRDLU была в том, что программа отвечала на несколько запросов подряд и учитывала уже сказанное. Это выглядело естественно для человека: мы тоже поддерживаем разговор, помним, что обсуждали только что, и не начинаем каждую реплику с нуля. Когда система поняла «тот блок» как конкретный объект из сцены, она словно разделила с пользователем общий контекст.

Но общий контекст у них был разным. Человек приносит с собой огромный запас фоновых ожиданий; у SHRDLU были геометрия блоков, набор допустимых действий и сведения о предыдущих репликах. Система могла разрешить неоднозначность, потому что набор кандидатов был мал, а все релевантные отношения уже находились в модели. Если в сцене всего несколько блоков и один красный куб, понять «красный куб» сравнительно легко. В обычной комнате красных предметов может быть несколько; в разговоре речь вообще может идти не о вещи, а о сообщении, решении или человеке.

Представим ресторан с пятью блюдами. Клиенту достаточно сказать «то острое, которое я брал в прошлый раз», и официант может найти подходящий вариант. Теперь заменим меню всем содержимым супермаркета, библиотеки, вокзала и семейной переписки. Вопрос «то, о чём мы говорили вчера» получает бесконечно больше возможных отсылок. Ограниченное пространство смыслов позволяет системе делать то, что в открытом разговоре потребовало бы гораздо большего знания о людях и мире.

Из этого не следует, что ограниченная система лишь имитирует понимание и потому неинтересна. Она показывает, какой именно вклад в понимание вносит контекст. Грамматика помогает определить структуру фразы, но сама по себе не сообщает, к какому предмету относится местоимение, какой смысл слова важен и какое действие возможно. Чтобы связать предложение с миром, нужно знать хотя бы некоторые свойства мира.

Виноград построил не готовую теорию всего человеческого понимания. Зато SHRDLU наглядно показала, почему языковая система, оторванная от предметов и действий, быстро упирается в пределы. Смысл фразы возникает не только из слов и их порядка. Он зависит от ситуации, предыдущего разговора, ожиданий и предположений о том, что обычно бывает.

Это наблюдение стало особенно важным, когда исследователи пытались перейти от маленьких искусственных областей к повседневным задачам. Утверждение «машина разобрала английскую команду» звучит одинаково, но за ним могут стоять совершенно разные возможности: разбор фразы в словаре из двухсот терминов или работа с произвольными отсылками в неизвестных обстоятельствах. Продемонстрировать первое можно в лаборатории. Второе требует общего знания, которое трудно даже перечислить.

SHRDLU стала для ИИ одновременно достижением и предупреждением. Она дала исследователям конкретную работающую систему, где язык и действие были связаны. И вместе с тем заставила смотреть внимательнее на сцену: какие части мира уже были заданы, какие вопросы исключены, сколько свободы оставалось пользователю. Вопрос «что программа сделала?» пришлось дополнить вопросом «какие предположения сделали это возможным?»

Что человек знает, не замечая

Проблема здравого смысла не сводится к словарю бытовых фактов. Если бы нужно было только записать, что стул обычно имеет сиденье, а птица обычно умеет летать, задача оставалась бы большой, но конечной. Трудность в том, что знания должны не просто присутствовать в памяти машины. Система должна понимать, когда они применимы, как они связаны с другими сведениями и что делать, когда они конфликтуют с наблюдением.

Допустим, программа знает, что стеклянные чашки хрупкие. Если человек спрашивает, сколько такая чашка весит, эта информация может не иметь значения. Если робот собирается смахнуть её со стола, она становится критичной. Если чашка специально изготовлена из ударопрочного стекла, общее знание требует оговорки. Значит, системе необходимо не только хранить факт, но и оценивать его отношение к текущей цели и ситуации.

У человека такая работа часто скрыта. Мы обращаем внимание на то, что может повлиять на задачу, а остальное оставляем фоном. Мы не перечисляем все свойства комнаты перед тем, как взять чашку. Если задачу меняют — например, просят перенести горячий напиток через тесную комнату, — внезапно становятся важны температура, равновесие, расстояние, занятые руки и присутствие других людей.

Здравый смысл можно рассматривать как огромный набор связей между предметами, действиями, причинами, целями и типичными последствиями. Но это описание не решает вопроса: какую часть этого набора программа должна использовать именно сейчас? Важность факта не хранится на нём как ярлык. Она зависит от намерения пользователя и от того, что может случиться дальше.

Этот выбор называется проблемой релевантности. Машина, которая хранит слишком мало, пропускает важные предпосылки. Машина, которая пытается учесть всё сразу, тонет в ненужных деталях. Человеку не приходится каждый раз анализировать все законы физики, потому что он умеет выделять то, что может изменить результат. В обычной беседе этот отбор кажется естественным; для программы он становится частью задачи рассуждения.

Другой невидимый компонент — знание о постоянстве. Если робот переместил чашку со стола на поднос, изменилось её положение. Цвет стены, имя владельца и форма ручки обычно не изменились. Человек не сообщает программе эти отрицательные факты после каждого действия. Мы предполагаем, что мир сохраняет большую часть своих свойств, пока что-то не дало основания думать иначе.

Всем этим знаниям есть практическое применение, но они не образуют аккуратную энциклопедию. В некоторых ситуациях они конкурируют. «Обычно дверь открывается» — полезное ожидание, пока дверь не заперта. «Обычно человек уходит из комнаты» — возможно, он возвращается через минуту. «Обычно птица летает» — но пингвин не летает, а раненая чайка может не взлететь. Программа должна различать общее правило, наблюдаемый факт и исключение.

Фреймы: знакомая сцена с пустыми ячейками

В 1974 году Марвин Минский предложил одну влиятельную схему представления знаний — фреймы. Фрейм можно представить как структуру, связанную с типичной ситуацией, объектом или сценой. Когда человек слышит «ресторан», в памяти активируются ожидания: люди приходят, занимают места, выбирают еду, делают заказ, получают блюда и платят. Не каждый ресторан устроен одинаково, но типичный сценарий помогает ориентироваться, пока новые сведения не опровергли ожидания.

Вычислительное представление такой сцены могло включать ячейки — или слоты — для участников, объектов и обычных значений. Некоторые значения задаются по умолчанию: стол обычно стоит в помещении, заказ обычно относится к блюдам меню, счёт появляется после еды. Если ситуация отличается — человек пришёл только забрать заказ, ресторан самообслуживания или счёт уже оплачен, — система должна изменить соответствующие ожидания.

Сильная сторона фреймов была в организации знания. Вместо того чтобы хранить тысячи разрозненных фактов, можно представить связанные ожидания как единый каркас. Один фрейм может наследовать части другого: ресторан — разновидность места, где люди покупают услуги и еду; конкретный ресторан добавляет собственные особенности. Такая структура должна была помочь машине заполнять пропущенное и понимать типичные сцены без отдельного правила на каждую деталь.

Для читателя это похоже на формы, которые мы заполняем в уме. Если собеседник говорит «я снял номер в гостинице», мы не спрашиваем, что такое кровать и зачем она там. Ситуация вызывает ожидания: есть комната, в ней можно ночевать, за неё платят, у неё есть номер или ключ. Мы способны воспользоваться этими ожиданиями, но не обязаны считать их безусловной истиной. Можно снять номер без кровати, если гостиница необычная или речь идёт о шутке.

Фрейм давал программе структурированный способ ожидать типичные детали. Он не избавлял её от необходимости иметь знания о мире. Он предлагал связать такие знания в более крупные блоки. Это важное различие: организация базы может уменьшить хаос, но не создаёт содержимое бесплатно. Чтобы фрейм «ресторан» был полезен, кто-то должен решить, какие участники и действия типичны, какие детали можно считать значениями по умолчанию и как описывать исключения.

Минский представлял фреймы как часть частичной теории мышления, а не завершённую машину здравого смысла. Это оговорка важна. Позднейшие пересказы иногда превращают фрейм в волшебную папку с готовым сценарием, но предложенная идея была шире и осторожнее: знание может быть организовано структурами ожиданий, которые допускают корректировку по мере того, как наблюдение расходится с типичной картиной.

Здесь легко перепутать два разных употребления слова «фрейм». Фреймы Минского — способ организовать знания о типичных ситуациях. «Проблема рамки», обсуждавшаяся Джоном Маккарти и Патриком Хейзом в 1969 году, — вопрос о том, как описать, какие свойства мира остаются неизменными после действия. Идеи связаны общей темой представления ситуации и её изменений, но это не одно и то же. Словарная похожесть не должна скрывать два разных инженерных вопроса.

«Обычно» не значит «всегда»

Значения по умолчанию делают рассуждение компактнее, но сразу приносят проблему исключений. Обычно птицы летают. Пингвин — птица. Из этих двух утверждений человек не обязан делать вывод, что пингвин летает: у него есть дополнительное знание об исключении. Если же программа хранит только общее правило и принадлежность к классу, она может прийти к неверному заключению.

Подобные случаи называют немонотонным рассуждением. В классической логике, если вывод сделан из набора посылок, добавление новых фактов не отменяет его автоматически. В повседневной жизни новые сведения часто заставляют пересмотреть прежние ожидания. «Птицы обычно летают» даёт рабочую догадку, но факт «это пингвин» меняет вывод. Система должна уметь отступить от того, что прежде считала приемлемым.

То же относится к чашке, двери и человеку, вышедшему из комнаты. Обычная чашка стоит на столе, пока нет признака, что она приклеена или закреплена. Дверь обычно можно открыть, если она не заперта. Человек не перестаёт существовать, уходя из комнаты, но его местоположение изменяется. Мы свободно сочетаем общие ожидания с частными наблюдениями; компьютерная программа должна определить, какой факт имеет приоритет и когда пересчитать прежние заключение.

Исключения быстро образуют цепочки. Если птица ранена, она может не летать. Но некоторые птицы не летают и без травмы; другие умеют летать лишь на короткое расстояние; третьи могут летать, но не в данный момент. Добавление правил помогает конкретным примерам и одновременно создаёт новые вопросы. Что считать травмой? Как отличить «не может летать» от «не летает сейчас»? Как проверить, что новая оговорка не изменила слишком много других выводов?

У этой задачи есть человеческая сторона. Мы говорим «обычно» не потому, что в голове хранится полный каталог всех исключений. Мы можем опереться на одно общее ожидание, проверить его по ситуации и пересмотреть, если обнаружили несоответствие. Даже ребёнок в быту постоянно уточняет картину мира: увидел, что пингвин не летает, и не обязан после этого забыть, что воробьи летают. Отделить правило, исключение и область применимости — часть обыденного обучения.

Для формальной системы каждое такое различие требует представления. Значения по умолчанию экономят записи, но затем приходится описывать условия отмены. Число строк сокращается в одном месте и растёт в другом. Главное — не объём самой базы, а управление изменением вывода, когда появляются новые сведения.

Проблема рамки: что изменилось, а что осталось

В 1969 году Джон Маккарти и Патрик Хейз обсуждали трудности построения программ, которые рассуждают о мире и действиях. Одну из них назвали проблемой рамки. Представим формальную модель, где действие изменяет состояние мира. Если робот берёт чашку со стола, нужно описать, что чашка больше не стоит на столе и теперь находится в руке. Но также хотелось бы зафиксировать всё, что не изменилось: стол всё ещё на месте, стена сохранила цвет, ложка не исчезла.

В небольшой демонстрации список неизменностей короткий. В реальной сцене число фактов может быть огромным. После каждого действия явно повторять, какие свойства остались прежними, означало бы переписывать почти весь мир. Но если не записать нужные отношения, формальная система может не иметь основания считать, что они сохранились.

Привычное описание задачи помогает увидеть масштаб. Человек говорит роботу: «Перенеси чашку на поднос». Роботу нужно узнать чашку, определить, где поднос, проверить, свободна ли рука, и решить, можно ли взять чашку. После выполнения он должен обновить её положение. Но не нужно менять цвет чашки, размер подноса, адрес комнаты или состояние окна — если, конечно, робот не задел их по пути. Как описать эту разницу так, чтобы программа не требовала полного списка всех неизменных фактов, но и не теряла важные изменения?

В ситуации исчисления, обсуждаемой Маккарти и Хейзом, действия связывают одну ситуацию с другой, а некоторые свойства мира меняются во времени. Задача — выразить последствия действия и определить, что сохраняется. Это не бытовая загадка о том, что «само собой остаётся»; это формальная трудность представления динамического мира. Для робота она особенно заметна, потому что план без корректного обновления мира скоро становится бесполезным: система продолжает действовать так, будто чашка всё ещё на столе, хотя уже держит её.

Вычислительная проблема здесь не только в том, что нужно записать слишком много фактов. Системе приходится определить, какие изменения важны для текущей задачи. Если робот переносит чашку, положение стола может быть несущественно. Если он рассчитывает маршрут, положение стола — одно из ключевых препятствий. Одна и та же информация бывает фоном для одного вывода и решающим условием для другого.

Поэтому проблема рамки связана с проблемой релевантности, но не сводится к ней. Вопрос о релевантности — какие сведения учитывать в рассуждении. Вопрос о рамке — как описать изменения и неизменности после действия. На практике вопросы пересекаются: чтобы обновить модель, программа должна понимать, какой участок состояния затронуло действие и какие другие выводы зависят от него.

Название «проблема рамки» закрепилось в философии ИИ и стало шире исходного примера. Оно напоминает, что формализованное действие всегда оставляет невысказанные предположения: мир сохраняет часть свойств, наблюдение неполно, а последствия не исчерпываются списком непосредственных эффектов. В отличие от человека, программа не может опереться на общее чувство непрерывности мира, если оно не представлено каким-то способом.

От базы фактов к карте понятий

Чтобы рассуждать о мире, программе нужна не только подборка предложений. Ей требуется карта того, о каких типах вещей идёт речь и как они связаны. Такая карта называется онтологией: формальным описанием классов объектов, отношений и понятий, используемых системой. Например, программа может различать людей, места, предметы, действия и свойства; знать, что один человек владеет объектом, что предмет находится в комнате, а действие меняет положение вещи.

Для небольшой задачи онтологию можно задать вручную и удерживать под контролем. В блоковом мире объектов мало, и отношения специально отобраны. В открытом повседневном мире «предмет» может быть физической вещью, событием, идеей, организацией или словом в метафоре. Система должна понять, в каком смысле употреблено понятие, иначе формально правильное правило может примениться не к тому объекту.

Словарь сам по себе не равен онтологии. Словарь говорит, что слова могут означать; онтология описывает типы сущностей и допустимые связи между ними. А база знаний содержит конкретные утверждения, связанные с этими понятиями. Например, можно описать класс птиц, отношение «является экземпляром класса» и отдельное утверждение о пингвине. Чтобы получить вывод, системе нужно применить правила к этим представлениям.

Чем больше система знает, тем важнее, чтобы части знания были совместимы. Новое утверждение может изменить не одну запись, а множество выводов, построенных на ней. Если программа узнаёт, что конкретная дверь заперта, общее ожидание «дверь можно открыть» отменяется для этой двери в данной ситуации. Если речь идёт о двери в игре или воображаемой истории, контекст меняется ещё раз. База знаний становится не энциклопедией, а взаимосвязанной моделью с разными точками зрения.

Это объясняет, почему здравый смысл нельзя решить, просто сохранив побольше фактов. Нужно определить типы понятий, условия их применения, отношения между ними, значения по умолчанию, способы работы с исключениями и правила выбора контекста. Каждый слой поддерживает следующий, а ошибка в одном из них способна исказить выводы по всей системе.

Наивная физика: мир до учебника

Здравый смысл нужен не только в языке. Чтобы взять стакан с водой, человеку не приходится решать уравнения гидродинамики, но он пользуется ожиданиями о твёрдых предметах и жидкостях: стакан удерживает воду, если его не наклонить слишком сильно; жидкость перетекает в открытый сосуд; уровень зависит от количества воды и формы ёмкости. Эти знания не заменяют физику и не обязаны описывать движение каждой частицы. Они помогают ориентироваться в повседневной среде без точного расчёта на каждом шаге.

В конце 1970-х Патрик Хейз предложил программу «наивной физики»: формально описывать те представления о мире, которыми люди пользуются до изучения теоретической физики. «Наивная» здесь не означает глупую или просто ошибочную физику. Речь о практической картине повседневных явлений: предметы поддерживают друг друга, жидкости текут, контейнеры ограничивают движение, а изменение положения предмета имеет последствия. Позднее Хейз подробно разбирал, например, как представить жидкости и их поведение.

Этот пример показывает, почему понятия, привычные человеку, трудно превратить в базу знаний. В разговоре мы можем сказать «налей воду в чашку». Для программы надо уточнить, что считается жидкостью, как она соотносится с сосудом, почему поток прекращается, когда ёмкость заполнена, и что произойдёт, если сосуд перевернуть. Если задача требует только ответить на простую фразу, такой уровень деталей может быть избыточен. Если система управляет роботом, который переносит открытый стакан, это уже практические условия безопасности.

Можно попытаться дать системе подробную количественную модель, но для обычного действия это часто слишком дорого. Чтобы понять, что стакан нужно держать вертикально, роботу необязательно рассчитывать молекулярные силы. Ему достаточно качественного описания: жидкость удерживается стенками, но может вытечь через край; наклон меняет поверхность и направление движения. Качественные модели оперируют отношениями и типичными изменениями, а не точными числами для каждого параметра.

Но и качественное описание должно различать много ситуаций. Пустая чашка, полная чашка, закрытая бутылка, отверстие банка и дырявый сосуд связаны с жидкостью по-разному. Одно правило о том, что «сосуд держит жидкость», недостаточно. Нужны условия, исключения и понятия о целостности, вместимости, отверстиях и ориентации.

Наивная физика делает общую проблему здравого смысла более наглядной. Люди не только запоминают, что происходит, но строят рабочие причинные ожидания: если опрокинуть сосуд, жидкость может пролиться; если убрать опору, предмет упадёт; если сильно нагреть лёд, он растает. Эти ожидания позволяют предсказывать последствия новых действий. Для машины же «знать, что такое стакан» ещё не значит знать, что произойдёт при его перемещении.

Сус: проект, которому досталась повседневность

В 1984 году Дуглас Ленат начал проект Сус в Microelectronics and Computer Technology Corporation, известной как МСС, исследовательском консорциуме, созданном американскими технологическими компаниями. Это было время, когда отрасль внимательно следила за японским проектом компьютеров пятого поколения и за большими государственными вложениями в вычислительные технологии. МСС задумывалась как совместная исследовательская площадка; Сус стала одной из самых амбициозных попыток превратить общий здравый смысл в инженерную программу.

Ленат исходил из наблюдения, что узкие системы часто ломаются, когда оказываются за пределами исходной задачи. MYCIN может рассуждать об инфекциях, но не о ремонте автомобиля. Программа, хорошо играющая в шахматы, не знает, что шахматная доска — это предмет на столе, а не окно в комнату. Чтобы отдельные приложения могли использовать обычные знания о мире, нужен общий слой понятий и правил.

Замысел Сус был не в том, чтобы создать одну большую энциклопедию и научить программу читать статьи. Команде требовалась база, где знания записаны в форме, пригодной для машинных выводов. В обычном тексте фраза «если человек уронил стакан, тот может разбиться» соседствует с объяснениями, примерами и оговорками, смысл которых читатель достраивает. Для логической системы условия и связи должны быть выражены явно.

Эта работа требовала особого языка представления — СусL — и разработанных для проекта способов описывать понятия, отношения и правила. Люди, занимавшиеся Сус, вносили утверждения, проверяли формулировки и связывали новые сведения с тем, что уже было в базе. Внешне многие записи могли казаться банальными. Компьютер не знал, что твёрдый предмет может повредить мягкий, пока соответствующее понятие и отношение не были представлены и соединены с другими фактами.

В статье 1995 года Ленат описывал масштаб проекта как около ста человеко-лет труда с 1984 года, примерно сто тысяч общих понятий и порядка миллиона вручную введённых аксиом здравого смысла. Он также отмечал, что система могла выводить и сохранять дополнительные утверждения. Эти числа были опубликованы руководителем проекта и показывают масштаб, которого достигла команда по собственному отчёту; они не означают, что база исчерпала повседневное знание или решила проблему общего интеллекта.

Сто человеко-лет — не сто лет календарного ожидания. Это условная сумма труда команды: несколько человек, работающих продолжительное время, в пересчёте на эквивалент полной занятости. Оценка всё равно поразительная. Она показывает, что «внести здравый смысл» оказалось не короткой настройкой программы, а долгой работой большого коллектива. База не появлялась от одного удачного алгоритма; она строилась запись за записью.

Так в истории ИИ возникла роль, отличная от традиционной научной работы над алгоритмами. Нужно было нанимать людей, которые умеют анализировать повседневные понятия, составлять формальные утверждения, находить двусмысленность и проверять логические последствия. Повседневный мир превратился в проект с инфраструктурой, внутренним словарём и редакционными решениями. Чем «обычнее» было знание, тем легче человеку не замечать, сколько предположений в нём скрыто.

МСС: где повседневное знание стало долгим проектом

Сус создавалась не в одиночной лаборатории одного профессора. Проект начался внутри МСС — консорциума крупных американских технологических компаний, основанного в Остине в начале 1980-х. Компании объединили ресурсы для долгосрочных исследований, в том числе на фоне опасений, вызванных японской программой компьютеров пятого поколения. В таком институциональном контексте идея вложить годы в общую инфраструктуру знания выглядела не отвлечённым философским хобби, а исследовательской ставкой на то, чего корпоративные вычислительные системы пока не умели.

Дуглас Ленат руководил попыткой построить базу общего знания, которую можно было бы использовать в разных приложениях. Если каждой программе приходится заново создавать представления о людях, предметах, времени и причинности, работу приходится повторять. Общая база обещала стать слоем, на который могли бы опираться более узкие системы. Вместо того чтобы снабжать каждое приложение своей отдельной коллекцией бытовых предположений, можно было дать ему доступ к общему словарю понятий и правилам.

Это обещание зависело от того, что общий слой действительно можно построить и поддерживать. Команда должна была решить, что относится к базовому знанию, как называть понятия и как кодировать связи, пригодные для логического вывода. Важно было и то, кто станет пользователем этой инфраструктуры: специалист, который пишет запросы на языке СусL, отдельная программа, которая обращается к движку, или конечный человек, который задаёт вопрос обычными словами. «Общая база для всех» предъявляет дополнительные требования к структуре и стабильности, потому что ошибка в основании может распространиться на разные приложения.

В статье 1995 года Ленат описывал около ста человеко-лет работы над Сус с момента старта в 1984 году. Эту оценку легко представить как коллективную смену: люди разбирали понятия, записывали утверждения, уточняли значения и следили за тем, как система их использует. Это не означает, что сто человек одновременно переписывали словарь. «Человеко-лет» — сумма усилий команды за годы проекта. Но сам масштаб показывает, как быстро задача «записать здравый смысл» перестаёт быть небольшим дополнением к программе.

Долгая продолжительность работы была одновременно достоинством и предметом вопросов. Нужна ли базе десятилетия, чтобы стать полезной? Как измерить прогресс, если полноту здравого смысла нельзя проверить конечным набором тестов? Может ли приложение получить пользу от неполной базы, пока работа продолжается? Ленат и коллеги стремились показать, что часть знания уже можно использовать, но само заявление о широкой компетентности было гораздо труднее доказать, чем успех в одной конкретной задаче.

К середине 1990-х Сус стала примером альтернативного типа исследовательского вложения. Вместо краткого цикла разработки отдельного продукта — длительная работа над тем, что должно было помочь многим будущим системам. Такая инфраструктура редко производит один наглядный момент триумфа. Её ценность определяется тем, можно ли повторно использовать знания, согласуются ли они между приложениями и окупает ли общая база стоимость её построения.

Что делать с противоречивым миром

Даже большую базу нельзя считать единым набором безусловных истин. Мы без труда говорим, что Дракула — вампир, а затем отвечаем, что вампиров в действительности нет. Это не ошибка в обычном рассуждении, потому что мы понимаем, что одно утверждение относится к вымышленному миру, а другое — к реальному. Строго формальная база должна объяснить машине, почему эти фразы не опровергают друг друга.

В Сус использовались контексты, называемые микротеориями: ограниченные области рассуждения, в которых связан набор утверждений и допущений. Контекст может относиться, например, к математике, к конкретной истории, к определённой модели мира или к иной области. Это позволяет системе не смешивать факт из одного набора допущений с утверждением из другого.

Само наличие контекстов не превращает базу в лёгкую задачу. Теперь программе нужно определить, в какой микротеории она рассуждает, какие утверждения можно перенести в соседний контекст и как поступить, если пользователь перескакивает между ними. Контекст не отменяет сложность повседневного знания; он делает одну из её частей явным элементом архитектуры.

Представим, что человек спрашивает: «Может ли вампир войти в дом без приглашения?» В разговоре о романе ответ определяется правилами данного вымышленного мира. В разговоре о реальном доме и реальном мире вопрос звучит иначе. Человеческий слушатель переключает контекст по названию персонажа, предыдущей фразе или тону беседы. Машине нужен способ представить эти сигналы и выбрать подходящий набор предпосылок.

Такое разделение позволяет сохранить локальную согласованность, не требуя, чтобы все истории, модели и факты были частью одного непротиворечивого описания реальности. Одновременно оно ставит вопрос о границах: какая информация остаётся общей между контекстами, а какая действует лишь внутри одного? Ответ зависит от того, как спроектирована система и какое рассуждение требуется от неё.

В повседневной жизни люди постоянно отличают действительность от игры, условного примера, цитаты или шутки. Мы можем сказать «представь, что двери исчезли» и затем на несколько минут вести рассуждение об этом мире. Позднее мы без труда возвращаемся к обычным допущениям. Для компьютерной системы такой переход требует дисциплины представления и управления контекстом. Иначе фантастическое утверждение может попасть в базу как факт, а вывод, верный в задаче, — перенестись туда, где он уже неуместен.

Город, который нужно не только строить, но и содержать

Большая система знаний похожа на город. Новые дома связаны с улицами, районы — с транспортом, а изменение одной магистрали влияет на маршруты в других местах. В базе знаний новая формулировка может менять выводы, которые зависят от понятия, правила или значения по умолчанию. Добавить утверждение сравнительно просто; гораздо сложнее выяснить, какие выводы на него опираются и какие из них теперь надо пересмотреть.

В малой экспертной системе несколько разработчиков могут помнить, почему введено то или иное правило. В проекте масштаба Сус ручное знание растёт до объёмов, где понадобятся стандарты и проверка согласованности. Если разные люди описывают похожие понятия по-разному, возникают дубли и конфликтующие отношения. Если общий термин определён слишком узко, новые случаи не помещаются в него; если слишком широко, правило начинает срабатывать там, где не должно.

Команда должна решать, какие знания считать достаточно универсальными, чтобы они вошли в общий слой. Некоторые утверждения зависят от культуры и исторического периода, другие — от конкретной профессии или организации. Даже «обычная» норма часто не одна на всех. Устройство семей, правила вежливости, способы обращения к незнакомцу и предположения о собственности могут отличаться между обществами и поколениями.

Здесь техническая работа оказывается ещё и редакторской. Нужно выбрать термин, определить границы понятия, решить, является ли утверждение безусловным или типичным, указать контекст, а затем проверить выводы. То, что одна команда называет «здоровым смыслом», включает взгляд конкретных людей на то, как устроен общий мир. Если база станет основой для разных систем, её предположения будут влиять на то, что приложения считают нормальным.

Такой отбор не обязательно делает проект предвзятым по злему умыслу. Но он лишает базы невидимой нейтральности. Если программа полагается на типичное устройство семьи, поведения или института, кто-то выбрал это как типичное. Если система не знает альтернативных практик, она может принять их за ошибку пользователя. Проблема формализации здравого смысла одновременно техническая и социальная: нельзя корректно представить то, чьи различия разработчики вообще не заметили.

У этого проекта была и человеческая цена труда. Человек должен был снова и снова превращать привычное знание в точную формулировку, разбираться, где начинается исключение, и проследить связи с другими утверждениями. Это не механический ввод «очевидных фактов». Такое занятие требует философской аккуратности и технической дисциплины, но часто выглядит со стороны скучно именно потому, что в его центре — вещи, которые обычно никто не обсуждает.

Знание о людях тоже нужно формализовать

Физические ожидания — лишь один слой здравого смысла. Люди рассуждают и о намерениях, знаниях, обещаниях, ошибках и целях друг друга. Если собеседник говорит «закрой окно, пожалуйста», мы понимаем не только грамматическую структуру просьбы. Мы предполагаем, что существует окно, что его можно закрыть, что собеседник просит изменить его состояние и что просьба относится к текущей комнате.

Но при этом человек может задавать уточняющие вопросы: окно уже закрыто? Его опасно трогать? Собеседник просит о буквальном действии или использует выражение как намёк? Мы способны учесть социальную ситуацию и предположить цель: например, человек просит закрыть окно, потому что ему холодно или с улицы шумно. Само предложение не содержит всех этих предпосылок.

Такое знание сложнее записать, чем связь между чашкой и столом, потому что люди могут ошибаться, скрывать намерения, говорить непрямо или менять цель. Если система приписывает человеку знание о факте, она должна понимать, откуда тот мог его узнать. Если кто-то обещал вернуться, обещание относится к будущему и может быть нарушено. Если один человек передал другому записку, получатель узнает её содержание лишь после того, как прочитает текст.

Для машины нужно представить не только «что истинно», но и «кто это знает», «когда узнал», «во что верит» и «какой цели добивается». В противном случае программа будет приписывать всем участникам одинаковую информацию и строить планы на ложных предпосылках. В повседневном разговоре мы постоянно учитываем, что собеседник не видел то, что видели мы, и не знает того, что нам только что сообщили.

Эти примеры не означают, что компьютер должен полностью копировать человеческую психологию. Они показывают, что даже практическое действие зависит от фоновых отношений между людьми. Робот, который понимает просьбу буквально, может выполнить её технически и всё же не помочь, если неправильно определит цель. Система, ведущая диалог, должна различать высказывание, просьбу, цитату, вопрос и вымышленный пример.

В итоге здравый смысл охватывает не одну энциклопедию бытовой физики. Он включает причинность, время, разговорные отсылки, планы, роли и ожидания. Разные задачи требуют разных частей этой картины. Но именно потому что в обычной жизни они смешиваются, их трудно полностью разделить заранее.

Проблема не в том, что люди знают слишком много

Иногда задача Сус пересказывалась так: человек знает миллионы фактов, компьютер тоже должен получить миллионы фактов. Но количество — только верхний слой трудности. Необходимо определить, что является объектом знания, какие понятия относятся к нему, когда утверждение действует, может ли его отменить наблюдение, какой контекст нужен и какую цель преследует система.

Если программа знает, что железо твёрже плоти, этот факт может помочь понять, почему металлический предмет способен причинить вред. Но сам факт не говорит, есть ли у человека травма, кто виноват, надо ли вызывать помощь, действовала ли ситуация в реальном мире или внутри игры. Каждый следующий вывод опирается на дополнительные предпосылки о причинности, намерениях, правилах и доступных действиях.

Именно поэтому здравый смысл нельзя измерить простой толщиной базы. Миллион записей может оставлять огромные пробелы, если в них нет нужных связей. Небольшое число хорошо организованных отношений может быть полезнее, если программа правильно использует их для своей задачи. Знание должно быть достаточно широким, чтобы поддержать вывод, но достаточно точно ограниченным, чтобы не вызывать ложные ассоциации.

То же относится к автоматическому извлечению знаний. Если получить закономерность из текстов, придётся спросить, как система отличает реальный факт от описания вымысла, шутки или ложного утверждения. Если пользоваться готовой базой, нужно понимать, кто её собрал и для какой аудитории. Если попросить специалиста написать правила, следует учитывать, что часть его опыта может быть неявной. Каждая стратегия переносит труд в другое место, но не устраняет его.

Система, которая умеет вывести ответ из одного набора правил, не обязательно умеет определить, какие правила подходят. А система, которая выучила частую закономерность, не обязательно поймёт, когда данный случай исключительный. В обоих случаях ключевым становится знание об области применимости: что программа знает, чего она не знает и при каких условиях ей нужно остановиться.

Чей именно здравый смысл?

Разницу можно увидеть в обычном приветствии. Где-то уместно сразу перейти к делу, где-то короткий обмен любезностями считается важной частью разговора. Одни собеседники воспринимают прямой отказ как ясность, другие — как резкость. Если программа примет одну привычку за универсальное правило, она будет систематически неправильно читать людей из другой среды. Дело не в том, что у одних есть здравый смысл, а у других его нет. Их ожидания о том, что в данной ситуации считается нормальным, могут различаться.

Даже физическое знание и социальное ожидание различаются по устойчивости. Тяжёлый предмет обычно падает вниз при земной гравитации независимо от языка общения. Правила обращения к незнакомцу могут зависеть от возраста, статуса, места и времени. База, которая хранит оба утверждения без контекста, рискует слишком уверенно применять локальную норму ко всем людям.

Поэтому разработчики должны определить не только содержание правила, но и его источник и область действия. Можно пометить утверждение как типичное для конкретной культуры, организации или периода; можно хранить несколько вариантов; можно спросить пользователя. Каждый выбор увеличивает сложность системы, зато помогает не выдавать спорное ожидание за нейтральный факт. Это особенно важно, если программой пользуются люди, которых не было среди авторов базы и обучающих примеров.

Правила, примеры и предел полноты

Позднее машинное обучение предложило другой способ получать знания о типичных связях. Вместо того чтобы вручную записывать, что чашки бывают хрупкими, что предметы падают и что люди обычно ищут укрытие под дождём, можно предоставить системе множество примеров и позволить ей извлечь повторяющиеся закономерности. Это переносит работу от формулирования каждой аксиомы к подготовке данных, выбору модели и проверке того, какие закономерности из них получились.

Разница существенна, но не сводится к противопоставлению «умные правила — глупые данные» или наоборот. Явное правило можно прочитать и изменить, но разработчикам нужно заранее определить понятия и условия. Статистическая модель способна извлечь связи, которые никто не формулировал вручную, но результат зависит от того, какие примеры ей доступны и насколько они отражают реальный мир. Она может обобщить частую связь и не распознать редкое исключение; может усвоить описание вымысла и не отличить его от факта, если задача этого не требует.

Данные тоже не прибывают на сервер с готовой меткой «это здравый смысл». Кто-то выбирает источник, удаляет или сохраняет дубликаты, определяет, как представлять контекст и что считать успешным выводом. Если система должна понимать, что человек иронизирует, но обучалась на буквальных формулировках, она может не уловить смысл. Если текст говорит «после ужина он заснул», модель может усвоить связь между едой и сном, но это не гарантирует, что она правильно рассудит о конкретном человеке или ситуации.

На этом фоне особенно заметна преемственность задач. Сус требовала вручную формализовать связи; машинное обучение извлекает часть связей из данных. Но обеим стратегиям нужно определить, какая информация относится к ситуации, как учитывать контекст, что считать исключением и как проверить ответ. Меняется источник знания и способ его представления, но вопрос о надёжном применении остаётся.

Экспертные системы и базы здравого смысла поэтому важны не как проигравший лагерь до «настоящего» ИИ, а как исследовательские ответы на устойчивую проблему. Они заставили отделить способность выдать правдоподобную фразу от способности поддержать её знанием, использовать вывод в действии и пересмотреть его при появлении новых фактов. Современная система может казаться гибче старой базы правил, однако широта ответа сама по себе ещё не сообщает, где у неё надёжное знание, а где — только знакомый шаблон.

Здесь нужен аккуратный вывод. Если машина отвечает неверно на обычный вопрос, причина может быть не в отсутствии одного бытового факта. Возможно, она не распознала контекст, выбрала неверное значение слова, не заметила исключение или связала событие с неподходящей причинной моделью. Исправить такую ошибку — значит понять, какой тип знания отсутствует и на каком этапе системы он нужен. Именно этому учит история здравого смысла: «добавить ещё фактов» — не диагноз, а лишь один из возможных способов лечения.

Проблема будет оставаться открытой и в системах, обученных на огромных коллекциях текстов, изображений и действий. Эти системы могут располагать несравнимо большим объёмом опыта, чем экспертные базы раннего ИИ, и всё же сталкиваться с ситуациями, где нужно удержать мир согласованным, отличить допущение от наблюдения и понять, что именно спрашивает человек. Урок прошлого не в том, что ручные правила надо вернуть. Он в том, что убедительная работа в одном сценарии не равна надёжному здравому смыслу во всех остальных.

Как проверить то, что по определению выходит за рамки

Для узкой программы можно составить набор случаев и проверить, насколько часто она выдаёт подходящий ответ. Для базы общего здравого смысла возникает сложнее задача: как выбрать тесты, если потенциальные ситуации охватывают физические объекты, язык, время, социальные отношения и тысячи исключений? Любой конечный набор примеров проверяет лишь часть пространства возможных случаев.

Если SHRDLU правильно выполняет команды в блоковом мире, это подтверждает, что её механизмы работают для заданного мира. Тест не может сам по себе показать, что программа разберётся с лестницей, собакой, пролившейся водой или сарказмом пользователя. Если Сус выводит верное следствие из формализованных предпосылок, это демонстрирует работу в соответствующем контексте. Но полнота базы — отдельный вопрос, который таким тестом не решается.

Особенно важно различать «ответ верен» и «система знает, почему он верен». Одна программа может вывести результат благодаря общему правилу, другая — потому что тестовый пример напрямую совпал с записью в базе. Для пользователя оба ответа выглядят одинаково. Чтобы проверить компетентность, нужно давать новые случаи, менять условия и смотреть, сохраняется ли вывод там, где он должен, и отменяется ли там, где появляется исключение.

Для повседневного знания нет очевидного списка всех допустимых исключений. Люди также расходятся в ожиданиях, а культурные и профессиональные нормы меняются. Поэтому проверка требует не одного числового результата, а анализа того, какая задача поставлена, чей стандарт использован и при каких предположениях. Иначе громкое слово «здравый смысл» превращается в оценку, которую невозможно опровергнуть: правильные случаи засчитываются, а ошибки объявляются следствием неполной базы.

Эта трудность не делает исследования бессмысленными. Она подсказывает, что общий интеллект нельзя подтвердить одной демонстрацией. Нужно изучать разные виды знания и проверять перенос: способна ли система использовать знакомую связь в новом случае; заметит ли изменение контекста; задаст ли уточняющий вопрос, если данных недостаточно; откажется ли от вывода, если он противоречит наблюдению.

Такой подход связывает старые проекты с сегодняшними дискуссиями о тестировании ИИ. Чем шире обещание системы, тем больше способов проверить его на границах. Хороший результат в одной задаче остаётся ценным, но не превращается автоматически в доказательство общей надёжности.

Неизвестно — не значит «нет»

Есть ещё одно различие, которое для человека кажется мелочью: система может не знать факт, но это не равносильно знанию его отрицания. Если в базе нет записи о том, что дверь заперта, из этого не следует, что дверь точно открыта. Возможно, её состояние ещё не проверяли. Если программа не знает, хрупкая ли конкретная чашка, ей нельзя автоматически заключать, что чашка небьющаяся.

Некоторые базы данных используют удобное допущение: если факт не записан, система может считать его ложным в пределах своей задачи. Для здравого смысла такое правило часто опасно. Отсутствие сведений об аллергии пациента не означает, что аллергии нет; отсутствие записи о вещи в комнате не гарантирует, что вещи там нет. Нужно различать отрицательный факт, отсутствие факта и неопределённость, вызванную неполными данными.

Люди постоянно управляют этим различием. Если не знаем, закрыта ли дверь, можем проверить ручку. Если неизвестно, кому принадлежит коробка, можем спросить. Если ответ нужен немедленно, иногда принимаем рабочее предположение и продолжаем. В каждом случае мы выбираем реакцию по цене возможной ошибки. Небольшое сомнение в цвете стены не мешает поставить чашку на стол; неизвестное состояние тормозов мешает выезжать на дорогу.

Чтобы действовать разумно, программа должна знать не только факты о внешнем мире, но и границы собственных сведений. Это называют метазнанием: система должна различать, что она знает, чего не знает и какие данные могут устранить сомнение. При этом фраза «я не знаю» сама по себе не решает задачу. Нужно выбрать, остановиться ли, запросить ли данные, передать ли решение человеку или применить безопасное действие по умолчанию.

В SHRDLU неопределённость была ограничена: программа знала, какие предметы существуют в её сцене, и работала со словарём, рассчитанным на эту среду. В открытой комнате неизвестно гораздо больше. Может ли программа считать, что проход свободен, если препятствие не распознано? Можно ли брать предмет, если материал не установлен? На такие вопросы невозможно ответить одним универсальным правилом, потому что цена ошибки зависит от действия и контекста.

Именно поэтому проблема здравого смысла связана с безопасностью, даже если исторические проекты часто ставили её как логическую или языковую задачу. Виртуальная модель может не заметить пропуск без последствий для человека; робот в физическом пространстве должен учитывать риск. Чем ближе программа подходит к действию, тем важнее отличать незнание от отрицания и уметь приостанавливать уверенный вывод.

От здравого смысла к языку

SHRDLU показала, что язык и знание о мире тесно связаны: слова означают больше, чем их место в предложении. Фреймы предложили организовать ожидания о знакомых ситуациях. Проблема рамки заставила точнее формулировать, как действия меняют состояние мира. Сус превратила попытку описать обыденные предположения в долгую работу над общей базой знаний.

Ни один из этих подходов не сделал здравый смысл готовым модулем, который можно вставить в любую программу. Они помогли разложить проблему на более ясные вопросы: как представить объекты и связи; как использовать значения по умолчанию; когда отменять прежний вывод; как выбирать релевантные сведения; что должно оставаться неизменным после действия; в каком контексте утверждение истинно.

Эта история также изменила представление о простоте. Задачи, которым люди годами учатся в школе, могут быть формализованы и проверены. А бытовое движение по комнате, понимание местоимения или различение реальности и вымысла требуют огромного количества фоновых допущений. Для человека они невидимы не потому, что не существуют, а потому, что становятся частью опыта до того, как мы начинаем их объяснять словами.

Когда мы слышим «поставь это туда», ситуация сообщает, что означают местоимения, куда направлено внимание и какие действия разумны. Контекст опирается на знание; знание связано с ожиданиями о мире; ожидания зависят от того, что уже произошло. Поэтому проблема языка не заканчивается на словаре и грамматике. Слова входят в ситуацию, а ситуация может быть понятна только тому, кто знает достаточно о мире вокруг неё.

Следующая глава вернётся к языку — на этот раз за пределами виртуальной комнаты с цветными блоками. Исследователи попробуют строить системы для перевода и обработки человеческих текстов, где нет заранее ограниченного словаря и единой сцены. Там выяснится, что между двумя языками лежит не только грамматика. Между ними — целый мир, который люди привыкли считать очевидным.

Глава 9. Язык, который не помещался в правила

7 января 1954 года в Нью-Йорке перед журналистами и гостями IBM компьютер IBM 701 переводил русские предложения на английский. Это была публичная демонстрация, подготовленная совместно исследователями Джорджтаунского университета и IBM. На экране и в газетных отчётах появлялся результат, который прежде требовал человека: русские слова входили в машину, английские — выходили.

Демонстрация состояла из специально отобранных предложений и очень ограниченного набора знаний. В словаре было около 250 словоформ и основ, а правила описывали лишь небольшое число языковых конструкций. Машина не могла взять произвольную русскую газету и перевести её целиком. Она работала с подготовленными фразами в пределах заранее созданной программы.

Это был настоящий успех — и настоящий предел. Компьютер впервые наглядно показал, что некоторые операции перевода можно формализовать и автоматизировать. Одновременно демонстрация создала образ, который быстро отделился от её условий: «машина переводит русский». Между этими двумя утверждениями лежала разница между экспериментом и универсальным переводчиком.

В этой разнице и начинается история машинного перевода. Исследователи хотели передать машине работу, которая на первый взгляд кажется последовательностью замен: найти значение слова, подобрать эквивалент, переставить фразу. Но язык не является текстом, который можно разобрать на детали, не зная, о чём говорят. Значение зависит от грамматики, контекста и предмета разговора. Переводчик должен не только знать два словаря, но и угадывать, какой мир имеется в виду.

Перевод становится государственной задачей

Машинный перевод привлекал внимание не только лингвистов. После Второй мировой войны научная информация быстро росла, а значительная часть важных публикаций появлялась на языках, которыми потенциальные читатели не владели. Во время холодной войны особенно остро ощущалась потребность быстро разбирать русскоязычные научные и технические тексты. В Советском Союзе была зеркальная задача: получать доступ к англоязычным исследованиям и публикациям.

Каждая сторона сталкивалась с узким местом, которое было очевидно и дорого. Переводчики нужны для точной работы, но их подготовка требует времени, а скорость чтения и перевода человеческая. Ведомство, библиотека или исследовательская группа могли получать документов больше, чем специалисты успевали обработать. Компьютер обещал превратить медленную ручную работу в масштабируемую процедуру: один раз записать правила и словарь, затем прогонять через машину большой поток текстов.

Обещание выглядело особенно убедительно потому, что компьютер уже быстро выполнял арифметические операции и обрабатывал данные. Если машина умеет сортировать, считать и выполнять инструкции, почему бы не поручить ей замену слов одного языка словами другого? Для инвесторов и государственных руководителей здесь виделась практическая цель, для инженеров — задача программирования, а для лингвистов — новый способ формализовать устройство языка.

Но вычислениям предстояло столкнуться с совершенно другим типом сложности. Слово может менять форму в зависимости от своей роли. Один и тот же набор букв имеет несколько значений. Связи между словами могут обозначаться окончаниями в одном языке и порядком слов в другом. Грамматически простое предложение может быть неоднозначным. Короткая фраза из нескольких слов иногда требует от читателя больше знания о мире, чем длинный технический текст.

Поэтому уже в начале машинный перевод был больше, чем подстановка английского слова вместо русского. Нужно было решить, как представить исходное предложение, как определить значение слов в этой фразе и как построить естественный результат на другом языке. Проблема состояла не только в скорости вычислений, но и в том, как разбить перевод на операции, которые компьютер способен выполнить.

До электронных компьютеров: механическая идея Троянского

Попытки механизировать перевод появились ещё до появления цифровых вычислителей. В 1933 году советский лингвист и инженер Пётр Смирнов-Троянский предложил устройство, которое должно было помогать переводить между языками. Его идея была не цифровым компьютером и не автономным переводчиком в современном смысле. Она предполагала работу человека вместе с механическим словарём и операциями над текстом.

По описанию Троянского, участники процесса могли разделить работу между собой. Один человек разбирал исходный текст, другой помогал сформулировать смысл в более нейтральной форме, механическое устройство поддерживало поиск соответствий, а специалист по языку перевода завершал работу и редактировал результат. Это была схема «человек — машина — человек», а не обещание, что устройство само прочитает, поймёт и переведёт книгу.

В его предложении есть важная для всей истории деталь. Троянский пытался определить, какую часть перевода можно стандартизировать, чтобы люди и машина выполняли каждый свою задачу. Словарь можно разложить по карточкам или полосам; грамматические роли можно обозначить специальными символами; человек может анализировать часть структуры; устройство — облегчать операции поиска и сопоставления.

Для этого Троянский рассматривал промежуточное представление, не совпадающее полностью ни с исходным, ни с целевым языком. Некоторые его грамматические обозначения опирались на эсперанто как на упрощённую систему форм. Это не означало, что он собирался переводить все тексты на эсперанто и обратно. Идея заключалась в том, чтобы получить нейтральную схему, через которую можно было связывать структуры языков.

У механического проекта не было доступной вычислительной платформы, которая могла бы выполнять сложные операции автоматически, а широкого внедрения не последовало. Троянский умер в 1950 году, до того как машинный перевод стал международным исследовательским полем. Его работу позднее нашли и переосмыслили советские исследователи, а историки стали рассматривать её как раннюю попытку разделить перевод на анализ, промежуточное представление и синтез текста на другом языке.

Не стоит делать из Троянского легенду о человеке, который в одиночку «изобрёл Google Translate». Его устройство не было реализованным универсальным переводчиком. Ценность проекта в другом: задолго до массовых вычислений он поставил вопрос, который не исчезнет в цифровую эпоху. Что машина действительно может сделать в переводе, если смысл всё равно должен быть разобран человеком?

Уоррен Уивер предлагает читать язык как код

В июле 1949 года математик Уоррен Уивер подготовил меморандум о возможности компьютерного перевода. Во время Второй мировой войны он руководил работой математиков, занимавшихся прикладными задачами и анализом сообщений. Опыт криптографии подсказал ему аналогию: возможно, незнакомый язык можно рассматривать как код, который компьютер способен расшифровать.

Аналогия не означала, что русский или китайский язык — всего лишь шифр. Уивер понимал, что у естественных языков есть структура, многозначность и разница грамматик. Но сравнение с криптоанализом помогало представить принципиальную возможность: в длинном тексте часть неоднозначности снимается контекстом и повторяющимися закономерностями. Если машина сможет учитывать достаточно большой фрагмент, возможно, она определит, какой вариант слова подходит.

Меморандум был важен не только содержанием, но и положением автора. Уивер работал в Фонде Рокфеллера и располагал влиянием на финансирование науки. Он не написал готовый алгоритм перевода; он сформулировал программу исследований и распространил её среди людей, способных поддержать новое направление. В последующие годы в нескольких американских университетах начали изучать автоматический перевод и формальные свойства естественного языка.

Идея Уивера соединяла два взгляда. Первый был инженерным: разобрать предложение на элементы и преобразовать их по правилам. Второй — статистическим: изучать частоты и контекст, чтобы выбрать вероятное значение. На раннем этапе ни один из подходов не был готов решить задачу целиком, но спор между ними возник почти сразу. Нужно ли вручную записать грамматику? Можно ли получать знание из наблюдений? Должна ли система иметь промежуточный язык? Или перевод можно построить как цепочку специальных операций от одного языка к другому?

В 1952 году Yehoshua Bar-Hillel, философ и логик, работавший в MIT, организовал первую крупную конференцию по машинному переводу. Обсуждали словари, грамматические анализаторы, универсальные языки и способы представления смысла. Уже на этой ранней встрече стало ясно: проблема не сводится к тому, чтобы набрать больше слов в память. Исследователи должны договориться, что именно компьютер будет делать с предложением, прежде чем система сможет выдать перевод.

Казалось, что новые вычислительные машины дадут ключ к задаче, которая веками была человеческой. Но чем ближе исследователи подходили к реальному языку, тем заметнее становилось, что «перевод» состоит из множества решений. Каждый из них можно было сделать отдельной задачей, но все они зависели друг от друга.

Что именно делала машина IBM 701

На демонстрации 1954 года IBM 701 работал с подготовленной русско-английской системой. Лингвистическую часть создавали специалисты Джорджтаунского университета, среди них Пол Гарвин; Леон Достерт был одним из организаторов проекта и его публичного показа. IBM предоставила компьютер и техническую поддержку. Этот состав уже говорит о том, что ранний машинный перевод требовал команды: лингвиста, программиста, инженеров и людей, которые могли определить, какие фразы стоит включить в опыт.

В словаре было около 250 лексических элементов, включая основы и окончания. Это число могло звучать как немало для одной машины, но покрывало лишь крошечную часть русского языка. Правил было шесть основных, и они применялись к заранее выбранным конструкциям. По опубликованным описаниям, в демонстрации использовался небольшой набор фраз, подготовленных так, чтобы показать возможности системы. Поэтому разные источники указывают то более шестидесяти тестовых предложений, то 49 предложений, перечисленных в материалах позднейшего анализа. Эта разница в подсчёте не меняет сути: выборка была маленькой и контролируемой.

Система не просто заменяла каждое русское слово отдельным английским. Она могла использовать словарные соответствия, учитывать отдельные грамматические признаки и представлять элементы фразы. Это было необходимо, потому что русский и английский по-разному показывают связи внутри предложения. Для демонстрации выбирали содержательные научные и технические высказывания, где ключевое значение можно было выразить сравнительно прямолинейно.

Представим предложение из строго ограниченной области, где встречаются слова «атом», «молекула», «углерод» или «кислород». Если словарь содержит нужные термины, а грамматика фразы соответствует предусмотренной конструкции, система способна получить осмысленный перевод. Она не обязана знать все значения слов, которые никогда не встретятся в этом наборе. Не требуется переводить идиому, шутку, газетную колонку или стихотворение. Ограничения не были помехой демонстрации; они составляли условие, которое сделало демонстрацию возможной.

Это не означает, что исследователи намеренно выдавали узкий эксперимент за готовый продукт. В публичном рассказе о технологии тоньше устроен механизм ожиданий: конкретная фраза «в этих условиях компьютер перевёл подготовленные предложения» легко превращается в общую фразу «компьютер переводит русский». Заголовку неинтересно объяснять, что в памяти 250 лексических единиц и шесть правил. Но читатель, услышав «машина переводит», естественно представляет возможность ввести любой текст.

Так демонстрация одновременно помогла науке и подняла планку будущих обещаний. Она убедительно показала, что машины способны выполнять не только арифметику. И она дала людям простой образ автоматического переводчика раньше, чем инженеры понимали, как обеспечить стабильную работу на реальных текстах.

Система не была бесполезным трюком. Она проверила гипотезу, позволила изучить технологию перевода, дала публике конкретный результат и привлекла внимание к вычислительной лингвистике. Но её успех отвечал на один вопрос: можно ли запрограммировать перевод некоторых подготовленных предложений? Он не отвечал на другой: можно ли переводить тексты, содержание которых неизвестно заранее?

Демонстрация как тщательно поставленный опыт

Публичный показ IBM 701 был рассчитан на то, чтобы сделать новую область видимой. Компьютер находился в центре внимания; приглашённые журналисты могли наблюдать, как подготовленные русские предложения превращаются в английские. В газетном изложении это легко звучало как событие, после которого перевод уже «поручен машине». Но между этим впечатлением и тем, что происходило в лаборатории, оставалась дистанция.

Любая демонстрация нуждается в ясном сюжете. Если показать тысячи страниц, зритель не сможет понять, почему один результат получился лучше другого. Если показать короткий набор фраз, легче увидеть операцию и оценить её. Исследователи выбрали научные и технические предложения, потому что в них можно было ограничить словарь и использовать сравнительно предсказуемую грамматику. Это было разумное экспериментальное решение: проверить конкретный механизм на примерах, а не обещать систему для всего языка.

Подготовленность фраз не делает демонстрацию фикцией. Она означает, что исследователи отвечали на узкий вопрос: можно ли запрограммировать процедуру, которая преобразует структуру определённых русских предложений в английский результат? Ответ был положительным. Компьютер не просто хранил заранее напечатанные ответы; программа применяла словарь и правила к входной фразе. Но из этого опыта нельзя было сделать вывод, что система сможет прочесть любой научный журнал без помощи лингвиста.

В этом различии часто теряется смысл первых публичных показов. Экспериментатор показывает, что механизм может работать при заданных условиях. Реклама и пресса, напротив, склонны описывать будущую возможность как уже доступную услугу. Между «машина обрабатывала эти фразы» и «машина переводит с русского на английский» лежит не одна техническая доработка, а расширение словаря, грамматики, жанров, контекста и способов проверки.

Если бы организаторы демонстрации показали систему, которая часами разбирает одну сложную фразу и в конце выдаёт спорный результат, интерес публики был бы меньше. Поэтому показу помогал отбор, как он помогает любому эксперименту. Этот отбор одновременно позволил увидеть реальную вычислительную идею и создал риск чрезмерного обобщения. Устройство сработало в хорошо очерченной рамке; будущее обещание начиналось там, где рамка заканчивалась.

Для истории ИИ такая сцена знакома. Маленькая система может быть впечатляющей именно потому, что её авторы контролируют условия. Важный вопрос после показа — не только «сработало ли?», но и «какие условия обеспечили успех, что произойдёт при их изменении и сколько человеческой работы осталось за кадром?». С этой точки зрения демонстрация IBM была началом исследований, а не доказательством скорого исчезновения переводческой профессии.

Слово — не маленькая коробка со значением

Перевод слова за словом выглядит заманчиво. Русское слово «лук» — на английском «opion» или «bow», в зависимости от смысла. Слово «ключ» может быть предметом, источником воды, инструментом для настройки или способом решить задачу. Словарь способен перечислить соответствия, но не выбрать правильное без контекста.

Человек использует сразу несколько подсказок. Смотрит на соседние слова, тему абзаца, жанр текста и ситуацию, в которой фраза сказана. В инструкции по инструменту «ключ» скорее окажется металлическим предметом; в музыкальной статье — знаком; в разговоре о загадке — ответом. Точная догадка часто возникает мгновенно, потому что десятилетия языкового опыта работают одновременно.

Для ранней программы каждый такой выбор требовал дополнительных правил. Система могла проверять соседние слова, учитывать грамматическую форму, использовать словарную метку и выбирать значение по установленному критерию. Но сколько контекста достаточно? Если рядом нет очевидной подсказки, нужно ли смотреть на всю фразу, предыдущие предложения, тему главы или знания о мире?

Грамматика также не снимает неоднозначность автоматически. В разных языках грамматическую информацию кодируют разными средствами: окончаниями, служебными словами, порядком слов и согласованием. Переводчик должен определить, какие части фразы связаны, кто действует, на что направлено действие и какое отношение обозначает предлог. Русский язык позволяет менять порядок слов для выделения темы или акцента; английский сильнее опирается на позицию слов. Формально одинаковая последовательность слов может передавать разные смысловые отношения.

Значит, программе нужно сначала получить представление структуры: определить слова и их формы, выделить части предложения, связать их друг с другом, выбрать значения, а затем построить фразу целевого языка. Каждая ступень может ошибиться, и ранняя ошибка часто переходит на следующий шаг. Неверно выбранное значение приводит к неверной структуре; неверная структура — к переводу, который выглядит грамматически правдоподобно, но сообщает не то.

Даже правила согласования и словарь имеют цену. Для каждого нового слова нужно определить формы, значения, грамматические категории и типичные сочетания. Для каждой конструкции — решить, как её обнаружить и что с ней делать. Чем шире язык, тем больше правил пересекаются. Слова живут не в изолированных ячейках; их поведение меняется в зависимости от соседей и контекста.

Лингвисты годами описывают такие различия для людей. Теперь от них требовалось ещё и представить описание так, чтобы компьютер мог его применить. Автоматизация не отменила лингвистический труд — она сделала его частью инженерного процесса.

Как предложение превращалось в программу

Чтобы понять ограничения ранних систем, полезно представить перевод не как кнопку, а как последовательность промежуточных решений. Пусть входная фраза сообщает: «Учёные изучают свойства нового материала». Даже в таком простом примере программа должна распознать словоформы, определить, какие слова относятся к каким частям предложения, решить, что «учёные» выполняют действие, а «свойства» являются его объектом, подобрать соответствия и собрать английскую фразу в привычном порядке.

В русском языке окончание часто помогает определить роль слова. Форма «материала» показывает, что речь идёт о родительном падеже, но это окончание само по себе не сообщает машине, какую именно связь автор имеет в виду. В английском многие грамматические отношения чаще выражаются фиксированным порядком слов и предлогами. Переводчик должен перенести не набор букв, а отношение между участниками фразы. Если он буквально оставит порядок исходного языка, результат может звучать неестественно; если изменит его неверно, изменится и смысл.

Первый возможный этап — морфологический анализ: программа определяет, какая форма слова перед ней и какие грамматические варианты допустимы. Затем синтаксический анализ строит связи. На следующем шаге семантический выбор решает, что значит каждое многозначное слово в данном окружении. После этого система формирует фразу целевого языка, включая порядок слов, артикли, предлоги и согласование. Такая схема выглядит аккуратно, но сами этапы не независимы. Чтобы выбрать значение, часто уже нужно знать структуру; чтобы построить структуру, иногда нужно угадать значение.

Представим слово «ключ» в фразе «ключ открывает дверь». Здесь предмет почти очевиден. Но в сообщении «ключ к новой установке находится в приложении» речь может идти о коде доступа, а не о металлическом инструменте. Если дальше говорится о воде, «ключ» может означать родник. Ранний словарь способен перечислить все варианты; выбор потребует распознавания темы, конструкции и знаний о том, какие предметы что делают.

Даже хороший разбор не всегда даёт единственную версию. «Старый друг учёного» может означать друга, который давно знаком с учёным, или пожилого друга, принадлежащего учёному. Человек использует контекст и интонацию; письменная фраза может не содержать достаточной подсказки. Машине иногда следовало бы сохранить неоднозначность или запросить уточнение, но системы обычно проектировали так, чтобы они всё равно выдавали один вариант. Это превращало неопределённость языка в уверенную ошибку.

В раннем компьютере каждый этап требовал формальной записи. Компьютер не мог «почувствовать» связь между словами; разработчик должен был указать признаки и действия, которыми программа воспользуется. Это превращало общее знание лингвиста в техническое задание: какую форму распознать, какое правило применить, какие варианты считать допустимыми. При добавлении нового явления старые правила могли конфликтовать с новыми.

Здесь возникал эффект накопления. Добавление правила для одного типа конструкции улучшало результат на знакомых примерах, но могло повредить другой разбор. Если правило слишком узкое, система пропускала варианты. Если слишком широкое, оно применялось там, где не должно. Чем больше становилась программа, тем важнее было управлять не только количеством правил, но и их приоритетами, исключениями и совместимостью.

Подобная задача напоминает не заполнение словаря, а создание целого конвейера решений, где ошибка в начале изменяет все следующие операции. Увеличение вычислительной мощности ускоряло обработку, но не подсказывало, какие анализы верны. Именно поэтому перевод стал одновременно испытанием компьютеров и лабораторией для новых идей о формальном представлении языка.

Прямая дорога, перевод через посредника и общий язык

В 1950-х и 1960-х исследователи спорили не только о том, какие правила нужны, но и о том, как должна быть устроена вся система. В одном подходе программа могла анализировать исходное предложение лишь настолько, насколько нужно для непосредственного перевода на конкретный целевой язык. Такой путь часто называют прямым переводом: правила специально связывают один язык с другим.

Этот способ прост в замысле и дешевле, когда пара языков и область применения ограничены. Можно создать систему для русско-английских текстов определённой тематики, не разрабатывая общий анализатор всех языков. Но если затем добавить японский, французский или арабский, потребуются новые словари и правила, а существующие решения нельзя просто переиспользовать без изменений.

Второй подход разделяет перевод на анализ и синтез. Сначала система разбирает исходный язык и строит промежуточное представление, затем по этому представлению создаёт целевой текст. Так работает идея перевода через перенос: единый анализ может быть переиспользован для нескольких языков, но пары языков всё ещё нуждаются в правилах преобразования между структурами.

Третий, более амбициозный вариант — интерлингва, или промежуточный язык смысла. Система сначала переводит предложение в представление, не привязанное к конкретному исходному или целевому языку, а потом по нему формирует любой выходной язык. Если создать хорошую интерлингву, можно не писать отдельную систему для каждой языковой пары. Анализатор русского соединяется с общим представлением, английский генератор — с тем же представлением; новые языки подключаются к уже существующей основе.

На схеме это похоже на транспортный узел. Если у каждого города есть прямой поезд во все остальные города, число маршрутов быстро растёт. Если существует центральная пересадка, достаточно провести линию от каждого города к узлу. Но метафора скрывает главный вопрос: что именно должно находиться в узле? Смысл не выглядит одной простой последовательностью символов. Нужно представить время, отрицание, роли участников, неопределённость, метафоры, культурные понятия и тему, которая может меняться по ходу текста.

Эти архитектурные споры проявились и в советских проектах. Группа Николая Андреева в Ленинграде изучала промежуточные представления. В Москве параллельно работали команды под руководством Дмитрия Панова и Алексея Ляпунова; лингвисты Ольга Кулагина и Игорь Мельчук занимались автоматическим анализом и переводом. Научная среда была разнообразной: одни исследователи уделяли больше внимания формальной лингвистике, другие — словарям и практическому преобразованию текста, третьи — языкам-посредникам.

Ни одна архитектура не устраняла необходимость знать оба языка и содержание текста. Выбор схемы только определял, где разместить знания и сколько частей системы можно использовать повторно. Прямой перевод складывает сложность в наборе правил для каждой языковой пары; интерлингва переносит сложность в разработку представления, достаточно универсального, чтобы через него выражать разные языки. Посредник сокращает число соединений, но сам может оказаться самой трудной частью всего проекта.

Спор о том, что именно нужно знать машине

За архитектурными схемами стоял более глубокий разногласный спор. Для одних исследователей язык прежде всего был системой грамматических отношений: если научиться надёжно распознавать формы и связи, можно построить правильный перевод. Для других центральной проблемой было значение: грамматически верная фраза не поможет, если система выбрала не то прочтение. Третьи считали, что выбор можно улучшить за счёт контекста и вероятностей, даже если невозможно вручную описать весь мир.

Эти позиции не всегда были взаимоисключающими, но каждая предлагала разный порядок действий. Сторонник грамматического анализа начинал бы с форм и синтаксиса. Сторонник семантического представления спрашивал бы, кто что сделал, чему подвергся и когда это произошло. Статистический подход искал бы регулярности в употреблении и надеялся, что достаточный контекст позволит предпочесть правдоподобный вариант. В 1950-е данных для таких методов было мало, а память и вычислительное время были ограничены. Поэтому первые системы преимущественно полагались на вручную подготовленные словари и правила.

Лингвисты тоже не всегда соглашались между собой. Разные теории по-разному делили слова на категории, описывали грамматические связи и трактовали исключения. Переводческая программа заставляла превратить эту дискуссию в точные решения. Человеку можно сказать: «в этой фразе слово относится к предыдущему предложению». Программа же требует указать, по каким признакам она поймёт, что это именно та фраза и именно та связь.

Проектам приходилось выбирать между полнотой описания и работоспособностью. Если сначала пытаться построить модель всего языка, задача могла остаться на бумаге. Если взять небольшой набор правил, программу можно запустить, но результат останется ограниченным. Демонстрация 1954 года стала заметной потому, что предложила второй путь: вместо завершённой теории языка показать систему, которая выполняет ограниченную, наблюдаемую операцию.

Ранние исследования при этом не были бесполезной подготовкой, которую позже отбросили. Они дали языковой обработке набор вопросов и технических приёмов: как хранить словоформы, как находить границы фраз, как описывать синтаксические связи, как отделять словарный выбор от построения результата. Многие системы позднее соединяли эти приёмы с новыми методами. Разница заключалась в том, где искали недостающую информацию и сколько труда требовалось для её добавления.

Когда исследователи спорили о «понимании», речь часто шла о вполне практическом выборе. Если машине требуется знать, что человек обычно держит книгу в руках, нужно ли записать это правило? Если текст сообщает о том, что книга лежит на столе, достаточно ли выучить словосочетание? Если в контексте появляется новая тема, может ли система вывести значение из окружающих слов? Каждый ответ означал другой объём базы знаний, иной способ обучения и новый тип проверки.

Поэтому история машинного перевода не развивается как простая лестница от плохой системы к хорошей. На каждом этапе менялась ставка: сначала правила для конкретной пары языков; затем разбор и промежуточные структуры; позже большие коллекции примеров и вероятностный выбор. Новые методы решали часть старых проблем, но приносили вопросы о данных, ошибках и контроле качества. Первые спорные решения продолжали влиять на устройство последующих систем.

Советская линия: не сноска к американской демонстрации

Советские исследователи включились в машинный перевод почти сразу после американской демонстрации. Первые группы появились в Москве и Ленинграде в 1955 году; к ним присоединялись специалисты в других научных центрах. Исследования развивались в условиях, отличных от американских, с собственными академическими институтами, вычислительной инфраструктурой и задачами доступа к иностранной литературе.

После смерти Сталина интерес к компьютерам, кибернетике и автоматизации начал расширяться. В ранний советский период машинный перевод связывался не только с отдельными лабораториями, но и с научной политикой и планами обработки растущего массива зарубежной информации. При этом развитие не было ровной государственной программой, в которой одинаково работали все институты. Исследователи спорили о лингвистике, проектировали разные системы и зависели от доступности компьютеров.

В Москве группа Дмитрия Панова строила систему английско-русского перевода для машины BESM, опираясь на ранние американские эксперименты. Алексей Ляпунов организовал работу, в которой участвовали лингвисты Ольга Кулагина и Игорь Мельчук; они изучали перевод с французского на русский и вопросы формального анализа. В Ленинграде группа Николая Андреева развивала собственный подход к промежуточным языковым представлениям. Уже этот краткий перечень показывает, что «советский машинный перевод» не был одним устройством или одной школой.

Именно параллельное развитие важно для истории науки. Американский эксперимент стал известным международным образцом машинного перевода, но он не породил все последующие идеи. В СССР шли споры о том, как представить грамматику, каким должен быть язык-посредник и можно ли строить перевод с опорой на глубокий лингвистический анализ. Некоторые советские программы оказались ближе к американскому подходу, некоторые стремились иначе организовать знание о языке.

Исследования Троянского обрели новую жизнь в этой среде. Его довоенная механическая схема была обнаружена и стала восприниматься как предшественник проектов, которые теперь можно было разрабатывать с помощью электронных вычислителей. Но позднее историк Майкл Гордин показал, что вокруг Троянского возникало несколько рассказов: его могли представить забытым изобретателем, доказательством самостоятельного советского первенства или участником более сложной истории международного обмена идеями. Полезнее рассматривать его работу конкретно, не превращая в соревнование «кто изобрёл машинный перевод первым».

Советская линия включала не только технические эксперименты, но и участие филологов, математиков, инженеров и лингвистов. Исследователи могли спорить о самой постановке задачи: считать ли язык набором грамматических правил, строить ли формальный промежуточный код, нужно ли описывать значение или достаточно обработать лексику и форму. Ответ на такой вопрос определял устройство программы не меньше, чем доступ к вычислительной мощности.

География советской работы помогает увидеть, что машинный перевод был частью более крупного движения по формализации языка. Разрабатывались методы автоматического анализа и поиска информации, создавались словари, исследовались грамматические структуры и языки программирования. Даже если коммерческий универсальный переводчик не появился, накопленные работы влияли на математическую лингвистику и последующие направления обработки естественного языка.

В течение 1950-х интерес к машинному переводу в СССР был масштабным, хотя фактические возможности машин и систем оставались ограниченными. Позднее, когда появились специализированные центры, часть исследований стала ориентироваться на практиче-

скую обработку текстов и черновой перевод. Разные периоды нельзя сводить к простой шкале «сначала энтузиазм, потом провал»: менялись институты, доступ к компьютерам и представления о том, какой результат можно считать полезным.

Вычислительная машина не равна исследовательской программе

Успех проекта зависел не только от качества лингвистической модели, но и от того, где находилась машина, сколько времени можно было на ней работать и кто имел доступ к результатам. Компьютеры 1950-х были редкими и дорогими системами. Их использовали для множества задач, а программирование и ввод данных требовали отдельных специалистов. Даже если исследовательская группа знала, какие правила хочет проверить, ей ещё нужно было превратить их в работающую программу и получить машинное время.

В Советском Союзе этот разрыв был особенно заметен в истории ранних проектов. Идеи могли быть математически оформлены и опубликованы, но практический опыт зависел от доступности подходящего компьютера. Это не означает, что советские исследования были исключительно теоретическими: система Панова запускалась на BESM. Но исследовательские группы работали в неодинаковых условиях, и путь от статьи до регулярной обработки текстов мог занимать годы.

Машинный перевод также требовал инфраструктуры за пределами лаборатории. Нужны были словари, подготовленные корпуса предложений, отраслевые специалисты и люди, способные оценить ошибки. Если цель — разбирать научную литературу, следует решить, какой поток документов важен, кто проверяет терминологию, где хранится перевод и как исправления возвращаются в словарь. Покупка компьютера решала только один участок этой цепи.

По мере роста интереса появились центры, где перевод понимался как часть более крупной работы с научной информацией. Для пользователя иногда было важнее быстро узнать содержание статьи, чем получить литературно совершенный текст. Черновик мог помочь решить, стоит ли заказывать полный перевод, или найти нужное описание в массиве документов. Такая задача отличается от публикации готового текста, но она не бессмысленна: ценность зависит от того, что читатель собирается делать с результатом.

Именно здесь разошлись ожидания исследователей, заказчиков и широкой публики. Заказчик мог считать полезным инструмент, который сокращает очередь на перевод, даже если редактор остаётся необходим. Публика могла услышать обещание автоматического переводчика, который заменит специалиста. Инженеры оценивали, работает ли программа на выбранных конструкциях. Эти критерии не совпадали, и разница между ними не всегда проговаривалась вслух.

Почему язык не поддавался окончательному словарю

Словарь — необходимая часть перевода, но не сам перевод. Обычное слово может иметь несколько значений, которые не связаны с его формой. «Коса» — причёска, сельскохозяйственный инструмент или узкая полоса суши; выбирать нужно по фразе. «Стол» в одном контексте предмет мебели, в другом — ведомство или отдел, а в разговоре «положить вопрос на стол» вообще означает рассмотреть его официально.

Грамматика помогает сузить варианты. Если слово согласуется с определённым родом или стоит в роли дополнения, некоторые прочтения становятся менее вероятными. Но часто контекст всё равно не решает задачу без знания мира. Фраза может быть правильно построена с обоими значениями, а смысл устанавливается тем, что происходит вокруг.

Язык также хранит идиомы — сочетания, смысл которых не получается простым сложением значений слов. Выражение «держать камень за пазухой» не описывает буквально предмет одежды и геологический объект. Его нужно понимать как образ скрытой неприязни или намерения. В английском будет другое устойчивое выражение или нейтральный эквивалент. Автоматическая подстановка слов даст дословный текст, который может быть грамматическим, но не передаст значение.

Ещё сложнее метафоры, ирония и многозначные высказывания. Когда человек говорит «отличная работа» после очевидной ошибки, он может иметь в виду прямо противоположное. Грамматика не предупреждает машину, что перед ней сарказм. Нужно учитывать отношения между собеседниками, ситуацию, предыдущее событие и тон — сведения, которые часто отсутствуют в письменном тексте.

Даже технические тексты не освобождают от контекста. В статье о химии слово может иметь специальное значение; в юридическом документе близкие термины могут различаться по правовому эффекту; в научном отчёте автор может намеренно вводить новое определение. Для каждой отрасли создают терминологические словари, но специалисты должны постоянно проверять, как конкретное слово употреблено в конкретном документе.

По мере добавления новых правил росла не только база, но и стоимость её согласования. Если система знает несколько значений слова, нужно определить, какие признаки позволят выбрать нужное. Если форма слова меняет его роль, надо распознать окончание и связать его с конструкцией предложения. Если предложение допускает несколько разборов, приходится ранжировать варианты, а для ранжирования нужна информация о вероятности или смысле.

Так появляется эффект лестницы. Сначала нужен словарь. Затем морфология, чтобы распознать формы. Затем синтаксис, чтобы установить отношения. Затем семантика, чтобы выбрать значения. Затем контекст, чтобы объединить предложения. Затем знания о мире, чтобы понять, что говорящий подразумевает. Каждая ступень решает реальную часть проблемы, но не заменяет остальные.

В этом смысле перевод стал одним из самых строгих способов исследовать язык. Чтобы автоматизировать преобразование текста, нужно было решить, какие свойства фразы сохраняются при переходе на другой язык, какие меняются и какую информацию нельзя восстановить по одним словам. Машинный перевод не просто просил лингвистов записать правила; он проверял, достаточно ли этих правил для результата, который читатель считает верным.

Перевод не останавливается на точке

Даже если система научилась разбирать отдельное предложение, текст продолжается за его границами. Местоимение «он» отсылает к кому-то, названному раньше. Слово «это» может указывать на целое событие или на одну деталь. Автор вводит термин, а дальше использует его короче. В русском и английском читателю иногда нужны разные подсказки, чтобы понять, о чём идёт речь.

Например: «Исследователь положил образец на стол. Он был холодным». Что было холодным — образец или стол? Грамматическая форма русского местоимения не всегда решает вопрос; смысл и ситуация помогают человеку выбрать. Перевод на английский требует указать, к какому предмету относится местоимение, либо перестроить фразу так, чтобы сохранить неопределённость. Если программа смотрит только на последнее предложение, она может сделать выбор без достаточных оснований.

Связь между предложениями важна и для темы. В одном абзаце автор может сначала назвать «систему», затем перейти к «программе», «она» и «механизму», говоря об одном объекте. Переводчик должен понять, относятся ли эти слова к одному предмету или появились новые участники. Названия и сокращения требуют похожей памяти: текст может один раз расшифровать аббревиатуру, а потом использовать её десятки раз.

Некоторые языки позволяют опустить то, что уже ясно из контекста. Другие чаще требуют обозначить участника или грамматическую связь. При переводе приходится добавлять слово, менять конструкцию или оставлять часть информации невыраженной. Значит, не все операции выполняются внутри предложения. Система должна сохранять сведения о предшествующих фразах, а затем решать, какие из них ещё важны.

Здесь возвращается вопрос об объёме памяти в широком смысле. Машине нужны не только ячейки для слов и правила для грамматики, но и способ поддерживать нить повествования: кто уже появился, какие события произошли, что автор считает известным читателю. Маленькая демонстрационная коллекция могла обойти эту задачу за счёт простоты фраз; открытый текст быстро обнаруживал её.

Для людей связность часто незаметна, потому что читатель автоматически связывает местоимения, темы и прежние сведения. Переводчик тоже может осознанно перечитать абзац, если связь неясна. Программа должна либо хранить контекст в пригодной для вычисления форме, либо полагаться на эвристики. Ошибка такого рода способна не испортить грамматику предложения, но поменять участника, к которому относится действие.

Так становится понятнее, почему речь о переводе так быстро приводит к вопросам о понимании. Система должна выбрать не только значение слова, но и то, как текущая фраза входит в уже рассказанную историю. Её задача растёт вместе с текстом: от знаков к словам, от слов к предложениям, от предложений к связному высказыванию и ситуации.

Бар-Хиллел и «коробка в загоне»

Йехошуа Бар-Хиллел стал одним из самых заметных ранних критиков идеи полностью автоматического высококачественного перевода. В 1950-е годы он работал над машинным переводом и изучал, что именно потребуется от системы, которая должна выдавать не только понятный черновик, но и текст, сопоставимый с работой профессионального переводчика.

Его аргумент был не в том, что компьютеры никогда не смогут переводить. Он разделял разные задачи: быстро получить приблизительное содержание, помочь переводчику и самостоятельно выдать высококачественный перевод любого текста — не одно и то же. Он сомневался, что машины того времени и доступные лингвистические методы способны обеспечить последний вариант без огромных знаний о мире.

В известном примере фигурирует английское слово «pen». Оно может означать пишущий инструмент или загон для животных. Если фраза сообщает, что коробка находится «в pen», грамматический разбор не решает, о каком значении идёт речь. В соседнем предложении может быть сказано, что ребёнок ищет игрушечный ящик; здравый смысл подсказывает, что тот, вероятно, находится в манеже или загоне, а не внутри авторучки. Но чтобы сделать этот вывод, машине нужно знать, что такое игрушечный ящик, что такое загон, кто такой ребёнок и какие предметы обычно помещаются друг в друга.

На таком примере видно, почему словарь не может быть единственным источником смысла. Оба значения слова корректны, оба могут соответствовать грамматике, но реальная ситуация делает одно прочтение абсурдным. Человек выбирает интерпретацию, соединяя фразу с представлением о мире. Компьютеру нужен либо формальный аналог этих знаний, либо иной способ получить их из большого числа примеров и контекстов.

В 1960 году Бар-Хиллел изложил критику амбиции «полностью автоматического высококачественного перевода». Важно не превращать этот тезис в фразу «Бар-Хиллел доказал, что компьютерный перевод невозможен». Его аргумент относился к максималистской цели и состоянию исследований того времени. Он не отрицал возможности чернового перевода, помощи человеку или систем для узких областей. Он предупреждал, что высокое качество в произвольном тексте требует не только мощного словаря и грамматики.

Дискуссия имела практическое значение. Одни исследователи считали, что достаточно разработать более глубокий синтаксический анализ и большую лингвистическую базу. Другие полагали, что система должна располагать общими знаниями о мире и здравом смысле. Третьи рассчитывали, что контекст и статистические закономерности помогут выбрать правильный вариант без полного моделирования мира. Во всех трёх случаях нужно было признать, что словарный перевод — лишь первый шаг.

Предупреждение Бар-Хиллела часто звучало пессимистично, особенно на фоне ожиданий после успешной демонстрации. Но его вклад был не в том, чтобы закрыть тему, а в том, чтобы потребовать точнее определить, что именно считается качественным переводом и насколько широкий текст система должна обрабатывать. Машинный перевод стал одновременно инженерной гонкой и спором о границах того, что можно формализовать.

Цена правила: кто переводит до и после компьютера

В ранних системах люди оставались внутри процесса на каждом этапе. Лингвисты выбирали словарные значения и грамматические конструкции; программисты превращали их в код; специалисты по предметной области помогали решить, какие термины встречаются в реальных документах. Пользователь мог подготовить текст, чтобы сделать его проще для системы. После получения результата редактор исправлял перевод.

Человеческий труд не исчезал. Он перемещался до запуска — в создание словарей, классификацию значений, разметку примеров и составление правил; а после запуска — в редактирование, исправление терминологии и выявление ошибок. Автоматизация могла ускорить некоторые этапы и изменить способ распределения работы, но она не отменяла необходимость человека, который понимал оба языка и предмет перевода.

Предобработка особенно важна для ранних систем. Если автор использовал короткие предложения, знакомую терминологию и предсказуемые формулировки, программа могла добиться заметно лучшего результата. Иногда текст можно было писать по специальным правилам — избегать многозначных слов, сложных оборотов и неясных местоимений. Так возникал компромисс: либо система учится работать с естественным текстом, либо люди адаптируют текст под то, что система может обработать.

Постредактирование давало другой путь. Машина быстро выдавала черновик, специалист исправлял ошибки и доводил стиль до нужного качества. В технической документации такой процесс может быть полезен, даже если перевод далеко не готов для публикации. В юридическом или литературном тексте цена неточности и стилистической потери намного выше. Качество нужно измерять относительно конкретной работы, а не одной общей шкалы.

В этом заключается один из устойчивых уроков машинного перевода. Вопрос «может ли машина переводить?» слишком широк. Важно спросить, какие языки, какие жанры, какой объём, с какой точностью, кто исправляет ошибки, сколько стоит подготовка ресурсов и кто несёт последствия неверного результата. Одна система может помочь исследователю просмотреть тысячи страниц, но быть непригодной для договора. Другая может переводить инструкции с ограниченным словарём, но не справляться с рекламным текстом.

История раннего перевода показывает, что каждая автоматизация создаёт новый участок человеческой работы. Когда компьютер ускоряет выбор слов, появляется потребность тщательно разметить словарь. Когда он строит черновик, возникает редакторская работа. Когда машина переводит быстро, организация должна определить, какие тексты можно поручать ей без проверки. Производительность не равна автономии, а скорость не равна качеству.

Узкая область — разумный инженерный компромисс

Сложность открытого языка не означает, что правила бесполезны. В технической документации термины повторяются, стиль стандартизирован, а структура предложений предсказуемее, чем в романе или личном письме. Если тексты ограничены одной предметной областью, можно создать специальный словарь и грамматические модели, которые дадут полезный результат.

Такой подход не обещает переводить всё. Он оптимизирует процесс для известной задачи. Например, оператор может использовать черновой перевод технического отчёта, чтобы найти нужную информацию; переводчик проверит критические места. Компания может стандартизировать написание инструкций, чтобы система чаще встречала знакомые конструкции. Пользователь видит экономию времени, хотя система не понимает все возможные способы человеческого выражения.

Для бизнеса узкая область означает контроль. Можно собрать типичные тексты, определить стандарт терминов, проверять перевод на примерах и обновлять словарь по мере появления новых моделей или требований. Чем свободнее исходный язык, тем больше вероятность, что система встретит конструкцию, на которую у неё нет правила. Ограничение формулировок снижает неопределённость.

Но сама идея «просто ограничить область» может быть обманчивой. Технический текст всё равно содержит неоднозначность, названия, ошибки, сокращения и ссылки на предыдущие части. У предметной области есть собственные метафоры и жаргон. Обновление словаря требует участия специалистов. Даже строго контролируемый язык создаёт исключения, потому что люди продолжают писать для других людей, а не только для программы.

И всё же ограничение — это стратегия, а не признание поражения. Большинство практических систем не обязаны решить весь язык. Они могут быть полезными, если их возможности точно обозначены и пользователь понимает, где требуется проверка. Ошибка ранней индустрии заключалась не в том, что исследователи начинали с маленьких систем, а в том, что успех маленькой системы порой воспринимался как доказательство близкого универсального перевода.

В этом смысле ранний опыт был созвучен SHRDLU из предыдущей главы. Маленький мир позволяет встроить знания и контекст непосредственно в проект. Перевод в ограниченной предметной области работает сходным образом: программа заранее знает, какой словарь и стиль наиболее вероятны. В обоих случаях возникает честная польза и соблазн экстраполировать её далеко за пределы испытанной среды.

Один результат — несколько разных оценок

Рассмотрим перевод отчёта о новом материале. Исследователь, которому нужно быстро понять, о чём статья, может считать приемлемым черновик с неуклюжими оборотами, если технические факты переданы верно. Редактор журнала предъявит другие требования: ясность, точность терминов и отсутствие двусмысленных фраз. Переводчик, который отвечает за инструкцию по безопасности, потребует ещё более строгой проверки, потому что ошибка может привести к неправильному действию.

Для системы одной отметки «хорошо» недостаточно. Можно отдельно оценить, правильно ли передано основное содержание; сколько предложений читатель понял без оригинала; сколько времени понадобилось на редактуру; сколько терминов пришлось исправлять; насколько часто программа пропускает отрицание, число, имя или единицу измерения. Такие показатели отвечают на разные вопросы. Система может быть быстрой, но требовать долгой редакции; может быть понятной, но менять оттенок утверждения; может редко ошибаться в общей теме, но опасно путать один критический термин.

Публичная демонстрация и рабочая эксплуатация поэтому измеряют разные вещи. Демонстрация показывает, что система способна на выбранном примере. Рабочая оценка проверяет, можно ли включить её в устойчивый процесс и что происходит с неотобранными текстами. Для этого нужны независимые примеры и заранее оговорённое назначение. Иначе у разработчиков есть соблазн оценивать систему по тем фразам, для которых её правила уже были подобраны.

У машинного перевода рано появилась ещё одна сложность: эталонный перевод не всегда единственный. Русскую фразу можно передать по-английски разными синтаксическими конструкциями; одна версия сохранит структуру оригинала, другая будет естественнее для читателя. Экспертам нужно было различать ошибку смысла и предпочтение стиля. Это требует времени и ясных критериев, а не только сравнения с единственной строкой ответа.

Поэтому практический результат должен включать не только программу, но и описание того, для каких документов она предназначена, какие ошибки допустимы, а какие требуют обязательной проверки. Такое описание может выглядеть менее захватывающе, чем заявление о переводе целого языка, но именно оно помогает понять реальную пользу. Узкая система с честно заданной областью может принести больше пользы, чем универсальное обещание, которое невозможно проверить.

Оценка качества превращается в отдельную проблему

Качество перевода не сводится к тому, совпадает ли каждое слово с одним заранее выбранным эталоном. Даже два профессиональных переводчика могут построить разные фразы и оба сохранить смысл. Выбор зависит от назначения текста, аудитории, стиля и цены ошибки. Поэтому оценка требует больше, чем списка правильных ответов.

Для лабораторного опыта можно заранее подготовить несколько предложений и показать, что машина их обрабатывает. Такая проверка удобна и полезна, но не показывает, что система справится с новым текстом. Чем ближе тестовый набор к структурам, на которых настраивали программу, тем меньше он говорит о поведении на незнакомом материале. Нужно проверять новые примеры, разные жанры и варианты формулировок.

Человек обычно оценивает перевод сразу по нескольким критериям: можно ли понять смысл, верны ли факты, естественно ли звучит фраза, не исчезли ли важные оттенки. Один текст может быть понятным, но стилистически неловким; другой — гладким, но с неверным термином. Одна метрика редко способна одновременно измерить все эти качества.

В ранних исследованиях качество часто оценивали экспертно и с учётом конкретного назначения. Машина могла помогать быстро просматривать документы, даже если её текст не годился для публикации. Но если результаты демонстрации представляли публике как «высококачественный перевод», возникало разногласие: речь о полезном черновике или о результате, который можно доверить без редактора?

Эта проблема помогла сделать машинный перевод более серьёзной дисциплиной. Исследователи стали тщательнее разделять тестовую демонстрацию и рабочую эксплуатацию. Нужно было считать не только секунды машинного времени, но и часы человеческой подготовки и редактуры. Система, которая переводит предложение за секунду, может оказаться медленнее, если на каждую строку приходится долго исправлять неправильный смысл.

На этом фоне разочарование середины 1960-х не было внезапным приговором языку как таковому. Оно возникло из-за разрыва между обещанным качеством, экономическими ожиданиями и результатами доступных систем. Отчёт ALPAC и его последствия подробно рассматриваются в главе о первой зиме ИИ; здесь важно, что перевод дал один из самых ясных примеров различия между красивой демонстрацией и устойчивым производственным процессом.

После большого разочарования осталась прикладная работа

К середине 1960-х ожидания от машинного перевода столкнулись с проверкой на стоимость и качество. Отчёт ALPAC, подробно рассмотренный в главе 6, критиковал американскую программу автоматического перевода и рекомендовал не наращивать финансирование универсального машинного перевода в прежнем масштабе. Документ стал символом «зимы» для области, хотя последствия в разных странах и учреждениях были неодинаковыми.

Для исследователей важен был не только размер бюджета. Возник вопрос, соответствует ли полученный результат реальной потребности. Если учёному нужен быстрый доступ к содержанию статьи, машина может дать полезный черновик. Если редакции нужен текст, который можно публиковать без проверки, тот же результат окажется неприемлем. Отчёт усилил требование сравнивать машинный перевод с человеческой работой и учитывать полную стоимость процесса: подготовку словарей, ввод исходных документов, время редактора и качество итогового текста.

Оценка ALPAC стала поводом для споров. Критики указывали, что исследование измеряло доступные на тот момент американские системы и не могло предсказать, что появится после новых методов и вычислительных ресурсов. Сторонники подчёркивали, что обещания уже влияли на финансирование и ожидания заказчиков; полагаться на возможное будущее было недостаточно. Так спор сместился от вопроса «можно ли когда-нибудь перевести языки?» к вопросу «какие результаты доступны сейчас и оправдывают ли они затраты?».

В последующие годы машинный перевод не исчез, но менялись места, цели и форма работы. В Канаде, Европе, Японии и Советском Союзе продолжали создавать системы для отдельных пар языков и задач. Один из устойчивых сценариев — перевод технической или научной документации с участием человека. Машина предлагает черновик либо облегчает поиск, а специалист проверяет важные места. Такой подход не отменяет автоматизацию: он задаёт ей более скромную и проверяемую роль.

Практическая полезность могла заключаться и в масштабе. Даже если специалист должен просмотреть каждый машинный результат, система способна помочь быстро отделить явно нерелевантные документы от тех, которые стоит переводить полностью. В других случаях ограниченный словарь и повторяющиеся формулировки делали достаточно предсказуемыми большие потоки технического текста. Не всякая экономия видна в качестве отдельного предложения; иногда она проявляется в том, сколько материала организация вообще успевает обработать.

Вместе с тем компромисс требовал прозрачности. Пользователь должен знать, является ли результат готовым переводом или черновиком, где он может ошибаться и какие фрагменты требуют обязательной проверки. Если это не объяснено, читатель склонен принимать гладкий машинный текст за надёжный. Ранние системы могли выдавать связный английский даже тогда, когда неверно разрешили неоднозначность. Внешняя уверенность результата иногда маскировала ограниченность его основания.

ALPAC не поставил точку в исследовании языка. Он помог сделать цену обещания видимой. Систему следовало оценивать не по тому, что она однажды смогла сделать на сцене, а по устойчивости в ежедневной работе, качеству на незнакомых материалах и объёму помощи, которую по-прежнему оказывал человек. Именно такое более строгое понимание пользы помогло машинному переводу пережить период снижения ожиданий.

Машинный перевод не закрылся вместе с финансированием

После переоценки ожиданий исследования машинного перевода не исчезли. Изменились объём финансирования и цели; больше внимания стали уделять вычислительной лингвистике, помощи переводчику и практическим системам для ограниченных задач. Советские исследования продолжались в другом институциональном ритме; в некоторых центрах позднее пытались выдавать черновой перевод больших массивов текстов, даже если его качество оставалось умеренным.

Это важно потому, что слово «зима» может создать неверный образ полного прекращения работы. Проект, который больше не обещает универсальную замену переводчика, всё ещё может принести пользу как инструмент поиска или черновой обработки. Технология не исчезает; исчезает часть прежней уверенности в сроках и масштабе.

В 1980-х и 1990-х машинный перевод получит новый импульс через большие коллекции уже переведённых документов. IBM, в частности, исследовала статистические методы, при которых система оценивала вероятные соответствия на параллельных текстах вместо того, чтобы полагаться только на вручную написанные правила. Идея вновь сместила центр тяжести: не расписывать каждую языковую конструкцию заранее, а извлекать регулярности из примеров.

Затем нейронные модели изменяют устройство переводчика ещё раз. Вместо цепочки отдельных словарных и грамматических компонентов система будет обучаться преобразованию целого предложения, используя большие массивы примеров и внутренние представления. Позднее механизм внимания поможет модели сопоставлять участки входного и выходного текста. Эти этапы относятся к дальнейшим главам книги; здесь достаточно заметить, что они не отменили вопросов, поставленных первыми переводчиками: что является контекстом, как измерять качество и кто отвечает за результат.

И сегодня кнопка «перевести» не означает, что язык стал простым. Машины покрывают больше языковых пар и жанров, чем их предшественники могли вообразить, но качество зависит от данных, предметной области, формулировки и назначения текста. Система может выдать убедительное предложение и перепутать важное имя, оттенок или отрицание. Этим она отличается от IBM 701 скорее масштабом и методом, чем отсутствием ограничений.

Язык потребовал разговора о понимании

Ранние эксперименты с переводом заставили исследователей уточнить, какой результат считать пониманием. Если программа верно преобразует предложение, нужно ли ей понимать мир в человеческом смысле? Если не нужно, достаточно ли статистической связи между формами? А если смысл зависит от ситуации, какие знания можно извлечь из текста, а какие требуется встроить отдельно?

В 1954 году публичная демонстрация ответила на узкий вопрос: компьютер может выполнять заранее определённые операции над предложениями и выдать приемлемый результат в рамках заданной системы. Следующие годы показали, что универсальное обещание требует намного больше: формальной лингвистики, контекста, словарей, знаний о предмете, человеческой оценки и масштабной работы по подготовке ресурсов.

Машинный перевод не проиграл одному набору правил. Он обнаружил, что естественный язык состоит из нескольких задач, переплетённых между собой. Правила помогли исследователям понять морфологию, синтаксис и структуру преобразования. Их предел проявился там, где значение зависело от знаний, которые люди обычно не проговаривают.

Именно поэтому заголовок этой главы говорит о языке, который не помещался в правила. Не потому, что правила были бесполезны, а потому, что язык каждый раз приносил с собой больше контекста, чем разработчики могли заранее перечислить. Ошибка не всегда находилась в грамматике. Иногда машине не хватало общего знания о мире; иногда человек ожидал слишком много от демонстрации; иногда переводчику был нужен не автономный заменитель, а помощник, которого можно контролировать.

После истории языка книги пора вернуться к вопросу о том, что происходит, когда подобные ожидания становятся рынком. В 1980-х экспертные системы и специализированные компьютеры снова обещали практический искусственный интеллект. На этот раз у индустрии появились заказчики, поставщики и большие бюджеты. Зима, которая последует, будет не кописей первой: уже существовали системы, за которые компании платили, и именно их стоимость и хрупкость станут частью следующего разочарования.

Глава 10. Вторая зима не была повтором первой

Представим обычный рабочий день в отделе, который готовит к отправке заказ на компьютер. Клиент выбрал процессор, память, диски и устройства ввода-вывода. Каждый пункт выглядит допустимым сам по себе. Вместе они могут оказаться несовместимыми: контроллер не поддерживает выбранный накопитель, памяти недостаточно для нужной конфигурации, а к одной модели нельзя подключить определённую периферию. Пока такие заказы проверяют вручную, поток ограничен временем и вниманием технических редакторов. Если ошибка обнаружится поздно, компьютер придётся пересобирать, а поставку — задерживать.

В начале 1980-х Digital Equipment Corporation, или DEC, начала использовать программу, которая помогала проверять именно такие заказы. В университете Карнеги-Меллон её называли R1, в самой компании — XCON. Программа применяла правила к списку выбранных компонентов и строила согласованную конфигурацию. Это не было интеллектом общего назначения и не было компьютерным инженером, спрятанным в корпусе. Это была система для ограниченной, дорогой и достаточно хорошо описанной задачи. Её успех показал, что формализованное знание иногда можно превратить в рабочий инструмент, а не только в эффектную демонстрацию.

У XCON, однако, был второй результат, менее удобный для рекламного буклета. Программа нуждалась в специалистах, которые выясняли, как меняются правила, помогали их записывать и проверяли последствия обновлений. Чем полезнее становилась система, тем больше компания зависела от работы по её сопровождению. Автоматизация не устранила людей из процесса: она переместила их усилия в разработку правил, обучение, контроль и исправление ошибок.

Вокруг экспертных систем в те годы вырос рынок специализированных рабочих станций, программных инструментов и консультационных компаний. Одновременно Япония вложилась в государственную программу компьютеров пятого поколения, а Соединённые Штаты и Великобритания разрабатывали собственные инициативы, видя в интеллектуальных системах и вычислительной технике вопрос промышленного и стратегического преимущества. Для многих наблюдателей будущее складывалось в убедительную картину: знания станут новым сырьём, компьютерные системы научатся применять их, а отрасль получит новую технологическую платформу.

Затем часть компаний потеряла рынок, государственные программы завершились, а заказчики начали задавать более жёсткие вопросы о стоимости и практической отдаче. Это время принято называть второй зимой искусственного интеллекта. Но сама метафора может обмануть: она создаёт впечатление, будто область замерла целиком, а затем однажды оттаяла. В реальности одни продукты продолжали работать, другие проекты оставили исследовательские результаты, а коммерческий спрос на некоторые платформы обвалился. Чтобы понять эту зиму, нужно увидеть разницу между тем, что действительно умело работать, и тем, что индустрия обещала поверх этого.

Специализированный компьютер для новой профессии

В лабораториях искусственного интеллекта в США одним из главных рабочих языков был Lisp. Его создавали в конце 1950-х для обработки символических выражений и списков. Это хорошо подходило для программ, которые манипулировали формулами, словами, правилами и структурами данных, изменявшимися во время выполнения. Lisp не был «языком ИИ» по своей природе, но многие ранние символические программы писали именно на нём.

Обычные компьютеры того времени не всегда удобно справлялись с Lisp-программами. Часть ограничений была связана со скоростью, часть — с тем, что языки и аппаратное обеспечение проектировались друг относительно друга не слишком тесно. Исследователи хотели среду, в которой программа могла оставаться запущенной, пока разработчик исследовал её состояние, исправлял объект или менял фрагмент кода. Для этого требовалось не только больше вычислительной мощности. Нужен был целый набор инструментов, рассчитанных на работу с Lisp.

В Массачусетском технологическом институте Ричард Гринблатт и Томас Найт создали CADR — Lisp-машину, которая стала важным прототипом для коммерческих систем. В начале 1980-х вокруг этих разработок появились компании Lisp Machines Inc. и Symbolics. Они предлагали не просто компьютеры, а связку оборудования, операционной системы, редактора, отладчика и языка. Для разработчика это могло означать более короткий цикл между идеей, запуском и исправлением программы.

Такой инструмент имел смысл, пока его преимущества ощущались каждый день. Если инженер мог быстрее исследовать большую символическую программу, специальная рабочая станция экономила время и расширяла круг задач, за которые лаборатория готова была браться. У компьютера была своя ниша — и в начале десятилетия эта ниша казалась будущим целого рынка.

Но коммерциализация лабораторной технологии изменила отношения внутри самой MIT AI Lab. Часть сотрудников ушла в Symbolics, другие поддержали проект Гринблатта, а между бывшими коллегами возникли споры о стратегии, ресурсах и том, кому принадлежит право развивать созданную в лаборатории систему. В этой истории не было простого конфликта между «учёными» и «бизнесменами». Многие участники хотели, чтобы их инструменты вышли за пределы университета; спорили они о том, как это сделать и что произойдёт с общей исследовательской средой.

Первые пользователи покупали Lisp-машину не потому, что на корпусе было написано «искусственный интеллект». Они покупали специализированную рабочую среду, которая ускоряла конкретный тип программирования. И всё же судьба этой техники постепенно привязалась к судьбе экспертных систем. Если компании будут массово создавать базы знаний и программы правил, им понадобятся инструменты, на которых такие системы удобно разрабатывать и запускать. Оптимизм по поводу одного рынка подпирал ожидания по поводу другого.

В этом и заключалась уязвимость модели. Специализированную машину оценивали не в сравнении с абстрактным компьютером, а с тем, что можно было купить и обслуживать на практике. Когда универсальные рабочие станции становились дешевле и мощнее, заказчику приходилось решать, оправдывает ли прирост скорости цену отдельной платформы. Кроме того, специальная машина часто тянула за собой собственную операционную систему, средства разработки и требования к обучению сотрудников. Если задача переставала нуждаться в максимальной производительности Lisp-среды, преимуществ могло оказаться недостаточно.

Смена рынка не была простой заменой медленного устройства быстрым. Обычная рабочая станция могла предложить компании сетевую совместимость, общие операционные системы и широкий выбор программ. Разработчики могли переносить приложения между

машинами или использовать коммерческое оборудование, доступное за пределами небольшой группы специализированных поставщиков. Такие преимущества редко попадают в таблицу сравнения вычислительной скорости, хотя именно они снижают стоимость владения и риск зависимости от одного производителя.

Для Lisp-машин это создавало структурную проблему. Их сильная сторона состояла в тесной связи языка с машиной и рабочей средой; тем же самым они отличались от платформ, уже привычных корпоративным отделам информационных систем. По мере того как обычная техника становилась способна запускать Lisp и другие инструменты, заказчик получал больше способов добиться приемлемого результата. Он мог согласиться на некоторую потерю удобства в обмен на совместимость и более широкий рынок оборудования. Выигрывали не обязательно лучшие машины по каждому техническому параметру, а те варианты, которые проще было встроить в существующую экономику компании.

Переход на универсальные системы не происходил в один день и не означал, что специализированные машины были технической ошибкой. Пока компьютеры общего назначения плохо подходили для символического программирования, отдельная архитектура давала разработчику то, чего иначе не было. Но преимущество платформы было временным: массовая техника быстро улучшалась, а установленную базу специального оборудования нужно было поддерживать. На рынке, где обещали быстрый рост экспертных систем, срок окупаемости казался приемлемым. При замедлении этого роста тот же парк устройств превращался в расход, который нужно было объяснить бухгалтерии.

Покупка Lisp-машины была похожа на выбор не отдельного инструмента, а целого рабочего места. В нём язык, редактор, отладчик и операционная система были рассчитаны друг на друга. Разработчик мог изучать внутреннее состояние работающей программы и изменять её частями, не возвращаясь каждый раз к началу обычного цикла компиляции. Для исследовательской команды это сокращало трение между экспериментом и исправлением ошибки.

Такая цельность имела и обратную сторону. Программа, написанная под одну среду, не обязательно переносилась на другую без переделки. Внутренние форматы, библиотеки и привычки команды могли привязать проект к поставщику. Пока специализированная станция давала заметный выигрыш, зависимость выглядела приемлемой ценой за производительность и удобство. Когда универсальные системы начали закрывать те же потребности, зависимость стала частью расчёта окупаемости.

Разговор о «лучшем компьютере» поэтому часто оказывался слишком узким. Заказчику предстояло оценить не один процессор, а цену обучения сотрудников, поддержки программ, замены оборудования и доступа к специалистам. Компания могла выбрать станцию с меньшей скоростью, если та запускала знакомую операционную систему, подключалась к уже существующей сети и позволяла нанимать разработчиков с более распространёнными навыками. Технологическое преимущество важно только до тех пор, пока оно перевешивает стоимость перехода и сопровождения.

XCON: полезная система, у которой появились обязанности

Задача конфигурации компьютеров DEC была подходящей для правил не потому, что она была примитивной, а потому, что её границы можно было очертить. У компании имелся ассортимент оборудования, отношения совместимости и процедура, по которой заказ превращался в конкретную спецификацию. Клиент выбирал компоненты. Технический редактор должен был установить, можно ли собрать их вместе, чего не хватает и какие инструкции нужны для дальнейшей работы.

Программа Джона Макдермотта из Карнеги-Меллон применяла к этой задаче так называемую производственную модель. Производственное правило обычно записывают как условие и действие: если выполняются указанные обстоятельства, сделай следующий шаг. Например, наличие одного компонента может требовать определённого контроллера; выбранная модель системы может ограничивать допустимый тип памяти. Механизм вывода проверяет условия и применяет подходящие правила. Он не «видит компьютер целиком», как человек-эксперт. Он выполняет цепочку формализованных проверок.

Первую версию R1 передали DEC в 1980 году. В описаниях участников проекта говорится о сотнях правил в начальной системе, затем база значительно расширялась. К середине десятилетия её объём измерялся уже тысячами правил. Эти числа характеризуют конкретную развивавшуюся программу, но сами по себе мало говорят о качестве. Тысяча правил не гарантирует ни правильности, ни полноты; небольшая база может быть достаточной для узкой задачи, если все важные варианты учтены и результат проверяется.

Для DEC XCON была не лабораторной игрушкой. Она включилась в производственный поток заказов на системы VAX. Система помогала снизить нагрузку на ручную проверку и обеспечивала повторяемый способ находить несовместимые комбинации. Экономическая ценность возникала не из таинственной способности «думать», а из сопоставления с прежним процессом: сколько времени уходит на проверку, сколько ошибок обнаруживают поздно, как часто возникает повторная работа и какой объём заказов отдел способен обработать.

Её результат, однако, нельзя оценивать одной цифрой. Для заказчика имели значение скорость обработки заказа, число конфигураций, которые система могла проверить, частота обнаруженных ошибок и доля случаев, которые всё ещё требовали ручной работы. Эти показатели могли улучшаться неравномерно. Система могла отлично отсеивать типичные несовместимости, но обращаться к специалисту при редких условиях; могла быть полезна отделу, но требовать заметных затрат от разработчиков и технических редакторов. Практический успех состоял не в абсолютной автономности, а в том, что общий процесс становился лучше с учётом всех его частей.

Сравнивать XCON с человеком напрямую тоже было непросто. Человек-редактор обладал контекстом, которого не было в базе правил, но мог уставать, по-разному трактовать инструкции или пропускать повторяющуюся проверку. Программа применяла записанные ограничения последовательно, зато не могла сама заметить, что каталог устарел или необычный запрос клиента лежит вне её модели. Поэтому надёжное внедрение требовало не соревнования человека и машины, а распределения работы: системе поручали повторяемые проверки, а сотрудникам оставляли исключения и контроль. При таком устройстве эффективность относилась ко всему процессу, а не к демонстрации отдельного алгоритма.

Публикации участников приводили заметные оценки экономии и полезности XCON. Эти сведения важны как свидетельства людей, которые внедряли систему и наблюдали её в работе, но их следует читать именно как оценки, сделанные внутри проекта и компании, а не как независимый финансовый аудит. Надёжнее всего виден сам факт организационного изменения:

DEC не ограничилась демонстрацией, а встроила программу в работу и выделила людей для её развития.

Именно на этом участке история XCON расходится с красивой формулой «загрузили экспертное знание — получили готового цифрового специалиста». Систему создали исследователи, но поддерживать её должна была компания, чьи продукты постоянно менялись. При появлении новых компонентов менялись комбинации, которые можно было продавать. Правило, верное для прежней версии оборудования, могло стать ошибочным после обновления. Чтобы программа не превращала старую инструкцию в автоматизированную ошибку, знания нужно было пересматривать.

DEC пришлось сформировать команду сопровождения. В начале передачи технологии многие сотрудники компании не знали в достаточной степени язык OPS-4, на котором была реализована система, и не имели опыта работы с экспертными программами. Исследователи из университета помогали коллегам в DEC освоить код и методы его изменения. Переход занял время: организация должна была вырастить не просто пользователей XCON, а специалистов, способных выяснить, почему система применила правило, где лежит нужное знание и как проверить, что новая версия не разрушила старые случаи.

Этот процесс показывает разницу между передачей программы и передачей способности её поддерживать. Файл можно скопировать за минуты. Рабочее знание о его устройстве и о границах его поведения создаётся дольше. Внутри компании нужно было договориться о процедуре тестирования: какие заказы считаются показательными, как фиксировать исключения, кто подтверждает новые правила и что делать, когда система не может надёжно выдать результат.

Работа с экспертами тоже не напоминала диктовку готовой энциклопедии. Специалист мог знать, что некоторую комбинацию деталей лучше избегать, но не всегда мог сразу сформулировать все условия. Иногда он вспоминал об исключении лишь после того, как программа давала неправильный ответ. В других случаях документация не совпадала с привычкой сотрудников. Инженеру знаний приходилось задавать вопросы, собирать примеры, предлагать формулировки, проверять их на реальных случаях и снова обращаться к специалисту.

Поэтому инженер знаний стал отдельной профессиональной ролью. Он переводил опыт предметного специалиста в формальные правила, помогал находить противоречия и строил тесты. Его работа лежала между программированием и интервьюированием: нужно было понимать систему достаточно хорошо, чтобы выразить знания, и предметную область достаточно хорошо, чтобы заметить, когда формулировка упрощает важное различие.

Здесь обнаруживалось, что формализация не удаляет человеческие решения. Команда выбирает, какие случаи включать, какие исключения считать значимыми, что называть корректным ответом и когда вмешательство человека обязательно. Программа может применять одни и те же правила последовательно; от этого сами правила не становятся нейтральными или исчерпывающими. Они отражают определённую версию работы, согласованную группой людей.

Иногда экспертная система вела себя как увеличительное стекло. Она делала заметными те элементы практики, которые прежде оставались неявными. Чтобы записать правило, нужно было уточнить, что именно сотрудник имеет в виду под «обычно», «совместимо» или «достаточно». Часто выяснялось, что разные специалисты используют один термин по-разному, а привычная процедура содержит неформальные обходные пути. Программа заставляла компанию увидеть, что рабочее знание не всегда лежит в виде готового списка в голове одного выдающегося эксперта.

В случае XCON это не отменяло практическую пользу. Наоборот, процесс уточнения правил мог улучшать знания о самой конфигурации. Но тот же процесс создавал постоянные расходы. Успешную программу нужно было обновлять вместе с ассортиментом, обучать новых

сотрудников и тестировать после изменений. Экспертная система не была покупкой «один раз и навсегда»; она напоминала инфраструктуру, которую надо обслуживать, как каталог или производственную линию.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.