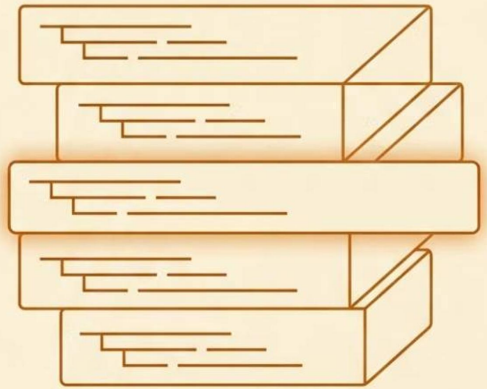
ИИ С НУЛЯ

Промпты для программиста

62 готовых запроса для нейросети



Виктория Коханистая

Виктория Коханистая

Работа с Claude - 62 готовых промта для программиста

https://litres.ru/74430223
SelfPub; 2026

Аннотация

Шестьдесят два готовых промта под работу разработчика: чтение чужого кода, отладка, тесты, написание, рефакторинг, ревью, проектирование, документация, задачи и общение с командой, собственный рост. Каждый промт — готовый текст: подставьте своё в квадратных скобках и отправьте.

У разработчика есть то, чего нет у юриста и врача: ошибку модели можно поймать автоматически. Это же и главная ловушка. Код, который запускается, — не то же самое, что код, который правильный: гонки, утечки, необработанные края и уязвимости спокойно проходят тесты на счастливом пути. А второй риск дороже первого и виден не сразу — код, который вы не понимаете, вы не сможете починить.

Поэтому в каждом промте здесь стоит требование объяснять решение, показывать альтернативы и честно называть, чего модель не знает. И ни одного «просто вставь и запусти».

Двенадцать разделов, восемь шаблонов, чек-лист перед мержем. Без привязки к версиям библиотек: они меняются быстрее, чем печатается текст.

Содержание

Виктория Коханистая
Работа с Claude - 62 готовых промта для программиста

Введение

Прочитайте это до первого промта

Разработчик — единственная профессия, где ошибку нейросети можно поймать автоматически: код либо работает, либо нет. Это делает работу с моделью безопаснее, чем у юриста или врача, — и одновременно создаёт главную ловушку.

Код, который запускается, не означает код, который правильный. Модель уверенно пишет решения с гонками, утечками, необработанными краями, неверной обработкой ошибок и уязвимостями — и всё это проходит тесты на счастливом пути. Она вызывает несуществующие методы и параметры, которых нет в вашей версии библиотеки. Она предлагает подходы, устаревшие несколько лет назад, потому что

их было много в обучающих данных.

Второй риск дороже первого и заметен не сразу: код, который вы не понимаете, вы не сможете починить. Приняв решение, которое не можете объяснить, вы взяли на себя обязательство поддерживать то, чего не понимаете. Поэтому во всех промтах здесь стоит требование объяснять, показывать альтернативы и честно называть, чего модель не знает.

И третье: то, что вы отправляете, уходит на чужой сервер. Про исходный код компании это отдельный разговор — раздел 2.

Где проходит граница

Нейросети — объём и разбор: прочитать чужой код, объяснить незнакомое, набросать тесты, сгенерировать однотипное, разобрать стек, переписать под другой стиль, написать документацию, найти краевые случаи.

Вам — решения и понимание: архитектура, выбор подхода, что попадёт в продакшн, безопасность и то, что вы готовы поддерживать.

Первое — большая часть рабочего дня. Второе — то, за что платят.

Разделы 1–3 читаются по порядку: там граница, данные и устройство промта. Дальше берите под текущую задачу.

Как устроено пособие

Каждый раздел начинается с блока «Какие проблемы решаем» — вашими словами.

Промты полные: скопировали, подставили своё в квадратных скобках, отправили.

После промтов, где легко получить правдоподобно неверное, — блок что проверить в ответе.

В конце раздела — чек-лист и задание на 10–15 минут на своём проекте.

Три честных ограничения

Первое. Пособие не сделает вас лучше как инженера. Оно снимет объём: чтение, перебор, рутину, документацию — и оставит время на то, что требует головы.

Второе. Ничего из полученного не попадает в продакшн без вашего понимания и проверки. «Работает» — не критерий приёмки.

Третье. Всё, что касается конкретных версий, API и безопасности, проверяется по документации и специалистам. Модель здесь систематически устаревает.

Раздел 1. Что поручать нейросети, а что нельзя

Граница, за которой скорость превращается в код, который вы не сможете починить.

Какие проблемы решаем в этом разделе

«Быстро получаю код, а потом два часа выясняю, почему он ведёт себя странно».

«Предлагает методы, которых нет в моей версии библиотеки».

«В команде спорят, можно ли вообще этим пользоваться, и аргументов ни у кого нет».

Почему «работает» — плохой критерий

Модель оптимизирует правдоподобие, а не корректность. Код, который она выдаёт, похож на правильный код — и в простом случае он правильный. Проблемы начинаются там, где правильность не видна с первого взгляда.

Что ломается	Почему не видно сразу
Краевые случаи: пустой ввод, ноль, отрицательные, очень большие значения	Тест на счастливом пути проходит
Обработка ошибок: проглоченные исключения, потерянный контекст	В нормальном режиме не проявляется
Конкурентность: гонки, дедлоки, неатомарные операции	Воспроизводится под нагрузкой, не на ноутбуке
Ресурсы: незакрытые соединения, утечки	Проявляется через дни работы
Безопасность: инъекции, отсутствие валидации, небезопасные значения по умолчанию	Не проявляется, пока не проявится
Несуществующие или изменившиеся методы и параметры	Ловится при запуске — это как раз лучший случай
Устаревшие подходы	Работает, но через год это ваш легаси

Модель обучена на всём коде, который существовал, — включая плохой, устаревший и небезопасный. Она воспроизводит распределение, а не лучшую практику. То, что решение встречалось часто, не значит, что оно верное; часто это значит, что его часто копировали.

Второй риск: код, который вы не понимаете

Он не про сегодня. Сегодня всё работает. Он про тот момент, через полгода, когда это упадёт в проде в три часа ночи, и чинить будете вы. Код, принятый без понимания, — это долг, который отдаётся в самый неудобный момент.

Практическое правило простое: если вы не можете объяснить коллеге, почему этот код работает и что будет, если входные данные окажутся другими, — этот код в проект не идёт. Разобраться можно тут же, промтами из раздела 4, — это занимает минуты.

Четыре зоны работы разработчика

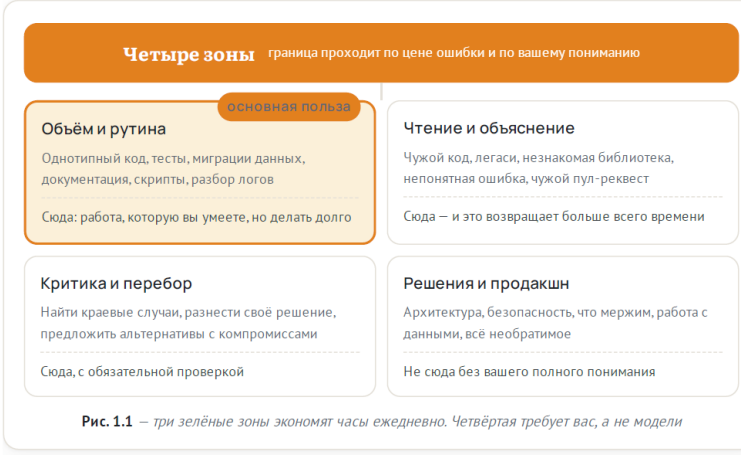


Рис. 1.1 — *три зелёные зоны экономят часы ежедневно. Четвёртая требует вас, а не модели*

Что реально экономит время

Задача	Как обычно	С нейросетью
Разобраться в незнакомом модуле	Полдня	Час
Понять стек ошибки из чужой библиотеки	40 минут поиска	10 минут
Написать тесты на существующий код	Откладывается	Час на модуль
Найти краевые случаи в своей функции	Находятся в проде	10 минут
Однотипный код: маппинги, DTO, обвязка	Час скуки	10 минут
Документация к модулю	Не пишется	30 минут
Разбор чужого пул-реквеста	Час	20 минут

Четыре запрета

Не поручайте

Код, который вы не понимаете, — в продакшн. Не потому что он плох, а потому что чинить его вам.

Решения по безопасности. Криптография, аутентификация, хранение секретов, права доступа — здесь модель уверенно предлагает устаревшее и небезопасное. Только документация, стандарты и специалисты.

Точные сведения об API и версиях. Сигнатуры, параметры, поведение — только официальная документация вашей версии. Модель здесь ошибается систематически.

Необратимые операции с данными. Миграции, удаления, скрипты в проде — сгенерированное проверяется на копии и читается построчно.

Самая недооценённая зона

Дайте модели свою функцию и попросите перечислить входные данные, на которых она сломается. Это занимает три минуты и находит то, что иначе нашлось бы в проде. Раздел 6 целиком об этом.

Чему вы научились

Понимать, почему «работает» не значит «верно»

Знать список того, что ломается незаметно

Не принимать код, который не можете объяснить

Различать три рабочие зоны и одну запретную

В следующем разделе: код и данные компании — что не уходит в чат.

Чек-лист раздела 1

Выписал задачи недели по зонам
Запомнил четыре запрета
Правило «не понимаю — не мержу» принято

Задания

Проверьте на честность. Спросите про метод библиотеки, которую хорошо знаете, и сверьте сигнатуру с документацией.

Дайте свою функцию и попросите перечислить, на каких входных данных она сломается.

Раздел 2. Код и данные компании

Что не уходит в чат — и как работать почти со всем остальным.

Какие проблемы решаем в этом разделе

«Вставил кусок кода, а там оказался ключ в конфиге».

«В компании запрет, но никто не объяснил, что именно нельзя».

«Хочу разобрать баг, а в логах данные пользователей».

Три категории риска

Секреты. Ключи, токены, пароли, строки подключения, приватные ключи, внутренние адреса. Попадают в чат случайно — вместе с куском конфига или лога. Правило: секрет, побывавший в чате, считается скомпрометированным и подлежит ротации. Это не паранойя, а стандартная процедура.

Исходный код компании. Здесь всё зависит от политики вашей организации и от сервиса. У части компаний есть корпоративные тарифы с договорными гарантиями, у части — запрет, у части — ничего не написано. Выясните, что у вас; «никто не запрещал» не равно «разрешено».

Персональные и продуктовые данные. Данные пользователей в логах и дампах, содержимое базы, внутренняя аналитика. Это не ваши данные.

Не загружайте

Файлы конфигурации, переменные окружения, любые секреты и токены

Логи и стеки без очистки — там регулярно оказываются данные пользователей и внутренние адреса

Дампы и выгрузки из баз

Исходный код, если политика компании этого не разрешает

Код и материалы под NDA — заказчика или партнёра

Внутренние схемы инфраструктуры, адреса сервисов, топологию сети

Данные о найденных уязвимостях до их закрытия

Персональные данные в любом виде, включая тестовые наборы, собранные из реальных

Что почти всегда можно

Абстрагированный фрагмент: та же логика на выдуманных именах и без бизнес-контекста. Публичные библиотеки и их код. Собственные пет-проекты. Сообщения об ошибках

после очистки. Общие вопросы о языке, алгоритмах и подходах.

Ключевой приём: воспроизвести проблему на минимальном примере. Это и безопаснее, и работает лучше — модель отвечает точнее на короткий изолированный вопрос, чем на тысячу строк вашего проекта. Заодно вы половину багов находите сами, пока сокращаете пример.

Промт 1. Абстрагировать фрагмент

Ты помогаешь подготовить фрагмент кода к обсуждению вне компании.

Код: [вставьте].

Задача: перепиши так, чтобы сохранилась только техническая суть.

Правила: имена доменных сущностей замени на нейтральные — Entity, Item, Record; названия внутренних сервисов и систем — ServiceA; поля с бизнес-смыслом — на абстрактные; строки, адреса, идентификаторы — на placeholder; комментарии с бизнес-логикой удали, отметив, что там было по смыслу.

Ограничения: структура, порядок вызовов, типы и логика должны сохраниться точно — иначе обсуждать нечего; ничего не упрощай и не «улучшай»; если встретишь что-то похожее на секрет, ключ, токен или пароль, не переноси это никуда и вынеси отдельным предупреждением; отметь места,

где абстрагирование потеряло важный контекст.

Формат: абстрагированный код, «найдено похожее на секреты», «потерян контекст».

Что проверить в ответе

Прочитайте абстрагированный код сами — модель может пропустить внутреннее имя или адрес.

Если найдено что-то похожее на секрет — считайте его скомпрометированным и ротируйте, даже если вы «почти уверены», что не отправляли.

Промт 2. Очистить лог или стек

Ты готовишь лог к обсуждению, убирая всё чувствительное.

Лог: [вставьте].

Задача: очисти.

Правила: удали токены, ключи, заголовки авторизации, строки подключения, пароли; идентификаторы пользователей и любые персональные данные замени на placeholder; внутренние адреса, имена хостов и путей — на нейтральные; сохранить: типы ошибок, сообщения библиотек, порядок событий, временные интервалы, стек вызовов.

Ограничения: не сокращай стек и не выбрасывай строки,

которые кажутся неважными; если сомневаешься, чувствительное ли значение, — удали и отметь; отдельно перечисли всё, что похоже на секрет, чтобы я знала, что ротировать.

Формат: очищенный лог, «похоже на секреты — ротировать», «удалено на всякий случай».

Промт 3. Минимальный воспроизводящий пример

Ты помогаешь свести проблему к минимальному примеру.

Что происходит: [опишите поведение]. Что ожидается: [что]. Окружение: [язык, версия, библиотеки с версиями]. Код: [абстрагированный фрагмент].

Задача: предложи, как сократить это до минимального воспроизводящего примера.

Ограничения: предлагай сокращать по одному элементу за раз и объясняй, что это проверяет; начинай с того, что вероятнее всего не связано с проблемой; на каждом шаге указывай, как понять, воспроизводится ли ещё; не переписывай код целиком; отметь, какие зависимости, скорее всего, можно заменить заглушкой; если из описания видно, что проблема, вероятно, не в этом коде, — скажи это сразу и предложи, где искать.

Формат: шаги сокращения — что убираем, что это проверит. Ниже — «возможно, проблема не здесь».

Промт 4. Правила для команды

Ты помогаешь написать правила использования нейросетей в разработке.

Что делаем с их помощью: [перечислите]. Какие данные и код у нас есть: [категории]. Что уже вызывало вопросы: [если было]. Какой тариф или сервис используем: [если известно].

Задача: напиши правила.

Ограничения: не формулируй правовых требований и не ссылайся на нормы — оставь [место для юриста и безопасности] со списком вопросов к ним; под каждым запретом — разрешённая альтернатива; обязательные пункты: код в продакшн идёт только понятый автором; секрет, побывавший в чате, ротируется; вопросы безопасности решаются по документации, а не в чате; сгенерированный код проходит ревью на общих основаниях; обязательный пункт про минимальный пример вместо загрузки проекта; не больше полутора страниц; без деклараций.

Формат: правила с альтернативами, ниже — «вопросы к юристу и безопасности».

Чему вы научились

Различать три категории риска: секреты, код компании,

данные

Абстрагировать фрагмент и очищать логи

Сводить проблему к минимальному примеру — безопаснее и эффективнее

Знать, что делать, если секрет всё же попал в чат

В следующем разделе: устройство промта разработчика.

Чек-лист раздела 2

Политика компании выяснена, а не додумана

Логи очищаются перед вставкой

Работаю с минимальным примером, а не с проектом

Задания

Выясните, что у вас разрешено. Пять минут разговора с руководителем или безопасностью снимают вопрос навсегда.

Сведите текущий баг к минимальному примеру промтом 3. Половина багов находится в процессе.

Раздел 3. Как устроен промт разработчика

Почему приходит устаревшее решение и какие ограничения это чинят.

Какие проблемы решаем в этом разделе

«Предлагает подход, который в нашем проекте не применяется уже три года».

«Даёт код, не спросив ни версии, ни контекста».

«Каждый раз объясняю стек, стиль и ограничения заново».

Три проблемы и как их чинить

Устаревание. Модель воспроизводит то, чего в обучающих данных было много, — а много было того, что писали годами. Новое API появилось недавно, старое обсуждали десять лет. Чинится указанием версий и требованием отмечать, в чём модель не уверена.

Отсутствие контекста. Она не знает вашего проекта: соглашений, слоёв, того, что уже есть в утилитах, чего нельзя добавлять в зависимости. Чинится постоянным контекстом.

Уверенность. Модель не говорит «не знаю» по своей инициативе. Она даёт ответ. Чинится прямым требованием: отметить, что проверить и в чём не уверена.

Самый полезный элемент промта разработчика — просьба перечислить, что в ответе надо проверить по документации. Модель довольно честно отмечает шаткие места, если попросить прямо, — но никогда не делает этого сама.

Промт 5. Блок ограничений разработчика

Ограничения:

— окружение: [язык и версия], [фреймворк и версия], [ключевые библиотеки с версиями]; не используй возможностей, появившихся позже, и не предлагай API, которого нет в этих версиях;

— не выдумывай методов, параметров и полей — если не уверена, что метод существует в указанной версии, отметь это явно;

— в конце отдельным списком: что проверить по документации и в чём ты не уверена;

— не добавляй новых зависимостей; если решение требует библиотеки, скажи это отдельно и предложи вариант без неё;

— объясняй, почему выбран этот подход, и назови альтернативу с её компромиссом;

— не пиши код без обработки ошибок и краевых случаев; отдельно перечисли, какие случаи не обработаны и почему;

— соблюдай стиль моего кода из приложенного примера, а не свой;

— не используй сокращённых имён и «умных» конструкций ради краткости — код должен читаться;

— не оптимизируй без моей просьбы; если видишь проблему производительности, отметь отдельно;

— в комментариях объясняй «почему», а не «что».

Промт 6. Постоянный контекст проекта

Контекст проекта, учитывай во всех ответах:

Стек: [язык и версия, фреймворк и версия, база, ключевые библиотеки с версиями].

Как устроен проект: [слои, где что лежит, как называются вещи].

Что уже есть и надо переиспользовать: [утилиты, хелперы, обёртки].

Чего нельзя: [новые зависимости, определённые подходы, что запрещено соглашениями].

Стиль: [приложите характерный файл как образец].

Как мы обрабатываем ошибки: [опишите принятый подход].

Как мы пишем тесты: [фреймворк, стиль, что принято проверять].

Всегда: не выдумывай API — отмечай, что проверить; альтернативу с компромиссом называй; краевые случаи перечисляй; стиль берёшь из образца; в конце — «проверить по документации».

Промт 7. Шесть формулировок второго хода

Перечисли всё, что в этом ответе нужно проверить по документации, и где ты не уверена.

Ты использовала API, которого может не быть в моей версии. Покажи такие места списком.

Это решение сложнее, чем нужно. Дай простой вариант и скажи, чем он хуже.

Какие входные данные сломают этот код? Перечисли, не переписывая его.

Объясни, почему ты выбрала этот подход, и назови две альтернативы с их минусами.

Перепиши в стиле приложенного файла: именование, структура, обработка ошибок.

Промт 8. Что проверить в этом ответе

Ты проверяешь собственный предыдущий ответ.

Задача: составь список того, что мне нужно проверить перед использованием.

Ограничения: перечисли все использованные методы, параметры и поведение библиотек, которые я должна сверить с документацией моей версии; перечисли предположения, которые ты сделала о моём коде и окружении; перечисли краевые случаи, которые код не обрабатывает; перечисли места, где поведение зависит от версии; честно отметь, в чём ты не уверена, — это важнее полноты; не переписывай ответ; не оправдывайся.

Формат: «сверить с документацией», «предположения», «не обработано», «зависит от версии», «не уверена».

Промт 9. Критика моего промта

Ты — старший разработчик, читающий постановку задачи перед тем, как её взять.

Мой запрос: [вставьте].

Задача: скажи, чего в нём не хватает.

Ограничения: сам запрос не выполняй; перечисли, что исполнитель будет вынужден предположить: версии, объёмы данных, поведение при ошибках, требования к производительности, кто вызывает этот код и что делает с результатом; отметь места, где формулировка допускает два разных решения; предложи вопросы, которые надо закрыть до начала; не переписывай запрос.

Формат: список — что придётся предположить, какой вопрос закрыть.

Чему вы научились

Указывать версии и запрещать API, которого в них нет

Требовать список «что проверить по документации» в каждом ответе

Держать постоянный контекст проекта со стилем и соглашениями

Просить альтернативу с компромиссом вместо единственного решения

В следующем разделе: чтение чужого кода.

Чек-лист раздела 3

Контекст проекта записан и вставляется в начало чата

Версии указаны во всех запросах о коде

Список «что проверить» запрашивается всегда и читается

Задания

Соберите постоянный контекст проекта. Он экономит по три сообщения в каждом чате.

Прогоните промт 8 после ответа, который вам понравился. Список «не уверена» обычно длиннее ожидаемого.

Раздел 4. Чтение чужого кода

Раздел, который возвращает больше всего времени: разобраться быстрее, чем гадать.

Какие проблемы решаем в этом разделе

«Пришёл в проект, где полмиллиона строк и никакой документации».

«Надо поправить функцию, которую написали пять лет назад и никто не помнит зачем».

«Читаю чужой код час и всё ещё не понимаю, что он делает».

Почему это самый выгодный раздел

Здесь модель работает с приложенным кодом, а не выдумывает. Риск ошибки минимальный, польза максимальная: чтение чужого кода — самая медленная и самая частая часть работы. Единственное требование — не верить объяснению на слово там, где от него зависит решение: проверяйте по самому коду.

Промт 10. Что делает этот код

Ты объясняешь незнакомый код.

Код: [вставьте абстрагированно]. Контекст, если известен: [что за проект, где это вызывается].

Задача: объясни, что он делает.

Ограничения: объясняй по порядку выполнения, а не по порядку строк; отдельно опиши, что на входе, что на выходе и какие побочные эффекты — запись в базу, файлы, сеть, изменение переданных объектов; отметь неочевидные места и объясни их отдельно; отметь, что ты предполагаешь, а что видно из кода, — это разные вещи; отметь, чего нельзя понять без остального проекта, и что для этого нужно посмотреть; не оценивай качество кода и не предлагай улучшений; не переписывай.

Формат: что делает по шагам, вход/выход/побочные эффекты, «неочевидное», «предположения», «нужно посмотреть ещё».

Что проверить в ответе

Раздел «предположения» — там модель достраивает то, чего не видит. Проверьте по коду.

Побочные эффекты — самое важное и самое часто про-

пускаемое при чтении вручную.

Промт 11. Зачем это здесь

Ты помогаешь понять назначение непонятного участка кода.

Код: [вставьте]. Окружающий контекст: [приложите вызывающий код или соседние функции]. История, если известна: [коммиты, комментарии].

Задача: предложи гипотезы, зачем это написано.

Ограничения: перечисли гипотезы, а не одну версию, и пометь их как гипотезы; для каждой напиши, что подтвердило бы её — какой тест, какой поиск по репозиторию, какой коммит; отдельно рассмотри вариант, что это обход давнего бага или требования, которого больше нет; отдельно рассмотри вариант, что это мёртвый код; скажи, что произойдёт, если это просто удалить, — по коду, а не по догадкам; не советуй удалять.

Конец ознакомительного фрагмента.