

Учитель Начальной

Курс: Нейросети для реверс-инжиниринга

Учитель Начальной

**Курс: Нейросети для
реверс-инжиниринга**

«Автор»

2026

Начальной У. Ш.

Курс: Нейросети для реверс-инжиниринга / У. Ш. Начальной —
«Автор», 2026

Учебное пособие посвящено применению нейросетей и больших языковых моделей в реверс-инжиниринге. В 20 главах последовательно рассматриваются основы машинного обучения для анализа бинарников, представления кода (байты, opcodes, CFG, IR), классические модели, Graph Neural Networks, трансформеры, восстановление границ функций, имён и типов, нейронная и LLM-декомпиляция, binary similarity, анализ malware и firmware, интеграция с Ghidra, radare2 и Binary Ninja, fine-tuning, агентные системы и этические ограничения. Материал сочетает теорию с практическими примерами, пайплайнами и актуальными инструментами 2025–2026 годов. Курс рассчитан на специалистов по безопасности, reverse engineers и исследователей, знакомых с Python и базовым ML. Пособие помогает перейти от ручного анализа к масштабируемому AI-assisted workflow и самостоятельно строить современные системы анализа бинарного кода.

© Начальной У. Ш., 2026

© Автор, 2026

Содержание

Введение	5
Глава 1. Введение в реверс-инжиниринг и роль нейросетей	6
1.1. Что такое реверс-инжиниринг	6
Классический пайплайн RE	6
1.2. Проблемы классического подхода	7
1.3. Как нейросети меняют RE	8
1.4. Пример: от ручного анализа к AI-assisted	9
Глава 2. Основы машинного обучения для анализа бинарников	10
2.1. Типы обучения	10
2.2. Основные задачи RE с точки зрения ML	11
2.3. Метрики	12
2.4. Подводные камни	13
2.5. Инструменты	14
Глава 3. Представление бинарного кода как данных	15
3.1. Низкоуровневые представления	15
3.2. Структурные представления	16
3.3. Canonicalization	17
3.4. Tokenization для трансформеров	18
3.5. Практический пример извлечения признаков	19
Глава 4. Классические ML-модели в RE	20
4.1. Feature engineering	20
Конец ознакомительного фрагмента.	21

Учитель Начальной

Курс: Нейросети для реверс-инжиниринга

Введение

Реверс-инжиниринг (RE) бинарных программ традиционно опирается на ручной анализ, эвристики и инструменты вроде IDA Pro, Ghidra, Binary Ninja и radare2. С появлением глубокого обучения и больших языковых моделей (LLM) процесс радикально меняется: нейронные сети помогают восстанавливать семантику, имена функций, типы данных, декомпилировать код и автоматизировать анализ malware и firmware.

Курс рассчитан на специалистов, знакомых с основами реверс-инжиниринга, Python и базовым машинным обучением. Цель - дать практическую картину: от извлечения признаков из бинарников до fine-tuning LLM и построения агентных систем.

Материал охватывает современное состояние области по состоянию на 2025-2026 годы: от классических ML-моделей до LLM4Decompile, Ventris, Graph Neural Networks, агентных пайплайнов с Neo4j и NL2GQL, а также интеграции с Ghidra, radare2 и Binary Ninja.

Каждая глава самодостаточна, но рекомендуется последовательное изучение: главы 1-7 дают фундамент, 8-15 раскрывают ключевые задачи, 16-20 посвящены интеграции, практике и перспективам. В расширенной версии добавлены практические примеры, описания инструментов, кейсы из реальных исследований и рекомендации по реализации.

Глава 1. Введение в реверс-инжиниринг и роль нейросетей

1.1. Что такое реверс-инжиниринг

Реверс-инжиниринг - процесс извлечения знаний из скомпилированного кода без исходников. Основные задачи: восстановление алгоритмов, поиск уязвимостей, анализ malware, firmware и протоколов, аудит стороннего ПО, восстановление потерянных исходников.

Типичный сценарий: аналитик получает исполняемый файл (PE, ELF, Mach-O) или прошивку, не имея исходного кода. Задача - понять, что делает программа, найти критические участки (криптография, сетевое взаимодействие, антиотладка) и либо задокументировать поведение, либо найти уязвимость.

Классический пайплайн RE

Triage - определение типа файла, архитектуры, наличия packer/protector, расчёт entropy секций.

Static analysis - дизассемблирование, построение CFG и call graph, декомпиляция, поиск строк и импортов.

Dynamic analysis - отладка (x64dbg, GDB, WinDbg), трассировка API, memory dumps, sandbox.

Semantic recovery - восстановление алгоритмов, структур данных, протоколов, написание отчёта.

1.2. Проблемы классического подхода

При компиляции теряется значительная часть информации: имена переменных и функций, типы данных, высокоуровневые структуры (классы C++, циклы в исходном виде, комментарии). Обфускация (VMProtect, Themida, кастомные виртуальные машины), упаковка (UPX, кастомные packers), оптимизация компилятора (O1-O3, LTO) и антиотладочные техники делают анализ крайне трудоёмким.

Масштаб современных бинарников (сотни тысяч функций в крупных приложениях и драйверах) делает полностью ручной анализ непрактичным. Аналитик вынужден фокусироваться только на интересных участках, рискуя пропустить важные детали.

1.3. Как нейросети меняют RE

Нейронные сети решают задачи классификации, сходства, восстановления и генерации.
Эволюция подходов:

-

2018-2020: RNN для function boundary detection и простых NMT-декомпиляторов.

-

2021-2023: Graph Neural Networks, BERT-подобные модели для type recovery и naming, первые серьёзные neural decompilers (Neutron, Coda, NeurDP).

-

2024-2026: LLM (LLM4Decompile, Ventris), agentic systems, hybrid symbolic+neural подходы, массовая интеграция с Ghidra/IDA.

Ключевые направления применения сегодня: binary classification, function similarity, type recovery, function naming, neural decompilation, malware detection и clustering, summarization, YARA generation, agentic automation.

1.4. Пример: от ручного анализа к AI-assisted

Раньше: аналитик открывает образец ransomware в IDA, ищет циклы шифрования вручную по характерным инструкциям (AES S-box, XOR-паттерны), тратит часы. Сейчас: специализированная нейросеть (см. главу 14) автоматически помечает encryption loops с точностью более 95%, после чего LLM генерирует краткое описание алгоритма и предлагает YARA-правило.

Глава 2. Основы машинного обучения для анализа бинарников

2.1. Типы обучения

Supervised learning: использует размеченные данные (пара бинарник - метка). Пример: 10000 samples с метками семейств malware - Random Forest или нейросеть предсказывает семейство нового файла.

Unsupervised learning: ищет структуры без меток. Кластеризация embeddings функций позволяет группировать похожие образцы malware даже без известных меток.

Self-supervised learning: Masked Language Modeling на больших корпусах assembly позволяет предобучать трансформеры без ручной разметки. Модель учится понимать язык инструкций.

2.2. Основные задачи RE с точки зрения ML

- Binary classification - malicious/benign или семейство malware.
- Function similarity / clone detection - поиск известного кода в новом бинарнике.
- Type recovery - предсказание типов переменных и параметров.
- Function naming и summarization - генерация имён и описаний.
- Neural decompilation - перевод assembly/P-Code в C-подобный код.
- Anomaly detection - поиск необычного поведения в firmware.

2.3. Метрики

Для классификации: accuracy, precision, recall, F1-score, ROC-AUC. Для декомпиляции критична re-executability rate (можно ли скомпилировать и выполнить восстановленный код с тем же поведением). Для embeddings - cosine similarity и retrieval metrics (Recall@K).

Важно измерять качество не только на удобных данных (OO, debug-символы), но и на stripped + O2/O3 бинарниках - именно они встречаются в реальной работе.

2.4. Подводные камни

- Data leakage: если одна и та же функция в разных компиляциях попадает и в train, и в test - метрики завышены.
- Domain shift: модель, обученная на GCC x86, может сильно деградировать на MSVC ARM64.
- Class imbalance: в malware-датасетах часто 95% benign - accuracy 95% ничего не значит.
- Overfitting на артефакты компилятора вместо семантики.

2.5. Инструменты

scikit-learn - быстрый baseline. PyTorch / TensorFlow - deep learning. Hugging Face Transformers - готовые трансформеры и LLM. Для бинарников: LIEF, pefile, capstone, Ghidra headless, Binary Ninja API.

Глава 3. Представление бинарного кода как данных

3.1. Низкоуровневые представления

Сырые байты и n-граммы: простейший способ. Модель видит последовательность байтов файла или секции .text. Хорошо работает для packer detection и простых классификаторов, но теряет структуру.

Opcode sequences: дизассемблер (Capstone, Ghidra) превращает байты в мнемоники (mov, call, xor). Последовательности opcodes сохраняют больше семантики и меньше зависят от конкретных адресов.

Intermediate Representations: P-Code (Ghidra), LLVM IR, Binary Ninja IL, ESIL (radare2). IR абстрагируется от архитектуры - одна модель может работать с x86, ARM и MIPS. Это критически важно для cross-architecture analysis.

3.2. Структурные представления

Control Flow Graph (CFG) - вершины = базовые блоки, рёбра = переходы. Data Flow Graph (DFG) показывает зависимости данных. Call graph - кто кого вызывает. Эти графы идеально подходят для Graph Neural Networks.

Дополнительные сильные признаки: строки (особенно URL, пути, сообщения об ошибках), таблицы импортов (какие API вызывает программа), экспорты, ресурсы, TLS-callbacks.

3.3. Canonicalization

Без нормализации модель переобучается на абсолютные адреса и конкретные регистры.
Типичные шаги:

- Замена адресов на символические метки (ADDR, IMM).
- Нормализация регистров (eax/rax -> REG0) или сохранение только классов регистров.
- Приведение констант к категориям (маленькие числа, указатели, магические константы кривые).
- Удаление NOP и выравнивающих инструкций.

3.4. Tokenization для трансформеров

Assembly рассматривается как язык. Нужен vocabulary: мнемоники, регистры, специальные токены. Пример токенизации короткой функции:

push REG | mov REG, IMM | call ADDR | add REG, REG | ret

BPE или WordPiece можно обучить на большом корпусе assembly. Важно не разбивать мнемоники на части - это снижает качество.

3.5. Практический пример извлечения признаков

С помощью LIEF + Capstone можно за минуту извлечь: список секций и их entropy, импорты, экспорты, первые N инструкций entrypoint, строки длиннее 4 символов. Эти признаки уже достаточны для baseline malware-классификатора.

Глава 4. Классические ML-модели в RE

4.1. Feature engineering

Классические признаки, которые хорошо работают годами:

- Entropy каждой секции (высокая entropy - возможно упаковка или шифрование).
- Размеры секций и их соотношения.
- Количество и типы импортов (наличие VirtualAlloc + WriteProcessMemory + CreateRemoteThread - injection).
- Метрики CFG: число базовых блоков, цикломатическая сложность, глубина вложенности.

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.