ChatGPT

# 1.1 Новый компилятор: LLM как уровень абстракции

## Исторический контекст

Классическое программирование (Software 1.0) строится на человеческом коде: разработчики пишут инструкции, которые затем компилируются в исполняемые файлы. Андрей Карпаты в эссе «Software 2.0» отмечает, что этот подход сменяет **Software 2.0** — программирование с помощью нейронных сетей [1]. В новой парадигме “исходный код” состоит из **датасета** и **архитектуры модели**. **Обучение** превращает этот «исходник» в готовый нейронный «бинарник» — набор весов, который исполняет заданное поведение. Большая часть работы сдвигается к сбору и маркировке данных, а разработчики поддерживают инфраструктуру обучения [1]. Эта смена парадигмы объясняет, почему сегодня проще собирать данные, чем писать программу, и почему большое количество существующих задач (например, распознавание речи, визуальное распознавание, машинный перевод и даже база данных) уже заменяется нейронными алгоритмами [2].

## Архитектурная основа: Transformer

Быстрый «взрыв» генеративных моделей стал возможен благодаря **архитектуре Transformer**, предложенной в 2017 г. Васвани и коллегами. Авторы отказались от рекуррентных и сверточных сетей в пользу **механизма внимания**; этот архитектурный шаг позволил параллелизировать обучение и добиться более высокой точности на машинном переводе [3]. Transformer показывал лучшие показатели BLEU на задачах перевода и хорошо обобщался на другие лингвистические задачи [3]. Именно он стал основой современных **Large Language Models (LLMs)**, позволив масштабировать параметрическое пространство до сотен миллиардов весов.

## LLM как «компилятор намерений» и Software 3.0

Карпаты в выступлении на AI Startup School (Y Combinator) проводит параллель между историей программирования и современной волной генеративного ИИ. В главе «Software evolution: From 1.0 to 3.0» он показывает дугу: **Software 1.0** — ручной код, **Software 2.0** — обучение нейросетей, **Software 3.0** — программирование намерениями. Программирование на английском языке становится реальностью: LLM преобразует высокоуровневое описание задачи в код, документацию или тесты. В разделе «Programming in English: Rise of Software 3.0» он утверждает, что мы находимся на заре эры, когда **спецификация на естественном языке** станет основным интерфейсом разработки [4].

Карпаты описывает LLM как новый *компилятор намерений*: пользователь формулирует проблему (например, «создать REST-API для управления задачами») — модель генерирует код, миграции баз данных, документацию и тесты. Лингвистическая спецификация заменяет низкоуровневые инструкции, подобно тому как обычный компилятор переводит код в машинные команды. Такой уровень абстракции превращает **LLM в инструмент проектирования**, а программиста — в архитектора, который формулирует требования и курирует результат. Переход от **Software 2.0** к **3.0** означает, что управление смещается от написания алгоритмов к **управлению контекстом и**

**данными**: важно снабжать модель правильными примерами и контролировать процесс обучения, а не писать каждую строку вручную [1].

## Последствия для разработчиков

- **Улучшение продуктивности.** Вместо того чтобы тратить время на шаблонный код, разработчики могут описывать задачу и получать прототип, фокусируясь на архитектуре и контроле качества. Это ускоряет создание MVP и позволяет быстрее тестировать гипотезы.
- **Новые роли и навыки.** Как пишет Карпаты, команды разделяются на специалистов, которые **курируют данные**, и инженеров, поддерживающих инфраструктуру обучения [1]. Для Software 3.0 важно уметь формулировать требования, подготавливать примеры, оценивать выводы модели и исправлять недочеты. Классическое программирование не исчезает: инженеры всё ещё пишут обвязку, инструменты тестирования и адаптируют модель под домен.
- **Лимиты и ответственность.** LLM предсказывает токены, а не исполняет детерминированные алгоритмы. Точность вывода зависит от качества данных и подсказок; модели склонны к галлюцинациям. Поэтому использовать LLM как компилятор нужно осторожно: проверять сгенерированный код, применять формальные проверки и контролировать безопасность.

## Пример сценария «Новый компилятор» (приблизительно 5 минут)

1. **С чего всё началось** (≈1 мин). Расскажите, что в Software 1.0 программисты писали каждую инструкцию. Затем появилась **Software 2.0**: нейронные сети, где “исходный код” — данные и архитектура; процесс обучения компилирует их в модель [1]. Благодаря этому подходу многие области (компьютерное зрение, распознавание речи, перевод, игры) были радикально улучшены [2].
2. **Роль Transformer** (≈1 мин). Объясните, что в 2017 году появился **Transformer**, отказавшийся от рекуррентных сетей и позволивший параллельно обучать модели. В машинном переводе он показал лучшую точность и скорость обучения [3], а затем стал фундаментом LLM.
3. **Появление Software 3.0** (≈2 мин). Кратко перескажите тезисы Карпаты: эволюция 1.0 → 2.0 → 3.0; программирование на английском; LLM как компилятор намерений. Приведите пример: задать задачу «сгенерируй API и документацию» и получить рабочий прототип. Укажите, что в Software 3.0 разработчик превращается в куратора: управляет контекстом, проверяет результаты, дообучает модель на специфических примерах [4].
4. **Вызовы и ограничения** (≈1 мин). Напомните, что LLM — это вероятностная модель. Она может ошибаться, генерировать небезопасный код или халлюцинировать. Поэтому нужен человек-в-контуре: проверка, тесты, инструменты. Отметьте, что дисциплины вроде системного программирования, алгоритмов и безопасности остаются востребованными.

## Заключение

Переход к Software 3.0 — **переход от программирования к управлению намерениями**. Лингвистические спецификации и LLM, основанные на Transformer, становятся новым «компилятором» человеческих замыслов. От нас требуется научиться формулировать задачи, собирать правильные данные и критически анализировать выходы модели. Только так можно раскрыть потенциал нового уровня абстракции и сделать ИИ настоящим помощником разработчиков.

1 2 Software 2.0. I sometimes see people refer to neural... | by Andrej Karpathy | Medium
https://karpathy.medium.com/software-2-0-a64152b37c35

3 [1706.03762] Attention Is All You Need
https://arxiv.org/abs/1706.03762

4 Andrej Karpathy: Software Is Changing (Again) : YC Startup Library | Y Combinator
https://www.ycombinator.com/library/MW-andrej-karpathy-software-is-changing-again