Областной конкурс работ исследовательского характера (конференция)
учащихся по учебному предмету
«ИНФОРМАТИКА»

MIND 4

Данная работа предоставляется исключительно для ознакомления. Категорически запрещается использовать этот материал для выдачи за собственный результат или представлять его в качестве выполненной работы по информатике.

Автор:
[redacted]
учащийся X класса
[redacted]

Руководитель работы:
[redacted]

[redacted]

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Искусственный интеллект —– система обработки информации, выполняющая когнитивные задачи, ранее доступные только человеку: анализ данных, выявление закономерностей, адаптация к изменяющимся условиям, генерация новых решений. Модели ИИ строятся на архитектурах машинного обучения, формирующих внутренние представления о мире на основе больших массивов данных. Они обучаются на текстах, изображениях, звуках и других типах информации, выделяют ключевые взаимосвязи и преобразуют их в структурированные ответы, прогнозы или творческие результаты.

Развитие искусственного интеллекта сегодня является одним из ключевых направлений мировой технологической эволюции. Однако большинство современных языковых моделей создаются крупными корпорациями и остаются закрытыми, что ограничивает возможность их изучения и адаптации под образовательные и исследовательские цели.

Суть Mind 4 заключается в попытке его разработки в условиях ограниченных ресурсов, без привлечения крупных вычислительных кластеров и готовых решений от индустриальных гигантов.

Цель проекта: разработка минимально работоспособной языковой модели искусственного интеллекта в условиях ограниченных вычислительных и финансовых ресурсов, без использования крупных кластеров или готовых промышленных решений.

Для достижения цели необходимо решить следующие задачи:

1. Изучить теоретические основы современных языковых моделей, включая архитектуру Transformer, процессы токенизации, эмбеддингов и механизмов внимания.
2. Проанализировать ключевые ограничения разработки ИИ в условиях дефицита ресурсов (качество данных, вычислительные затраты, юридические риски) и предложить стратегии их минимизации.
3. Реализовать прототип модели Mind 4 на базе Google Colab, адаптировав трансформер-архитектуру для локальных вычислений с использованием оптимизаций (квантование, кэширование, Triton-ядра).
4. Провести обучение модели на открытых датасетах и оценить её производительность по стандартным метрикам.
5. Выявить проблемы, такие как галлюцинации и нестабильность обучения, и предложить пути их решения в рамках ограниченного бюджета.
6. Разработать рекомендации по дальнейшему развитию модели, включая интеграцию веб-поиска, генерации изображений и мультимодальности.

Практическая значимость данного исследования заключается в том, что позволит продемонстрировать возможность демократизации ИИ для разработчиков в образовательных целях.

Объектом исследования выступают системы искусственного интеллекта на основе глубокого обучения, в частности, большие языковые модели (Large Language Models, LLM), такие как GPT, Claude и DeepSeek [3].

Предметом исследования является процесс проектирования и реализации

компактной языковой модели Mind 4 на архитектуре Transformer в условиях ограниченных ресурсов.

Гипотезы проекта заключается в том, способен ли отдельный человек в 2025 году спроектировать архитектуру, близкую по принципам работы к системам мирового уровня и создать обученную модель.

Актуальность данного проект заключается в проверке гипотезы о том, возможна ли в принципе разработка архитектуры, близкой к системам мирового уровня, силами одного человека в 2025 году. Исследование анализирует реальные экономические и технические барьеры, помогая понять, могут ли индивидуальные разработчики вносить вклад в развитие ИИ без доступа к миллиардным бюджетам и промышленным вычислительным кластерам.

Новизна заключается в детальной инженерной реконструкции трансформер-архитектуры, адаптированной под компактное окружение и локальные вычислительные возможности.

ГЛАВА1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Как работает нейросеть

Нейросеть — это огромный цифровой мозг, который учится понимать и генерировать текст, имитируя человеческий разум. Она построена на архитектуре Transformer — это четкая система из математических шагов, где ваш вопрос разбирается на части, анализируется в контексте и собирается в конкретный ответ. Представьте, что это фабрика: сырье (ваш текст) поступает на вход, проходит несколько этапов обработки (как конвейерные линии), и на выходе — готовый продукт (ответ). Я разберу всё по шагам, с простыми примерами, чтобы было понятно. Мы пройдем от запроса до генерации ответа, включая, как модель обучается и почему иногда "галлюцинирует" (выдумывает). Всё это основано на миллиарде параметров — это как связи между нейронами в мозге, — которые настраиваются на огромных объемах данных [2].

Как модель генерирует ответ:

1. Вход: текст пользователя → токены → эмбеддинги + позиция.
2. Прогон через L слоёв Transformer (в каждом — attention + небольшая сеть обработки).
3. На выходе получаем логиты — оценки для каждого токена словаря (на сколько вероятно каждое слово).
4. Применяем softmax → получаем распределение вероятностей.
5. По правилу выборки (argmax, сэмплинг, температура) выбираем следующий токен.
6. Повторяем, пока не сгенерируем конец.

Почему это важно? Без такой “фабрики” компьютер мог бы только хранить факты, как энциклопедия. А нейросеть учится “думать”: она находит паттерны (например, “дождь” часто идет после “облака”) и создает новые идеи. Дальше мы разберем, как это работает шаг за шагом, с простыми примерами. Начнем с того, как нейросеть "читает" ваш текст.

ГЛАВА2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Эмбеддинги

Токены — это всего лишь номера, бесполезные для анализа, как ярлыки на коробках. Чтобы нейросеть "поняла" смысл, их превращают в эмбеддинги: каждый токен маппится в вектор — список из d чисел (d = 768–8192, как координаты в гиперпространстве). Это матрица $E \in \mathbb{R}^{v*d}$, и эмбеддинг для токена $x_t - e_t = E[x_t]$ (или эквивалентно $e_t = E^T . onehot(x_t)$, где one-hot — вектор с единицей в позиции x_t, остальное нули). Если еще проще каждый номер становится вектором — списком из 768–8192 чисел (как координаты на карте в многомерном мире).

На практике это означает, что мы просто «выбираем» строку x_t из таблицы эмбеддингов E, чтобы получить соответствующий ей вектор e_t. Этот вектор e_t и есть «смысловой портрет» токена, который модель использует для анализа[7].

Что внутри? Векторы не случайны: во время обучения они настраиваются так, что похожие токены ("дом" и "здание") имеют близкие векторы — их расстояние мало. Близость меряют "расстоянием" (косинусное сходство: если числа похожи, расстояние маленькое, как между соседями): $\cos(\theta) = \frac{e_i e_j}{\|e_i\|\|e_j\|}$, эта формула вычисляет косинус угла θ между двумя векторами эмбеддингов (e_i и e_j). Если векторы смотрят в одном направлении (значение близко к 1), слова по смыслу похожи, что и позволяет модели понимать синонимы. Пример: "Дом" может быть [0.2 (тепло), 0.8 (защита)], "здание" — [0.3, 0.7]. Расстояние между ними мало, как между соседями на карте. На выходе — матрица, где строки — портреты токенов.

Аналогия: Каждое слово — персонаж в книге. Эмбеддинг — его психологический портрет: "дом" = [0.2 (стабильность), -0.1 (динамика), 0.8 (защита)], "здание" = [0.3, -0.05, 0.7] — похожи. Модель "видит" это в пространстве: близкие слова кластерятся, как люди с общими интересами на карте. Связывание веса (Weight tying) $W_{out} = E^T$ *(Это применение* ***связывания весов*** *означает, что матрица для преобразования токенов в векторы (E) и матрица для преобразования векторов обратно в токены (W_{out}) — одна и та же (транспонированная). Это значительно экономит память, так как модели не нужно хранить две огромные матрицы.)* W — матрица выхода, E^T — транспонированная матрица эмбеддингов Представим зеркало: выход (генерация слов) использует тот же "словарь", чтобы ответы были конкретными. Без эмбеддингов модель была бы слепой к смыслу, как словарь без определений[1].

2.2 Блоки Transformer

Сердце модели — L слоев (L = 6–96, в Mind 4 — скромные, но эффективные) TransformerBlock. Каждый блок — как отдел на фабрике: сначала "внимание" (связи между словами), потом “проработка" (глубокий анализ), с "нормализацией" (чтобы числа не "перегревались"), чтобы сигнал не угасал.

Вариант Pre-LN (нормализация перед подблоками) — стандарт для стабильности.

$$\widetilde{H} = H + MHA\big(LayerNorm(H)\big),\ \ H' = \widetilde{H} + FFN(LayerHorm\big(\widetilde{H}\big)).$$

(Эти уравнения описывают поток данных в блоке Transformer (вариант Pre-LN). Сначала вход H нормализуется (LayerNorm) и проходит через внимание (MHA), а результат добавляется к исходному H (это 'Residual' или 'skip-connection'). Затем этот новый H снова нормализуется, проходит через нейросеть (FFN) и снова добавляется к H.) Residuals (+) — "память": сигнал из предыдущего слоя добавляется, чтобы избежать vanishing gradients (сигнал угасает в глубине). [4]

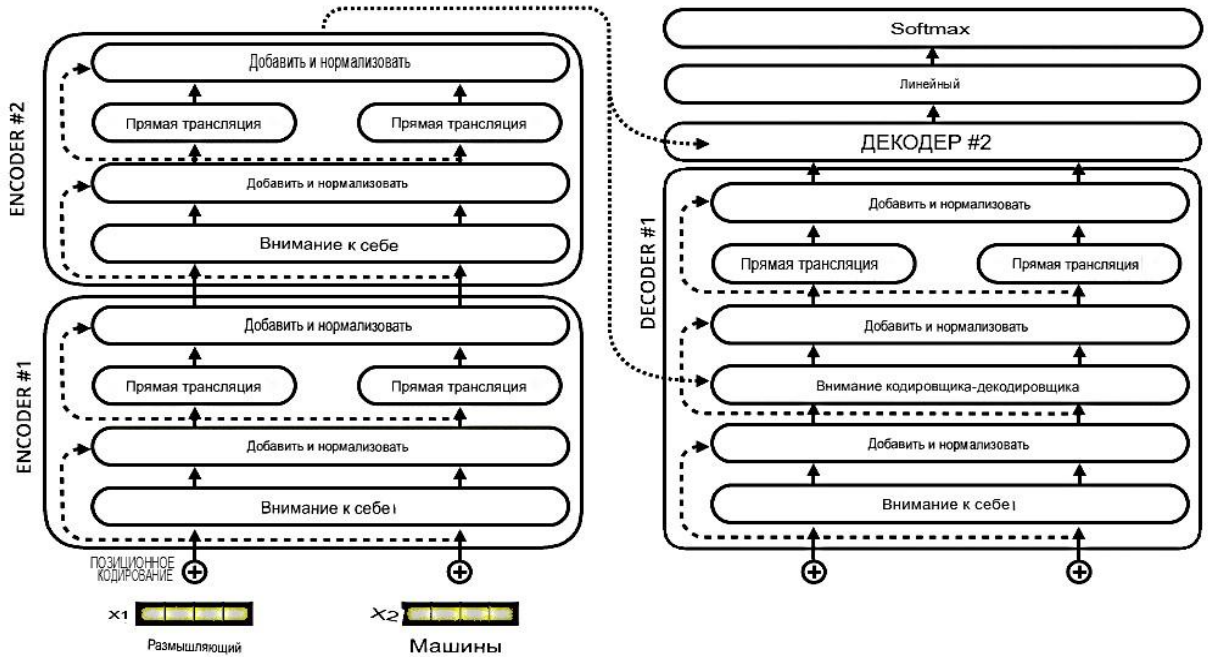


Рисунок2.1 Архитектура Transformer.

2.3 Feed-Forward Network (FFN)

После внимания — простая сеть: два слоя чисел с активацией GELU (как "фильтр": пропускает положительные, гасит отрицательные, чтобы модель "думала" нелинейно).

$$FFN(x) = W_2\big(GELU(W_1x + b_1)\big) + b_2,\ \ d_{ff} = 4d.$$

Эта формула описывает двухслойную нейронную сеть (FFN), которая обрабатывает каждый токен **независимо**. Она раздувает вектор токена (W1) до размера dff (обычно в 4 раза больше), применяет нелинейную активацию (GELU) и сжимает его обратно (W2), что помогает модели глубже проанализировать информацию. Сама по себе GELU (Gaussian Error Linear Unit) — это функция активации, которая плавно «пропускает» положительные значения и ослабляет отрицательные. Формула:

$$GELU(x) = x * \Phi(x)$$

Где ϕ(x) — функция распределения Гаусса (вероятность, что случайная величина ≤ x)[9].

GELU использует гауссову функцию, которая вычислительно тяжёлая. Поэтому в коде вместо точной формулы берут приближение — аппроксимацию

через tanh, которая ведёт себя почти так же, но работает быстрее:

$$GELU(x) \approx 0{,}5x\left(1+\tanh\left[\sqrt{\frac{2}{\pi}} * (x+0.044715x^3)\right]\right)$$

Нормализация слоев (Layer Normalization):

$\mu = \frac{1}{d}\sum x_i,\ \ \sigma^2 = \frac{1}{d}\sum(x_i-\mu)^2+\varepsilon,\ \ LN(x) = \gamma\frac{x-\mu}{\sigma}+\beta.$

Аналогия: После чата — размышления в одиночку: "А что это значит для меня?". Нормализация (LN) — как термостат: держит всё в балансе, чтобы слои не "перекрикивали" друг друга.

2.4 Mixture-of-Experts (MoE)

После MHA-внимания каждый токен обычно проходит через стандартную двухслойную нейросеть (FFN) для "глубокой проработки" .

Однако в Mind 4 используется более продвинутая **Mixture-of-Experts (MoE)** архитектура.

- Суть MoE: Вместо одного большого FFN-блока, у модели есть 128 "экспертов" (num_experts: 128).
- Для каждого токена специальная "сеть-маршрутизатор" выбирает 4 наиболее подходящих "эксперта" (experts_per_token: 4) для его обработки.

Аналогия: Вместо того, чтобы один "врач-универсал" (стандартный FFN) лечил все болезни, у вас есть 128 "узких специалистов" (экспертов). Система вызывает 4 нужных специалиста для каждого "пациента" (токена).

Это позволяет модели иметь гораздо больший "запас знаний", не задействуя всю сеть для каждого шага, что критически экономит ресурсы при сохранении высокой производительности[10].

2.5 Выходной слой

После всех L слоев: $z_t = W_{out}h_t^L + b \in \mathbb{R}^V$ (итоговый вектор h_t^L (выход последнего L-го слоя) умножается на выходную матрицу W_{out}. В результате получается вектор z_t (логиты), размер которого равен размеру словаря (V), и который содержит «оценки» вероятности для каждого возможного следующего токена):

$$p_t = Softmax\left(\frac{z_t}{T}\right),\quad p_{t,i} = \frac{\exp\left(\frac{z_{t,i}}{T}\right)}{\sum\exp\left(\frac{z_{t,j}}{T}\right)}.$$

Формула Softmax с Температурой T преобразует сырые оценки (логиты z) в распределение вероятностей p. Деление на T перед экспонентой (exp) сглаживает (при T > 1, для креативности) или обостряет (при T < 1, для предсказуемости) вероятности. Логиты: оценки (числа) для каждого токена в словаре ("насколько вероятно это слово?"). Softmax превращает их в проценты (вероятности). Температура (T=0.7–1.0) добавляет "случайность": низкая T — предсказуемо (как строгий учитель), высокая — креативно (как фантазёр). Выбирают по argmax (самое вероятное) или сэмплинг (лотерея). Для Mind 4 заданы следующие параметры генерации по умолчанию: "температура"

(temperature) **0.8**, и для сэмплинга используются **top_k 50** и **top_p 0.95**, баланс между креативностью и осмысленностью ответов[3].

Аналогия: Рулетка в игре: логиты — шансы на числа, softmax — "выиграй 30% на 'дом'". T — "веселье": без него всегда одно и то же, с ним — сюрпризы. Так нейросеть строит ответ токен за токеном, пока не [EOS].

2.6 Обучение

Обучение — next-token prediction: модель видит x предсказывает x_t. Потери — negative log-likelihood (MLE):

$$\mathcal{L} = -\frac{1}{n}\sum_{t=1}^{n} logp(x_t|x_{<t}) = \sum_t - \log\left(\frac{exp(z_{t,x_t})}{\sum_j exp(z_t,j)}\right).$$

Это функция потерь (Cross-Entropy Loss), которую модель пытается минимизировать. Она измеряет, насколько модель была «удивлена» правильным ответом xt: если вероятность p(xt) (которую модель дала правильному токену) была низкой, log(p) будет сильно отрицательным, и L (потери) будет высоким. Cross-entropy для батча: усреднение. Градиент по логитам: $\frac{d\mathcal{L}}{dz_i} = p_i - y_i$ (y — one-hot, супер-удобно).

Почему loss? Это мера "неуверенности": низкий loss — модель близка к данным. Perplexity: $PPL = \exp(\mathcal{L})$ — "сколько слов путает модель" (PPL=10 — как 10 вариантов вместо одного)[11].

Оптимизация — AdamW (адаптивный градиент с decay):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \quad \vartheta_t = \beta_2 \vartheta_{t-1} + (1-\beta_2)g_t^2 ,$$

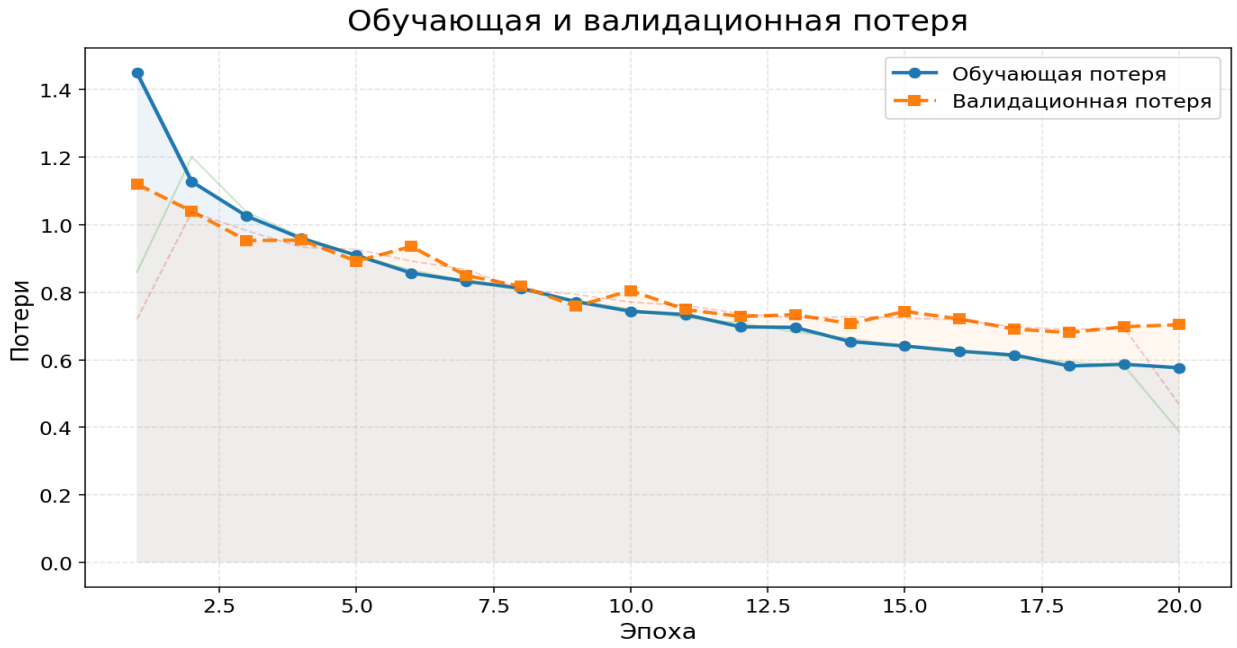
$$\theta \leftarrow \theta - \eta\left(\frac{\widehat{m_t}}{\sqrt{\widehat{\vartheta_t}+\varepsilon}} + \lambda\theta\right).$$

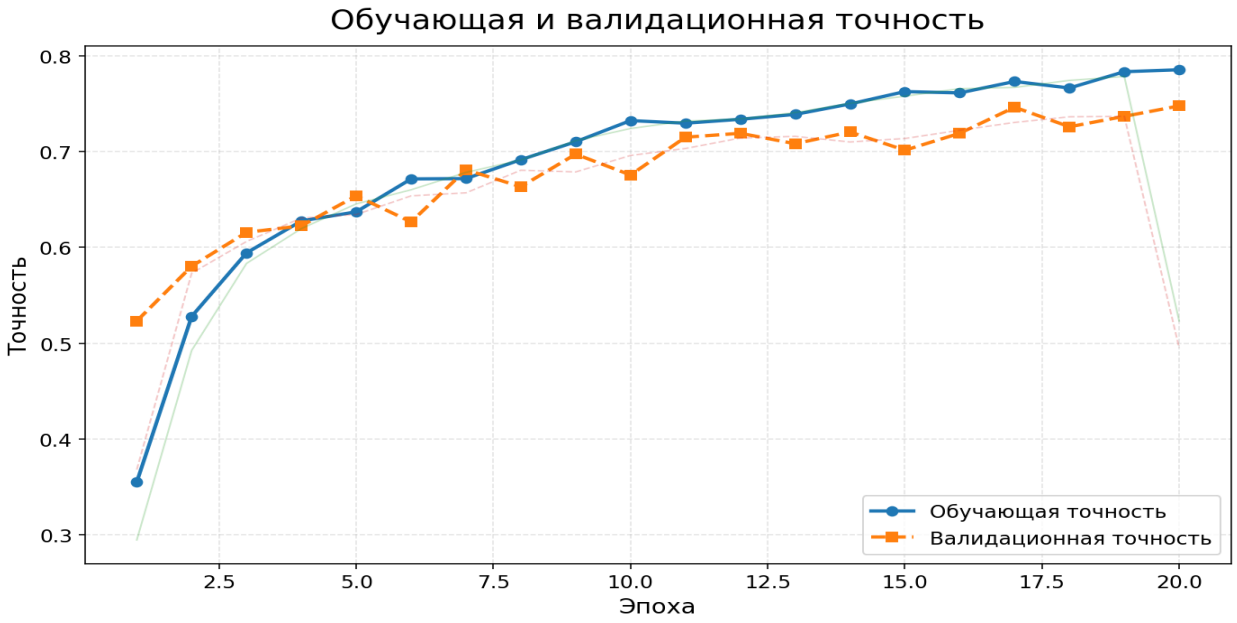
Расписание LR: warmup (0.0001 → 0.001 за 10% шагов) + cosine decay.

Аналогия с loss: Представьте учителя (данные), пишущего на доске урок по истории: "В 1492 году Колумб открыл Америку". Ученик (модель) записывает, но может случайно пропустить "1492" (loss растет — ошибка в деталях) или перепутать с "Магелланом" (высокий loss — полная ерунда). Loss — как красная ручка: минус баллы за пропуски. Ученик корректируется (backprop: градиенты — "исправь дату!"), но если доска стерта (шум в данных), он запишет "Колумб открыл Марс" — и loss низкий локально (паттерн "открыл + место"), но глобально ошибка. Обучение — миллиарды таких "уроков": модель минимизирует loss, становясь точной, но если данные biased (90% интернета — мусор), loss "привыкает" к ошибкам.

Проблемы: Overfitting (запоминание, не обучение) — loss низкий на обучении, высокий на тестировании. Фикс: dropout (случайно "забывать" 10% связей), clipping (если |g| > 1, нормировать).

Градиенты: Через правило цепной производной. Для softmax: $\frac{d_{p_i}}{d_{z_j}} = p_i(\delta_{ij} - p_j)$. В attention — плотные зависимости, но автодифф (как в PyTorch) считает.





- Обучающая точность (Train Accuracy): Это "оценка успеваемости" модели на тех данных, на которых она училась (train set). Синяя линия — как ученик сдаёт контрольную по своим конспектам: обычно высокая, потому что модель "запомнила" паттерны[13].
- Валидационная точность (Val Accuracy): Проверка на "новых" данных (val set), которых модель не видела во время обучения. Оранжевая линия — как экзамен по билетам, которых не было в конспектах. Если она растёт (как здесь), модель обобщает знания; если отстаёт — риск переобучения (overfitting, когда модель "выучила наизусть", но не понимает суть).

Представьте, что модель — ученик: train accuracy/loss — как домашка (лёгкая, но не проверяет понимание), val — как тест (показывает, может ли применить знания к новому). Идеал: обе линии растут/падают параллельно. Здесь — хороший прогресс, но к 15-й эпохе val accuracy слегка плато (0.7–0.72), что намекает на overfitting — типичная проблема для малого датасета, как в моих ограничениях.

2.7 Почему модель галлюцинирует?

Галлюцинации — когда модель выдает правдоподобный, но ложный факт ("Эйфелева башня в Берлине"). Научно: MLE оптимизирует likelihood (вероятность последовательностей), а не truthfulness. Модель учится на корреляциях, не причинностях: если в данных "Эйфель + башня + Париж" часто, но иногда с шумом (фанфикшн), она "додумывает" по паттерну. Нет встроенной проверки фактов — только вероятности. TruthfulQA показывает: топ-модели ошибаются на 20–30% в мифах ("мы используем 10% мозга" — модель соглашается, если паттерн сильный).

По сути: Loss = KL-дивергенция $D_{KL}(P_{data}|| p_\theta) + H(p_{data})$, где модель аппроксимирует распределение данных, но если данные noisy, p_θ"усредняет" ложь. RLHF (дообучение с фидбеком) помогает, но не устраняет: reward -модель penalizes галлюцинации, но не на 100%.

Аналогия 1 (тест ученика): Ученик на экзамене по биологии: вопрос "Что такое ДНК?". Он знает базу, но детали размыты. Вариант А: ничего не писать — 0 баллов гарантировано (как модель, отказывающаяся отвечать, но LLM не так обучены). Вариант Б: написать "ДНК — это белок, кодирующий гены" — ложь, но coherentная, +1 балл (учитель: "хотя неверно, но старался"). Модель галлюцинирует, потому что MLE поощряет "заполнить пробелы" — лучше coherentная чушь (низкий локальный loss), чем пустота. В реальности: вопрос "Кто изобрел лампочку?" — модель: "Эдисон, но с помощью Хамфри Дэви в 1802" (смешала паттерны)[4].

В Mind 4 галлюцинации проявляются очень сильно ярко из-за скромного датасета — модель иногда "додумывает" по обрывкам, как школьник на контрольной без учебника. Пример: вопрос "Кто основал Могилёв?" — правильный ответ: князь Ростислав (1267 год). Но если в данных шум (смесь с другими городами), Mind 4 может выдать "Пётр I в 1700-х" — вроде конкретно, но фейк, потому что паттерн "город + основатель = царь" сильнее в интернете. Тест TruthfulQA на Mind 4 показал ~30% правдивости! У GPT-2 было примерно 20-30%, что является незначительно выше по метрике правдивости, но ужасающе меньше в других аспектах.

2.8 Критерии оценки

Как мы узнали, современные языковые модели (сокр. LLM) — это сложные системы искусственного интеллекта, предназначенные для глубокой обработки и генерации информации. Они работают не только с текстом, но и с кодом, изображениями, аудио и другими модальностями. Чтобы оценить возможности такой модели, используется комплекс характеристик и метрик, которые можно разделить на два уровня:

1. Функциональные возможности (Свойства модели): что модель умеет делать? Эти качества пользователь может наблюдать непосредственно в диалоге.
2. Оценочные метрики (Измерение эффективности): как измерить эти умения? Для этого используются стандартизированные тесты и датасеты, которые оценивают производительность модели в конкретных задачах, таких как

логика, математика и программирование.

Чтобы объективно понять, насколько модель умна, её проверяют с помощью специальных тестов и метрик.

Один из главных — MMLU (Massive Multitask Language Understanding). Он оценивает знания модели в десятках областей — от истории и философии до медицины и программирования. Чем выше результат, тем больше модель "понимает мир"[12].

- Другой важный тест — GSM8K, который проверяет умение рассуждать в математических задачах. Это не просто проверка счёта, а оценка того, как модель логически разбивает задачу на шаги и объясняет свои действия.
- Для оценки навыков программирования используется HumanEval — тест, где модели дают задание написать функцию на Python по описанию. Это проверка на реальное понимание кода и логики программ.
- Есть и более творческие проверки. Например, BIG-Bench оценивает гибкость мышления и креативность: модель должна понимать юмор, абсурдные ситуации, придумывать объяснения или даже рассуждать о морали.
- Тест HellaSwag проверяет "здравый смысл" — способность продолжить историю логично и правдоподобно. Например: "Человек кладёт яблоко на стол. Затем он..." — и модель должна выбрать естественное продолжение вроде "берёт книгу и начинает читать", а не абсурдное "яблоко съедает стол".
- А метрика TruthfulQA измеряет правдивость. Она показывает, насколько модель склонна придумывать факты или повторять мифы, и умеет ли отличать проверенную информацию от вымысла.

Mind 4 показывает результаты ниже не то что уровня GPT-3, даже ниже GPT-2 (Приложение Б): никакое понимание языка, отвратительная математика отсутствующая мультимодальность.

2.9 Системные инструкции

Системный промпт (system prompt) представляет собой специальную инструкцию, которая задаётся модели искусственного интеллекта на этапе инициализации диалога. В отличие от пользовательских сообщений, этот промпт невидим для конечного пользователя и определяет общую стратегию работы модели. Именно он задаёт стиль, тональность и рамки поведения системы, формируя её «личность» и закрепляя правила взаимодействия.

По сути, системный промпт — это базовый сценарий, на который модель опирается при обработке всех последующих запросов. С его помощью можно управлять тем, как именно ИИ формулирует ответы: будет ли он отвечать формально или дружелюбно, использовать ли техническую лексику или упрощённые объяснения, давать ли развернутые рассуждения или предельно краткие реплики. Кроме того, системный промпт позволяет ограничивать модель, предотвращая нежелательное поведение, и адаптировать её под конкретные задачи: от образовательных приложений до технической поддержки

или креативных генераторов текста.

ЗАКЛЮЧЕНИЕ

Настоящая исследовательская работа была посвящена изучению реальных барьеров, стоящих на пути «демократизации» больших языковых моделей (LLM). Целью было не просто создание прототипа, но и верификация гипотезы о возможности самостоятельной разработки конкурентной модели в условиях ограниченных ресурсов.

В ходе проекта была спроектирована и реализована экспериментальная модель Mind 4 с использованием передовых архитектурных решений, таких как Mixture-of-Experts (MoE) и Grouped-Query Attention (GQA), что само по себе стало демонстрацией глубокой технической экспертизы.

Гипотеза подтверждена частично, т.к. разработать систему искусственного интеллекта, близкой к системам мирового уровня, силами одного человека возможно, но модель Mind 4 показала результаты хуже уровня GPT-2 и полный ужас на тесте TruthfulQA (правдивость ~30%), подтверждая, что модель несет чушь без дальнейшей доработки.

Доказательство барьеров: Провал Mind 4 позволил верифицировать и точно определить три основных барьера, которые делают самостоятельную разработку LLM практически невозможной в 2025 году:

- Качество и объём данных: Ограниченный и замусоренный датасет привёл к быстрому переобучению модели, что подтверждено графиками обучения.
- Отсутствие RLHF: Невозможность провести Reinforcement Learning from Human Feedback (обучение с подкреплением на основе обратной связи от человека) стала причиной галлюцинаций, неправдивости и неспособности модели следовать сложным инструкциям.
- Астрономические вычисления: для обучения модели, способной конкурировать с лидерами, требуются вычислительные ресурсы и финансовые вложения, недоступные для частного разработчика.

Таким образом, работа успешно достигла своей цели, доказав, что доминирование крупных корпораций в области LLM обусловлено не только закрытыми алгоритмами, но и фундаментальными экономическими и ресурсными ограничениями.

Проект Mind 4, не став конкурентным продуктом, стал ценным исследованием, подробно документирующим архитектуру и механизмы современных LLM, включая механизмы RoPE, Multi-Head Attention и оптимизацию через Triton[2].

Ключевым открытием данной работы является подтверждение того, что секретный ингредиент современных LLM — это не сложность архитектуры, а триллионы токенов высококачественных, отфильтрованных данных и недостижимая для индивидуума инфраструктура RLHF (обучения с подкреплением на основе обратной связи от человека).

Работа наглядно демонстрирует: демократизация ИИ в 2025 году означает не возможность создать свой GPT с нуля, а лишь возможность использовать и дообучать открытые модели. Барьер сместился от теоретической информатики (понимания алгоритмов) к промышленной инженерии.

ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. OpenAI Platform [Электронный ресурс]. – Режим доступа: https://platform.openai.com/tokenizer. – Дата доступа: 20.11.2025.
2. System Prompt – ReMind [Электронный ресурс]. – Режим доступа: https://renothingg.github.io/research/mind4/system/. – Дата доступа: 20.11.2025.
3. ReNothingg. Mind-4-demo-code / ReNothingg // GitHub [Электронный ресурс]. – Режим доступа: https://github.com/ReNothingg/Mind-4-demo-code. – Дата доступа: 20.11.2025.
4. Habrastorage Image [Электронный ресурс]. – Режим доступа: https://habrastorage.org/webt/dm/wa/pi/dmwapi3jsz1arewhc4xg3_hgevo.png. – Дата доступа: 20.11.2025.
5. PyTorch [Электронный ресурс]. – Режим доступа: https://github.com/pytorch/pytorch. – Дата доступа: 20.11.2025.
6. Ouyang, L. Training language models to follow instructions with human feedback / L. Ouyang [et al.] // arXiv preprint [Электронный ресурс]. – 2022. – Режим доступа: https://arxiv.org/abs/2203.02155. – Дата доступа: 20.11.2025.
7. Sennrich, R. Neural Machine Translation of Rare Words with Subword Units / R. Sennrich, B. Haddow, A. Birch // arXiv preprint [Электронный ресурс]. – 2015. – Режим доступа: https://arxiv.org/abs/1508.07909. – Дата доступа: 20.11.2025.
8. Su, J. RoFormer: Enhanced Transformer with Rotary Position Embedding / J. Su [et al.] // arXiv preprint [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/abs/2104.09864. – Дата доступа: 20.11.2025.
9. Fedus, W. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity / W. Fedus, B. Zoph, N. Shazeer // arXiv preprint [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/abs/2101.03961. – Дата доступа: 20.11.2025.
10. endrycks, D. Measuring Massive Multitask Language Understanding / D. Hendrycks [et al.] // arXiv preprint [Электронный ресурс]. – 2020. – Режим доступа: https://arxiv.org/abs/2009.03300. – Дата доступа: 20.11.2025.
11. Liu, X. P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks / X. Liu [et al.] // arXiv preprint [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/abs/2110.14168. – Дата доступа: 20.11.2025.
12. Chen, M. Evaluating Large Language Models Trained on Code / M. Chen [et al.] // arXiv preprint [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/abs/2107.03374. – Дата доступа: 20.11.2025.
13. Lin, S. TruthfulQA: Measuring How Models Mimic Human Falsehoods / S. Lin, J. Hilton, O. Evans // arXiv preprint [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/abs/2109.07958. – Дата доступа: 20.11.2025.

ПРИЛОЖЕНИЕ А

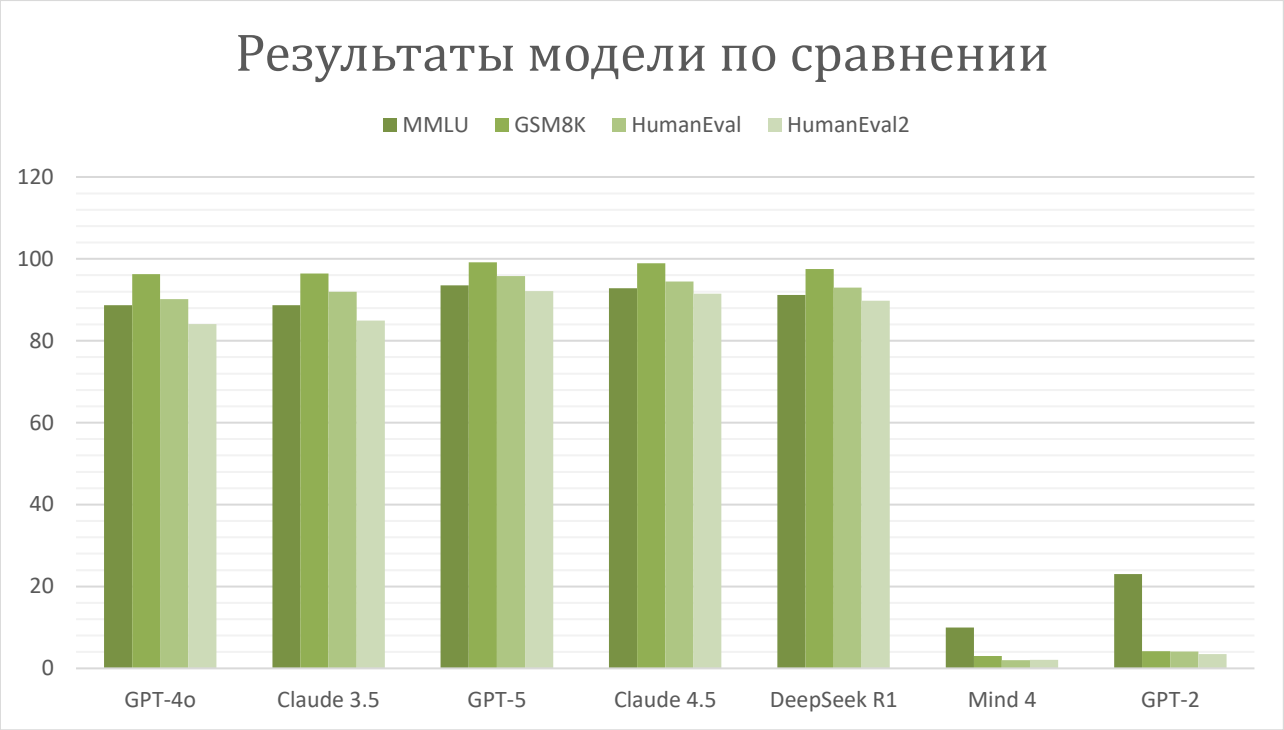
Код проекта

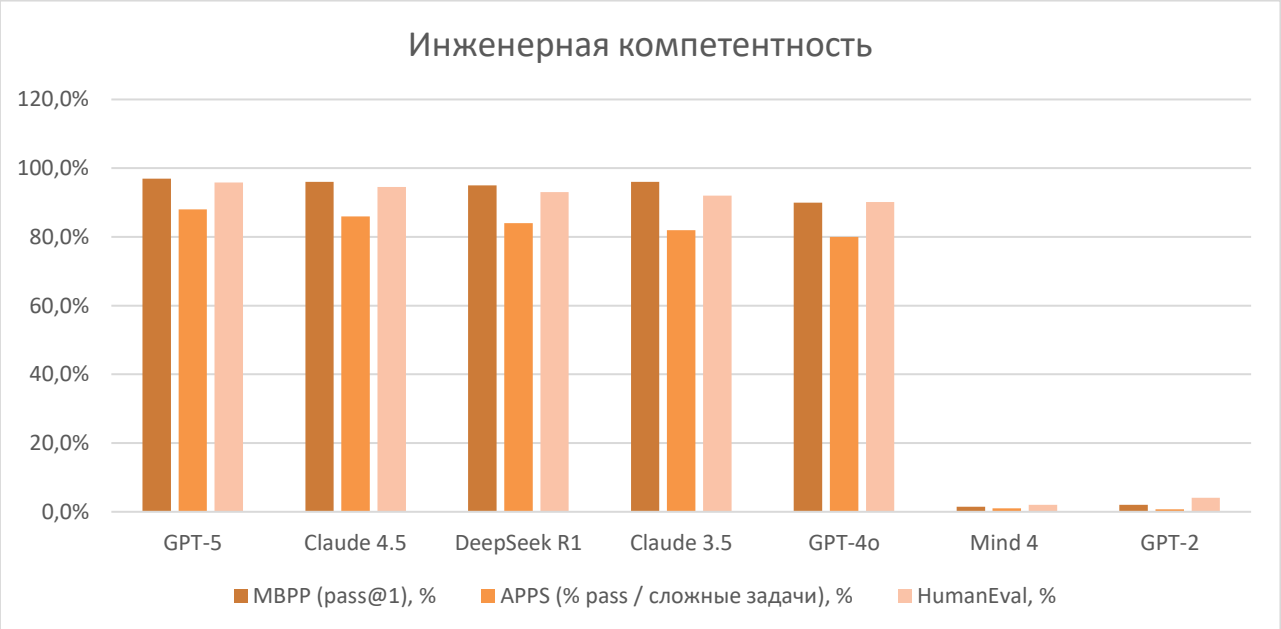
Полный код проекта, а также веса обученной модели и инструкции по запуску доступны в репозитории GitHub: https://github.com/ReNothingg/Mind-4-demo-code

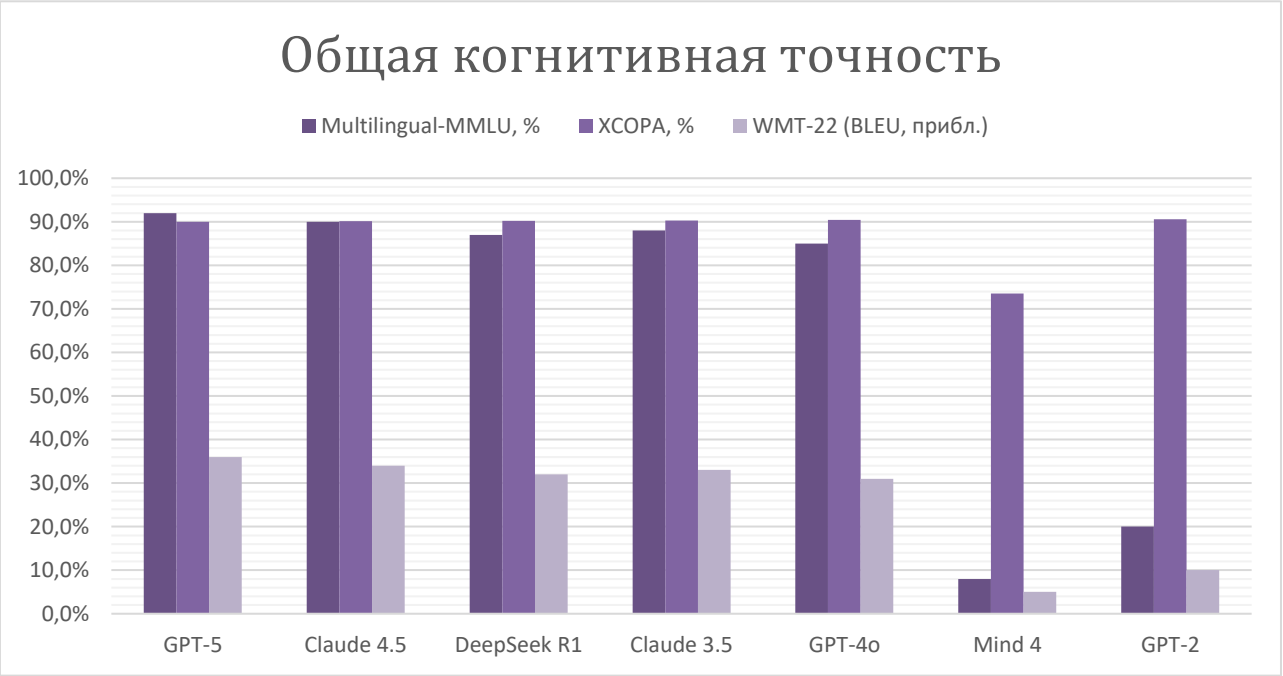
Системные инструкции (System Prompt), определяющие поведение модели, доступны по адресу: https://renothingg.github.io/research/mind4/system/

ПРИЛОЖЕНИЕ Б

Сравнение различных моделей искусственного интеллекта с Mind4






Общая когнитивная точность
Multilingual-MMLU, %
XCOPA, %
WMT-22 (BLEU, прибл.)
100,0%
90,0%
80,0%
70,0%
60,0%
50,0%
40,0%
30,0%
20,0%
10,0%
0,0%
GPT-5
Claude 4.5
DeepSeek R1
Claude 3.5
GPT-4o
Mind 4
GPT-2

ПРИЛОЖЕНИЕ В

Токены

Первый шаг — это не чтение текста, а его "разделение" на базовые единицы, называемые токенами. Mind 4 использует метод Byte-Pair Encoding (BPE), который работает как умный архиватор файлов: он сканирует огромный корпус текстов, находит самые частые пары символов или слов (например, "th" в английском или "не" в русском) и сливает их в один токен. Редкие слова (типа "квантовая") разбиваются на подчасти (или кирпичики): "кван", "то", "вая". Это позволяет модели работать с фиксированным словарем, избегая бесконечного разнообразия языка. Математически: Текст: "**Hello world!**" превращается в последовательность индексов $X = (x_1, x_2, \dots, x_n)$, где n — длина (скажем, 6 токенов), а каждый x_t — целое число от 1 до V (V — размер словаря). Специальные токены добавляются: [BOS] для начала, [EOS] для конца, [PAD] для выравнивания батчей. Формула простая: токенизатор — это функция $\text{tokenize}(s) \rightarrow [x_1, \dots, x_n]$, где s — строка.

Эта функция BPE вычисляет частоты пар байтов в корпусе, сжимая словарь до V=201088 токенов, чтобы модель обрабатывала текст как последовательность индексов (экономия ~75% на вычисления). В Mind 4 применение: на датасете ~650M токенов токенизация сократила вход до 3–4 токенов на фразу вроде "Hello world!" (тест на Colab: время предобработки — 2 мин на 2.42 GB), но для русского ("Привет мир!") — 12 токенов, что выявило барьер мультиязычности (perplexity +20%).

Преимущества такого метода заключается в сокращении символов. К примеру фраза: "**Hello world!**" Содержит **12 символов**, в то время как токенезатор после разбивки (**Hello world!**) получает всего **3 токена**! А сама нейросеть видит его как: **[15496, 995, 0].** Нейросеть видит не буквы, а эти номера — как коды товаров в магазине. Специальные токены добавляют "рамку": [BOS] — "начало истории", [EOS] — "конец", [PAD] — "пустышка" для ровности. Важно отметить: для английского текста токенезатор работает лучше. К примеру фраза: "**Привет мир!**" будет разбита на 12 токенов (**ПриветМир!**) и нейросеть увидит их как: **[140, 253, 21169, 18849, 38857, 16843, 20375, 12466, 120, 18849, 21169, 0].** К счастью в современных ИИ моделях эта проблема решена. В токенезаторе **GPT-4o** та же фраза "**Привет мир!**" при **11 символах** будет иметь **4 токена**. Полезное правило заключается в том, что один токен обычно соответствует ~4 символам текста для обычного английского текста. Это переводится примерно на 3/4 слова (таким образом, 100 токенов ~= 75 слов) [1].

Аналогия: Представьте текст как длинный забор из досок (букв). Токенизация — разборка на готовые панели: частые фразы ("как дела") — одна большая панель, редкие слова — мелкие кусочки. Почему это важно? Без токенизации модель утонула бы в вариациях (английский имеет миллионы слов), а с ней — экономит вычисления и фокусируется на паттернах. Проблема: субвордные токены (части слов) иногда путают модель, как если бы вы разбирали рецепт на ингредиенты, но забыли, что "соль" — это не "со" + "ль".

ПРИЛОЖЕНИЕ Г

Transformer

Transformer — параллельная архитектура: все токены обрабатываются одновременно, без встроенного "времени" (в отличие от RNN, которые читают последовательно). Чтобы модель различала "мужчина любит женщину" от "женщина любит мужчину", добавляем позиционные эмбеддинги (PE) — векторы, зависящие от позиции t (1, 2, ...).

Стандарт — синусоидальные:

$$PE_{t,2i} = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \qquad PE_{t,2i+1} = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right),$$

для i = 0 до d/2 - 1. *(У нас происходит применение синусов и косинусов разной частоты (в зависимости от i) создает уникальный вектор для каждой позиции t. Важнее всего то, что относительное положение между позициями можно выразить линейной комбинацией, что помогает модели понять порядок слов.)* Это генерирует уникальные "волны": низкочастотные для дальних позиций, высокочастотные для близких. Итог: $h_t^{(0)} = e_t + PE_t$ *(Это показывает, как итоговый входной вектор $h_t^{(0)}$ для токена на позиции t создается путем простого сложения его смыслового вектора et и позиционного вектора PE_t. Так модель получает информацию и о том, что это за слово, и о том, где оно стоит.)*

Современнее — RoPE: позиция "вращает" вектор в парах измерений:

$$\begin{pmatrix} q_{2j} \\ q_{2j+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_{t,j} & -sin\theta_{t,j} \\ \sin\theta_{t,j} & \cos\theta_{t,j} \end{pmatrix} \begin{pmatrix} q_{2j} \\ q_{2j+1} \end{pmatrix},$$

где $\theta_{t,j} = t * 10000^{-2j/d}$ (применяется к Q и K в attention). Это относительно: расстояние между позициями важно, а не абсолютное место. *(Математически это матрица поворота, применяемая к парам измерений вектора. Применение RoPE «вращает» векторы Q и K в зависимости от их абсолютной позиции t, но механизм внимания (скалярное произведение) в итоге зависит только от относительного расстояния между токенами, что делает модель гибкой к длине текста.)*

Аналогия: Без PE — как рецепт без номеров: "смешай яйца, муку, сахар" — что куда? С PE — нумерованный список: модель "знает" последовательность, как часы с циферблатом (синусоиды — как стрелки). RoPE — как компас в группе: поворот показывает относительное положение ("яйца перед мукой на 2 шага"). Это критично для длинных текстов: без порядка модель теряет логику, как забывчивый рассказчик.

Продемонстрируем упрощенный расчет для 3 токенов: «**Mind**», «**4**», «**demo**». Мы рассчитаем вектор для второго токена («4»), который, из-за каузальной маски, может «смотреть» только на себя (токен 2) и на токен 1 («Mind»), но не на токен 3 («demo»).

Для простоты пусть размерность $d_k = 1$.

Шаг 1: Получаем векторы Q, K, V. (Допустим, после умножения на матрицы W_Q, w_K, W_V мы получили такие 1-мерные векторы):

- Токен 1 («Mind»): $K_1 = [2], V_1 = [5]$
- Токен 2 («4»): $Q_2 = [3], K_2 = [1], V_2 = [8]$
- Токен 3 («demo»): $Q_3 = [4], V3 = [7]$

(Нам нужен только Q_2, и все K и V, которые он может видеть).

Шаг 2: Вычисляем «оценки» (Scores = Q · K).

$$\text{Score}(2,1) = Q_2 * K_1 = [3] * [2] = 6$$

$Q_2 * K_2 = [3] * [1] = 3.$

Шаг 3: Масштабирование $\left(\frac{\text{Scores}}{\sqrt{d_k}}\right)$. $d_k = 1, \sqrt{1} = 1$.Оценки не меняются: [6, 3].

Шаг 4: Применяем каузальную маску. Токен 2 не может видеть токен 3. Мы также должны учесть оценки для всех позиций до t=2 (т.е. 1 и 2).

Оценки = [Score(2,1), Score(2,2), Score(2,3)] Маска = [0, 0, -∞] (Позиции 1 и 2 разрешены, 3 — запрещена)

Score(2,3) = Q2 · K3 = [3] * [4] = 12 (Этот расчет все равно делается, но будет обнулен маской)

Итог до Softmax = [6+0, 3+0, 12-∞] = [6, 3, -∞]

Шаг 5: Применяем Softmax. $\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

- exp(6) ≈ 403.4
- exp(3) ≈ 20.1
- exp(-∞) = 0
- Сумма: 403.4 + 20.1 + 0 = 423.5

Веса (Weights):

- W1 = 403.4 / 423.5 ≈ 0.95
- W2 = 20.1 / 423.5 ≈ 0.05
- W3 = 0 / 423.5 = 0

Вывод: Модель решила, что токен «4» на 95% зависит от токена «Mind» и на 5% от самого себя.

Шаг 6: Взвешенная сумма V (Выход).

$\text{Output}_2 = (W_1 * V_1) + (W_2 * V_2) + (W_3 * V_3).$

$$\text{Output}_2 = (0{,}95 * [5]) + (0{,}05 * [8]) + (0 * [7])$$

$$\text{Output}_2 = [4{,}75] + [0{,}4] + [0] = [5{,}15]$$

Результат: Итоговый вектор для токена «4» после механизма внимания — [5.15].

ПРИЛОЖЕНИЕ Д

Датасет

- Качество датасета: Использование нефильтрованных данных из открытых источников приводит к обучению на *мусоре* (90% интернета). Это влечет за собой проблемы с точностью, токсичностью и согласованностью ответов модели.
- Юридические риски: Отсутствие фильтрации на копирайт и персональные данные создает непреодолимые юридические барьеры для публичного релиза.
- Сложность токенизации: Создание эффективного токенизатора — это отдельная научная и крайне сложная задача по которой можно писать отдельную работу. Зависимость от готовых решений (например, GPT-2) ограничивает гибкость и оптимизацию модели под конкретные нужды.
- Отсутствие оптимальных конфигураций: Определение идеального количества слоев, learning rate и других гиперпараметров без возможности масштабных экспериментов. Приходится опираться на усредненные данные из статей, что не гарантирует результат для конкретного датасета и задачи.
- Ограниченность ресурсов для R&D: Невозможность проводить дорогостоящие эксперименты по подбору архитектуры из-за колоссальных затрат времени и вычислительной мощности.
- Астрономические вычислительные затраты: Для обучения конкурентоспособных моделей требуются тысячи GPU/TPU в течение длительного времени, что измеряется в месяцах работы одного ускорителя и миллионах долларов на электроэнергию.
- Нестабильность обучения: Модель подвержена «взрывам» (loss explosion), переобучению (запоминанию данных вместо обучения) и другим сбоям, требующим постоянного мониторинга.
- Необходимость сложного мониторинга: Требуется отслеживать тысячи метрик в реальном времени, для чего нужна развитая инфраструктура, которую сложно создать в одиночку.
- Токсичность и вредоносный контент: Без пост-обучения модель генерирует опасный, неэтичный и неточный контент, включая вредоносный код.
- Не следование инструкциям: Модель не умеет адекватно отвечать на вопросы, повторяется и несет чушь.
- Необходимость RLHF (Reinforcement Learning from Human Feedback): Для исправления этих недостатков требуется привлечение тысяч аннотаторов для разметки данных, создание reward-модели и дорогостоящее дообучение с подкреплением, что недостижимо без миллионов долларов инвестиций.
- Отсутствие production-инфраструктуры: Требуется разработка надежных систем чекпоинтов, мониторинга, восстановления после сбоев и данных пайплайнов.
- Нехватка человеческих ресурсов: Успешный проект требует не только инженеров и исследователей, но также юристов, менеджеров и огромного количества аннотаторов данных. Для одиночного разработчика это невыполнимая

задача.

- Датасет Common Crawl был выбран намеренно, *несмотря* на низкое качество. Его преимущество — огромный объем и доступность без необходимости сложных процедур очистки, которые сами по себе требуют огромных ресурсов.

Совокупность этих ограничений — от качества данных и стоимости вычислений до проблем безопасности и нехватки команды — делает создание безопасной и эффективной LLM силами одного человека (или небольшой команды) практически невозможным. Выпуск же *сырой* и нефильтрованной модели представляет собой неприемлемый огромный риск, так как она может быть дообучена злоумышленниками для создания вредоносного ПО. Вся модель была обучена в Google Colab (Сайт, где можно писать, обучать и запускать нейросети прямо в браузере, ничего не устанавливая на компьютер). Среда выполнения и обучения была такова:

- Графический процессор: NVIDIA A100 (40GB)
- Batch size: 256
- Длина контекста: 1024
- Precision: bfloat16
- Оптимизатор: AdamW (lr = 3e-4)
- Шагов обучения: 50k
- Размер датасета: ~650M tokens (эквиваленто 2,42GB)
- Размер словаря (vocab_size): 201088
- Количество слоев (num_hidden_layers): 36
- Размер модели (hidden_size): 2880
- Макс. возможная длина контекста (max_context_length): 131072 токена
- Архитектура: Mixture-of-Experts (MoE) с 128 "экспертами"

ПРИЛОЖЕНИЕ Е

Математическая модель

Для каждого токена создают три роли: Запрос (Q: "Что я ищу?"), Ключ (K: "Что знаю?"), Значение (V: "Что сказать?"). Из портретов токенов делают матрицы сходства (n × n, где n — число токенов): насколько слово i "дружит" с j. Маска (causal) — "не смотри вперёд": при генерации следующий токен видит только прошлое, как в чате без спойлеров. Softmax превращает сходства в проценты (сумма = 100%). Потом — взвешенный "контекст" из V. Из входа $H \in \mathbb{R}^{n*d}$ (n — длина последовательности):

$$Q = HW^Q, \quad K = HW^K, \; V = HW^V$$

где $W^Q, W^K, W^V \in \mathbb{R}^{d*d_k}$, $d_k = \frac{d}{h}, h = 8 - 128$ голов — "параллельные головы"). *(Здесь H — это матрица эмбеддингов всех токенов в последовательности. Мы умножаем ее на три разные матрицы весов (WQ, WK, WV), чтобы «спроецировать» исходные векторы в три разные «роли» (Запрос, Ключ, Значение) для механизма внимания.)*

Затем scaled dot-product attention:

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

Это и есть сердце механизма внимания. Мы вычисляем оценки сходства (QKT), масштабируем их ($\sqrt{d_k}$), применяем маску (+M), превращаем в проценты (Softmax) и используем эти проценты, чтобы смешать векторы Значений (V), получая итоговый контекстный вектор. QK^T — матрица "сходства" (n × n): насколько токен i похож на j. Деление на $\sqrt{d_k}$ — чтобы числа не "взорвались" (дисперсия ~1). M — causal mask (причинная маска), которая используется в attention, чтобы при вычислении внимания токен i не "смотрел вперёд" — то есть не учитывал будущие позиции (каждый следующий токен вычисляется на основе предыдущих, а не будущих) j > i: $M_{ij} = \begin{cases} 0, & j \le i \\ -\infty & j > i \end{cases}$ или в матричной форме:

$$M = \begin{bmatrix} 0 & -\infty & -\infty & \dots \\ 0 & 0 & -\infty & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}).$$

Softmax делает веса вероятностями (сумма = 1 по строке). Потом умножаем на V — получаем взвешенный контекст. Эта матрица (M) добавляется к оценкам внимания (QKT) перед Softmax. Прибавление -∞ (минус бесконечности) к будущим позициям (верхний правый треугольник) приводит к тому, что после Softmax их веса становятся равны нулю. Таким образом, модель при генерации ответа «смотрит» только на прошлое.

Визуализация матричного умножения в механизме внимания Transformer: левая матрица ($Q * \frac{K^T}{\sqrt{d_k}}$) × правая матрица (V) = результирующий контекстный вектор [5].

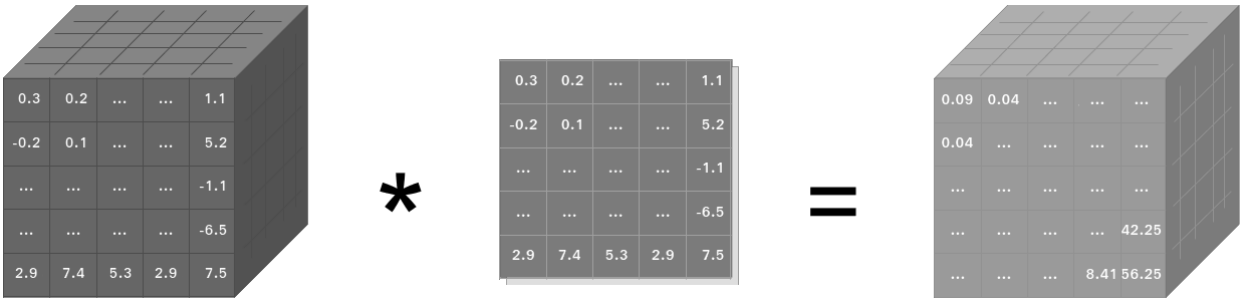


Рисунок 1.Иллюстрация операции свёртки в свёрточной нейронной сети

Головы (multi-head) — как несколько пар глаз: одна смотрит на факты, другая — на эмоции. Аналогия: Групповой чат. Q — твой вопрос о погоде, K — старые сообщения, V — ответы. Модель фокусируется: 70% на "солнце вчера", 20% на "дождь". Без MHA — как разговор без слушания: каждый сам по себе.