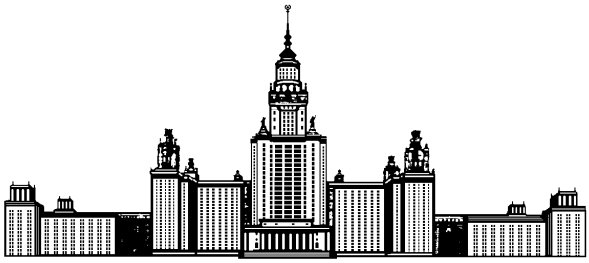

Московский государственный университет имени М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

Косарев Евгений Александрович

Оптимизация распределенного обучения больших языковых моделей

Distributed Large Language Models training optimization

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
Профессор РАН, д.ф.-м.н.,
Воронцов Константин Вячеславович

Москва, 2024

Содержание

1 Введение

Технологии анализа текстов непрерывно развиваются и помогают все более качественно решать целый спектр задач: от суммаризации документов [4] до их семантического парсинга [5]. В 2017 году была опубликована работа [1] Attention is all you need, которая представила архитектуру моделей машинного обучения, называемую трансформер. Архитектура подразумевает даличие двух блоков - Encoder и Decoder, которые могут встречаться как вместе внутри модели, так и по отдельности. Таким образом, сформировались три направления развития трансформеров:

1. **Encoder-only** модели. Они получили свою известность под названием BERT [2]. С их помощью возможно переводить тексты в векторное представление [8], исследовать модальности документов [7] и проводить классификацию текстов на любые домены [9]. В семействе трансформеров эти модели наиболее компактные по числу обучаемых параметров, в основном от 110M [10] до 1B у xxl версий (здесь и далее 1M - равен одному миллиону параметров, а 1B - одному миллиарду).

2. **Encoder-Decoder** модели. Наиболее известные модели с такой архитектурой это T5 [11, 12] и UL2 [13]. Они сочетают в себе оба блока трансформеров, и в отличие от BERT, способны генерировать текст. На момент написания диссертации в открытом доступе имеются модели T5 с 20B параметров.

3. **Decoder-only** модели. На сегодня это самый популярный класс трансформерных моделей, а его самые яркими и всемирно известными представителями являются ChatGPT и GPT-4. Эти алгоритмы способны с высоким качеством решать любые текстовые задачи. Самые большой представитель Decoder-only моделей в открытом доступе насчитывает 176B параметров [14].

Чем больше обучаемых параметров у модели, тем сложнее и дороже ее обучать. Масштаб проблемы продемонстрировала одна из первых популярных Decoder-only GPT-3. [3]. Для своего времени модель содержала рекордные 175B параметров, на обучение которых, по заявлениям авторов, было потрачено более 86 тысяч GPU-дней [6]. Таким образом, одним из главных барьеров на пути к качественным алгоритмам уровня GPT-4 становится стоимость обучения и его сложность.

1.1 Постановка задачи

Языковая модель — это алгоритм, который обучается предсказывать следующее слово (или часть слова) в тексте на основе предыдущего контекста. Под такими алгоритмом здесь и далее будем обозначать Decoder-only или, более кратко, семейство GPT моделей. Также, важно уметь не только правдоподобно продолжать текст, но и давать действительно качественные и логичные ответы. Исследователи из OpenAI предложили общую схему [16] обучения языковой модели, способной вести осмысленный диалог с пользователем на любую тему. Он состоит из трех этапов:

1. **Pretrain**. На этом этапе получается большая языковая модель, способная продолжать текст. Итоговое качество алгоритма зависит как от обучающих данных (терабайты текстовых данных), так и от архитектуры, скорости и стабильности обучения. Это самый ресурсоемкий этап обучения языковой модели, который может длиться месяцы.

2. **SFT (Supervised Fine-Tuning)**. С помощью асессоров или синтетических данных собирается множество инструкций, в котором модели задается формат общения, например диалоговый, и она учится правильно отвечать на поставленные вопросы. На собранных данных дообучается pretrain-модель. Объем обучающих на этом этапе обучения на порядки меньше, чем на стадии Pretrain, однако их качество значительно выше.

3. **RL (Reinforcement Learning)**. С помощью обучения с подкреплением (например, метода RLHF [17]) авторы предлагают выравнивать поведение модели после SFT этапа. Например, учить GPT давать политически нейтральные, вежливые и более полные ответы.

Такая схема обучения ChatGPT-подобных моделей последовательная, а значит напрямую зависит от качества модели, которая получится после pretrain-а. Диссертация будет сконцентрирована на оптимизации и улучшении алгоритмов больших языковых моделей на стадии pretrain, но ее результаты в области ускорения обучения и методик сбора данных можно будет использовать как на этапе SFT, так и в RL.

Введем некоторые понятия, которые позволят определить сложность обучения LLM.

Токен — атомарная единица текста, содержащая в себе минимальный объем информации. Это может быть как целое слово, так и подстрока (в случае эмодзи - последовательность байт). Чем больше токенов обработает модель на этапе претрейна, тем больше текста она увидит и тем лучше научится понимать окружающий мир и продолжать текст.

Бюджет — количество вычислительных ресурсов, которые могут быть выделены на обучение алгоритма. Под бюджетом понимают разные величины:

- Количество вычислительных ресурсов, GPU ускорителей, с помощью которых может быть обучен алгоритм
- Максимально допустимое время обучения алгоритма
- Число обработанных моделью токенов (другими словами, размер обучающего множества)

Внесем определенность в эти величины:

- Будем обучать модели для сравнения с open-source аналогами на 1024 GPU ускорителях NVIDIA TESLA A100 80GB и зафиксируем обучающие данные с общим объемом в 2T (два триллиона) токенов.

- Эксперименты с данными, гиперпараметрами обучения и архитектурой модели будем проводить на 128 GPU ускорителях NVIDIA TESLA A100 80GB и 100B (одна десятая триллиона) и 300B токенов.

Первая цель данного диссертационного исследования - оптимизировать обучение на доступных ресурсах, чтобы тратить бюджет максимально эффективно. Также важно минимизировать время работы кластера над обучением модели в этом бюджете, чтобы добиться лучших результатов. Таким образом, оптимизация распределенного обучения будет складываться из нескольких этапов:

1. Оптимизация архитектуры модели: при сохранении затраченных временных ресурсов на обучение и пройденного числа токенов во время обучения модель должна демонстрировать лучшее качество за счет оптимизации архитектуры.

2. Оптимизация инфраструктуры обучения: многие вычисления можно проводить более эффектино, экономнее расходуя вычислительный ресурс видеокарт, а также оптимизируя открыто доступные фреймворки. Эти оптимизации уменьшат время, затрачиваемое на обучение.

3. Оптимизация обучающих данных: гипотеза состоит в том, что чем чище и информативнее данные, тем быстрее и эффективнее обучается модель и тем лучшие результаты она демонстрирует. В диссертации предлагаются методы фильтрации данных, которые позволят улучшить качество модели.

Вторая цель - получить модель качественнее двух самых современных на текущий момент open-source моделей:

1. Сбербанк в июле 2023 года выложил в общий доступ самую новую русскоязычную GPT модель под названием ruGPT-3.5 13B [15]

2. Также в июле 2023 года в открытом доступе появилась одна из самых сильных по производительности англоязычных моделей LLaMa2 7B [20].

1.2 План решения

Для получения конкурентноспособной модели необходимо провести исследование доступных решений с точки зрения:

1. Архитектуры pretrain-модели так, чтобы при заданном бюджете и заданном обучающем множестве минимизировать функцию потерь.

2. Эффективности кодовой базы обучения модели на кластере.

3. Разнообразия и качества обучающих данных.

Так как обучение на кластере довольно дорогое, то тестирование всех изменений (кроме оптимизации кодовой базы) будем проводить на более дешевых моделях с точки зрения числа обучаемых параметров и бюджета обучения. Для сравнения качества моделей будут выбраны бенчмарки, отражающие самые разные аспекты знаний, содержащихся в языковых моделях. Итоговым результатом диссертации станет обучение большой языковой модели со всеми предложенными оптимизациями на бюджете в 2T токенов.

2 Архитектура модели

Для начала необходимо выбрать одну из трех парадигм обучения трансформерных моделей. Одной из задач диссертации ставится обучение модели более качественной, чем ruGPT-3.5 13B и LLaMa2 7B. Это значит, что итоговый алгоритм должен иметь способность генерировать текст, поэтому Encoder-only архитектуры не подходят. Согласно [19], PALM, имеющая архитектуру Encoder-Decoder, проигрывает при сопоставимом числе обучаемым параметрам современным Decoder-only моделям по качеству, поэтому остановим выбор на Decoder-only архитектурах.

Decoder-only моделей обучено довольно много, наиболее известные из них это GPT3 [3], BLOOM [14], MPT [21], LLaMa и LLaMa2 [19, 20]. Все они отличаются числом и порядком слоев, числом обучаемых параметров, разными функциями, которыми осуществляются внутренние преобразования результатов вычислений внутри моделей. Статьи [19, 20] демонстрируют преимущество LLaMa-архитектуры над остальными в множестве бенчмарков, поэтому их способ построения модели будет проанализирован и улучшен в этой работе.

2.1 Decoder-only

Языковые модели при генерации текстов оперируют частями слов, то есть токенами, каждый из которых имеет свой уникальный номер. Принцип, по которому тексту сопоставляется кодирование токенами, называется **токенизацией**, а алгоритм, выполняющий эту процедуру, — **токенизатором**. На Рис. 4 продемонстрировано, как может работать токенизатор:

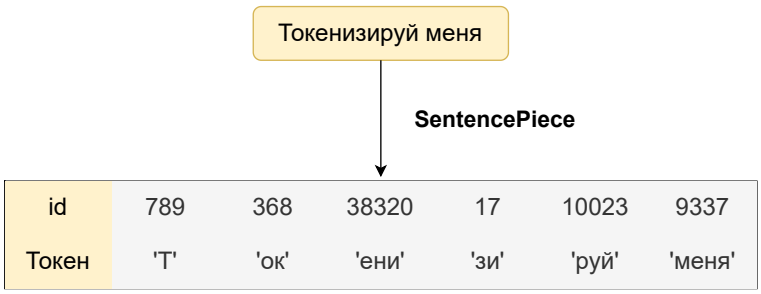

Рис. 1: Пример работы токенизатора

Токенизатор всегда имеет **словарь** — известный конечный набор токенов (здесь

и далее размер словаря равен V), на который он может разбить входной текст. Раз набор конечен, то каждому токену из него в языковой модели сопоставляется вектор определенной размерности, а тексту - набор этих представлений, то есть матрицу. Формализируем эту концепцию:

1. На вход подается текст T, после его токенизации получим $Tokenize(T) = [t_1, \ldots, t_n]$, где t_i - номер i-го токена, а n - необходимый объем токенов, которыми кодируется данный текст. Набор $[t_1, \ldots, t_n]$ называют **контекстом** языковой модели, а n - **размером контекста**.

2. Языковая модель — вероятностный алгоритм $f(T)$, поэтому его задача получить вероятностное распределение следующего токена t_{n+1} по контексту:

$$f(t_{n+1}|T) = \mathbb{P}(t_{n+1}|t_n, t_{n-1}, ..., t_1)$$

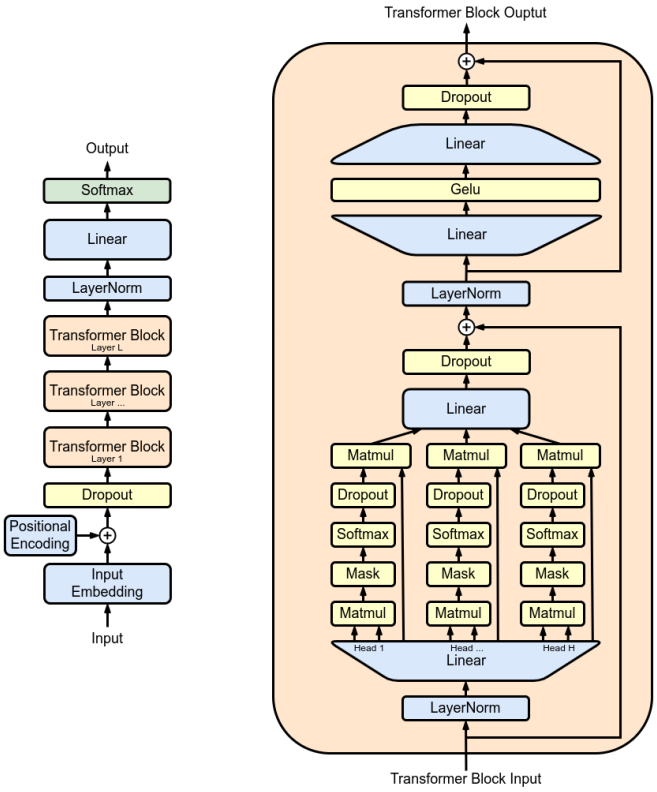

Рис. 2: Архитектура GPT-2

Изначально, GPT-архитектура выглядела так, как показано на Рис. 2. Распишем подробнее, как происходит обработка токенов на каждом блоке слоев:

1. Слой Input Embedding сопоставляет каждому токену вектор размера $[hidden\_size]$, конкатенирует их и выдает матрицу размером $[n, hidden\_size]$.

2. Слой Positional Encoding добавляет к представлениям векторов позиционную информацию (например, как это делается в [1]). Так как все дальнейшие операции внутри GPT инвариантны с точки зрения результата работы модели к порядку токенов в матрице эмбедингов, то это единственное место, в котором в модель подается информация о порядке следования токенов. У входа в слой та же размерность, что и у выхода из него.

3. Действие слоя Dropout описано в статье [22]. У входа та же размерность, что и у выхода.

4. Слой трансформера (Transformer Block) композитный - в нем вычисляется блок внимания (Attention) и проводится внутреннее преобразование (MLP).

   (a) Производится пре-нормализация входной матрицы через LayerNorm [23]. У входа та же размерность, что и у выхода.

   (b) Вычисляется блок Attention, здесь представлена его модификация Multi-Head Attention [1].

   (c) Слой MLP сочетает в себе два линейных преобразования $f(X) = W \times X + b$ и функцию активации между ними. Здесь формула выглядит следующим образом:

   $$MLP(X) = Gelu(X \times W_{in} + b_{in}) \times W_{out} + b_{out},$$

   где $W_{in}^{[hidden\_size, intermediate\_size]}$, $b_{in}^{[intermediate\_size]}$, $W_{out}^{[intermediate\_size, hidden\_size]}$, $b_{out}^{[hidden\_size]}$, а Gelu сохраняет размерность.

5. Последний Linear слой — линейное преобразование $f(X) = W_{lin} \times X + b_{lin}$, где матрицы имеют размеры: $W_{lin}^{[hidden\_size, V]}$, $b_{lin}^{[V]}$.

6. Операция Softmax нормирует результат предыдущего слоя по размерности V:

   $$Softmax(X)_{i,j} = \frac{e^{X_{i,j}}}{\sum_j e^{X_{i,j}}}$$

   Таким образом, модель генерирует вектор-распределение следующего токена, основанный лишь на информации о предыдущих токенах в контексте.

Отметим, что открытая русскоязычная модель ruGPT-3.5 13B была построена на этой архитектуре.

3 Обучение языковой модели

Языковая модель предсказывает следующий токен на основе предыдущих. Обучение производится в режиме «без учителя»: для этого необходимо собрать текстовые данные, затем токенизировать их и обновить веса одним из методов оптимизации через минимизацию функции потерь.

3.1 Подготовка батча

Для ускорения обучения алгоритмов используется понятие батчей, когда модель одновременно обрабатывает несколько документов одновременно. Это эффективно с точки зрения вычислительных ресурсов, так как варьирование размера батча позволяет по максимуму использовать вычислительные ресурсы GPU. В случае языковых моделей батч — это матрица размера $B \times N$, где N - размер контекста, а B - размер батча.

Пусть имеются текстовые документы T^1, T^2, после токенизации они кодируются последовательностями токенов K^1 и K^2 $[t_1^i, t_2^i, ..., t_{K_i}^i], i \in \{1, 2\}$, где нижний индекс токена обозначает порядковый номер в кодировании, а верхний - принадлежность тексту. Модель обучается на текстовых данных с размером контекста - N, и пусть без ограничения общности $K^1 \geq N$. Тогда тексты разбиваются на последовательно идущие подпоследовательности токенов длины N (последний кусок будет короче) и упаковываются в батч. Если кусок текста короче контекста, то он дополняется специальными токенами <PAD>, называемыми **паддингами**:

t_1^1	t_2^1	t_3^1	...	t_N^1
t_{N+1}^1	t_{N+2}^1	t_{N+3}^1	...	t_{2N}^1
t_{2N+1}^1	<PAD>	<PAD>	...	<PAD>
t_1^2	t_2^2	t_3^2	...	t_N^2
t_{N+1}^2	t_{N+2}^2	<PAD>	...	<PAD>

Таблица 1: Заполнение токенами батча рарзмером $5 \times N$

Важно заметить, что токен паддинга не считается функциональным токеном: модель не обучается его генерировать, по нему не считается значение функции потерь.

3.2 Функция потерь

По определению языковая модель авторегрессионна, то для последовательности токенов $[t_0, t_1, ..., t_N]$ она должна уметь:

- По токену $[t_0]$ предсказывать следующий токен t_1
- По токенам $[t_0, t_1]$ предсказывать следующий токен t_2
- По токенам $[t_0, t_1, t_2]$ предсказывать следующий токен t_3
- ...
- По токенам $[t_0, ..., t_{N-1}]$ предсказывать следующий токен t_N

Пусть X - входные данные языковой модели f, y - ожидаемый результат работы. По рассуждениям выше можно заметить, что X и y это фактически одна и

та же матрица, сдвинутая первая относительно второй на один токен. Формализуем вычисление функции потерь для языковых моделей: если матрица X имеет форму $B \times N$, то $f(X)$ выдает матрицу размера $B \times N \times V$, так как результат работы GPT - матрицу вероятностей следующих токенов. Составим матрицу $\hat{y}$, которая отличается от y дополнительной размерностью, состоящей из 0 и 1: номеру правильного токена y_j сопоставим вектор, состоящий из 0 и 1 на позиции j. Тогда, матрицы $f(X)$ и $\hat{y}$ имеют одинаковую форму и можно записать кросэнтропийную функцию потерь:

$$L_{cross-entropy}(f(X), \hat{y}) = -\frac{1}{BN} \sum_{b=1}^{B} \sum_{n=1}^{N} \sum_{v=1}^{V} \hat{y}_{b,n,v} * \log \frac{e^{f(X)_{b,v,n}}}{\sum_{c=1}^{V} e^{f(X)_{b,c,n}}}$$

Минимизация этой функции позволяет алгоритму максимизировать вероятность того токена, который доложен следовать из контекста. Еще раз напомним, что обучение происходит в режиме «без учителя»: по входу в модель конструируется целевой результат, а раз так, то для обучения GPT достаточно скачать множество текстовых данных из интернета (или вообще весь интернет).

4 Обучающие данные

Обучающие данные возьмем из открытых источников. Языковая модель тем эффективнее обучается, чем более информативные текстовые данные были поданы ей на обучение. Цель диссертации - получение русскоязычной модели на заданном бюджете в 2T токенов, поэтому большая часть обучающих данных должна быть на русском языке. Этапы фильтрации данных, определение языка, причины выбора именно этих наборов данных и их пропорций обсудим в этом и 8-м параграфе. Таким образом, обучающее множество будет составлено из следующих доменов:

1. **CommonCrawl (CC)**. Веб-данные будем брать из CommonCrawl [24]. Это ресурс, который занимается скачиванием всех веб данных в одно хранилище. В моей работе были скачаны текстовые данные с января 2020 по август 2023 года включительно.

   en-CC. Часть текстовых данных, написанных преимущественно на английском языке.

   ru-CC. Часть текстовых данных, написанных преимущественно на русском языке.

2. **wikipedia.org**. Тексты статей были взяты в открытом доступе в агрегированным виде с ресурса HuggingFace [25]. В нем содержатся множество актуальных страниц с википедии.

3. **Proof-Pile-2**. Этот открытый набор обучающих текстовых данных [26] состоит из многих доменов, в обучение возьмем три из них:

   Lemma. Набор математических данных, содержащий задачи, формулы и теоремы.

arxiv.org. Сборник научных статей на самые разные темы.

algebraic-stack. Текстовые данные математического кода, включающий численные вычисления, линейную алгебру и формальную математику. Код представлен на более чем пятнадцати языках программирования, включающих C++, Python, Matlab и другие.

4.1 Dataset-1

По ходу диссертации обучающие данные, их состав и качество будут меняться и оптимизироваться. Первоначальное обучающее множество назовем Dataset-1, пропорции данных отображены на Рис. 3.

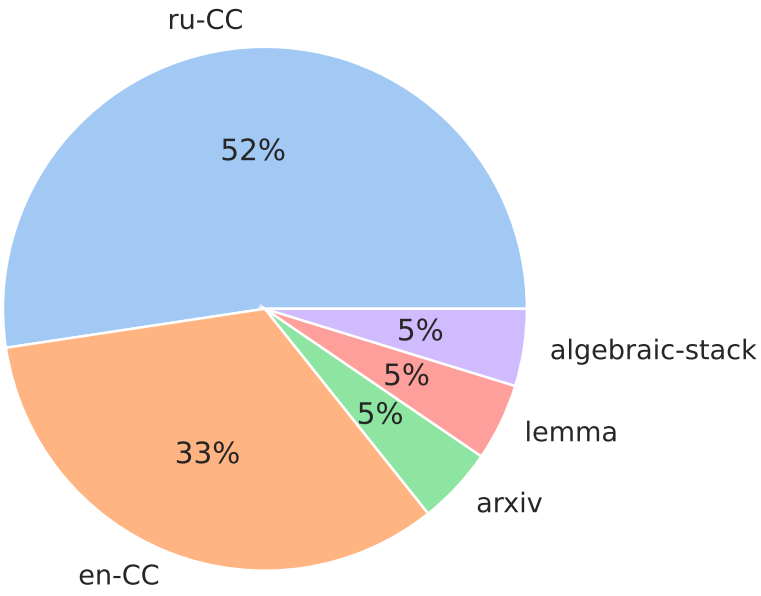

Рис. 3: Разбиение данных по доменам в обучающем множестве Dataset-1

Здесь и далее обучающие множество будет составляться по следующим правилам:

- Каждый домен преобразуется в токены с помощью выбранного токенизатора
- Домен перемешивается в случайном порядке
- Далее каждый домен разбивается на микробатчи размера $1 \times N$, где N - размер контекста модели

- Пропорции можно воспринимать как вероятностное распределение, семплированием из которого выбирается домен, из которого следующий микробатч размера 1 отправится на формирование в обучающее множество.

5 Оптимизация архитектуры

За несколько лет архитектуру трансформеров пытались по-разному оптимизировать, изменяя слои и их размерности, однако для русского языка такого исследования еще не проводилось. Зафиксируем бюджет в 100B обучающих токенов из Dataset-1 (сохраняем пропорции данных), будем сравнивать качество модели по значениям функции потерь, меньше - лучше. В базовой архитектуре выберем параметры, как в Таблице 2. Колонка Params обозначает число обучаемых параметров модели при заданных параметрах архитектуры. Здесь и далее контекст модели на этапе обучения будет равен 4096 токенам.

Params	vocab_size	hidden_size	intermediate_size	n_heads	n_layers
2.6B	50272	2560	10240	20	32

Таблица 2: Архитектура для тестов модификации модели

5.1 Позиционные эмбединги

Позиционные эмбединги отвечают за кодирование позиций токенов, чтобы языковая модель понимала, в каком порядке они идут друг относительно друга. Матрица позиционных эмбедингов имеет размеры $[n, hidden\_size]$, где n - размер контекста, равен 4096, и вычисляется по следующим формулам:

$$W_{PE}[pos, i] = \begin{cases} sin(\frac{pos}{1000^{2k*hidden\_size}}) & \text{,если i} = 2\text{k} \\ cos(\frac{pos}{1000^{2k*hidden\_size}}) & \text{,если i} = 2\text{k} + 1 \end{cases}$$

У таких эмбедингов имеются следующие проблемы:

1. Позиционная информация добавляется единожды в начале модели и после каждого слоя трансформера сигнал о позиционном кодировании ослабевает.

2. В тексте важна не абсолютная, а относительная нумерация токенов друг относительно друга. Если добавить предложение в середину или в конец текста, то модель воспримет его по-разному, так как текущий вариант кодирования зависит от абсолютной позиции в контексте.

Первую часть можно попытаться исправить, добавляя позиционные эмбединги перед каждым Attention слоем. Однако, это ухудшит значения loss функции:

Заменим позиционное кодирование с помощью роторных эмбедингов RoPE [27]. Они исправляют оба недостатка прежних эмбедингов, так как используются на

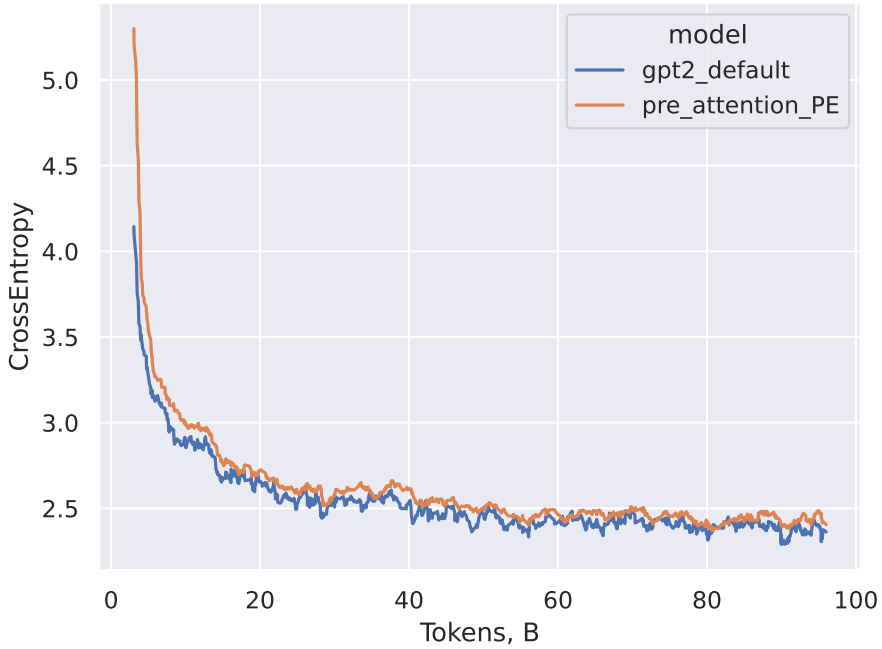

Рис. 4: Добавка абсолютных эмбедингов до блока внимания

этапе вычисления блока внимания. Пусть имеются два токена на позициях m и n в контексте (токены x_m и x_n). В блоке внимания они преобразуются путем домножения Query и Key матрицы: $f(x)_Q^m = x_m Q^T$ и $f(x)_K^n = x_n K^T$, а далее эти результаты перемножаются. Внесем в перемножегние позиционную информацию: $g(x_m, x_n, m-n) = Re[f(x)_Q^m (f(x)_K^n)^T e^{i(m-n)10000}]$. Функция потерь отображена на Рис. 7 под названием gpt2_rope. Получено уменьшение функции потерь.

5.2 Пре-нормализация

В обучении больших языковых моделей важна численная стабильность вычислений. В работе [28] авторы утверждают, что изменения порядка нормализации уменьшает в $\sqrt{L}$ раз нормы градиентов, где L - число слоев в трансформере, а также снижает значение функции потерь. Изменения в архитектуре схематично изображены на Рис. 5.

В рассматриваемой модели на русском языке такая модификация также придает эффект уменьшения значений функции потерь на Рис. 7 под названием gpt2_rope_preln.

5.3 Замена LayerNorm на RMSNorm

LayerNorm [29] центрирует по среднему и разбросу приходящую в него матрицу. Таким образом, распределение чисел в матрице остается инвариантным, что способствует улучшению сходимости нейронных сетей. В отличие от LayerNorm, RMSNorm

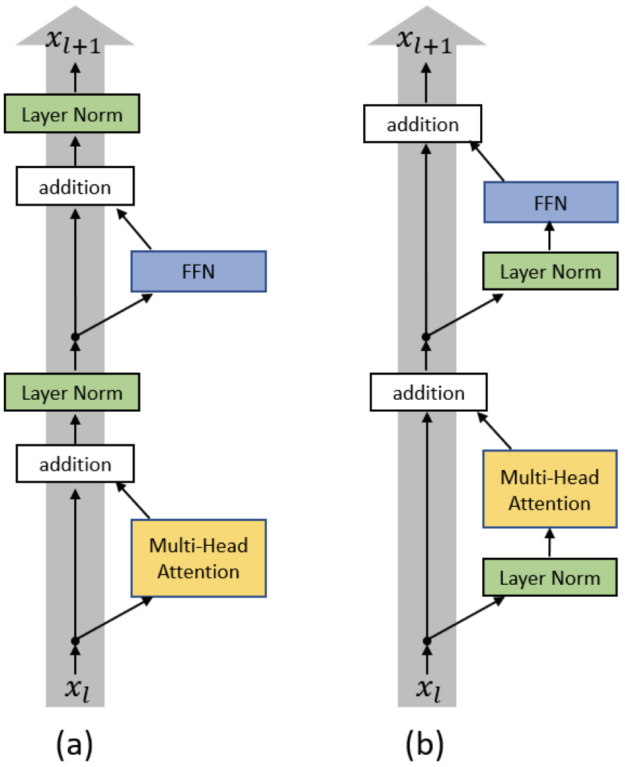

Рис. 5: Изменение порядка нормализации слоев [28]. (a) - исходная архитектура, (b) - обновленная.

[30] не центрирует входные матрицы по среднему.

Пусть $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$. Тогда нелинейное преобразование в общем виде выглядит так:

$$a_i = \sum_{j=1}^{m} w_{ij} x_j$$

$$y_i = f(a_i + b_i)$$

И LayerNorm можно описать следующими формулами:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2} y_i = f\left(\frac{a_i - \mu}{\sigma} g_i + b_i\right)$$

В свою очередь экспериментально RMSNorm демонстрирует лучшие результаты в сравнении с LayerNorm и высчитывается так:

$$y_i = f\left(\frac{a_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}} g_i + b_i\right)$$

Вместе с заменой нормализующего слоя будет также убран dropout. Гипотеза состоит в том, что в языковой модели очень много параметров, поэтому dropout просто

замедляет обучение и является избыточным механизмом регуляризации. Модификация под названием gpt2_rope_preln_rmsnorm на Рис. 7 также уменьшает функцию потерь.

5.4 Замена функций активации

Функция активации в языковых моделях - это нелинейное преобразование, находящееся внутри MLP слоя. В исходной архитектуре используется функция GeLU. Напомним, что $MLP_{GeLU}(X) = Gelu(X \times W_{in} + b_{in}) \times W_{out} + b_{out}$. В статье [31] авторы рассматривают различные функции активации для Encoder-Decoder архитектуры трансформера. Проведем аналогичное исследование на Decoder-only архитектуре. Рассмотрим стандартные функции активаций [32, 33]:

$$\text{ReLU}(X) = Max(0, X) \text{ - здесь максимум поэлементный}$$

$$\text{GeLU}(X) = x\Phi(x) \text{ - здесь } \Phi(x) = P(X \leq x), X \sim \mathcal{N}(0, 1)$$

$$\text{Swish}(X) = X\sigma(X) \text{ - здесь } \sigma(X) = \frac{1}{1 + e^{-X}} \text{ применяется поэлементно}$$

А также gated активации, то есть обучаемые [34] ($\otimes$ - поэлементное произведение матриц):

$$\text{ReGLU}(X) = \text{ReLU}(X) \otimes (X \times W + b)$$

$$\text{GEGLU}(X) = \text{GeLU}(X) \otimes (X \times W + b)$$

$$\text{SwiGLU}\,(X) = \text{Swish}(X) \otimes (X \times W + b)$$

Дополнительно попробуем отключить линейную добавку из всех слоев в нашей архитектуре(bias и no_bias). В этом эксперименте значения функции потерь близки друг к другу. Для наглядности составим таблицу со отобразим среднее значение функции за последние 100 градиентных шагов для каждого эксперимента:

Функция активации	ReLU	GeLU	Swish	ReGLU	GEGLU	SwiGLU
Loss with bias	2.259	2.246	2.252	2.211	2.196	2.191
Loss with no_bias	2.274	2.249	2.257	2.214	2.193	2.192

Таблица 3: Значения функции потерь для различных функций активации

По таблице 3 можно сделать вывод, что gated активации имеют меньшие значения функции потерь за счет увеличения числа обучаемых параметров. С точки зрения качества, линейная добавка (bias) влияла сильнее на обучение со стандартными функциями активаций, наименьшее значение функции потерь имеют архитектуры с GEGLU и SwiGLU. GEGLU вычисляет функцию ошибок Гаусса, что является вычислительно более трудоемким процессов, чем подсчет сигмоиды в SwiGLU. С точки зрения размера модели - он вырос до 3.5B, так как в MLP появилась

новая функция активации и скорость обучения замедлилась на 8% относительно gpt2_rope_preln_rmsnorm. В дальнейших двух пунктах будет показано, как уменьшить размер модели и еще больше повысить качество обучения. Текущий эксперимент на Рис. 7 называется gpt2_rope_preln_rmsnorm_swiglu.

5.5 Модификация MLP

Произведем еще одну модификацию слоя MLP. В статье [19] утверждается, что на итоговый результат влияет коэффициент $C = \frac{\text{intermediate\_size}}{\text{hidden\_size}}$ и лучший C = 8/3, вместо C=4, как сейчас в модели. Проверим это утверждение для выбранной архитектуры и текущего обучающего множества. Переберем значения C по сетке, а также усредним значения функции потерь по последним 100 градиентным шагам: результаты в Таблице 4. У LLaMa 2 этот коэффициент равен 8/3, что близко к 2.5, поэтому эксперимент с C=2.5 не будем проводить.

Лучший коэффициент C=3.5, это уменьшило размер модели до 3.2B обучаемых параметров и уменьшило значение функции потерь. Текущий эксперимент на Рис. 7 называется gpt2_rope_preln_rmsnorm_swiglu_mlpdim.

	C = 1	C = 1.5	C = 2	C = 8/3	C = 3.0	C = 3.5	C = 4.0
Loss	2.389	2.339	2.289	2.223	2.185	2.181	2.192

Таблица 4: Значения функции потерь для различных функций активации

5.6 Модификация Attention

Согласно [1], блок внимания вычисляет по следующей формуле:

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

Здесь матрицы Q, K, V получены следующим линейным преобразованием входа в слой X:

$$Q = X * W_Q, K = X * W_K, V = X * W_V$$

В работе [35] предлагается внедрить оптимизацию Grouped Query Attention: она ускоряет как обучение, так и инференс языковой модели, снижая число параметров и вычислений, и практически не влияет на качество обучения. Матрица Q, K, V имеют размер $[intermediate\_size, hidden\_size] = [intermediate\_size, head\_size \times num\_heads]$. Предлагается для K и V матриц сократить число голов для Multi-Head Attention путем группировки: если ранее каждая голова K и V матриц соответствовала одной Q голове, то теперь каждая голова K и V матриц соответствует уникальной группе из q голов матрицы Q. Схематично это показано на Рис. 6.

Выберем характерный размер группы равный 4. Таким образом, матрицы W_K и W_V становятся в 4 раза меньше, модель теперь имеет ровно 3B обучаемых параметров

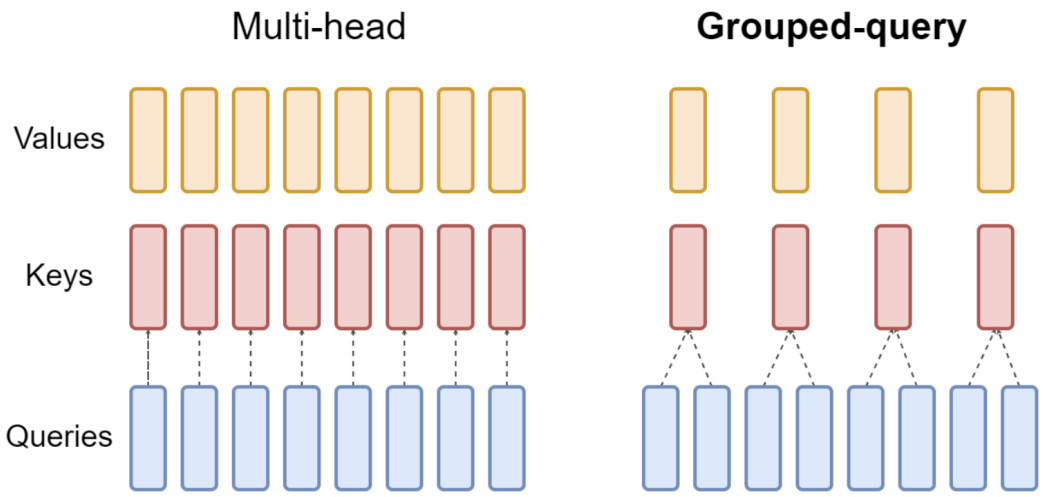

Рис. 6: Иллюстрация работы Grouped Query Attention в сравнении с Multi-Head Attention [35]

и обучается быстрее на 10 процентов. В итоге, число параметров увеличилось по сравнению с экспериментов gpt2_rope_preln_rmsnorm, однако скорость обучения практически не поменялась.

Новый эксперимент gpt2_rope_preln_rmsnorm_swiglu_mlpdim_qga на Рис. 7 имеет такое же значение функции потерь, как и gpt2_rope_preln_rmsnorm_swiglu_mlpdim.

5.7 Выводы

Исходная модель имела 2.6B обучаемых параметров, новая - 3B. При этом скорость обучения у них одинаковая и оптимизацией архитектуры удалось добиться снижения функции потерь на 13% до значения 2.181. Все эксперименты и графики обучения сгруппированы на Рис. 7.

6 Оптимизация вычислений и стабильность обучения на кластере

Обучение большой языковой модели на кластере подразумевает оптимизацию вычислений с точки зрения повышения эффективного кода, анализа расхождений модели и поиска узких мест. Разработка архитектуры обучения большой языковой модели велась на фреймворке PyTorch с использованием технологии распределенного обучения FSDP [45]. Была проделана работа по каждому из трех направлений. Замеры будем проводить для 7B модели, архитектура которой представлена в Таблице 12, и 29B модели, она получена увеличением числа слоев и их внутренних размерностей.

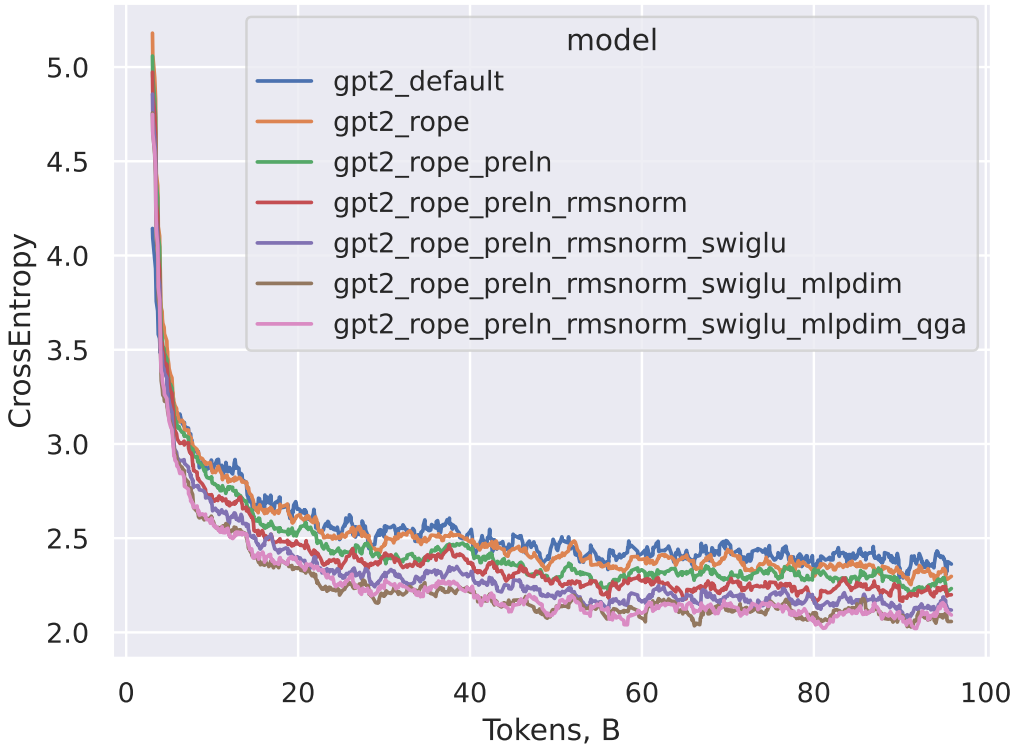

Рис. 7: Сравнение последовательных улучшений архитектуры друг с другом

6.1 Fused операции

Фреймворк PyTorch можно рассматривать как интерфейс, который вызывает команды, исполняющиеся на видеокартах. Общая идея объединенных (Fused) операций заключается в том, то вместо последовательных вызовов операций из PyTorch, можно написать на CUDA объединенную (Fused) операцию и использовать ее. Схематично реализация Fused операций продемонстрирована на Рис. 8.

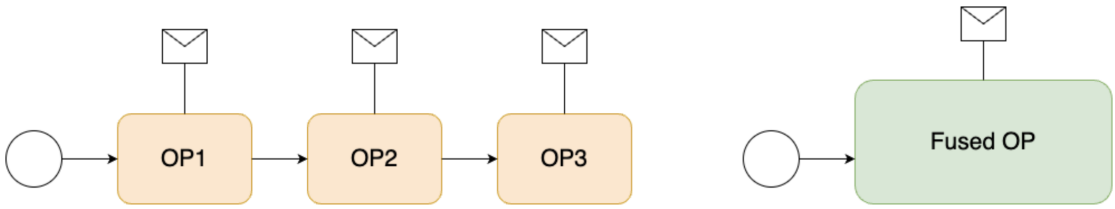

Рис. 8: Демонстрация общей идеи fused операций

Автор алгоритма Flash Attention [46] использует эту идею и на Рис. 9 демонстрирует, что использование Fused операций может существенно сократить время вычисления блока Attention и эффективнее использовать вычислительные ресурсы GPU.

В диссертации удалось реализовать еще несколько Fused операций: Rotary

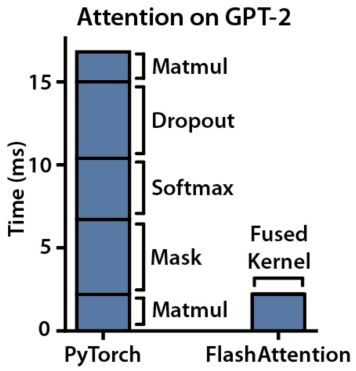

Рис. 9: Сравнение времени вычисления Attention напрямую и через ядро Flash-Attention [46]

Embedding, RMS-Norm и Cross Entropy Loss. Для замеров ускорения используем 29B модель и запустим измерения на 128 GPU Tesla A100. Базовая реализация модели исключительно на PyTorch имеет самую низкую скорость, а последовательное добавление Fused операций дает ускорение до 75.7%. Результаты продемонстрированы в Таблице 5.

—	throughput (batches/sec)	+ %
PyTorch	0.109	0.00
Flash-Attention v2	0.177	62.73
Fused Rotary Embedding	0.181	66.15
Fused RMS-Norm	0.189	73.29
Fused Cross Entropy Loss	0.192	75.71

Таблица 5: Кумулятивное ускорение от добавления fused операций

6.2 NaN в обучении

Во время обучения выполняется множество умножений матриц, а также других численных операций, которые могут приводить к численным нестабильностям, а именно к появлению значений NaN. На графиках обучения на Рис. 10 можно заметить треугольники - так отображаются NaN значения функции потерь. Такое значение может возникать как на Forward, так и на Backward шаге обучения, и обновлять веса такими значениями нельзя - это приведет к остановке обучения модели.

С одной стороны можно пропускать такой батч, где возникла численная нестабильность. Однако на практике это приводит к тому, что до 25 % батчей пропускаются и кластер перестает эффективно использоваться, а число NaN по ходу обучения не

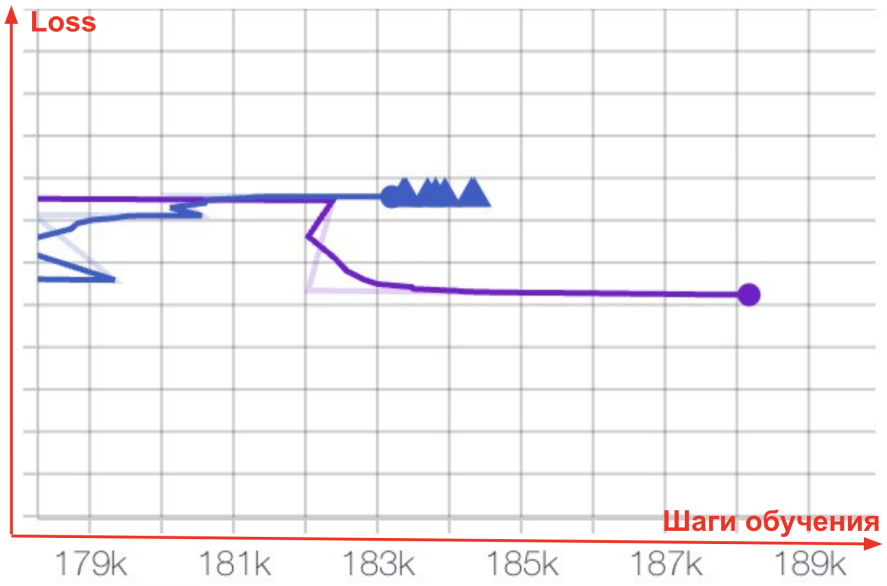

Рис. 10: Появление NaN в значении функции потерь при обучении (изоюражен сглаженный график)

уменьшается. Чтобы иметь возможность обучать модель эффективно дальше, предлагается оптимизировать часть Backward шага, а именно операцию ReduceScatter. Она показана на Рис. 11 и выполняет следующие действия:

1. Принимает на вход градиенты с каждой видеокарты.
2. Усредняет градиенты по всем входящим данным в рамках одного батча обучения.
3. В распределенном режиме рассылает на каждую видеокарту принадлежащую ей часть усредненного градиента.

Изменим шаг 2 и будем полностью заполнять нулями матрицы градиентов, в которых присутствует NaN. Таким образом, модель будет обучаться лишь на той части батча, которая была вычислена корректно.

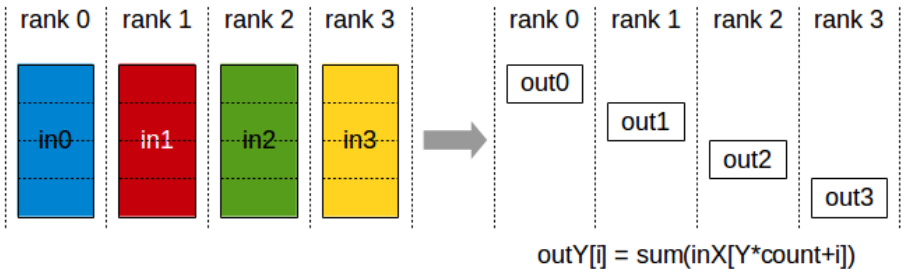

Рис. 11: Операция Reduce Scatter

Введем график количества занулений градиентов, содержащих NaN (Рис. 12). Можно заметить, что в начале, когда батчи просто пропускались, график рос, а

после включения оптимизации упал до нуля. Таким образом, удалось восстановить эффективность обучения и исключить холостую работу кластера.

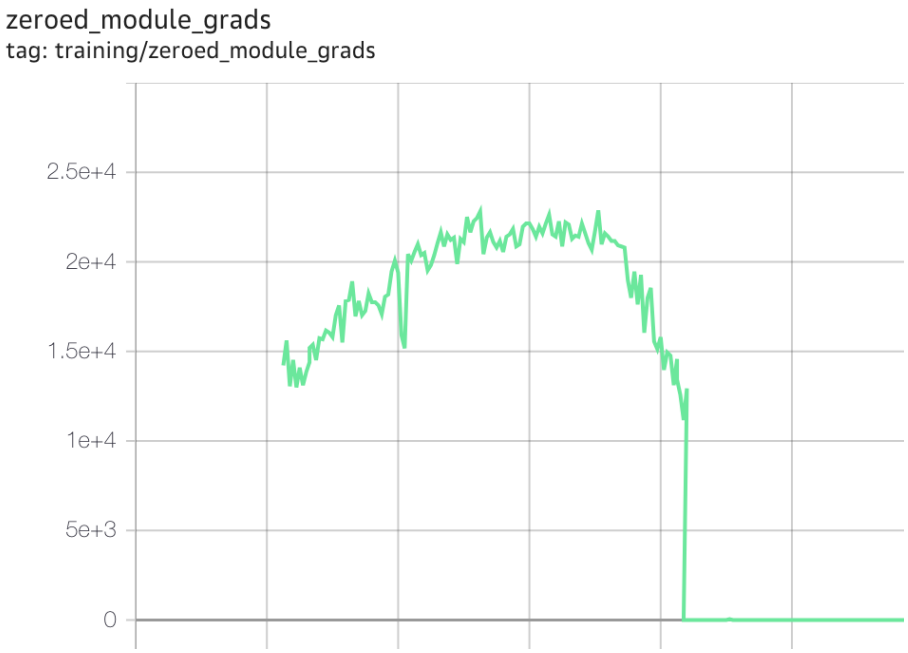

Рис. 12: NaN в обучении

6.3 Инициализация модели

Распределенное обучение исполняется на сотнях или даже тысячах видеокарт на кластере, который в свою очередь состоит из множества взаимосвязанных компонент. При наличии столь сложной системы высока вероятность выхода из строя некоторых компонент, что остановит обучение и вызовет его перезапуск на работоспособной части кластера. На протяжении исследования случались ситуации, когда за неделю происходило до 20 остановок, что существенно тормозило обучение. Исследование показало, что при перезапусках самой долгой частью является загрузка чекпоинта и инициализация модели, эта часть обучения будет оптимизирована.

6.3.1 CPU и GPU инициализации

Стандартные инициализации модели происходят полностью на одном модуле - либо на процессоре, либо на видеокарте. Далее модель рассылается по вычислительным узлам.

CPU

Достоинства

- При инициализации модели можно использовать до 2TB оперативной памяти.

Недостатки

- После инициализации требуется провести GPU-CPU коммуникации, чтобы расположить модель на видеокартах.

GPU

Достоинства

- Отсутствие коммуникаций с процессором при инициализации модели, она сразу располагается на GPU.
- У GPU быстрая память, что сокращает время инициализации модели

Недостатки

- Память ограничена 80GB

Можно заметить, что инициализации в определенном смысле противоположны друг другу: GPU быстрее, но имеют меньше памяти, а процессор, наоборот, медленнее, но на нем можно инициализировать большие модели.

6.3.2 Meta инициализация

Реализация этого подхода основана на meta-init технологии PyTorch FSDP. Основная идея заключается в том, что нет необходимости инициализировать модель на процессоре полностью. Так как алгоритм разбиения модели между видеокартами детерминированный, то можно заранее определить, на каком устройстве окажется та или иная часть весов модели. Поэтому, происходит отложенная инициализация: на CPU создается модель без реальной инициализации весов, далее происходит распределение весов между GPU, и наконец на видеокартах происходит инициализация принадлежащей им части весов. Случается и так, что инициализация весов бывает зависимая между видеокартами. Тогда на каждой видеокарте создается полная копия тензора согласно логике инициализации, и дальше используется лишь та часть весов, которая соответствует видеокарте.

Этот подход использует все преимущества CPU и GPU инициализаций - сокращает потребление оперативной памяти и сохраняет высокую скорость инициализации модели. Сравним все подходы на 7B и 29B моделях, результаты находятся в Таблицах 6,7.

—	Init	Init с чекпоинта (мин)	Peak CPU mem (GB)	Peak GPU mem (GB)
CPU	4.6	10.0	130	0.12
GPU	1.6	3.8	53	79
Meta	0.8	2.6	53	0.12

Таблица 6: 7B sharded

—	Init	Init с чекпоинта (мин)	Peak CPU mem (GB)	Peak GPU mem (GB)
CPU	16.5	31.3	389	0.5
GPU	–	–	–	–
Meta	3.1	6.9	58	0.5

Таблица 7: 29B sharded

По итогу потребление памяти, используемой процессором, на 7B снижается в 2,45 раз, а на 29B в 6,7 раз. Время инициализации с чекпоинта снижается на 7B снижается в 3,84 раза, а на 29B в 4,53 раза. Важно заметить, что на GPU инициализировать 29B модель не получилось, так как не хватило объема памяти видеокарты.

7 Токенизатор

Выбор токенизатора определяет те единицы текста, которыми оперирует модель. В предыдущих секциях был выбран токенизатор от ruGPT-3.5-13B. Одной из проблем этого токенизатора является обработка чисел и формул, как это представлено на Рис. 13. Можно заметить, что различным числам ставятся в соответствие отдельные токены и всего числовых токенов в словаре содержится 253 штуки. Одна из гипотез этой диссертации состоит в том, что при таком подходе не наберется достаточно статистики о со встречаемости токенов и математические задачи будут решаться на более низком уровне. Решением этой проблемы является токенизация чисел по цифрам, то есть введение 10 токенов от ’0’ до ’9’. Таким образом, число 1234 будет разбиваться на токены [1, 2, 3, 4].

```
11 + 4 = ? -> [1967, 3015, 1019, 3367, 2563]
22 + 4 = ? -> [3966, 3015, 1019, 3367, 2563]
```

Рис. 13: Токенизация арифметического выражения.

Также, в выбранном ruGPT-3.5 токенизаторе неизвестно, какое распределение языков было выбрано на этапе построения токенизатора и насколько эффективно обрабатывается русский и английский языки. Чтобы проверить это, обучим выбранную модель на Dataset-1 и сравним производительность модели на разных токенизаторах. Для этого сравнение производительности по значениям функции потерь не подходит, так как вычисление функции зависит от размера словаря модели (он же равен размеру токенизатора) и не отражает улучшение или ухудшение понимания модели различных текстовых данных. Для этого определим набор замеряемых бенчмарков.

7.1 Бенчмарки

Рассмотрим несколько бенчмарков, отражающих основные аспекты работы модели:

1. **MMLU** - бенчмарк [36] оценивает общие знания модели о мире в формате теста в 57 тематиках - дается вопрос и 4 варианта ответа, модели необходимо выбрать верный. Оценивается по метрике accuracy в режиме 5-shot.

2. **ruMMLU** - бенчмарк MMLU, переведенный на русский язык. Проверяет понимание тех же знаний, что и MMLU при переходе на русский язык. Оценивается по метрике accuracy в режиме 5-shot.

3. **GSM8K** - бенчмарк [37] содержит математические задачки школьного уровня, модели необходимо в формате рассуждения прийти к правильному ответу и напечатать его отдельно. Оценивается по метрике accuracy в режиме 8-shot.

4. **HumanEval** - бенчмарк [38] оценивает умение модели писать код: дается задание и тесты, написанный моделью код должен их пройти. Замеры составляются по метрике pass@1.

5. **ruOpenBookQA** - бенчмарк [39] проверяет базовые естественнонаучные знания о мире на русском языке в формате теста аналогично MMLU. Оценивается по метрике accuracy.

Таким образом, по выбранным бенчмаркам можно оценить уровень общих и научных знаний модели, а также умения писать код и решать математические задачи.

7.2 Эксперименты

Обучать будем подобранную в предыдущем пункте 3B модель на бюджете в 320B токенов с несколькими вариантами токенизаторов. Попытаемся подобрать оптимальный размер токенизатора, а обучающие данные токенизатора возьмем из Dataset-1.

Будем отталкиваться от Open-Source моделей при выборе размера токенизатора:

- GPT-2, GPT-3 [18] и ruGPT-3.5-13B[15] модели в основном моноязычные, всего 50272 токена в словаре.
- BLOOM [14] мультиязычная модель, содержит 250880 токенов в словаре.
- LLaMa и LLaMa2 [19, 20] модели в основном англоязычные, всего 32000 токена в словаре.

Словари имеют разный размер, однако можно выделить несколько закономерностей. Во-первых, при добавлении новой модальности (другого языка или математики) необходимо увеличивать словарь. Во-вторых, большинство современных моделей имеют в основе BPE токенизатор [40]. Таким образом, в качестве базового алгоритма

для построения токенизатора будет выбран BPE, а число обучаемых токенов будем варьировать от 10к до 120к токенов, также сравним их с ruGPT-3.5 токенизатором. Результаты описаны в Таблице 9.

—	ruGPT-3.5	10 тыс.	20 тыс.	40 тыс.	60 тыс.	120 тыс.
MMLU	0.244	0.251	0.248	0.235	**0.253**	0.236
ruMMLU	**0.254**	0.243	0.224	**0.253**	0.227	0.251
GSM8K	0.012	0.004	0.013	**0.016**	**0.016**	0.015
HumanEval	0.034	0.007	0.037	**0.051**	**0.052**	0.050
ruOpenBookQA	0.253	0.259	0.251	**0.278**	**0.279**	**0.277**

Таблица 8: Измерения качества моделей, обученных с различными токенизаторами

Один из важных признаков качества токенизаторов — значения фертильности. Определим ее как средние число букв содержащееся в токене. Дыля выбранных токенизаторов значения фертильности на Dataset-1 посчитаны в Таблице 9.

—	ruGPT-3.5	10 тыс.	20 тыс.	40 тыс.	60 тыс.	120 тыс.
Фертильность	3.3	6.7	4.2	3.4	3.1	2.7

Таблица 9: Измерения качества моделей, обученных с различными токенизаторами

7.3 Выводы

Обучение 3B моделей показало, что для получения корректных ответов на MMLU бенчмарке 320B обучающих токенов недостаточно вне зависимости от токенизатора. Так как правильный ответ - один из четырех вариантов, то случайное гадание дало бы значение 0.25 на этом бенчмарке. Можно заметить, что результаты бенчмарков GSM8K, HumanEval и ruOpenBookQA не отличаются на 40, 60 и 120 тыс. токенов, однако GSM8K и HumanEval лучше всего себя показывают на размерах токенизатора больше 40 тыс. токенов.

Токенизатор ruGPT-3.5 на GSM8K, HumanEval показывает результаты ниже, чем токенизаторы больше 40 тыс. токенов, а на ruOpenBookQA выдает качество, сравнимое со случайным гаданием (аналогично MMLU). Значение фертильности показывает, что ruGPT-3.5 и токенизаторы больше 40 тыс. токенов одинаково хорошо сжимают текстовую информацию, и чем больше токенизатор - тем ожидаемо меньше становится значение фертильности.

В качестве итогового токенизатора предлагается выбрать токенизатор на 40 тыс. токенов, потому что он сравним по качеству с большими токенизаторами и показывает качество лучше, чем токенизатор от ruGPT-3.5.

8 Улучшение обучающих данных

Напомним, что обучающие данные состоят из пяти доменов: ru-CC, en-CC, arxiv, lemma, algebraic-stack. Изначально, Common-Crawl не разделен по языкам. Чтобы определить язык и выбрать из исходных файлов только русский и английский языки, используется библиотека polyglot [41]. После формирования каждый из доменов проходит следующие стадии перед тем, как попасть в Dataset-1:

1. **Скачивание**. Скачивается домен из открытого источника и разбивается на маленькие файлы (150 MB), таким образом, чтобы они помещались в оперативную память и могли обрабатываться в многопроцессном режиме на CPU.

2. **Exact-дедупликация**. Так как бюджет обучения ограничен, то необходимо обучаться на уникальных чистых данных, дублирование текста нежелательно. Схема дедупликации представлена в парадигме Map-Reduce:

   (Map) Вычисляем значение md-5 функции от каждого текста параллельно на разных устройствах.

   (Reduce) Группируем по значению md-5 дубликаты и возвращаем лишь один из них. Таким образом, алгоритм может обрабатывать большие последовательности данных, которые не помещаются в оперативную память устройств. Этот этап фильтрует 26% изначальных данных.

3. **Чистка данных**. Документы сортируются по длине и из обучающего множества исключаются документы, которые короче 150 символов. Примеры случайно выбранных коротких документов можно посмотреть на Рис. 14. Этот этап дополнительно фильтрует 21% изначальных данных.

```
Текст №98856867. -Т 7.25  - Г - В - А-∞ - З-µ-љ - М
Текст №92708958. Продажа! Комнаты раздельные 13, и 17 , кухня 7м
Текст №51590446. №193-ФЗ
Текст №37486502. VR_Таллин - Стокгольм - Хельсинки      Круиз Минск
```

Рис. 14: Примеры коротких текстов

В итоге, после первичной обработки, в Dataset-1 попадает лишь 47% от исходных данных, который были скачены из открытого доступа.

8.1 SemDedup

Попробуем продолжить развивать идею чистки данных дальше: было обнаружено, что слишком короткие документы неинформативные или содержат ошибки парсинга, а также, что в исходных скачанных данных 26% данных дублируются точь-в-точь. Возникает предположение, что примерно столько же, а может и больше, данных имеют семантические дубликаты и также подлежат изъятию из обучающей выборки.

Идея подхода SemDedup [42] заключается в том, что семантические дубликаты образуют плотные кластера, поэтому их можно выделить с помощью алгоритмов кластеризации и пофильтровать. Фильтрация состоит из следующих шагов:

1. Перевести обучающие данные в единое признаковое пространство (выбрать преобразователь в числовые вектора) и провести кластеризацию.

2. Посчитать попарные расстояния между документами внутри кластера, выбрать порог thr, выше которого все документы считаются дубликатами.

3. По каждой группе дубликатов внутри кластера отсортировать документы по косинусному расстоянию по убыванию от центроида кластера и оставить из них тот, у кого это расстояние больше всех.

Пункт 3 оставляет самые разнообразные документы внутри кластера и удаляет дубликаты, которые на него похожи. Таким образом, авторы подхода обещают уменьшение объема обучающих данных и повышение их информативности. Иллюстрация работы подхода продемонстрирован на Рис. 15.

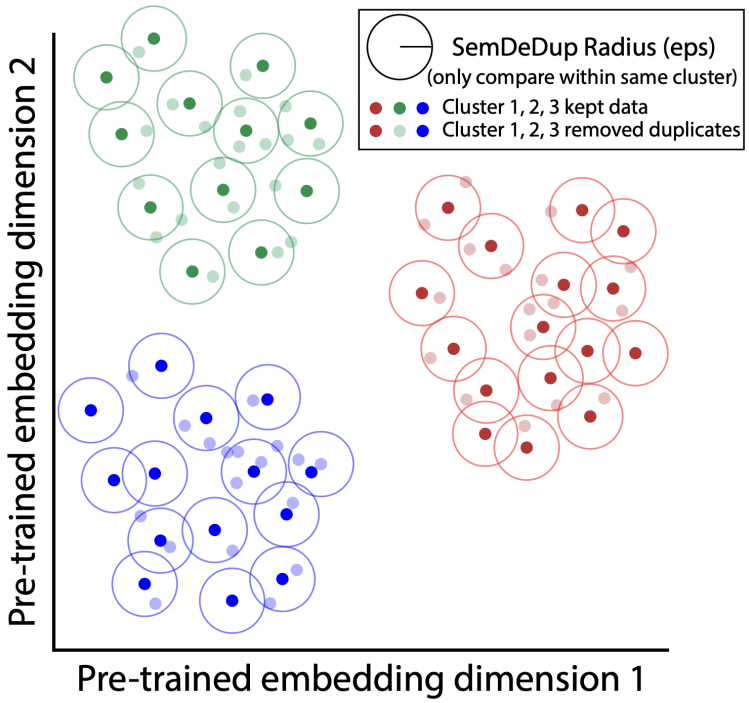

Рис. 15: Иллюстрация работы алгоритма SemDedup [42]

Интересно, что исходный код авторов, который они приводят в статье, не работает по нескольким причинам: не все части кода попали на GitHub, а также нет пояснений, как производить кластеризацию данных в тех случаях, когда они не помещаются в GPU или CPU память.

В качестве доработок и оптимизаций в диссертации предлагаются следующие изменения:

- Разбиение документов на группы по 2.5 млн документов. Такой объем помещается в оперативную память на 1 вычислительный узел.

- При подсчете попарных расстояний проводить вычисление матрицы на CPU долго, поэтому будем делать это на GPU в режиме Tensor Parallel.

  Пусть имеется матрица X размером N×D, причем N $\gg$ D. Тогда, если вектора заранее нормализованы, попарное произведение близости вычисляется по формуле $Y = X \times X^T$ и занимает память N×N.

  Вычисления на GPU ускорителе проходят в сотни раз быстрее, чем на CPU, поэтому необходимо, что вычисление влезало по памяти в 80GB видеопамяти Tesla A100. Режим Tensor Parallel предполагает вычислять матрицу-результат в цикле: $Y = \text{concat}[X_i \times X^T]_{i=1}^{K}$,где X_i -срез матрицы X по оси N. Таким образом, если производить одно вычисление на CPU, то при N=2,500,000 и D=768 вычисление занимает более 40 минут времени, а на GPU при K = 10 примерно 3 минуты.

- Производить обучение кластеризующего алгоритма будем на отложенной выборке в 2.5 млн документов, а применять его всех оставшихся данных. В статье используетеся алгоритм K-MEANS в эффективной реализации из библиотеки faiss [47].

При этих оптимизациях, обработка 320B токенов стала занимать 11 часов на 16 вычислительных узлах. Параметры по кластерам оставим такими же как в статье - 11 тысяч кластеров, а параметр фильтрации будет подбираться таким, чтобы алгоритм фильтровал 30% данных. Еще одним гиперпараметром алгоритма является выбор базового алгоритма-векторизатора данных. Авторы предлагают для текстов использовать OPT-векторизатор [43], однако на момент написания работы он считается устаревшим и неясно его качество на русском языке. К сожалению, не удалось найти рейтинг токенизаторов на русском языке, поэтому для поиска быстрых и качественных моделей был использован MTEB: Massive Text Embedding Benchmark [44] на английском языке, а также поиск по ресурсу HuggingFace. Исходя из результатов, будут перебраны такие векторизаторы: facebook/opt-125m, intfloat/e5-large-v2, cointegrated/rubert-tiny2, ai-forever/ruElectra-large. Дедупликации подвергались все домены, после - на данных обучалась 3B модель на бюджете в 320B токенов и считались выбранные бенчмарки, результаты представлены в Таблице 10.

Можно сделать вывод, что facebook/opt-125m плохо кластеризует русский, математику и код, а intfloat/e5-large-v2 лучше всех моделей выделяет семантические дубликаты. Также важно обратить внимание, что даже на улучшенных данных моделям все еще не удается решать лучше случайного гадания бенчмарки MMLU. Применяя модифицированный алгоритм дедупликации SemDedup поверх векторизатора intfloat/e5-large-v2Dataset-2, получается лучший набор обучающих данных, назовем его Dataset-2.

—	Dataset-1	facebook/ opt-125m	ai-forever/ ruElectra-large	cointegrated/ rubert-tiny2	intfloat/e5-large-v2
MMLU	0.235	0.247	0.251	0.244	0.249
ruMMLU	0.253	0.238	0.254	0.258	0.247
GSM8K	0.016	0.013	0.017	0.017	**0.019**
HumanEval	0.051	0.046	0.035	0.055	**0.061**
ruOpenBookQA	0.278	0.239	0.281	0.289	**0.301**

Таблица 10: Результаты обучения моделей после оптимизированного алгоритма SemDedup, основанного на разных векторизаторах.

8.2 SemDedup HDBSCAN

Проведя визуальный анализ кластеров, можно сделать вывод, что они получаются недостаточно интерпретируемыми. Это означает, что внутри одного кластера может находиться меньше дубликатов, чем реально существует в обучающем множестве.

Гипотеза состоит в том, что алгоритм кластеризации K-MEANS недостаточно хорошо работает в плотном многомерном пространстве документов. Визуальный анализ и перебор алгоритмов кластеризации позволил выбрать другое решение - алгоритм HDBSCAN [48]. У алгоритма есть ряд недостатков - он намного более медленный, чем K-MEANS (поэтому его не удастся применить ко всем обучающим данным) и часть данных, до 20%, он отправляет в множество, которое не может кластеризовать. Модифицируем предыдущий эксперимент:

1. HDBSCAN не работает в пространстве признаков больше двух, поэтому перед использованием необходимо провести их редукцию. Для этого обучается алгоритм UMAP [49].

2. Часть данных HDBSCAN не может кластеризовать, эти данные не исключатся из обучающего множества.

3. На 11 тысячах кластеров большинство из них остаются неинтерпретируемыми и содержат мало точек. Поэтому выберем число кластеров, раваным 764 и получим визуально интерпретируемую кластеризацию. Пример применения кластеризации на части Common Crawl увидеть на Рис. 16.

4. В виду кратно более высокой сложности вычислений ансамбля UMAP+HDBSCAN, применим его только на доменах с небольшим числом токенов arxiv, lemma, algebraic-stack. Остальные домены отфильтруем способом из Dataset-2.

Домены, на которых применялся HDBSCAN, являются достаточно чистыми с точки зрения наличия дубликатов, однако все же удалось уменьшить их на 7.4%. Назовем этот набор данных Dataset-3 и сравним результаты с обучением на Dataset-2,

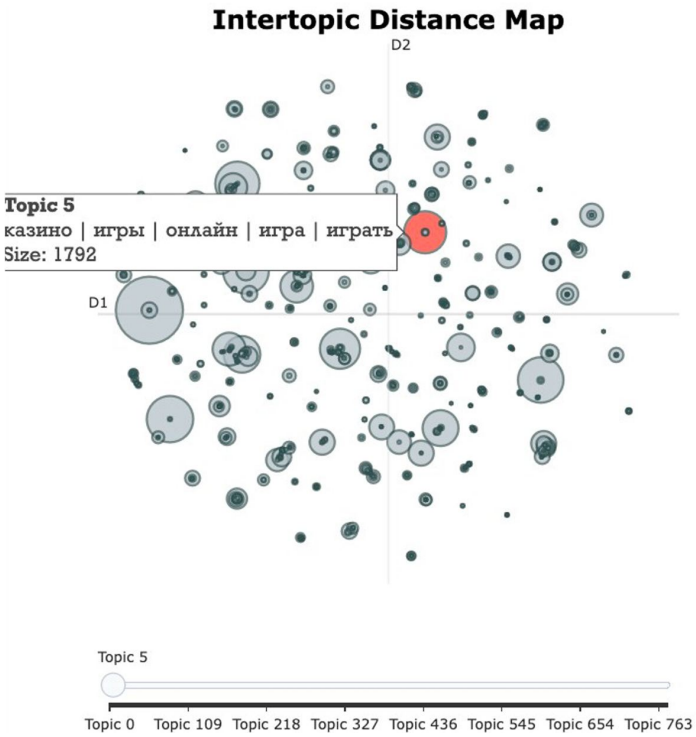

Рис. 16: Демонстрация кластеризации HDBSCAN

результаты в Таблице 11. Можно сделать вывод, что визуальное улучшение кластеризации дало прирост в качестве на всех бенчмарках. Финальную 7B модель будем обучать на Dataset-3 и подготовим под него 2T обучающих токенов.

—	Dataset-1	Dataset-2	Dataset-3
MMLU	0.235	0.249	0.251
ruMMLU	0.253	0.247	0.246
GSM8K	0.016	0.019	**0.021**
HumanEval	0.051	0.061	**0.062**
ruOpenBookQA	0.278	0.301	**0.312**

Таблица 11: Результаты обучения моделей после оптимизированного алгоритма SemDedup, основанного на разных векторизаторах.

8.3 Выводы

Так как по MMLU бенчмаркам качество получилось на уровне случайного гадания, то оценивать прогресс будем по трем оставшимся. Можно сделать вывод, что усложнение алгоритмов фильтрации давало постоянный рост метрик, а переход от Dataset-1 к Dataset-3 принес средний выигрыш на бенчмарках равный 21.7%.

9 Обучение 7.3B модели

На этом этапе подготовлены данные, оптимизирована инфраструктура обучения и подобрана оптимальная архитектура модели. Одной из целей диссертации было обучения модели, которая превосходит по качеству ответов LLaMa2 7B модель, поэтому выберем параметры модели так, чтобы количество обучаемых слоев было близко к 7B параметрам (Таблица 12).

Params	vocab_size	hidden_size	intermediate_size	n_heads	kv_heads	n_layers
7.3B	40000	4096	14336	32	8	32

Таблица 12: Архитектура 7.3B модели

9.1 Детали обучения

Модель обучалась на бюджете в 2T токенов (как LLaMa2 7B) полученных из Dataset-3 с размером батча в 4 млн токенов и контекстом 4096 токенов. Оптимизатор Decoupled AdamW [50] использовался с параметрами betas = [0.9, 0.95], learning rate = 3.0e-4, epsilon = 1e-08, weight decay = 0.1, все они были получены из статей LLaMa и LLaMa2 [19, 20]. График обучения 7.3B модели представлен на Рис. 17.

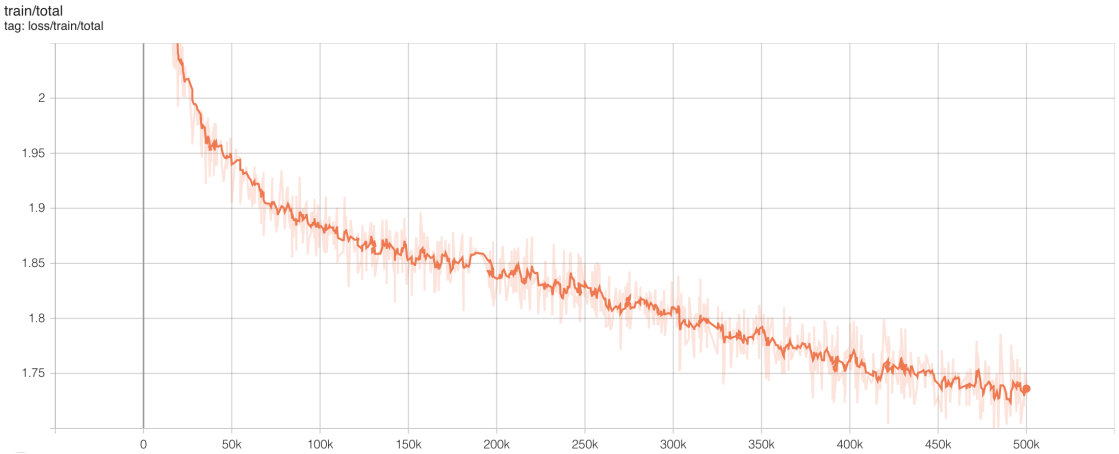

Рис. 17: Падение функции потерь 7B модели

9.2 Сравнение с ruGPT-3.5-13B и LLaMa2 7B

Измерения качества трех моделей на бенчмарках представлены в Таблице 13. Основные выводы:

1. Несмотря на то, что ruGPT-3.5-13B почти в два раза больше конкурентов с точки зрения числа обучаемых параметров, она показывает самые низкие результаты среди всех замеряемых моделей как на английском, так и на русском языке.

2. Авторы LLaMa-2 в статье утверждают, что модель обучалась преимущественно на англоязычных данных, однако показывает неплохие результаты и на русскоязычных бенчмарках.

3. Полученная 7.3B модель демонстрирует паритет с LLaMa-2 в математических задачах и значительное преимущество на всех бенчмарках по сравнению с остальными моделями.

—	ruGPT-3.5-13B	LLaMa2 7B	7.3B model
MMLU	0.26	0.453	**0.524**
ruMMLU	0.246	0.361	**0.531**
GSM8K	0.016	**0.146**	**0.148**
HumanEval	0.012	0.128	**0.226**
ruOpenBookQA	0.208	0.471	**0.639**

Таблица 13: Сравнение производительности ruGPT-3.5-13B, LLaMa2 7B и 7.3B model на бенчмарках

10 Результаты, выносимые на защиту

1. Методы подбора оптимальной архитектуры для языковой модели, позволившие достичь снижения функции потерь на 13%.

2. Методы оптимизации вычислений, позволившие ускорить на 75.7% обучение модели и ее перезапуски в 4.53 раза.

3. Алгоритмы фильтрации данных, повышающие качество и разнообразие обучающих данных, позволившие повысить качество обучения на бенчмарках на 21,7%.

4. Обученная русскоязычная модель, которая демонстрирует значительно лучшее качество, чем открытые ruGPT-3.5-13B и LLaMa2 7B.

Список литературы

[1] Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – Т. 30.

[2] Tenney I., Das D., Pavlick E. BERT rediscovers the classical NLP pipeline //arXiv preprint arXiv:1905.05950. – 2019.

[3] Brown T. et al. Language models are few-shot learners //Advances in neural information processing systems. – 2020. – Т. 33. – С. 1877-1901.

[4] Tarasov D. S. Natural language generation, paraphrasing and summarization of user reviews with recurrent neural networks //Proc. Mater. Int. Conf. Dialog. – 2015. – С. 1-20.

[5] Yih S. W. et al. Semantic parsing via staged query graph generation: Question answering with knowledge base //Proceedings of the Joint Conference of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing of the AFNLP. – 2015.

[6] Dong X., Yang Y. Searching for a robust neural architecture in four gpu hours //Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. – 2019. – С. 1761-1770.

[7] Pota M. et al. An effective BERT-based pipeline for Twitter sentiment analysis: A case study in Italian //Sensors. – 2020. – Т. 21. – №. 1. – С. 133.

[8] Reimers N., Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks //arXiv preprint arXiv:1908.10084. – 2019.

[9] Koroteev M. V. BERT: a review of applications in natural language processing and understanding //arXiv preprint arXiv:2103.11943. – 2021.

[10] Liu Y. et al. Roberta: A robustly optimized bert pretraining approach //arXiv preprint arXiv:1907.11692. – 2019.

[11] Raffel C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer //The Journal of Machine Learning Research. – 2020. – Т. 21. – №. 1. – С. 5485-5551.

[12] Xue L. et al. mT5: A massively multilingual pre-trained text-to-text transformer //arXiv preprint arXiv:2010.11934. – 2020.

[13] Tay Y. et al. Unifying language learning paradigms //arXiv preprint arXiv:2205.05131. – 2022.

[14] Workshop B. S. et al. Bloom: A 176b-parameter open-access multilingual language model //arXiv preprint arXiv:2211.05100. – 2022.

[15] Сбер открывает доступ к нейросетевой модели ruGPT-3.5 // Хабр[сайт] — режим доступа: https://habr.com/ru/companies/sberbank/articles/746736/ свободный (дата обращения: 10.12.2023) — Загл. с экрана.

[16] Ouyang L. et al. Training language models to follow instructions with human feedback //Advances in Neural Information Processing Systems. – 2022. – Т. 35. – С. 27730-27744.

[17] Ouyang L. et al. Training language models to follow instructions with human feedback //Advances in Neural Information Processing Systems. – 2022. – Т. 35. – С. 27730-27744.

[18] The Illustrated GPT-2 (Visualizing Transformer Language Models) // сайт - режим доступа: https://jalammar.github.io/illustrated-gpt2/ свободный (дата обращения: 28.11.2023) — Загл. с экрана.

[19] Touvron H. et al. Llama: Open and efficient foundation language models //arXiv preprint arXiv:2302.13971. – 2023.

[20] Touvron H. et al. Llama 2: Open foundation and fine-tuned chat models //arXiv preprint arXiv:2307.09288. – 2023.

[21] MosaicML NLP Team, "Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs," 2023. [Online]. Available: www.mosaicml.com/blog/mpt-7b. [Accessed: 2023-10-14]

[22] Hinton G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors //arXiv preprint arXiv:1207.0580. – 2012.

[23] Ba J. L., Kiros J. R., Hinton G. E. Layer normalization //arXiv preprint arXiv:1607.06450. – 2016.

[24] Common Crawl // Common Crawl[сайт] — режим доступа: https://www.commoncrawl.org/ свободный (дата обращения: 28.08.2023) — Загл. с экрана.

[25] Dataset Card for Wikipedia // HuggingFace[сайт] — режим доступа: https://huggingface.co/datasets/wikipedia свободный (дата обращения: 18.02.2024) — Загл. с экрана.

[26] Dataset Card for Proof-Pile-2 // HuggingFace[сайт] — режим доступа: https://huggingface.co/datasets/EleutherAI/proof-pile-2 свободный (дата обращения: 18.02.2024) — Загл. с экрана.

[27] Su J. et al. Roformer: Enhanced transformer with rotary position embedding //Neurocomputing. – 2024. – Т. 568. – С. 127063.

[28] Xiong R. et al. On layer normalization in the transformer architecture //International Conference on Machine Learning. – PMLR, 2020. – C. 10524-10533.

[29] Ba J. L., Kiros J. R., Hinton G. E. Layer normalization //arXiv preprint arXiv:1607.06450. – 2016.

[30] Zhang B., Sennrich R. Root mean square layer normalization //Advances in Neural Information Processing Systems. – 2019. – T. 32.

[31] Shazeer N. Glu variants improve transformer //arXiv preprint arXiv:2002.05202. – 2020.

[32] Hendrycks D., Gimpel K. Gaussian error linear units (gelus) //arXiv preprint arXiv:1606.08415. – 2016.

[33] Ramachandran P., Zoph B., Le Q. V. Searching for activation functions //arXiv preprint arXiv:1710.05941. – 2017.

[34] Dauphin Y. N. et al. Language modeling with gated convolutional networks //International conference on machine learning. – PMLR, 2017. – C. 933-941.

[35] Ainslie J. et al. Gqa: Training generalized multi-query transformer models from multi-head checkpoints //arXiv preprint arXiv:2305.13245. – 2023.

[36] Hendrycks D. et al. Measuring massive multitask language understanding //arXiv preprint arXiv:2009.03300. – 2020.

[37] Cobbe K. et al. Training verifiers to solve math word problems //arXiv preprint arXiv:2110.14168. – 2021.

[38] Chen M. et al. Evaluating large language models trained on code //arXiv preprint arXiv:2107.03374. – 2021.

[39] Taktasheva E. et al. TAPE: Assessing few-shot Russian language understanding //arXiv preprint arXiv:2210.12813. – 2022.

[40] Sennrich R., Haddow B., Birch A. Neural machine translation of rare words with subword units //arXiv preprint arXiv:1508.07909. – 2015.

[41] https://pypi.org/project/polyglot/

[42] Abbas A. et al. Semdedup: Data-efficient learning at web-scale through semantic deduplication //arXiv preprint arXiv:2303.09540. – 2023.

[43] Zhang S. et al. Opt: Open pre-trained transformer language models, 2022 //URL https://arxiv. org/abs/2205.01068. – 2023. – T. 3. – C. 19-0.

[44] Muennighoff N. et al. MTEB: Massive text embedding benchmark //arXiv preprint arXiv:2210.07316. – 2022.

[45] Zhao Y. et al. Pytorch fsdp: experiences on scaling fully sharded data parallel //arXiv preprint arXiv:2304.11277. – 2023.

[46] Dao T. et al. Flashattention: Fast and memory-efficient exact attention with io-awareness //Advances in Neural Information Processing Systems. – 2022. – T. 35. – C. 16344-16359.

[47] Douze M. et al. The faiss library //arXiv preprint arXiv:2401.08281. – 2024.

[48] Campello R. J. G. B., Moulavi D., Sander J. Density-based clustering based on hierarchical density estimates //Pacific-Asia conference on knowledge discovery and data mining. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – C. 160-172.

[49] McInnes L., Healy J., Melville J. Umap: Uniform manifold approximation and projection for dimension reduction. arXiv 2018 //arXiv preprint arXiv:1802.03426. – 1802. – T. 10.

[50] Loshchilov I., Hutter F. Decoupled weight decay regularization //arXiv preprint arXiv:1711.05101. – 2017.