ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет об исследовательском проекте

на тему: Инференс больших языковых моделей типа декодер

(промежуточный, этап 1)

Выполнил:
Студент группы БПМИ223 _____ И.А.Бобошко
Подпись И.О.Фамилия

04.02.2025
Дата

Принял:
Руководитель проекта Потапов Иван Андреевич
Имя, Отчество, Фамилия
Senior Software Engineer
Должность, ученое звание
Zalando SE
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки _____ _____ _____
Оценка (по 10-ти бальной шкале) Подпись

Москва

Содержание

Аннотация

Цель данного проекта заключается в изучении устройства больших языковых моделей для генерации текста и современных методов оптимизации их инференса. В рамках работы изучается влияние различных методов квантизации и оптимизаций памяти на скорость и качество инфересна LLM.

1 Введение

Современные большие языковые модели стали неотъемлемой частью множества приложений, от автоматической генерации текста до интеллектуальных помощников и систем машинного перевода. Их развитие привело к значительному улучшению качества текстовой генерации, однако высокая вычислительная сложность и ресурсозатратность инференса (процесса генерации предсказаний) остаются серьезными вызовами.

В данной работе рассматриваются ключевые принципы устройства больших языковых моделей, их архитектурные особенности и современные методы оптимизации инференса: квантизация и оптимизации механизма внимания. Были поставлены следующие задачи:

1. Изучить устройство больших языковых моделей
2. Исследовать и описать современные методы квантизации больших языковых моделей
3. Изучить подходы к оптимизации механизма внимания в LLM
4. Провести эксперименты и сравнить выбранные методы
5. Разворачивание инференса языковой модели и проведение бенчмарков.

Поставленные задачи были выполнены: изучено устройство архитектуры трансформер и больших языковых моделей, были разобраны такие оптимизации attention, как KV-cache и PagedAttention. Была изучена базовая теория квантизации нейросетей, а также современные методы квантизации LLM: GPTQ, LLM-8bit, NF4. Описаны основные стратегии генерации текста и их влияние на инференс языковых моделей. Были проведены эксперименты и бенчмарки, а также запуск настоящего инференса большой языковой модели с помощью vLLM.

2 Устройство больших языковых моделей

2.1 Языковая модель

Определение 1. Пусть $X = (x_1, \ldots, x_n)$ - последовательность токенов, Θ - параметры модели, тогда задача языкового моделирования заключается в предсказании следующего токена по текущему и предыдущим. Пользуясь методом максимального правдоподобия, можно свести задачу к классификации предсказанного токена на каждом шаге. [15]

$$p_\theta(X) = \prod_{t=1}^{n} p_\theta(x_t | x_{<t}) \longrightarrow \max_\Theta$$

$$-\sum_{i=1}^{n} \log(p_\theta(x_t | x_{<t})) \longrightarrow \min_\Theta$$

Замечание 2. Данное определение описывает определенный тип языковых моделей, а именно left-to-right или регрессионная модель, в работе будут рассмотрены так же и другие варианты, например, masked (для обучения BERT)

Замечание 3. В контексте данной работы под токеном будут подразумеваться составляющие текстовых данных, например, символы, слова, предложения и другие лексемы.

2.1.1 Обучение нейросетевых языковых моделей

1. Подаем модели на вход эмбеддинги предыдущих токенов, то есть для i-ого токена, передаем $(x_0, \ldots, x_{i-1})$
2. Получаем векторное представление контекста как выход нейросети
3. Обучаем линейный классификатор на предсказание следующего токена с помощью закодированного контекста. Зачастую применяют это линейное отображение, потому что размер эмбеддинга не совпадает с количеством классов, а также это дает больше возможностей при обучении. [18]

2.1.2 Оценка качества

Определение 4. Cross-Entropy

$$CE(y_1, \ldots, y_N) = -\frac{1}{N} \sum_{i=1}^{N} \log p(x_i | x_{<i}) \tag{1}$$

Чем ниже значение Cross-Entropy, тем лучше модель предсказывает правильные токены, то есть тем лучше она отражает структуру языка.

Определение 5. Perplexity

$$Perplexity(y_1, \ldots, y_N) = 2^{CE(y_1, \ldots, y_N)} \tag{2}$$

Чем меньше, тем лучше. Минимальное значение - 1, когда мы предсказали все токены правильно, то есть нулевой лосс. Самый худший сценарий $Perplexity(y_1, \ldots, y_N) = |Vocab|$, то есть равномерное распределение.

2.2 Attention

2.2.1 Self-Attention

Пусть $X \in \mathbb{R}^{l \times d_k}$ - входные эмбеддинги, где l - длина последовательности, d_k - размер эмбеддинга токена. Из входной матрицы получим три новых с помощью линейный преобразований, где $W \in \mathbb{R}^{d_k \times d_k}$:

$$X \longrightarrow XW_Q = Q \in \mathbb{R}^{l \times d_k}$$

$$X \longrightarrow XW_K = K \in \mathbb{R}^{l \times d_k}$$

$$X \longrightarrow XW_V = V \in \mathbb{R}^{l \times d_k}$$

Тогда слой attention можно задать следующим образом:

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

То есть сначала мы получаем три новых представления для исходной последовательности, которые будут выполнять разные функции:

- W_Q отвечает за запросы
- W_K отвечает за ключи
- W_V отвечает за представления, которые будут использоваться для получения выходов слоя

Далее мы считаем QK^T - скалярные произведения Q-эмбеддингов и K-эмбеддингов, то есть для каждого запроса q_i (строка матрицы Q) мы считаем скалярное произведение со всеми ключами, нормируем на d_k - необходимо для того, чтобы в softmax не поступали слишком большие числа, далее, применяя softmax, мы получаем "веса важности"каждого ключа для данного запроса.

В конце мы домножаем на V, тем самым мы берем каждое значение с весами полученными из прошлого шага.

2.2.2 Multihead-Attention

Вместо трех матриц W_Q, W_K, W_V у нас будет W_Q^h, W_K^h, W_V^h, где $h = 1, \ldots, H$, $W^h \in \mathbb{R}^{\frac{d_k}{H} \times \frac{d_k}{H}}$. Таким образов у нас параллельно будут считаться несколько attention score, что позволяет извлечь больше информации из последовательности:

$$\text{head}_h(Q_h, K_h, V_h) = \text{softmax}(\frac{Q_h K_h^T}{\sqrt{d_k}})V_h$$

Тогда итоговый результат будет конкатенацией результатов, полученных с помощью каждой головы, также применяют линейное преобразование для сконкатенированного выхода

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_H)W_O$$

Данный прием позволяет извлечь больше различных аспектов связи между токенами

2.2.3 Masked Attention

Данная модификация нужна, чтобы не учитывать токены из будущего, достигается за счет применения маски, которая зануляет в softmax ненужные токены

2.3 Transformer

В оригинальной статье данная модель была предложена для решения задачи машинного перевода. [17]

2.3.1 Attention

В данной модели используются разные виды attention в encoder и decoder

- Encoder-Encoder - обычный MultiHeadAttention
- Decoder-Decoder - masked MultiHeadAttention
- Encoder-Decoder - Q из decoder; K, V из encoder

2.3.2 Positional Encoding

В трансформере мы никак не учитываем порядок токенов, поэтому авторы статьи добавляют еще один вид эмбеддинга, который решает эту проблему

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{3}$$

где pos — позиция слова, i — индекс измерения, а d_{model} — размерность эмбеддинга. Эти функции позволяют представить позиции так, чтобы они могли быть легко интерпретируемы моделью, сохраняя различия между последовательностями различных длин.

2.3.3 Add and Norm

Механизм *Add and Norm* используется в архитектуре трансформеров для стабилизации обучения и улучшения сходимости модели. Он применяется после каждого self-attention и FFN сети. Этот механизм состоит из двух частей:

- Residual Connection : к выходу слоя добавляется его входное значение:

$$y = x + f(x) \tag{4}$$

  Это позволяет сохранять информацию из исходного входа, облегчая процесс обучения.

- Layer Normalization : нормализует выходные данные по их среднему и дисперсии:

$$\hat{y} = \frac{y - \mu}{\sigma} * \gamma + \beta \tag{5}$$

  где μ — среднее значение, σ — стандартное отклонение, а γ и β — обучаемые параметры.

Этот процесс помогает модели быстрее сходиться, улучшает стабильность градиентов и делает обучение более эффективным.

2.3.4 Feed-forward block

После подсчета attentiona в каждом блоке в конце применяется полносвязная нейросеть:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

Можно считать, что attention извлекает важные связи, а FFN обрабатывает их. Стоит отметить, что в данном блоке мы сначала увеличиваем ширину в 4 раза, а потом возвращаем в исходное состояния, поэтому данный блок можно считать один из самых затратных мест в архитектуре.

2.4 Архитектура больших языковых моделей

Современные большие языковые модели в большинстве случаев принадлежат к трансформерной архитектуре, адаптированной под решение задачи генерации текста. Зачастую LLM представляют из себя декодерную часть из трансформера, такой выбор обусловлен авторегрессионной природой задачи генерации текста. Также стоит отметить, что некоторые составляющие оригинального трансформера были переработаны для того, чтобы модели лучше масштабировалась.

3 Оптимизации механизма внимания

В этом разделе описаны наиболее популярные и состоявшиеся методы оптимизации ускорения инференса больших языковых моделей.

3.1 KV cache

В больших языковых моделях генерация текста происходит последовательно: каждый новый токен предсказывается на основе предыдущих. Для подсчета attention блока для текущего токена необходимо также посчитать KV значения для предыдущих токенов в последовательности, что приводит к квадратичной сложности ($O(n^2)$) и высоким вычислительным затратам. Для оптимизации этого процесса используется KV-Cache (Key-Value хранилище) — механизм кэширования ключей (K) и значений (V) в блоках attention, позволяющий избежать их пересчета для прошлых токенов [3]. Таким образом, данный алгоритм позволяет избавиться от квадратичной сложности, но при этом требует потребления $O(b \cdot h \cdot n)$, где b - размер батча, h - размер скрытого слоя, n - длина последовательности.

Algorithm 1 Алгоритм KV-Cache

Require: x_n - текущий токен,
W_Q, W_K, W_V,
cache $= \{K_{cache} = [K_1, \ldots, K_{n-1}]$, $V_{cache} = [V_1, \ldots, V_{n-1}]\}$
Ensure: y_n, cache
Вычислить key, value и query для текущего токена
$q_n \leftarrow x_n W_Q$
$k_n \leftarrow x_n W_K$
$v_n \leftarrow x_n W_V$
Обновить cache:
$K_{cache} = \text{concat}(K_{cache}, k_n)$
$V_{cache} = \text{concat}(V_{cache}, v_n)$
Вычислить attention:
$y_n \leftarrow \text{Softmax}\left(\frac{q_n \cdot (K_{cache})^T}{\sqrt{d_k}}\right) \cdot V_{cache}$
return y_n, cache

3.2 Paged Attention

Как мы выяснили ранее KV-cache действительно позволяет более эффективно вычислять attention, но при этом такой подход требует $O(b \cdot h \cdot n)$ памяти. При наивной реализации алгоритма KV-cache для каждой последовательности мы выделяем $b \cdot h \cdot$ max-length, что влечет за собой неэффективное использование памяти. Например, для последовательностей маленькой длины, при изначально большом контексте, такая проблема будет особенно актуальна. В статье [13] показано, что при наивном использовании KV-cache действительно используется только 20-40 процентов от выделенной под KV-хранилище памяти. Был предложен Paged Attention - эвристика, вдохновленная концепцией виртуальной памяти из операционных систем, которая позволяет эффективно утилизировать память. Авторы предлагают вместо хранения матриц K и V как непрерывных кусков в памяти, разбить их на блоки(страницы) фиксированного размера, которые будут храниться в памяти произвольным образом. Таким образом, данный подход позволяет оптимально расходовать память: у исследователей получилось достичь практически полной утилизации памяти (96%). Более того, данный подход позволяет обрабатывать очень длинные контекст эффективно: для последовательностей маленькой длины будет выделено небольшое количество блоков, то есть недоутилизация памяти для одно последовательности не может превосходить размер блока.

Таблица 1: Сравнение подходов

Метод	Память	Время доступа
Обычный KV-Cache	$O(n)$	$O(1)$
PagedAttention	O(активные страницы)	$O(\lceil n/P \rceil)$

4 Квантизация

Определение 6. Квантизация - это способ построения отображения из большого, более мощного (зачастую "непрерывного") множества в менее мощное ("дискретное") множество.

Замечание 7. В контексте данной работы нас в основном будут интересовать методы, которые отображают floating point числа в тип данных, которые меньше весят, например, из fp32 в fp8 или int8.

4.1 Квантизация больших языковых моделей

На данный момент передовые большие языковые модели зачастую состоят из миллиардов параметров, например, одна из самых популярных open-source моделей - LLaMa [16] на 7 миллиардов параметров, загруженная в fp32, весит примерно 28gb, а модель BLOOM(176b) [19] в fp16 требует около 350gb видеопамяти, что делает инференс такой модели ресурсозатраным. Таким образом, квантизация используется для более эффективного инференса и хранения LLM.

В больших языковых моделях значительную часть параметров занимают веса линейных слоев из attention и FFN, поэтому в основном квантизуют именно их. Более того, есть подходы, которые помимо слоев квантизуют также и активации, то есть выходы предыдущего слоя.

4.1.1 Основные виды квантизации

В контексте квантизации LLM выделяют три основных семейства методов [11]:

1. Post-Training Quantization (PTQ) - квантизация после обучения, наиболее простой подход, не требующий дообучения модели.

2. Mixed Precision - можно квантизовать не все части модели, а только, например, некоторые слои или определённые типы операций (активации, веса). Это помогает найти баланс между производительностью и качеством

3. Quantization Aware Training - метод для достижения минимальной потери качества. В отличие от PTQ, квантизация происходит в процессе обучения. В QAT архитектура модели изначально модифицируется для хранения весов и активации модели в двух типах: исходном и квантизованном. В forward pass на вход в модель передают квантизованные входы, что позволяет модели испытывать эффекты квантизации, но при этом в backward pass градиенты считаются в плавающей арифметике, тем самым повышая устойчивость модели к квантизации.

4.2 Инференс квантизованных моделей

4.2.1 Simulated квантизация

В данном подходе мы квантизуем только веса модели, тем самым сокращая размер модели. Во время инференса мы деквантизуем веса и производим вычисления с плавающей точкой. Такой подход позволяет постичь хорошей точности, но при этом не ускоряет инференс модели. Пусть $X_{in} \in \mathbb{FP}^{nm}$ - активация прошлого слоя, тогда вычисления можно записать так:

$$\hat{W} = s(W - z)$$

$$X_{out} = X_{in}W \approx X_{in}\hat{W}$$

4.2.2 Int-Based квантизация

В отличие от Simulated подхода, происходит квантизация входящей активации, тем самым наиболее трудозатраные вычисления, а именно GEMM, происходят в целочисленной арифметике за счет чего получается ускорить инференс модели.

$$\hat{W} = s(W_q - z)$$

$$\hat{X}_{in} = s(X_{in} - z)$$

$$X_{out} = X_{in}W \approx \hat{X}_{in}\hat{W}$$

4.3 Линейная квантизация

В данном разделе описаны самые базовые методы квантизации: zero-point и absmax квантизация. Эти методы являются фундаментальными и тем или иным образом используются во всех современных методах квантизации.

Определение 8.

$$Q_{(s,z)} : \mathbb{X} \to \mathbb{Y}$$

$$x \to y = cast(x/s + z, \mathbb{Y})$$

$$DeQ_{(s,z)} : \mathbb{Y} \to \mathbb{X}$$

$$y \to x = s(x - z)$$

$$s \in \mathbb{X} - scale, z \in \mathbb{Y} - \text{zero point}$$

Замечание 9. В данном случае операция cast обозначает приведение к типу $\mathbb{Y}$. Например, в случае $\mathbb{X} = \mathbb{FP}32, \mathbb{Y} = \mathbb{INT}8$ - это сначала операция округления, а потом отображение множество значений $\mathbb{Y}$

4.3.1 Поиск scale и zero point

Пусть $x \in \mathbb{FP}^n$, тогда опередим $x_{min} = min(x)$, $x_{max} = max(x)$. Будем отталкиваться от того, что при обратном отображении мы хотим получить из максимального обратно максимальное и наоборот, то есть:

$$\begin{cases} x_{min} &= s(y_{min} - z) \\ x_{max} &= s(y_{max} - z) \end{cases}$$

$$\begin{cases} s &= (x_{max} - x_{min})/(y_{max} - y_{min}) \\ z &= cast(y_{min} - x_{min}/s, \mathbb{Y}) \end{cases}$$

На картинке изображено распределение весов, а красных точки из квантизованного множества.

4.4 NF4

В статье QLORA [8] авторы предложили метод квантизации NF4, который использует квантили нормального распределения вместо равномерной сетки как в линейной квантизации. Основная интуиция данного алгоритма заключается в том, что зачастую веса обученных моделей распределены нормально с нулевым средним. Таким образом, используя данный факт, можно уменьшить ошибку квантизации за счёт более плотного размещения сетки квантизации в областях с высокой концентрацией весов.

Квантизация NF4 определяется следующим образом:

$$q_i = \frac{1}{2}\left(Q_x\left(\frac{i}{2^k + 1}\right) + Q_x\left(\frac{i+1}{2^k + 1}\right)\right) \tag{7}$$

где:

- q_i — i-й уровень квантования,
- Q_x — квантильная функция стандартного нормального распределения $\mathcal{N}(0, 1)$,
- k — количество бит на вес (в случае NF4 $k = 4$, что даёт $2^4 = 16$ возможных значений).

4.5 LLM.8bit()

В статье LLM.int8() [7] был предложен метод, который позволяет квантизовать большие LLM (от 7 миллиардов параметров) в 8 бит без потери качества. В основе данного алгоритма лежит идея, что у моделей есть признаки (столбцы) с большой магнитудой, которые значительно влияют на итоговое качество модели. Таким образом, был предложен метод, в котором обычные признаки квантизуются с помощью vector-wise 8-битной квантизации (в статье были рассмотрено оба варианта: симметричный и асимметричный), а признаки с большой магнитудой не квантизуются.

4.5.1 Vector-wise квантизация

Пусть $X_{in} \in \mathbb{FP}^{nm}$ - активация прошлого слоя, $W \in \mathbb{FP}^{mk}$. Идея данного подхода заключается в том, чтобы квантизовать каждую строку X_{in} и каждый столбец W. Тогда деквантизовать матричное произведение $X_{in}W$ можно по формуле:

$$X_{out} = X_{in}W \approx \frac{1}{s_{X_{in}} s_W^T} * \hat{X}\hat{W}$$

4.6 GPTQ

GPTQ — это метод PTQ квантизации, в котором используется калибровочный датасет для достижения наименьшей ошибки [9]. В основе данного метода лежит идея, что можно рассматривать процесс квантизации как оптимизационную задачу:

$$\hat{W} = argmin_{\bar{W}} \|XW^T - X\bar{W}^T\|_2^2$$

где

1. X - активация из предыдущего слоя
2. W - оригинальные веса
3. $\bar{W}$ - квантизованные веса

В статье используют подход, который называется row-wise: квантизуется одна строка матрицы весов, остальные строки поправляются так, чтобы минимизировать ошибку из-за квантизации строки.

Algorithm 2 GPTQ (row-wise)

Require: X - активации с прошлого слоя (калибровочный датасет),
W - оригинальные веса слоя
Ensure: $\hat{W}$

```
for i in rows do
    Ŵ_i = Q(W_i)
    H = 2X^T X
    W = W − (W_i − Q(W_i)) / [H_ii^{-1}] H^{-1}_{i,:}
    убрать квантизованную строку из матрицы
end for
return Ŵ
```

5 Стратегии генерации текста

Обученная языковая модель выдает нам распределение следующего токена. Базовый метод заключается в семплинге следующего токена из полученного распределения. Для более качественной генерации можно выдавать:

$$x^* = \underset{x}{argmax} \prod_{t=1}^{n} p_\theta(x_t|x_{<t}) \tag{8}$$

В связи с тем, что поиск такого x требует полного перебора в этом разделе будут рассмотрены некоторые эвристики, которые дают хорошее качество.

5.1 Жадный алгоритм

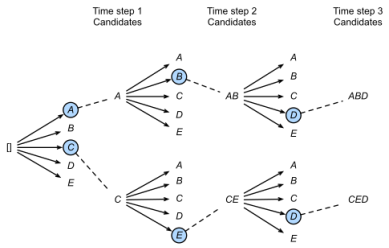
На каждом шаге выбираем наиболее вероятный токен:

$$x^* = \prod_{t=1}^{n} argmax_{x_t}(p_\theta(x_t|x_{<t})) \tag{9}$$

Данный метод быстро работает, но при этом уступает в качестве. Также стоит отметить, что он зачастую зацикливает генерацию.

5.2 Beam Search

В отличие от greedy в данном методе мы рассматривает несколько вариантов параллельно. На каждом шаге мы оставляем только топ-k из этих веток.



5.3 Temrature sampling

Данный метод заключается в шкалировании логитов модели:

$$softmax(x) = \frac{exp(x)}{\sum_i exp(x)} \longrightarrow \frac{exp(x/T)}{\sum_i exp(x/T)} \tag{10}$$

Чем больше гиперпараметр T, тем ближе будет итоговое распределение к равномерному и наоборот, чем меньше T, тем ближе к вырожденному (то есть к жадной генерации).

5.4 Top-k

В данном подходе в качестве ответа выбирается класс из топ-k классов. У данного метода есть проблема при фиксированных k, например, бывает, что у нас будет равномерное распределение выходов, тогда мы потеряем часть информации, а также когда у нас есть несколько наиболее вероятных классов, тогда мы можем выдать в качестве ответа класс с около нулевой вероятностью

5.5 Top-p

Выбирает слова из динамически изменяемого набора, включающего суммарно p вероятности [12]:

$$P(x) \in \{x_i | \sum P(x_i) \leq p\} \tag{11}$$

Плюсы: баланс между случайностью и вероятностью. Минусы: возможные несвязные тексты при высоких значениях p.

6 Метрики качества инференса LLM

6.1 Метрики производительности

Метрики производительности позволяют оценить эффективность работы LLM.

Latency

- **Time To First Token Latency** — время между отправкой запроса и получением первого токена ответа. Критически важна, например, для чат-ботов.

- **Time Per Token** — время генерации токена после первого.
- **End-to-End latency (E2E latency)** — время, которые было потрачено на генерацию всего запроса.

Throughput

- **Tokens Per Second (TPS)** — количество токенов, генерируемых моделью за одну секунду.

Замечание 10. Стоит отметить, что также нужно учитывать, как система утилизирует выделенные ресурсы, например, утилизация видеопамяти GPU. Данная метрика может быть полезна в ситуации ограниченных ресурсов.

6.2 Метрики качества генерации

В контексте инференса больших языковых моделей нас будут интересовать метрики качества генерации, которые позволяют оценить насколько упало качество генерации при использовании определенного метода квантизации.

- Perplexity на датасете WikiText2 [6] — используется во многих статьях [7, 11, 9] для сравнение методов квантизации. Весь текст конкатенируется через '/n/n', токенизируется и разбивается на непрекращающиеся куски фиксированной длины, для каждого куска вычисляется логарифм правдоподобия. Результатом будет экспоненциальное среднее от всех записей. Данный подход помогает стандартизировать вычисления и гарантировать сравнимость разных моделей.
- Zero-shot Accuracy на Lambda Dataset [14]. Lambda - это датасет из 10 тысяч отобранных примеров, сред которых: задачи на понимание большого контекста, головоломки. Для вычисление метрик на этом датасете использовалась библиотека Language Model Evaluation Harness [10], которая является одном из самых популярных средств подсчета качества больших языковых моделей. Например, она используется для подсчете метрик качества в лидерборде huggling face.

7 Практическая часть

7.1 vLLM

vLLM [5] — это библиотека с открытым исходным кодом для эффективного и быстрого запуска больших языковых моделей (LLM). Она реализует инновационную технологию управления памятью (PagedAttention [13]), что позволяет ускорить генерацию текстов и обслуживать больше параллельных запросов на современных видеокартах. vLLM поддерживает популярные модели, такие как Llama, GPT, Falcon и другие, и интегрируется с интерфейсами HuggingFace Transformers и OpenAI API. Благодаря этому vLLM подходит для развертывания LLM в продакшене с минимальными задержками и высокой производительностью. Также стоит отметить, что она полностью open-source (Apache 2).

7.2 Развертывание инференса LLM с помощью vllm

Используемые инструменты:

- Prometheus [4] — это система мониторинга и сбора метрик времени выполнения, которая автоматически собирает и хранит показатели из различных сервисов и приложений. Она удобна для отслеживания производительности, выявления узких мест и анализа состояния инференса LLM
- Grafana [2] — это система визуализации данных, которая позволяет строить дашборды на основе метрик, полученных от Prometheus. С помощью Grafana можно удобно отслеживать производительность и загрузку моделей в реальном времени, а также настраивать оповещения на основе собранных данных.
- Docker Compose [1] — это инструмент, который позволяет управлять многоконтейнерными приложениями с помощью одного конфигурационного файла (обычно docker-compose.yml).В нашем случае с помощью Docker Compose можно одновременно запускать вместе Prometheus, Grafana и vLLM, упростив тем самым развертывание и управление всеми сервисами.

  Разберем на примере запуска модели **deepseek-ai/DeepSeek-R1-Distill-Qwen-7B**

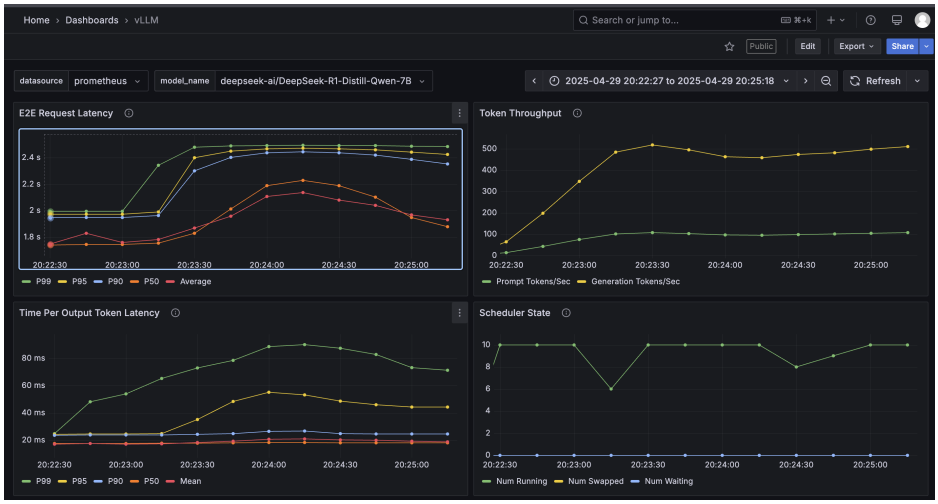


Рис. 1: Вывод метрик во время бенчмарка

1. Необходимо установить Docker, Docker-Compose и vLLM
2. Запуск сервера: можно запускать через докер, но в самом простом случае достаточно: vllm serve deepseek-ai/DeepSeek-R1-Distill-Qwen-7B –max-model-len 2048. Логирование метрик Prometeus включено по умолчанию.
3. Запускаем Prometheus, Grafana через docker compose (примеры в репозитории)
4. На данном этапе можно посмотреть на метрики производительности через ручку: http://localhost:8000/metrics.
5. Настроив Grafana можно смотреть на метрики через dashboard: http://localhost:3000/dashboard/import

7.3 Бенчмарк на производительность

Цель данного эксперимента заключалась в исследовании поведения инференса при имитации реальной жизни, а также в сравнении производственных метрик для обычной модели и квантизованной (метод: GPTQ, 4 бита, group-size=128), а также модель с весами в $\mathbb{FP}8$

Входные параметры:

- Модель: deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
- Квантизованная модель : boboxa/DeepSeek-R1-Distill-Qwen-7B-GPTQ-4bit
- Модель$\mathbb{FP}8$: deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
- Видеокарта: A800 80G
- Количество запросов: 1000
- Количество тредов: 10

Устройство бенчмарка:

1. Генерируем случайные промпты
2. Отправляем их нашей модели параллельно
3. Считаем метрики: latency (mean, p50, p90, p99), throughput

Замечание 11. p50, p90, p99 — перцентили задержки (например, p99 = 99% запросов выполняются быстрее этого значения).

Результаты:

Лучший результат по пропускной способности и задержке показала модель, загруженная в FP8, потом идет исходная модель, на последнем месте идет GPTQ, который выполнялся практически на 20 процентов дольше, чем оригинальная модель. Это связано с тем, что сейчас в vLLM нет хорошей поддержки данного метода квантизации. Но при этом стоит отметить, что он занимает меньше, так как веса представлены 4-битным типом данных. Это может быть важнее в ситуации нехватки видеопамяти.

Таблица 2: Benchmark Comparison of Three Models

Metric	**Original**	**GPTQ(4bit)**	**FP8**
Total requests	1000	1000	1000
Concurrency	10	10	10
Success rate	100.0%	100.0%	100.0%
Total time (sec)	191.07	236.76	160.33
Throughput (req/sec)	5.23	4.22	6.24
Avg latency (sec)	1.90	2.36	1.60
P50 latency (sec)	1.89	2.34	1.70
P90 latency (sec)	2.18	2.73	1.98
P99 latency (sec)	2.44	2.96	2.36
Avg input tokens	20.74	20.75	20.68
Avg output tokens	99.46	99.08	99.06

7.4 Анализ влияния KV-cache на скорость инференса

В рамках данного эксперимента было исследовано влияние KV-cache на скорость инференса и на размер потребляемой памяти.

- Модель: facebook/opt-125m
- Видеокарта: A800 80G
- Размер промпта: [10, 50, 100, 200]
- Максимальное количество токенов: 100
- Размер батча: [1, 4, 8, 16]

Описание эксперимента: Модель запускалась с kv-cache и без, а также с разными размерами входной последовательности и размером батча, при этом максимальный размер генерируемой последовательности оставался одинаковым.

Результат эксперимента: Действительно, KV-cache оказывает существенно влияние как на задержку, так и на количество токенов в секунду. Уже для батча размером 8 разница в latency достигает несколько секунд, а throughput в два-три разы выше.

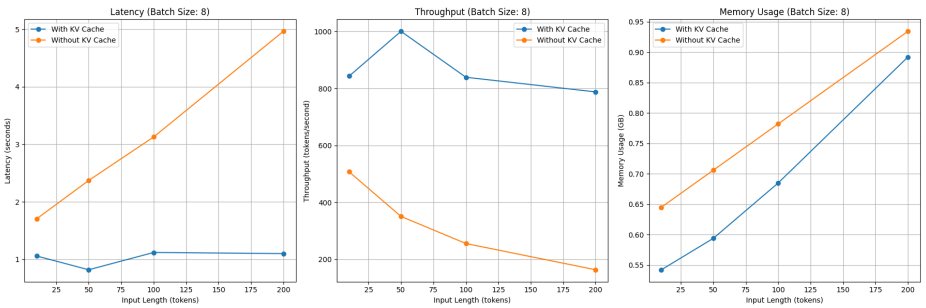
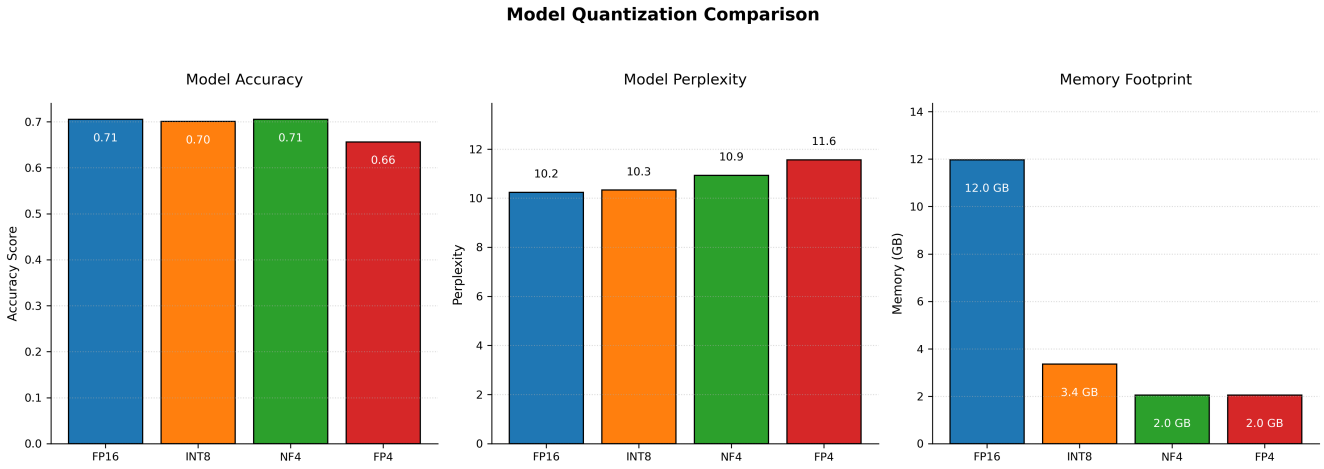


Рис. 2: Метрики производительности с KV-cache и без

7.5 Влияние квантизации на качество генерации

- Модель: meta/llama3.2-3b
- Видеокарта: A800 80G
- Метод квантизации: LLM-8bit(), NF4, FP4

В рамках данного эксперимента была квантизована модель llama3.2-3b с помощью методов из библиотеки bitsandbytes, вычислены метрики качества генерации для всех моделей, а также размер памяти, который они занимают. Таким образом, для данной модели оптимальным методом квантизации можно считать NF4, так как при минимальном, среди предложенных моделей, занимаемом месте показывает сопоставимые с оригинальной моделью результаты: по accuracy на LAMBDA без потерь относительно исходной модели, а относительная разница перклексити составляет 6 процентов. Также стоит отметить, что метод LLM8bit() показал самые близкие к оригиналу значения перплексити, но при этом работал в несколько раз медленнее, чем остальные методы. Как мне кажется, это может быть связано с тем, что метод изначально предполагался к применению на больших моделях.



Список литературы

[1] Docker. URL: https://docs.docker.com/compose/.

[2] Grafana. URL: https://prometheus.io/.

[3] Hugging face. URL: https://huggingface.co/docs/transformers/kv_cache.

[4] Prometheus. URL: https://grafana.com/.

[5] vllm. URL: https://github.com/vllm-project.

[6] Wikitext2. URL: https://huggingface.co/datasets/mindchain/wikitext2.

[7] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL: https://arxiv.org/abs/2208.07339, arXiv:2208.07339.

[8] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023. URL: https://arxiv.org/abs/2305.14314, arXiv:2305.14314.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL: https://arxiv.org/abs/2210.17323, arXiv:2210.17323.

[10] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL: https://zenodo.org/records/12608602, doi:10.5281/zenodo.12608602.

[11] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021. URL: https://arxiv.org/abs/2103.13630, arXiv:2103.13630.

[12] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020. URL: https://arxiv.org/abs/1904.09751, arXiv:1904.09751.

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL: https://arxiv.org/abs/2309.06180, arXiv:2309.06180.

[14] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL: https://arxiv.org/abs/1606.06031, arXiv:1606.06031.

[15] I. Sadrtdinov. Deep learning 1. 2023. URL: https://github.com/isadrtdinov/intro-to-dl-hse/tree/2023-2024/lecture-notes.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL: https://arxiv.org/abs/2302.13971, arXiv:2302.13971.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL: https://arxiv.org/abs/1706.03762, arXiv:1706.03762.

[18] Elena Voita. NLP Course For You, Sep 2020. URL: https://lena-voita.github.io/nlp_course.html.

[19] BigScience Workshop. Bloom: A 176b-parameter open-access multilingual language model, 2023. URL: https://arxiv.org/abs/2211.05100, arXiv:2211.05100.