Технологии оптимизации современных LLM моделей

А. С. Алексеев

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

unn-alex@yandex.ru

Аннотация. **Исследование посвящено системному анализу технологий оптимизации крупных языковых моделей (LLM), таких как GPT и DeepSeek, с целью снижения их вычислительной сложности, энергопотребления и затрат на эксплуатацию. Рассмотрены современные методы оптимизации, включая сжатие входных данных, квантование (LLM.int8(), SmoothQuant), управление памятью (PagedAttention, KV-кэш), Grouped-Query Attention (GQA) и Sliding Window Attention (SWA). Особое внимание уделено аппаратным ускорителям (HLS, HIDA) для повышения производительности и энергоэффективности. Результаты показывают, что предложенные методы позволяют значительно снизить затраты на эксплуатацию LLM, делая их более доступными конечному пользователю и более энергоэффективными. Работа вносит вклад в развитие методов системного анализа и оптимизации сложных систем.**

Ключевые слова: крупные языковые модели (LLM), оптимизация, производительность, энергопотребление, квантование, управление память, KV-кэш, сжатие предложений, механизмы внимания, параллелизация, высокоуровневый синтез

I. Введение

В последние годы крупные языковые модели (LLM), такие как GPT и BERT, стали неотъемлемой частью современных технологий, находя применение в обработке естественного языка, автоматизации бизнес-процессов и генерации текста. Однако их использование сопряжено с высокими финансовыми и экологическими затратами. Например, обучение GPT-3 потребовало энергии, эквивалентной выбросу 552 тонн углекислого газа, а эксплуатация ChatGPT обходится в более чем 700,000 долларов США в день [9].

Эти факторы подчеркивают необходимость оптимизации LLM для снижения затрат и повышения экологической устойчивости. LLM основаны на архитектуре трансформера, которая включает механизм внимания (attention), позиционные кодировки, слои Feed-Forward Neural Networks (FFN), нормализацию и остаточные связи, а также кэш ключей и значений (KV-кэш) [3]. Эти компоненты требуют значительных вычислительных ресурсов, что делает оптимизацию LLM критически важной для снижения затрат и повышения экологической устойчивости. В данном исследовании рассмотрены современные методы оптимизации, направленные на улучшение производительности и энергоэффективности LLM.

II. Объект и предмет исследования

Объект исследования – архитектура крупных языковых моделей (LLM) и аппаратные ускорители, используемые для их оптимизации.

Предмет исследования – сравнительный анализ методов оптимизации LLM, включая сжатие данных, квантование, управление памятью и аппаратные ускорители. Анализ будет проводиться по критерию ускорения работы LLM по сравнению с её стандартной производительностью.

III. Методы оптимизации LLM

A. Сжатие входных данных

Одним из ключевых направлений оптимизации является сокращение количества входных токенов без потери смысловой информации. Метод сжатия предложений на основе обучения с подкреплением (Sentence Compression with Reinforcement Learning – SCRL) позволяет уменьшить объём данных, поступающих на вход модели, что снижает вычислительную нагрузку. Этот подход демонстрирует высокую скорость работы на этапе инференции, что делает его привлекательным для реальных приложений. Однако метод имеет ограничения, такие как чувствительность к различиям между данными обучения и применения, а также возможные грамматические ошибки при сжатии предложений до очень короткой длины [10]. В статье не упоминается конкретное применение данного алгоритма к работе с LLM, но зная сложность механизма внимания относительно длины контекста ($O(n^2)$) [13], то при указанной способности сжатия входных предложений в 3–4 раза и учитывая возможные накладные расходы, можем вывести итоговое ускорение работы LLM в 2–5 раз при сталкивании с длинными запросами.

B. Квантование

Важным аспектом оптимизации LLM является снижение вычислительной сложности матричных операций, которые занимают значительную часть ресурсов. Методы квантования, такие как LLM.int8() и SmoothQuant, позволяют уменьшить требования к памяти и ускорить выполнение операций. LLM.int8() [1] использует двухэтапный подход: векторное квантование и смешанная точность, что позволяет эффективно обрабатывать выбросы в активациях моделей. SmoothQuant, в свою очередь, переносит сложность квантования на веса модели, что позволяет квантовать как веса, так и активации в формате INT8 [2]. Оба метода

демонстрируют высокую эффективность в снижении требований к памяти и ускорении выполнения операций, достигая ускорения в 2–4 раза для моделей с миллиардами параметров. И способны производить квантование моделей свыше 6.7 млрд параметров, в отличие от более ранних алгоритмов квантования, что имели большие потери в точности:

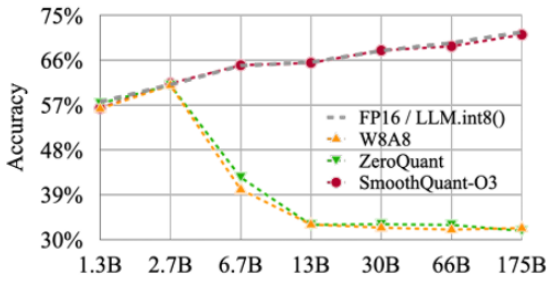

Рис. 1. Сравнение точности модели OPT при применении методик квантования W8A8, ZeroQuant, SmoothQuant и LLM.int8() – последние два не показывают спад точности после 6.7 млрд параметров [2]

Оба метода демонстрируют высокую эффективность в снижении требований к памяти и ускорении выполнения операций. Однако SmoothQuant показывает лучшие результаты в плане задержки генерации токенов и потребления памяти по сравнению с LLM.int8(). Например, SmoothQuant обеспечивает увеличение скорости вплоть до 1.5 раз и снижение потребляемой памяти вплоть до 1.96 раз по сравнению с оригинальной моделью. [2]

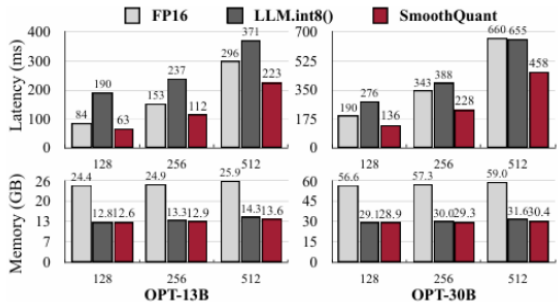

Рис. 2. Сравнение задержки генерации (сверху) и потребляемой памяти (снизу) при использовании LLM.int8() и SmoothQuant (средний и левый столбики соответственно) [2]

C. *Управление памятью (KV-кэш)*

Управление памятью, выделяемой для хранения ключей и значений (KV-кэш) в механизме внимания, также является важной проблемой. Традиционные системы управления памятью страдают от фрагментации и неэффективного использования ресурсов. Метод PagedAttention, вдохновлённый техникой подкачки страниц, позволяет гибко управлять памятью, разделяя KV-кэш на блоки фиксированного размера и устраняя как внутреннюю, так и внешнюю фрагментацию [3].

На основе PagedAttention авторы разработали систему vLLM, которая представляет собой высокопроизводительный движок для обслуживания LLM. vLLM использует централизованный планировщик для координации выполнения запросов на распределённых GPU. Ключевым компонентом системы является менеджер KV-кэша, который управляет памятью KV-кэша в виде блоков. Менеджер KV-кэша поддерживает таблицы блоков, которые отображают логические блоки на физические блоки памяти. Это позволяет системе динамически выделять память для новых токенов и освобождать память после завершения генерации последовательности [3].

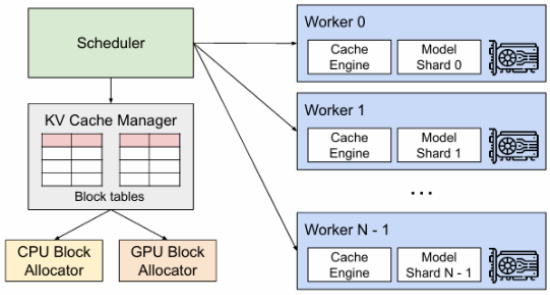

Рис. 3. Архитектура системы vLLM [3]

D. *Оптимизация механизмов внимания*

Оптимизация использования памяти декодером также была рассмотрена в исследовании. Метод Grouped-Query Attention (GQA) [11] предлагает компромисс между многоголовым вниманием (Multi-Head Attention, MHA) и многозапросным вниманием (Multi-Query Attention, MQA). GQA группирует запросы в несколько групп, каждая из которых использует общий ключ и значение, что позволяет снизить нагрузку на память, сохраняя при этом высокое качество модели.

GQA позволяет преобразовать существующие модели с многоголовым вниманием в модели со сгруппированным вниманием с минимальными затратами на дообучение. Создатели метода [11] описывают процесс преобразования, который включает два этапа: сначала ключевые и ценностные проекционные матрицы всех голов объединяются в одну матрицу путём усреднения (mean pooling), а затем модель дообучается на небольшой части исходных данных (например, 5 % от общего объёма предобучения). Это позволяет значительно сократить вычислительные затраты по сравнению с обучением модели с нуля. Этот метод особенно эффективен для больших моделей, где количество голов внимания может быть очень большим, и используется в наиболее успешных современных LLM, таких как Mistral 7B или Microsoft Phi-4 Mini [12]. GQA позволяет ускорить процесс декодирования вплоть до 5.39 раз быстрее оригинала, сохраняя качество, близкое к многоголовому вниманию (MHA), при этом скорость работы близка к многозапросному вниманию (MQA) [11].

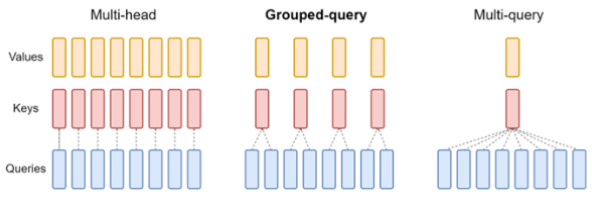

Рис. 4. Сгруппированное внимание как компромисс между многоголовым и многозапросным [11]

В контексте оптимизации алгоритма вычисления внимания был рассмотрен метод Sliding Window Attention (SWA), который позволяет значительно снизить вычислительные затраты и требования к памяти, сохраняя при этом высокую производительность модели. SWA ограничивает область внимания для каждого токена фиксированным окном из предыдущих токенов, что делает сложность линейной относительно длины последовательности [12]. Этот метод особенно полезен в задачах, где требуется обработка длинных последовательностей, таких как анализ документов или генерация текста. Метод используется в модели Mistral 7B (7 млрд параметров), которая знаменита тем, что превосходит в производительности и качестве аналогичные по архитектуре модели в 13 млрд параметров.

E. Параллельное предсказание токенов (MEDUSA)

Особое внимание уделено методу MEDUSA, который ускоряет вывод LLM за счёт параллельного предсказания нескольких токенов. MEDUSA добавляет к основной модели несколько дополнительных декодирующих голов, каждая из которых отвечает за предсказание токена на определённой позиции в будущем. Это позволяет модели генерировать несколько токенов за один шаг, что значительно сокращает количество шагов декодирования. Например, если основная модель предсказывает токен на позиции t+1, то первая дополнительная голова предсказывает токен на позиции t+2, вторая — на t+3, и так далее. Каждая декодирующая голова представляет собой однослойную нейронную сеть с остаточным соединением, что делает её простой в реализации и эффективной с точки зрения вычислений [5].

Для обработки множественных кандидатов, генерируемых MEDUSA Heads, используется древовидный механизм внимания. Этот механизм позволяет одновременно обрабатывать несколько кандидатов, что повышает вероятность принятия более длинных последовательностей токенов за один шаг декодирования. В отличие от традиционного каузального внимания, где каждый токен зависит от всех предыдущих токенов, в древовидном внимании каждый токен зависит только от своих предшественников в рамках одной ветви дерева. Это позволяет эффективно обрабатывать несколько кандидатов без необходимости увеличения размера батча [5].

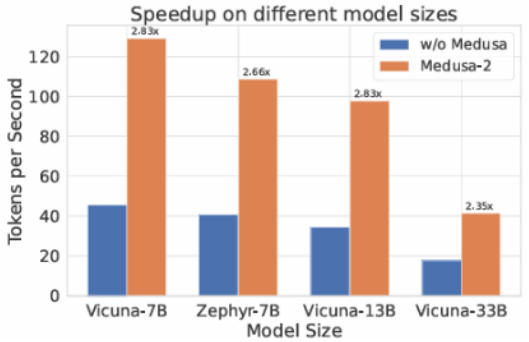

Рис. 5. Результаты скорости работы моделей Vicuna-7B, Zephyr-7B, Vicuna-13B и Vicuna-33B с и без фреймворка MEDUSA (левый и правый столбики соответственно) [5]

MEDUSA демонстрирует ускорение вывода в 2.35–2.83 раза без потери качества генерации. Например, на модели Vicuna-7B Medusa-1 достигает ускорения в 2.18 раза, а Medusa-2 – до 2.83 раза. На модели Vicuna-13B результаты аналогичны, что подтверждает эффективность метода для моделей различного размера. MEDUSA особенно полезна для задач, требующих быстрой обработки запросов, таких как чат-боты и системы автоматического перевода [5].

F. Аппаратные ускорители

Наконец, в исследовании были рассмотрены методы оптимизации аппаратных ускорителей, такие как High-Level Synthesis (HLS) и HIDA. HLS позволяет автоматически генерировать аппаратные ускорители для LLM, оптимизированные под конкретные задачи, что значительно повышает производительность и снижает энергопотребление [6]. Например, для LLM можно создать ускорители, специально предназначенные для выполнения операций внимания или матричных умножений, которые являются ключевыми для таких моделей. Эти ускорители могут быть реализованы на FPGA, что позволяет достичь значительного ускорения вычислений по сравнению с традиционными CPU или GPU. Кроме того, FPGA обеспечивают высокую степень параллелизма, что особенно важно для задач, связанных с обработкой больших объёмов данных, таких как LLM. [6]

Метод HIDA, в свою очередь, предлагает новый подход к автоматизации высокоуровневого синтеза, используя иерархическое промежуточное представление для моделирования потока данных. Этот метод позволяет эффективно оптимизировать доступ к памяти и распараллеливание задач, что делает его особенно полезным для LLM-моделей, где производительность и эффективность использования ресурсов являются критическими факторами [8].

Помимо этого, в исследовании также рассмотрен метод ScaleHLS, который использует многоуровневое промежуточное представление (MLIR) для оптимизации сложных иерархических структур, таких как LLM. ScaleHLS позволяет автоматически исследовать пространство проектирования (DSE) и находить оптимальные конфигурации для различных параметров оптимизации, таких как размер блоков для разбиения циклов и степень распараллеливания [7]. Этот метод демонстрирует ускорение вычислений в 768 раз по сравнению с базовыми реализациями, что делает его мощным инструментом для оптимизации LLM.

IV. Заключение

В заключение можно сказать, что современные методы оптимизации LLM, рассмотренные в данном исследовании, представляют собой важный шаг на пути к созданию более доступных, энергоэффективных и экологически устойчивых языковых моделей. Однако, несмотря на значительные успехи в этой области, остаются нерешённые проблемы, такие как адаптация методов оптимизации к различным архитектурам моделей и языкам, а также дальнейшее снижение вычислительных затрат без потери качества работы моделей.

Также в заключение можно подвести итоги сравнительного анализа методов оптимизации по критерию прироста производительности по сравнению со стандартной, неоптимизированной LLM. Итоговые результаты анализа вышеперечисленных методов представлены в следующей таблице повышения производительности. Прирост обозначен в формате xN, где N – положительное число и такая запись означает, что LLM производит ответ на запрос в N раз быстрее по сравнению со стандартным состоянием. Для стандартного состояния коэффициент N будет считаться равным единице:

ТАБЛИЦА I.

Технология	**Прирост производительности**
Стандартная аппаратная архитектура (CPU + GPU)	
Стандартная LLM	x1
SCRL	x2 – x5 (x1 при коротком входном запросе)
LLM.int8()	x1.5
SmoothQuant	x1.5 (но в 1.96 раз меньше расход памяти)
PagedAttention + vLLM	x2 – x4
MEDUSA	x2.35 – x2.83
GQA	x5.39
SWA	Трудно оценить.
Настраиваемая аппаратная архитектура (FPGA)	
HLS (ScaleHLS, HIDA)	x768.1

Также, следует отметить, что многие из этих методов комбинируются между собой. Например, оптимизированная модель Mistral 7B использует технологии как SWE, так и GQA вместе. Также теоретически возможно при использовании модели применить к ней квантование и запустить через фреймворк vLLM, что даст максимальный прирост производительности.

Дальнейшего изучения требует влияние оптимизации структуры LLM на экологический аспект использования и на доступность технологий для малого и среднего бизнеса.

Благодарность

Выражаю благодарность Шошкову Н.О. за возможность участия в конференции и за помощь с составлением материала и необходимой документации.

Список литературы

[1] Tim Dettmers, Mike Lewis, Younes Belkada, Luke Zettlemoyer LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale, 2022. arXiv preprint arXiv:2208.07339.

[2] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. 2023. In ICML.

[3] Woosuk Kwon, Zhuohan Li1, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. 2023 In SOSP.

[4] Shivanshu Shekhar, Tanishq Dubey, Koyel Mukherjee, Apoorv Saxena, Atharv Tyagi, Nishanth Kotla. Towards Optimizing the Costs of LLM Usage. 2024. arXiv:2402.01742v1.

[5] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, Tri Dao. MEDUSA: Simple LLMInference Acceleration Framework with Multiple Decoding Heads. 2024. arXiv:2401.10774v3.

[6] Hanchen Ye, Hyegang Jun, Jin Yang, Deming Chen. High-level Synthesis for Domain Specific Computing. 2023. In ISPD.

[7] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, Deming Chen. ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation. 2021. arXiv:2107.11673v4.

[8] Hanchen Ye, Hyegang Jun, Deming Chen. HIDA: AHierarchical Dataflow Compiler for High-Level Synthesis. 2023. arXiv:2311.03379v1.

[9] Kingsum Chow, Yu Tang, Zhiheng Lyu, Anil Rajput, Khun Ban. Performance Optimization in the LLM World 2024. 2024. In ICPE

[10] Demian Gholipour Ghalandari, Chris Hokamp, Georgiana Ifrim. Efficient Unsupervised Sentence Compression by Fine-tuning Transformers with Reinforcement Learning. 2022. arXiv:2205.08221v1

[11] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, Sumit Sanghai. GQA:Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. 2023. arXiv:2305.13245v3

[12] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed. Mstral 7B release Paper. 2023. arXiv:2310.06825v1

[13] Daniel Jurafsky, James H. Martin Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3-е изд. 2024. 599 с. https://web.stanford.edu/~jurafsky/slp3/