Научная статья

УДК 004.89
DOI 10.25205/1818-7900-2025-23-4-44-61

Исследование методов оптимизации скорости исполнения больших языковых моделей для задачи распознавания команд

Александр Игоревич Гончаренко[1]
Максим Иванович Чупров[2]
Евгений Семенович Нежевенко[3]

[1]Институт интеллектуальной робототехники НГУ
Новосибирск, Россия

[2]ООО «Экспасофт»
Новосибирск, Россия

[3]Институт автоматики и электрометрии СО РАН
Новосибирск, Россия

a.goncharenko@expasoft.tech; https://orcid.org/0009-0000-5087-8506
m.chuprov@expasoft.tech
nedj@iae.nsk.su

Аннотация

Целью данной работы являлось исследование и реализация методов оптимизации (особенно методов прунинга) больших языковых моделей для задачи function calling, а также сравнение точности и скорости работы полученных моделей.

В качестве базовой модели была выбрана модель Mistral-7B. Для эффективной тренировки модели использовался датасет glaive-function-calling-v2, предназначенный для задачи function calling. Для обучения базовой модели использовалось квантование до 4 бит в формате nf4 и двойное квантование в сочетании с методом QLoRA (Quantized Low-Rank Adaptation).

Оптимизация модели проводилась несколькими способами: (1) с использованием метода ShortGPT, (2) с помощью критерия Тейлора для послойного прунинга, (3) методом LLM-Pruner, который отбрасывает параметры модели поканально, оставляя при этом количество слоев модели неизменным, и (4) методом PowerInfer, который использует свойство контекстуальной разреженности в больших языковых моделях. Для всех перечисленных способов оптимизации были построены оптимизированные модели, и проведено сравнение точности и скорости работы полученных моделей.

Результаты экспериментов показали, что наибольшая точность была достигнута на модели, которая была оптимизирована с помощью метода послойного прунинга по критерию Тейлора важности слоя. Для данного метода был проведен ряд экспериментов, в которых исследовалась разная расстановка гейтов внутри слоя декодера, а также различные способы агрегирования важности слоя на гейтах. По итогам экспериментов можно сделать вывод, что расстановка гейтов после блоков Multi-Head Attention и использование агрегирования важности с помощью L2-нормы вектора градиентов дают наибольшую точность по сравнению с другими возможными вариантами.

Научная значимость работы состоит в сравнении передовых методов прунинга, исходя из соотношения качество/скорость модели, и получении ускоренной версии модели для задачи function calling.

Ключевые слова

прунинг, квантование, ряд Тейлора, большие языковые модели, механизм внимания, function calling, PowerInfer

Благодарности

Авторы выражают благодарность Чеблаковой Елене Анатольевне за помощь в оформлении статьи.

Для цитирования

Гончаренко А. И., Чупров М. И., Нежевенко Е. С. Исследование методов оптимизации скорости исполнения больших языковых моделей для задачи распознавания команд // Вестник НГУ. Серия: Информационные технологии. 2025. Т. 23, № 4. С. 44–61. DOI 10.25205/1818-7900-2025-23-4-44-61

Research of inference speed optimization methods of large language models for function calling task

Alexander I. Goncharenko[1], Maxim I. Chuprov[2]
Evgeniy S. Nejevenko[3]

[1]Institute of Intelligent Robotics of Novosibirsk State University
Novosibirsk, Russian Federation

[2]Expasoft LLC
Novosibirsk, Russian Federation

[3]Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences
Novosibirsk, Russian Federation

a.goncharenko@expasoft.tech; https://orcid.org/0009-0000-5087-8506
m.chuprov@expasoft.tech
nedj@iae.nsk.su

Abstract

This work is devoted to study and practical implementation of optimization methods (especially pruning) for large language models (LLM) in the context of function calling task, as well as comparison of accuracy and speed of the obtained models.

Authors chose Mistral-7B as the basic model; glaive-function-calling-v2 – as dataset for training. 4-bit quantization in nf4 format and double quantization were used in combination with QLoRA (Quantized Low-Rank Adaptation) method. Four different pruning methods were applied for model optimization. The first method, ShortGPT, focuses on reducing the model size by trimming less significant parts. The second method is based on Taylor's criterion for layer-by-layer pruning. The third method, LLM-Pruner, removes parameters channel-by-channel maintaining the total number of layers. The fourth method, PowerInfer, uses contextual sparsity of large language models. Optimized models were implemented for all these methods; the accuracy and speed of resulting models were compared.

Results of experiments show that the highest accuracy was achieved using the layer-by-layer pruning according to Taylor's criterion of layer importance. This method was tested with different placement of gates within the decoder layer and different ways of aggregation of layer importance on the gates. Experiments show that best results were achieved by placing the gates after Multi-Head Attention blocks and using the L2 norm of the gradient vector to aggregate layer importance.

Scholarly importance of the work includes comparison of advanced pruning methods in the context of quality/speed ratio and obtaining a speed up version model for the function calling task.

Keywords

pruning, quantization, Taylor series, large language models, attention mechanism, function calling, PowerInfer.

Acknowledgements

The authors thank Elena A. Cheblakova for help in drafting the article.

For citation

Goncharenko A. I., Chuprov M. I., Nejevenko E. S. Research of inference speed optimization methods of large language models for function calling task. *Vestnik NSU. Series: Information Technologies*, 2025, vol. 23, no. 4, pp. 44–61 (in Russ.) DOI 10.25205/1818-7900-2025-23-4-44-61

Введение

В последнее время большую популярность приобрели большие языковые модели (Large Language Models, LLM). Они стали ключевым инструментом в различных областях, начиная с обработки естественного языка и генерации текста и заканчивая агентными системами и база-

ми знаний. Такие модели, как GPT (Generative Pretrained Transformer) [1] и BERT (Bidirectional Encoder Representations from Transformers) [2], показали впечатляющие результаты в области понимания и обработки человеческого языка.

Отличительно чертой больших языковых моделей является использование огромных объемов данных и вычислительных ресурсов для достижения высокой производительности и универсальности. Целью данной работы являлось исследование и реализация методов оптимизации больших языковых моделей для повышения их эффективности и доступности в использовании.

1. Анализ предметной области

1.1. Языковое моделирование и большие языковые модели

На данный момент в области обработки естественного языка (Natural Language Processing, NLP) одной из самых распространенных задач является задача языкового моделирования. Языковое моделирование является основой для многих приложений NLP, таких как машинный перевод, суммаризация, ответы на вопросы (Question answering, QA) и др.

Языковое моделирование представляет собой предсказание следующего слова или последовательности слов в заданном контексте на основе статистического анализа языка. Для решения данной задачи существуют различные языковые модели, начиная от ранних Word2Vec [3] и заканчивая современными большими языковыми моделями.

В основе больших языковых моделей обычно лежит архитектура «трансформер» [4], точнее, ее авторегрессионная часть для генерации последовательностей в виде текста, также называемая декодером. Трансформер – это архитектура глубокой нейронной сети, предназначенная для обработки последовательностей, таких как тексты или временные ряды. Она основана на механизме внимания (self-attention), который позволяет модели фокусироваться на различных частях входных данных в зависимости от их важности для конкретной задачи. Механизм внимания в трансформере реализуется блоками Multi-Head Attention. Иллюстрация работы этих блоков приведена на рис. 1.

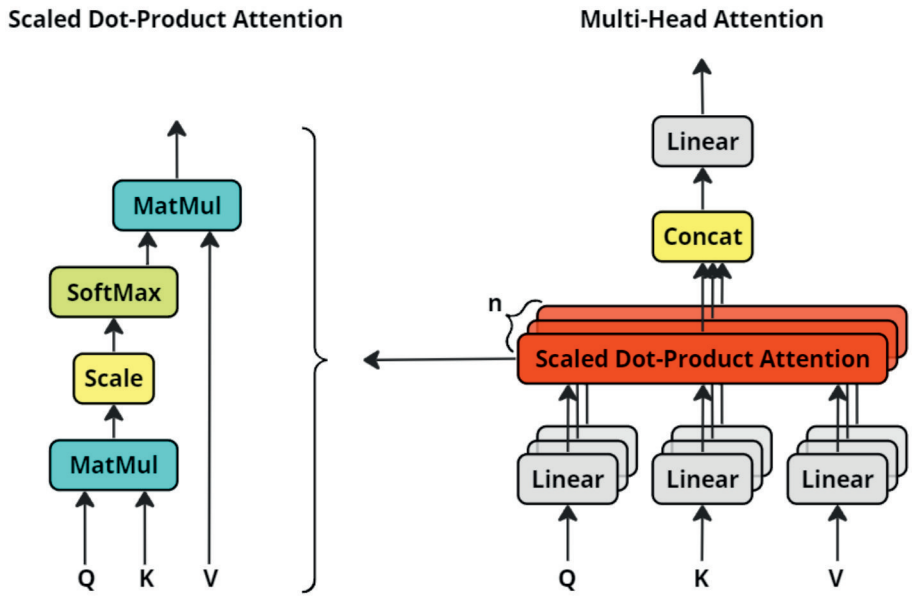

Рис. 1. Блок Multi-Head Attention
Fig. 1. Multi-Head Attention block

1.2. Методы оптимизации нейронных сетей

Основными методами оптимизации нейронных сетей являются прунинг и квантование.

1.2.1. Прунинг

Прунинг – это метод оптимизации, используемый в искусственных нейронных сетях для удаления отдельных параметров или групп параметров из существующей сети, чтобы сохранить точность сети и повысить ее эффективность. Существует два вида прунинга: структурированный и неструктурированный.

Структурированный прунинг предполагает удаление параметров из сети с определенной структурой, такой как удаление целых слоев или блоков параметров. Например, можно удалять целые сверточные фильтры в сверточных слоях или целые нейроны в полносвязных слоях. Этот подход обычно более прост в реализации и может обеспечить более стабильные результаты при сохранении производительности сети.

Неструктурированный прунинг, напротив, удаляет отдельные параметры независимо от их структуры, что может привести к более разреженным моделям. Например, это может быть удаление отдельных весов внутри сверточного фильтра или отдельных весов в полносвязных слоях. Хотя неструктурированный прунинг позволяет обеспечить большую степень сжатия модели, он может быть более сложен в реализации и требователен к вычислительным ресурсам.

Пример обоих видов прунинга изображен на рис. 2.

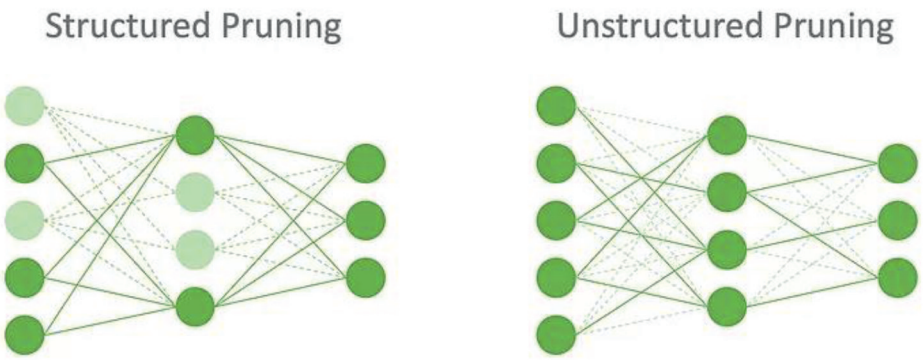

Рис. 2. Структурированный и неструктурированный прунинг
Fig. 2. Structured and unstructured pruning

Помимо этого, существуют различные критерии прунинга нейронных сетей, например, на основе информации об L2-норме параметра или их градиенте.

Так, при использовании L2-критерия, важность параметра определяется как

$$I_w = w^2,$$

где w – значение параметра; I_w – важность параметра w, определяемая как квадрат значения данного параметра. Важность группы параметров определяется как сумма важностей параметров:

$$I_W = \sum_{w \in W} w^2,$$

где W – группа параметров; I_W – важность группы параметров.

Критерий на основе градиента основывается на том, что важность параметра может быть оценена по ошибке, вызванной его удалением. Эта ошибка может быть измерена как разность значений функций потерь с параметром и без него следующим образом:

$$I_w = \left| L_w - L_{w=0} \right|,$$

где L_w – значение функции потерь с параметром w; $L_{w=0}$ – значение функции потерь без параметра w; I_w – важность параметра w. Если представить функции потерь в виде разложения Тейлора, то получим следующее:

$$I_w = \left| L_{w=0} + \frac{\partial L_w}{\partial w} w - L_{w=0} - \frac{\partial L_{w=0}}{\partial \left(w = 0 \right)} \cdot 0 \right| = \left| \frac{\partial L_w}{\partial w} w \right| = \left| g_w w \right|,$$

где g_w – значение градиента для веса w; I_w – важность веса w, которая считается как модуль произведения веса на значение градиента этого веса.

Приведенные примеры критериев для прунинга имеют свои достоинства и недостатки, которые следует учитывать при выборе конкретного метода. Так, к достоинствам критерия на основе L2-нормы весов можно отнести простоту применения и интуитивную интерпретацию. Недостатком критерия является то, что он не учитывает взаимосвязь между параметрами, что может привести к потере важной информации при прунинге.

У критерия на основе градиента наоборот, благодаря использованию градиента, появляется возможность оценить, насколько параметр влияет на функцию потерь, что позволяет сохранить наиболее важные параметры, но при этом сам критерий является вычислительно затратным из-за необходимости вычислять этот градиент.

1.2.2. Квантование

Квантование – это процесс уменьшения размера весов, смещений и активаций, обычно с 32-битных значений с плавающей точкой до более низких битов, например 16 или 8. Такое снижение точности позволяет получить более компактное представление модели, что приводит к снижению потребления памяти и повышению скорости вычислений.

В общем виде операция квантования определяется следующим образом:

$$Z = \left[q_{\min} - \frac{r_{\min}}{S} \right],$$

$$S = \frac{r_{\max} - r_{\min}}{2^b - 1},$$

$$X_q = \left[\frac{X}{S} + Z \right],$$

где Z – константа квантования, соответствующая нулевому значению; S – константа квантования, отвечающая за масштаб преобразования; $[r_{\min}, r_{\max}]$ – вещественный диапазон значений во входных данных; b – количество бит в квантованном типе данных; $q_{\min}$ – минимальное значение в квантованном типе данных; X – входные данные; X_q – квантованные данные.

Операция деквантования определяется как

$$X = S(X_q - Z).$$

Квантование имеет важное значение для развертывания больших нейронных сетей на устройствах с ограниченными ресурсами, таких как микроконтроллеры или одноплатные компьютеры, без значительного снижения точности.

1.3. Особенности оптимизации больших языковых моделей

При оптимизации больших языковых моделей следует учитывать особенности, связанные с их работой. Прежде всего это авторегрессионная природа больших языковых моделей. Авторегрессионные модели, такие как GPT, генерируют текст последовательно, токен за токеном, основываясь на ранее сгенерированных токенах. Это усложняет задачу параллелизации процесса генерации.

Другой важной особенностью является то, что для предварительного обучения больших языковых моделей требуются огромные вычислительные мощности и большие корпуса текста. Поскольку такие модели состоят из миллиардов параметров, предварительное обучение может стать трудоемкой или даже невозможной задачей в рамках оптимизации модели. Это происходит потому, что модели данного типа являются универсальными и предназначены для решения большинства типов задач без предварительного обучения.

1.4. Обзор существующих методов оптимизации

1.4.1. Метод LLM-Pruner

LLM-Pruner [5] представляет собой метод статического структурированного прунинга больших языковых моделей. Метод состоит из трех основных этапов: обнаружение зависимостей, оценка важности весов и восстановление качества работы.

В рамках этапа обнаружения зависимостей необходимо разбить языковую модель на независимые группы нейронов, а чтобы сгруппировать их, необходимо определить, как одни нейроны зависят от других. Авторы [5] вводят два вида зависимостей:

1) если нейрон N_j исходит только от нейрона N_i, то N_j зависит от N_i;

2) если нейрон N_i входит только в нейрон N_j, то N_i зависит от N_j.

Принцип зависимости заключается в том, что если текущий нейрон зависит исключительно от другого нейрона и этот другой нейрон подвергается прунингу, то и текущий нейрон также должен быть подвергнут прунингу. Используя такое определение зависимости, появляется возможность автоматически анализировать связанные структуры в языковой модели и затем группировать их для последующего прунинга.

После обнаружения всех зависимостей в модели наступает этап оценки важности весов. В рамках данного этапа из языковой модели удаляются те группы, которые имеют наименьшую важность по некоторому заданному критерию. Метод поддерживает множество критериев важности весов, однако основными являются критерии на основе L2-нормы весов и на основе градиентов. Определив важность каждого отдельного веса, появляется возможность определить важность группы, состоящей из этих весов. Авторы предлагают агрегировать информацию о важности группы четырьмя разными способами: суммированием важности весов в группе, произведением важности весов в группе, взятием максимума важности весов внутри группы и взятием важности последней структуры в группе. После оценки важности каждой группы авторы ранжируют группы по их важности, а затем прунят группы с более низкой важностью на основе заранее заданного коэффициента прунинга.

Для восстановления качества работы модели используется метод LoRA или QLoRA для минимизации количества обучаемых параметров, что позволяет сократить сложность обучения.

1.4.2. Метод ShortGPT

В отличие от метода LLM-Pruner, авторы метода ShortGPT [6] предложили метод структурированного прунинга по слоям, а не по каналам большой языковой модели. Делается это в два этапа:

1. Расчет меры важности каждого слоя.
2. Удаление определенной доли наименее важных слоев.

Авторы метода определяют меру важности с помощью косинусной близости между скрытыми состояниями на входе и выходе слоя. Чем меньше косинусная близость, тем более важен данный слой в большой языковой модели. Как показано на рис. 3, важность i-го слоя может быть посчитана как

$$score_i = 1 - \frac{1}{n}\sum_{t=1}^{n}\frac{X_{i,t}{}^{T} X_{i+1,t}}{\left\|X_{i,t}\right\|_2 \left\|X_{i+1,t}\right\|_2},$$

где $X_{i,t}$ – вектор скрытого состояния для t-го токена в последовательности на слое i; $X_{i+1,t}$ – вектор скрытого состояния для t-го токена в последовательности на слое $i + 1$; n – количество токенов в последовательности.

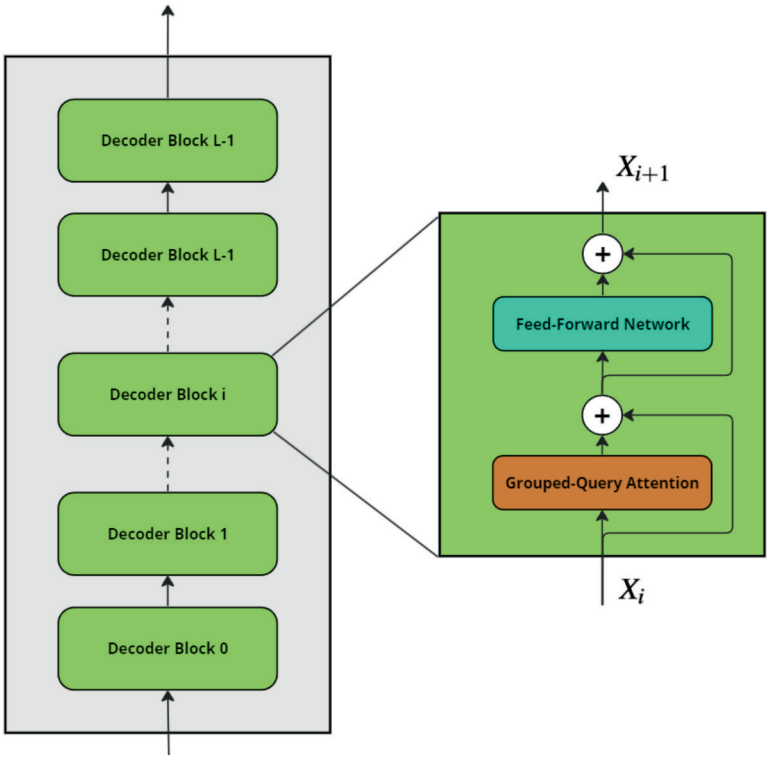

Рис. 3. Иллюстрация для пояснения формулы расчета важности
Fig. 3. Illustration to explain the importance calculation formula

Такой подход позволяет более эффективно сжимать модели, так как в нем, в отличие от LLM-Pruner, сокращается не только количество параметров и вычислений, но и количество последовательных операций.

1.4.3. Метод SparseGPT

SparseGPT [7] предлагает новый метод статического неструктурированного прунинга для моделей типа GPT, позволяющий провести быструю оптимизацию модели без дополнительного дообучения даже для моделей с сотнями миллиардов параметров. Это стало возмож-

ным из-за разреженности активаций в больших языковых моделях, когда только небольшая часть нейронов в каждом слое имеет влияние на активацию слоя.

Достигается такой результат благодаря применению подхода Mask Selection & Weight Reconstruction, где после прунинга части весов обновляются оставшиеся веса, чтобы компенсировать запруненные. В своем методе авторы используют несколько нововведений, таких как аппроксимация гессиана и итеративный выбор маски прунинга, которые снижают вычислительную сложность реконструкции весов без сильной потери качества запруненной модели.

Однако SparseGPT имеет несколько недостатков. Во-первых, он использует неструктурированный прунинг, в котором присутствуют разреженные вычисления для достижения ускорения нейросети. Даже при прунинге 50 % весов удается добиться ускорения не более чем в 1,6–1,7 раза по сравнению с исходной моделью. При большем проценте прунинга качество модели начинает существенно ухудшаться. Во-вторых, не все устройства эффективно поддерживают разреженные вычисления, из-за чего возникает ограниченность использования данного метода, особенно на встраиваемых устройствах.

1.4.4. Метод DejaVu

Авторы метода DejaVu [8] используют особенность LLM, связанную с разреженностью активаций. Они обнаружили, что активность нейронов на каждом слое напрямую зависит от входных значений на этих слоях. Эту особенность авторы назвали контекстуальной разреженностью, и в своей работе они исследуют контекстуальную разреженность в LLM для достижения ускорения без ухудшения качества модели.

Суть предложенного метода состоит в том, чтобы на основании входных данных предсказывать, какую малую часть нейронов в MLP и MHSA блоках необходимо активировать для достижения ускорения за счет уменьшения итоговых вычислений. Делается это за счет создания небольших MLP-предикторов поверх каждого блока в LLM, которые во время инференса модели предсказывают, какие нейроны нужно активировать, благодаря чему нет необходимости использовать все нейроны для предсказания модели. Это позволяет добиться значительного ускорения модели без существенного снижения качества ее работы.

1.4.5. Метод PowerInfer

PowerInfer развивает идеи предыдущего метода и также основывается на использовании контекстуальной разреженности для оптимизации модели. Однако, помимо оптимизации скорости инференса модели, PowerInfer [9] снижает потребление GPU, делая представленный фреймворк для инференса больших языковых моделей наиболее предпочтительным для запуска на устройствах с ограниченными ресурсами среди всех представленных выше методов.

PowerInfer представляет собой гибридный GPU/CPU-метод для инференса больших языковых моделей, использующий разреженное распределение активаций нейронов для достижения более быстрой скорости инференса на одном графическом процессоре. Предварительное размещение наиболее активных нейронов на графическом процессоре и менее активных нейронов на центральном процессоре, а также использование онлайн-предикторов для выбора нейронов для активации на графическом процессоре и центральном процессоре обеспечивает эффективную работу даже на пользовательских GPU, таких как RTX 4090 или RTX 2080Ti.

Такое решение позволяет запускать достаточно большие языковые модели (более 70 миллиардов параметров), добиваясь ускорения в 8–12 раз по сравнению с исходными моделями.

2. Материалы и методы

2.1. Выбор задачи и модели

Для дальнейших экспериментов в качестве задачи языкового моделирования была выбрана задача function calling, которая в контексте больших языковых моделей означает возможность модели вызывать определенные функции или API в зависимости от контекста запроса. Это позволяет большим языковым моделям предлагать вызов функций, что расширяет их возможности взаимодействия на естественном языке.

Function calling – это задача, приближенная к реальному практическому использованию больших языковых моделей в отличие от популярных бенчмарков с выбором ответа или генерацией односложного ответа. Function calling требует не только хорошего понимания пользовательского запроса, но и генерации сложного и точного ответа.

В качестве модели была выбрана Mistral-7B [10], которая является сильной базовой моделью для многих задач языкового моделирования, включая задачу function calling.

2.2. Тренировочные и тестовые данные

Для эффективной тренировки модели для задачи function calling использовался датасет glaive-function-calling-v2[1]. Он включает в себя 113 тысяч примеров и содержит обширный набор функций общего назначения. Помимо примеров с вызовом функции, в датасете также есть примеры, где либо отсутствует доступ к внешним функциям, либо отсутствуют подходящие для запроса функции.

Разбиение датасета на валидационную и тренировочную выборку проводилось следующим образом: выбиралось случайное подмножество функций, участвующих в вызовах в датасете, и все примеры с вызовами данных функций отбирались для валидационной выборки. Остальные примеры использовались для непосредственной тренировки модели. Такое разбиение позволяет избежать «утечки» данных, когда модель тренируется также на примерах из валидационной выборки – в данном случае, на примерах, где в тренировочной и валидационной выборках вызываются одни и те же функции.

2.3. Предобработка данных

Для выбранного датасета были выполнены определенные шаги предобработки данных, чтобы привести его к необходимому формату для последующего использования в обучении модели.

Предобработка glaive-function-calling-v2 производилась в несколько шагов:

– фильтрация примеров, где отсутствуют вызовы внешних функций, чтобы обеспечить наличие только релевантных данных для обучения;

– удаление текста после вызова функции. Данный шаг представляет возможный ответ системы в виде вызова функционального API;

– преобразование из формата мессенджера в формат mistral-instruct, чтобы стандартизировать представление данных.

После создания датасета и разбиения его на тренировочную и валидационную выборки была произведена синтетическая генерация примеров с множественными вызовами функций. Это было сделано для того, чтобы модель обрела возможность по единственному пользовательскому запросу вызывать несколько функций одновременно. При этом учитывалось, что в тренировочной выборке отсутствовали примеры с множественными вызовами функций.

[1] https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2 (дата обращения: 15.03.2024).

Аналогичная ситуация была в валидационной выборке, где также наблюдалось недостаточное количество таких примеров.

2.4. Метрики качества модели

В качестве метрик, по которым будет оцениваться качество модели, были выбраны следующие метрики:

1. Exact match. Это метрика, которая сопоставляет один в один ответ модели с правильным ответом.
2. Время генерации. Эта метрика показывает, сколько времени в миллисекундах уходит в среднем на генерацию одного токена моделью.

2.5. Обучение базовой модели

Обучение больших языковых моделей напрямую сопряжено с трудностями из-за их огромного размера. Например, чтобы обучить модель с 7 миллиардами параметров, потребуется приблизительно 28 гигабайт видеопамяти на графическом процессоре, что существенно превышает возможности современных пользовательских видеокарт.

Для преодоления этих трудностей в данной работе использовался метод квантования в сочетании с QLoRA (Quantized Low-Rank Adaptation). Квантование позволяет существенно сократить размер модели путем уменьшения количества байт, используемых для представления весов и активаций. В данной работе использовалось квантование до 4 бит в формате nf4 и двойное квантование. Формат nf4 – это тип данных, при котором значения представляют собой квантили стандартного нормального распределения в предположении, что веса модели распределены соответствующим образом. Квантование весов выполняется блоками по 64 элемента, и для каждого элемента хранятся константы в формате fp32 для последующего деквантования весов. Суть двойного квантования состоит в том, что дополнительно квантуются хранимые константы блоками по 256 элементов, и тем самым сокращается потребление памяти еще на 0,4 бита на параметр.

QLoRA, в свою очередь, позволяет дообучать квантованные модели с помощью обучаемой низкоранговой добавки к весам модели. Таким образом, основные веса остаются квантованными, что помогает сократить размер модели, а обучаемая добавка к весам позволяет дообучать модель на тренировочных данных. QLoRA применялась ко всем линейным слоям модели с рангом 8 и alpha, равным 16.

Иллюстрация работы метода QLoRA приведена на рис. 4.

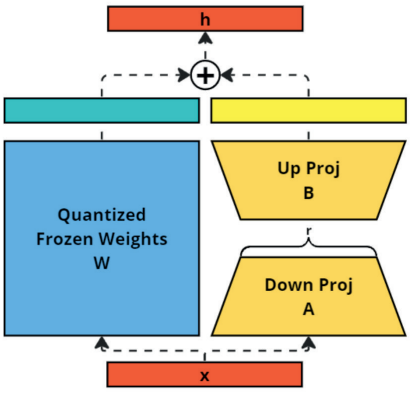

Рис. 4. Иллюстрация работы метода QloRA
Fig. 4. Illustration of QloRA method work

Для обучения модели использовались следующие гиперпараметры:

– batch_size: 1 (с накоплением градиента в 32 шага)
– steps: 4000
– warmup steps: 2000
– learning rate: 7e-5
– weight decay: 5e-5
– scheduler: линейный
– optimizer: paged AdamW 8bit

Процесс обучения базовой модели включал несколько шагов, а именно:

1. Загрузка предобученной модели Mistral с оптимизированными весами.
2. Квантование с использованием библиотеки bits&bytes, которая сжимает веса модели до 4 бит.
3. Добавление QLoRA-слоев на каждый линейный слой модели.
4. Загрузка подготовленного датасета, описанного в пункте «Предобработка данных».
5. Создание класса Trainer из библиотеки transformers для обучения модели.
6. Обучение QLoRA-слоев по заданным гиперпараметрам для минимизации ошибки во время обучения.
7. Сохранение обученной модели, в которой веса QLoRA объединены с весами модели.

2.6. Оптимизация методом ShortGPT

Оптимизация модели с использованием метода ShortGPT проходит в несколько шагов, начиная с калибровки на тренировочном датасете. Этот процесс начинается с прогона каждого примера из тренировочного датасета через модель, что позволяет получить скрытые состояния на входе и выходе каждого слоя модели для каждого примера. Затем для каждого примера и каждого слоя вычисляется косинусное расстояние между выходным и выходным скрытым состоянием, что позволяет оценить степень изменения состояний на разных слоях модели. После этого находится среднее значение между всеми примерами для каждого слоя, что позволяет оценить важность каждого слоя модели.

Вторым этапом оптимизации является прунинг, который основан на статистике важности каждого слоя, накопленной во время калибровки. Этот процесс начинается с сортировки слоев по степени их важности от наименее значимого до наиболее значимого. Затем выбираются 10 наименее важных слоев, которые удаляются из модели, тем самым сокращаются вычислительные и аппаратные требования к инференсу модели.

Последний этап оптимизации – дообучение. На этом этапе берется запруненная модель, из которой были удалены наименее важные слои, и проводится дообучение стандартной процедурой дообучения, тем самым восстанавливая потерянную способность к решению задачи function calling.

2.7. Оптимизация с помощью критерия Тейлора

Метод ShortGPT неплохо показал себя в качестве метода оптимизации моделей, используя в качестве критерия для прунинга косинусное расстояние между скрытыми состояниями на входе и выходе слоев. Однако такой критерий не учитывает информацию о влиянии удаленных слоев на итоговое значение функции потерь. В связи с этим была предложена идея использовать критерий Тейлора для послойного прунинга.

Суть метода заключается в добавлении «гейтов» [11] после блока GQA в каждом слое модели перед skip connection. Гейт – это слой в нейронной сети, представляющий собой вектор, инициализированный единицами, веса которого умножаются на входные значения и переда-

ются дальше. Такой подход позволяет накапливать градиенты на этих весах для дальнейшей калибровки модели.

Выясним, почему градиент, накопленный на гейте, действительно показывает важность той группы весов, которая с ним связана. Определим следующие функции:

$$y = \sum_{i=1}^{n} w_i x_i,$$
$$z = hy,$$

где y – выход группы весов; n – количество нейронов в группе; w_i – i-й вес в группе весов, x_i – вход для i-го веса; z – выход на гейте; h – вес на гейте, равный единице.

Найдем производные:

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial z} y = \frac{\partial L}{\partial z} \sum_{i=1}^{n} w_i x_i,$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \frac{\partial y}{w_i} = \frac{\partial L}{\partial z} x_i h = \frac{\partial L}{\partial z} x_i,$$

где L – значение функции потерь.

Важность параметра определяется как

$$I_w = g_w w.$$

Важность группы параметров определяется как сумма важностей всех параметров:

$$I_L = \sum_{i=1}^{n} w_i \frac{\partial L}{\partial w_i} = \sum_{i=1}^{n} w_i \frac{\partial L}{\partial z} x_i = \frac{\partial L}{\partial z} \sum_{i=1}^{n} w_i x_i = \frac{\partial L}{\partial h},$$

где I_L – важность данной группы. Таким образом, градиент, накопленный на гейте, равен сумме важностей параметров, входящих в данную группу.

При использовании гейтов возможно их различное расположение внутри слоя декодера. Всего можно представить четыре варианта расстановки: гейт после блока Multi-Head Attention, гейт после блока Feed-Forward Network, два разных гейта после обоих блоков, один и тот же гейт после обоих блоков (рис. 5–8).

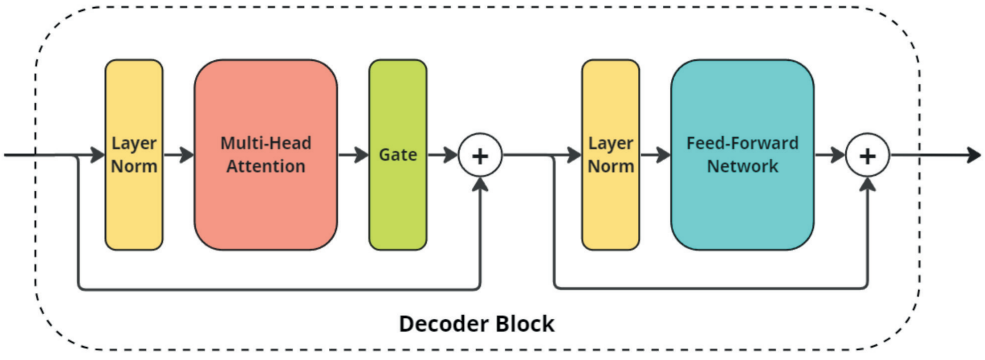

Рис. 5. Расстановка гейта после блока Multi-Head Attention
Fig. 5. Placing the gate after Multi-Head Attention block

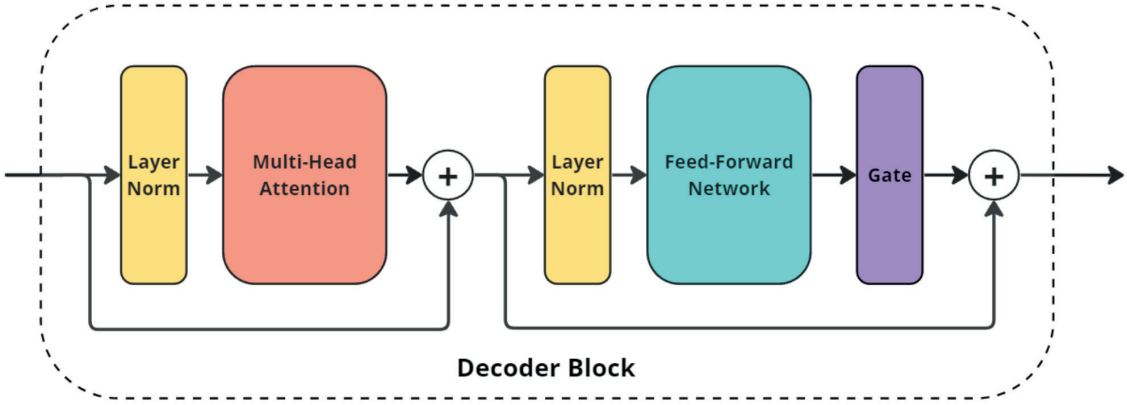

Рис. 6. Расстановка гейта после блока Feed-Forward Network
Fig. 6. Placing the gate after Feed-Forward Network block

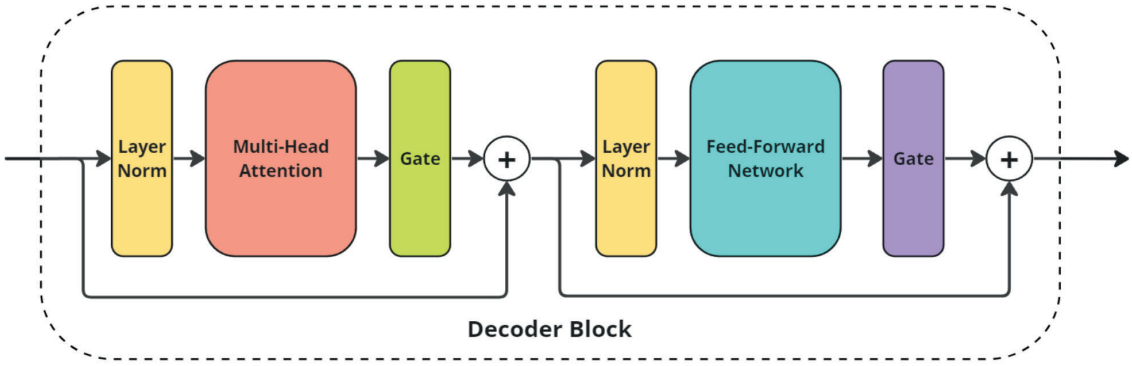

Рис. 7. Расстановка двух разных гейтов после блоков Multi-Head Attention и Feed-Forward Network
Fig. 7. Placing two different gates after Multi-Head Attention and Feed-Forward Network blocks

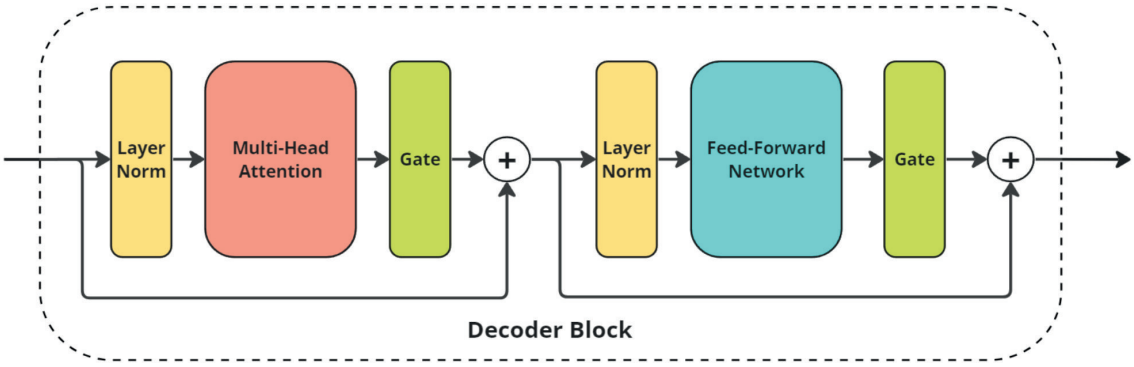

Рис. 8. Расстановка одного и того же гейта после блоков Multi-Head Attention и Feed-Forward Network
Fig. 8. Placing the same gate after Multi-Head Attention and Feed-Forward Network blocks

Важность слоя можно определить двумя разными способами: как модуль суммы градиентов на гейте и как L2-норму вектора градиентов. В первом случае важность слоя будет меньше зависеть от важности каждого канала по отдельности, во втором случае – наоборот, больше учитывается наибольший вклад каждого отдельного канала.

Калибровка модели производится следующим образом: на тренировочном датасете прогоняется каждый пример через модель, вычисляется ошибка с помощью функции потерь, применяется обратное распространение ошибки, и сохраняются градиенты на каждом гейте. В самом конце, когда все градиенты посчитаны, определяется важность каждого слоя.

Процедура прунинга и дообучения повторяет процедуру прунинга методом ShortGPT, однако использует статистику важности, накопленную в процессе калибровки критерием Тейлора. Данный метод представляет собой способ оптимизации больших языковых моделей, учитывающий информацию о градиенте.

2.8. Оптимизация методом LLM-Pruner

Помимо послойного прунинга, также был опробован метод LLM-Pruner, который отбрасывает параметры модели поканально, оставляя при этом количество слоев модели неизменным. Метод был выбран для эксперимента, чтобы узнать, будет ли он, имея меньшую эффективность в плане ускорения модели, обладать большей точностью на задаче function calling благодаря сохранению глубины оригинальной модели.

Первым шагом был выбор критерия прунинга. Было решено использовать критерий важности на основе L2-нормы весов, состоящий в том, что чем меньше значение веса, тем меньше его важность. Важно отметить, что существует и альтернативный критерий Тейлора, который оценивает важность на основе градиента веса. Однако было решено не использовать этот критерий из-за его повышенных требований к памяти видеокарты, на которой запускается и оптимизируется модель.

Следующим шагом было выявление и удаление наименее важных каналов с использованием выбранного критерия прунинга. Количество запруненных каналов было рассчитано таким образом, чтобы общее количество параметров примерно совпадало с количеством параметров моделей, оптимизированных послойными методами оптимизации.

После прунинга модели была применена стандартная процедура дообучения, которая применялась ранее для других методов оптимизации.

2.9. Оптимизация методом PowerInfer

Помимо привычных методов структурированного прунинга, когда из модели удаляются группы параметров, был опробован метод PowerInfer, использующий свойство контекстуальной разреженности в больших языковых моделях.

Прежде всего было решено использовать предварительно обученные веса модели, предоставленные авторами метода PowerInfer [9]. Это решение позволило сократить время и ресурсы, которые бы потребовались на самостоятельное дообучение модели после конвертации в совместимый с методом формат. Далее, уже на основе этих весов, модель дообучалась под искомую задачу. Процедура дообучения повторяла процедуру дообучения базовой модели для остальных методов оптимизации. Точность полученной модели оказалась ниже, чем исходной базовой модели. Это объясняется тем, что модель для PowerInfer имеет гораздо большую разреженность активаций, из-за чего обучение такой модели представляет собой более сложную задачу.

Далее необходимо было обучить онлайн-предикторы, которые играют ключевую роль в методе PowerInfer. В ходе экспериментов были подобраны процедуры обучения и параметры предикторов, которые обеспечили допустимое качество работы модели с онлайн-предикторами. В качестве гиперпараметров предикторов были выбраны следующие значения:

– epochs: 10
– learning rate: 0.02

– weight decay: 0.0001
– optimizer: RAdam
– predictor hidden size: 2048
– criterion: weighted binary cross entropy

После обучения модели и онлайн-предикторов все необходимые компоненты метода были готовы к запуску на представленном авторами метода ПО для инференса.

2.10. Измерение скорости инференса

Для измерения скорости моделей использовался фреймворк для инференса больших языковых моделей – llama.cpp [12]. Этот инструмент предоставляет удобный и понятный интерфейс для запуска больших языковых моделей, а также вывод основных характеристик работы модели, в том числе скорости работы.

Для измерения скорости моделей был использован пример из валидационной выборки длиной 322 входных токена и 27 токенов на генерацию. Все вычисления производились на CPU в однопоточном режиме и на GPU.

Методология измерения времени генерации моделей представляет собой использование двух прогонов модели для «прогрева» и пять прогонов для тестирования. В конце результаты усредняются для получения средних показателей скорости работы моделей.

Инференс моделей происходил с весами в формате fp16 с использованием метода декодирования greedy search.

3. Результаты

Тестирование точности exact match моделей проводилось на валидационной выборке датасета. В качестве целевого устройства для замеров скорости работы моделей был выбран персональный компьютер со следующими характеристиками:

– процессор: Intel Core i7-11700K 3.60 GHz
– видеокарта: Nvidia RTX 4090 24GB
– оперативная память: 60 GB
– операционная система: Linux

Для метода оптимизации с помощью критерия Тейлора был проведен ряд экспериментов, где рассматривалась разная расстановка гейтов внутри слоя декодера, а также различные способы агрегирования важности слоя на гейтах. Результаты приведены в табл. 1.

Таблица 1

Результаты экспериментов с критерием Тейлора

Table 1

Results of experiments with Taylor criterion

Модель	**Параметры**	**Способ агрегирования критерия**	**Расстановка гейтов**	**Точность, %**
Базовая модель	7241732096	–	–	81,83
L2 + FFN	5060612096	L2 norm	после FFN	76,83
L2 + ATTN	5060612096	L2 norm	после Attention	80,34
sum + FFN + ATTN	5060612096	sum	после FFN и Attention	78,30
L2 + FFN + ATTN	5060612096	L2 norm	после FFN и Attention	77,34

По итогам экспериментов (см. табл. 1) было выявлено, что расстановка гейтов после блоков Multi-Head Attention и использование агрегирования важности с помощью L2-нормы вектора градиентов дают наибольшую точность среди остальных опробованных вариантов.

Также было проведено сравнение опробованных методов оптимизации. В табл. 2 представлены результаты тестирования базовой и оптимизированных моделей по разным метрикам качества.

Таблица 2

Результаты тестирования моделей

Table 2

Results of model testing

Метод оптимизации	**Кол-во параметров**	**Точность, exact match, %**	**Скорость генерации токенов на CPU, ms**	**Скорость генерации токенов на GPU, ms**
Базовая модель	7241732096	81,83	491,73 (+–32,75)	21,334 (+–0,14)
ShortGPT	5060612096	79,21	326,15 (+–11,36)	12,06 (+–0,03)
Критерий Тейлора (L2 + ATTN)	5060612096	80,34	326,15 (+–11,36)	12,06 (+–0,03)
LLM-Pruner L2	5035266848	79,13	306,87 (+–4,76)	16,12 (+–0,12)
PowerInfer	8449691648	67,69	342,81 (+–11,12)	69,51 (+–3,12)

Из результатов экспериментов можно сделать следующие выводы.

1. Наибольшая точность для оптимизированной модели была достигнута с помощью метода послойного прунинга по критерию Тейлора важности слоя. При этом потери составляли всего 1,49 % точности. Этому же методу, вместе с методом ShortGPT, удалось достичь наилучшей скорости на GPU – ускорение модели составило 43,47 %. Таким образом, можно сказать, что на ускорение языковых моделей на GPU больше влияет их глубина, нежели количество каналов.

2. При работе на CPU глубина модели влияет на ее ускорение не так сильно, как общее количество параметров модели. Это видно по ускорению моделей методами LLM-Pruner и ShortGPT. При схожем количестве параметров модели имеют сопоставимые значения ускорения.

3. Метод PowerInfer показал себя худшим образом из всех использованных методов оптимизации, уступая остальным методам из табл. 2 как в скорости, так и в точности.

Таким образом, можно сделать вывод, что методы послойного прунинга оказались лучшими по соотношению точность/скорость генерации среди всех опробованных методов.

Заключение

В результате исследования были изучены и применены на практике различные методы оптимизации больших языковых моделей для задачи function calling. Был разработан программный код, который позволяет воспользоваться представленными в работе методами оптимизации. Код написан на языке Python с использованием современных инструментов для создания, обучения и инференса больших языковых моделей.

В качестве результатов применения методов оптимизации были получены оптимизированные модели и составлена таблица сравнения точности и скорости работы полученных моделей. Сделаны выводы о применимости использованных методов оптимизации для каждого конкретного случая.

Список литературы / References

1. **Radford A. et al.** Improving language understanding by generative pre-training. 2018.
2. **Devlin J. et al.** Bert: Pre-training of deep bidirectional transformers for language understanding // arXiv preprint arXiv: 1810.04805. 2018. DOI: 10.18653/V1/N19-1423
3. **Mikolov T. et al.** Efficient estimation of word representations in vector space // arXiv preprint arXiv: 1301.3781. 2013. https://doi.org/10.48550/arXiv.1301.3781
4. **Vaswani A. et al.** Attention is all you need // Advances in neural information processing systems. 2017. T. 30. DOI/10.5555/3295222.3295349
5. **Ma X., Fang G., Wang X.** Llm-pruner: On the structural pruning of large language models // Advances in neural information processing systems. 2023, vol. 36, pp. 21702–21720. https://doi.org/10.48550/arXiv.2305.11627
6. **Men X. et al.** Shortgpt: Layers in large language models are more redundant than you expect // arXiv preprint arXiv: 2403.03853. 2024. https://doi.org/10.48550/arXiv.2403.03853
7. **Frantar E., Alistarh D.** Sparsegpt: Massive language models can be accurately pruned in one-shot // International Conference on Machine Learning. PMLR, 2023. P. 10323–10337. https://doi.org/10.48550/arXiv.2301.00774
8. **Liu Z. et al.** Deja vu: Contextual sparsity for efficient llms at inference time // International Conference on Machine Learning. PMLR, 2023. P. 22137–22176. DOI/10.5555/3618408.3619327
9. **Song Y. et al.** Powerinfer: Fast large language model serving with a consumer-grade gpu // arXiv preprint arXiv: 2312.12456. 2023. https://doi.org/10.1145/3694715.3695964
10. **Jiang A. Q. et al.** Mistral 7B // arXiv preprint arXiv: 2310.06825. 2023. https://doi.org/10.48550/arXiv.2310.06825
11. **Molchanov P. et al.** Importance estimation for neural network pruning // Proceedings of the IEEE / CVF conference on computer vision and pattern recognition. 2019. P. 11264-11272. DOI: 10.1109/CVPR.2019.01152
12. **Gerganov G.** GitHub – ggerganov/llama.cpp: Port of Facebook's LLaMA model in C/C++ – github.com. 2023.

Сведения об авторах

Гончаренко Александр Игоревич, старший преподаватель Института интеллектуальной робототехники НГУ

Чупров Максим Иванович, разработчик-исследователь систем искусственного интеллекта компании ООО «Экспасофт»

Нежевенко Евгений Семенович, доктор технических наук, ведущий научный сотрудник тематической группы оптико-электронных специализированных процессоров Института автоматики и электрометрии Сибирского отделения Российской академии наук

Information about the Authors

Alexander I. Goncharenko, Senior lecturer, Institute of Intelligent Robotics of Novosibirsk State University

Maxim I. Chuprov, Artificial intelligence systems developer/researcher, Expasoft LLC

Evgeniy S. Nezhevenko, PhD., Leading researcher of the subject group of optical-electronic specialized processors, Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences

Статья поступила в редакцию 09.06.2025;
одобрена после рецензирования 23.08.2025; принята к публикации 23.08.2025

The article was submitted 09.06.2025;
approved after reviewing 23.08.2025; accepted for publication 23.08.2025