Разработка вычислительного бэкенда для Llama.cpp на базе Rockchip NPU с поддержкой низкобитных вычислений

Е.Н. Антонянц, И.А. Каштанов, Р.Н. Волошин

***Аннотация* — Статья посвящена решению задачи энергоэффективного инференса больших языковых моделей (LLM) на маломощных устройствах. В работе рассматривается проблема высоких вычислительных затрат при генерации текста и представлен обзор существующих методов аппаратного ускорения. Особое внимание уделено использованию нейронного процессора (NPU) платформы Rockchip для оптимизации ресурсоёмких тензорных операций без существенной потери качества.**

В рамках исследования разработан специализированный вычислительный бэкенд для фреймворка Llama.cpp, интегрируемый в модульную архитектуру GGML. Предложенное программное обеспечение делегирует операции умножения матриц на NPU с поддержкой вычислений в форматах FP16, INT8 и INT4. Реализованы механизмы предварительной обработки весов с адаптацией под аппаратный формат, алгоритмы сглаживания выбросов в активациях с применением ортогональных преобразований Адамара, а также конвейер выполнения с параллелизацией вычислений. Проведено экспериментальное тестирование и сравнительный анализ разработанного решения с традиционным выполнением на центральном процессоре (CPU).

Результаты экспериментов на моделях масштаба от 350 млн до 8 млрд параметров подтверждают превосходство предложенного подхода: зафиксировано ускорение обработки входного контекста более чем в 3 раза по сравнению с CPU-бэкендом. Перенос вычислений на специализированный ускоритель позволил снизить энергопотребление системы более чем в 2 раза при минимальной деградации точности для моделей свыше 1 млрд параметров. Разработанный бэкенд демонстрирует высокую эффективность и делает реалистичным развёртывание современных LLM на устройствах с пассивным охлаждением и ограниченным запасом питания.

***Ключевые слова* — большие языковые модели, инференс, энергоэффективность, нейронный процессор, квантизация, маломощные устройства.**

I. Введение

В последние годы большие языковые модели (LLM), основанные на архитектуре Transformer, продемонстрировали прорывную производительность в области обработки естественного языка [1]. Модели, такие как LLaMA, Mistral и DeepSeek, находят многочисленные применения в сценариях принятия решений и генерации контента. Они способны выполнять широкий спектр задач, включая перевод, обобщение текстов, ответы на вопросы и создание программного кода. Однако использование этих моделей на практике, известное как «инференс», требует значительных вычислительных ресурсов, памяти и энергии. Совокупные затраты на инференс в некоторых случаях могут даже превышать затраты на первоначальное обучение моделей [2].

Широкое внедрение LLM ограничено их высокими вычислительными требованиями и объемом памяти. Эта проблема особенно остра для маломощных устройств, таких как смартфоны и одноплатные компьютеры, где ресурсы строго ограничены. Например, модель с 7 миллиардами параметров требует около 28 ГБ памяти для работы с полной точностью, что выходит далеко за рамки возможностей обычного потребительского оборудования. Одним из ключевых методов решения этой проблемы является квантизация – процесс снижения числовой точности параметров модели, что позволяет значительно сократить как вычислительную сложность, так и требования к памяти. Однако существуют и недостатки – в процессе квантизации часть информации безвозвратно теряется [3].

Главным препятствием для низкобитной квантизации является сохранение высокой точности модели. Одной из причин этого является наличие выбросов – экстремальных значений – в весах и активациях нейронной сети. Активации часто более чувствительны, так как имеют большой динамический диапазон, что делает их квантизацию значительно сложнее, чем квантизацию весов модели. Эти выбросы могут искажать диапазон квантизации, что приводит к значительной потере точности для большинства остальных значений и, как следствие, к снижению общей производительности модели [3]. Таким образом, разработка методов, позволяющих эффективно запускать LLM на устройствах с ограниченными ресурсами при сохранении их точности, является одной из наиболее актуальных задач в области прикладного искусственного интеллекта.

II. Существующие решения

Ускорение больших языковых моделей (LLM) с

Статья получена 6 марта 2026 г.

Антонянц Егор Николаевич, Новосибирский государственный технический университет, Россия (e-mail: bax201438@gmail.com).

Каштанов Иван Александрович, Новосибирский государственный технический университет, Россия (e-mail: kashtanoff254@gmail.com).

Волошин Ростислав Николаевич, Новосибирский государственный технический университет, Россия (e-mail: rostislav.voloshin7@gmail.com).

использованием нейронных процессоров (NPU) является одним из самых быстро развивающихся направлений в индустрии. NPU как специализированные ускорители обеспечивают гораздо большую энергоэффективность по сравнению как с традиционными процессорами (CPU), так и даже некоторыми графическими процессорами (GPU), что очень важно для устройств с ограниченными вычислительными ресурсами [4].

A. Поддержка сторонних NPU

Для своих процессоров Core Ultra, оснащенных встроенным NPU, компания Intel развивает инструментарий OpenVINO [5]. Этот фреймворк предоставляет разработчикам инструменты для оптимизации, квантизации и развертывания моделей, абстрагируя сложность прямого взаимодействия с аппаратным обеспечением. Однако в стандартной форме OpenVINO ориентирован на модели со статической структурой графа вычислений, что не совсем подходит для динамических LLM. Для решения этой проблемы был представлен фреймворк NITRO, который разбирает LLM на более мелкие статичные куски, чтобы заставить его работать с NPU от Intel.

Компания AMD интегрировала NPU на базе архитектуры XDNA в свои мобильные процессоры Ryzen. Для их поддержки была представлена платформа Ryzen AI Software [6]. Она позволяет задействовать аппаратное ускорение для задач искусственного интеллекта, интегрируясь со стандартными фреймворками, такими как PyTorch и TensorFlow, через среду исполнения ONNX Runtime.

В мобильных процессорах Snapdragon компания Qualcomm использует Hexagon NPU. Для работы с ним предназначен Qualcomm AI Engine Direct SDK [7]. Этот набор инструментов позволяет разработчикам получать низкоуровневый доступ к аппаратным ускорителям, оптимизируя выполнение LLM и других моделей непосредственно на устройстве. Популярный фреймворк Llama.cpp получил экспериментальную поддержку NPU Qualcomm Hexagon [8]. Это позволяет использовать аппаратное ускорение для запуска GGUF-моделей на большом количестве устройств с процессорами Snapdragon.

B. Поддержка Rockchip NPU

Платформа Rockchip, в частности популярные системы на кристалле RK3588/RK3576/RK3562, оснащена мощным NPU с высокой производительностью и поддержкой INT4/INT8/FP16 вычислений, что делает ее привлекательной для запуска LLM на одноплатных компьютерах. Однако программная поддержка для этой цели до недавнего времени была ограничена.

Основным решением, предложенным производителем, является библиотека rknn – Rockchip Neural Network. Для работы с большими языковыми моделями был выпущен фреймворк rknn-llm [9].

Для развертывания модели на устройстве с чипом Rockchip необходимо выполнить следующие шаги:

1) С помощью инструмента RKLLM Toolkit на ПК-хосте модель из формата Hugging Face конвертируется в собственный формат .rkllm.

2) В процессе конвертации может применяться квантизация до поддерживаемых для выполнения типов.

3) Полученный .rkllm файл загружается на устройство, где с помощью RKLLM Runtime и драйвера RKNPU выполняется на нейропроцессоре.

Несмотря на функциональность, rknn-llm обладает рядом фундаментальных ограничений, которые затрудняют ее использование в исследовательских и кастомных проектах:

1) Фреймворк поставляется в виде предварительно скомпилированных бинарных файлов. Это не позволяет сообществу вносить изменения, проводить глубокую оптимизацию или адаптировать библиотеку под новые, не поддерживаемые официально архитектуры моделей.

2) Пользователь ограничен теми моделями и форматами квантизации, которые были заложены разработчиками. Отсутствует возможность для экспериментов с новыми методами квантизации или тонкой настройки процесса под конкретную задачу. Библиотека работает по принципу «черного ящика».

3) Фреймворк поддерживает лишь небольшой и строго определенный перечень моделей, таких как Llama, Qwen и Gemma. Чтобы добавить поддержку новых архитектур, требуется обновление самого инструментария от Rockchip. Это лишает разработчиков необходимой гибкости.

Анализ существующих решений показывает четкий тренд: для эффективного использования NPU необходим специализированный программный стек. Однако для распространенной и доступной платформы Rockchip существующее решение (rknn-llm) представляется полностью закрытым как по архитектуре, так и по методам оптимизации и квантизации: библиотека работает по принципу «черного ящика», не давая исследователям никакого контроля над этими процессами. Взаимодействие же с аппаратным обеспечением возможно лишь через проприетарные библиотеки, что связано с отсутствием открытого кода и документации со стороны производителя.

При этом экосистема Llama.cpp предоставляет идеальную основу для интеграции аппаратного ускорения благодаря своей модульной архитектуре вычислительных бэкендов. На данный момент в Llama.cpp существуют такие бэкенды как CUDA, Metal, OpenCL, Vulkan, BLAS [8]. При этом нативная поддержка Rockchip NPU отсутствует.

Таким образом, возникает очевидная потребность в открытом, гибком и высокопроизводительном решении, которое бы объединило универсальность Llama.cpp с вычислительной мощью и энергоэффективностью NPU Rockchip. [10, 11].

III. Теоретические основы

A. Архитектура Transformer и вычислительная сложность LLM

Современные большие языковые модели практически полностью основаны на архитектуре Transformer [10]. Ее ключевыми строительными блоками являются механизм внимания (attention) и полносвязные нейронные сети (Feed-Forward Networks, FFN). С вычислительной точки

зрения, подавляющая часть операций в этих блоках сводится к одной фундаментальной операции – умножению матриц.

В механизме внимания входная матрица активаций умножается на матрицы весов для получения запросов (Q), ключей (K) и значений (V). Затем вычисляется матрица весов внимания путем умножения Q на транспонированную K^T. Наконец, полученные веса умножаются на матрицу V.

Каждый блок полносвязных сетей состоит из двух линейных слоев с функцией активации между ними. Каждый линейный слой представляет собой умножение матрицы входных активаций на матрицу весов слоя.

В процессе генерации ответа модель последовательно обрабатывает входные данные, и на каждом шаге выполняются десятки таких матричных умножений. Размеры этих матриц напрямую зависят от гиперпараметров модели и длины контекста.

Именно операция умножения матриц является основным потребителем вычислительных ресурсов и значительно ограничивает производительность. Следовательно, ускорение именно этой операции является наиболее эффективным способом оптимизации инференса LLM [10].

B. Фреймворк Llama.cpp и архитектура бэкендов GGML

Llama.cpp – это библиотека, написанная на языке C++, предназначенная для эффективного инференса LLM на потребительском оборудовании. Ее высокая производительность и кроссплатформенность обеспечиваются лежащей в ее основе тензорной библиотекой GGML (см. рисунок 1).

Ключевой особенностью GGML является представление всех вычислений в виде графа вычислений [11]. В этом графе узлами являются операции (например, умножение, сложение, активация), а ребрами – тензоры (многомерные массивы данных), которые передаются между операциями.

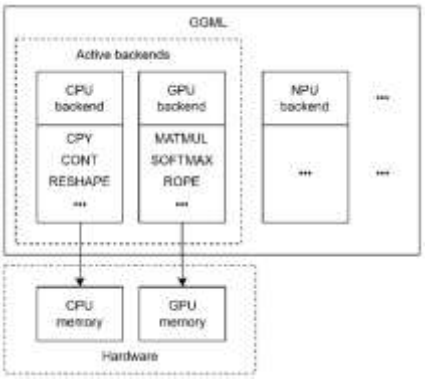

Рисунок 1 – Архитектура библиотеки GGML

Для взаимодействия с различным аппаратным обеспечением GGML использует модульную архитектуру вычислительных бэкендов. Эта архитектура позволяет делегировать выполнение определенных операций из графа вычислений специализированным аппаратным ускорителям. Стандартный бэкенд выполняет все операции на центральном процессоре (CPU). Однако, если в системе присутствует, например, графический процессор NVIDIA, можно активировать бэкенд CUDA. В этом случае GGML будет автоматически отправлять операции, которые могут быть ускорены на GPU, на исполнение этому бэкенду, в то время как остальные операции продолжат выполняться на CPU.

Такой подход обеспечивает гибкость и расширяемость. Он позволяет добавлять поддержку нового оборудования путем реализации нового бэкенда, не изменяя основную логику работы библиотеки. Именно эта архитектурная особенность делает Llama.cpp идеальной платформой для интеграции поддержки различных ускорителей.

C. Аппаратная платформа Rockchip NPU

Нейронный процессор (NPU) от Rockchip представляет собой специализированный сопроцессор, спроектированный для эффективного выполнения операций, характерных для нейронных сетей [12]. Его ключевые особенности:

1) Архитектура NPU содержит большое количество вычислительных блоков, способных параллельно выполнять тысячи простых математических операций (умножение-сложение), что идеально подходит для матричных вычислений.

2) NPU аппаратно оптимизирован для работы с низкобитными целочисленными форматами, такими как INT8 и INT4, а также с 16-битным форматом с плавающей точкой FP16. Выполнение вычислений в этих форматах значительно быстрее и энергоэффективнее по сравнению с 32-битными вычислениями на CPU.

3) Для эффективной работы NPU требует, чтобы данные располагались в физически непрерывных областях памяти. Для управления такой памятью используется механизм DMA (Direct Memory Access), который позволяет передавать большие объемы данных между основной оперативной памятью и NPU без участия центрального процессора, минимизируя задержки.

Сочетание этих факторов делает Rockchip NPU мощным инструментом для ускорения LLM, при условии наличия программного обеспечения, способного правильно использовать его возможности.

D. Квантизация и методы ее применения

Квантизация – это процесс преобразования тензоров из формата с высокой точностью в формат с более низкой точность [13]. Основная цель этого процесса – уменьшить объем занимаемой моделью памяти и ускорить вычисления за счет использования специализированных целочисленных инструкций процессора.

В контексте LLM применяется аффинное преобразование, где каждому значению с плавающей точкой r ставится в соответствие целочисленное значение q. Наиболее распространенной является симметричная квантизация, которая описывается формулой (1).

$$g = round\left(\frac{r}{s}\right), \qquad (1)$$

где s – масштабный коэффициент, определяющий, как диапазон вещественных чисел отображается на диапазон целочисленных.

Масштаб вычисляется на основе абсолютного максимального значения в исходном тензоре по формуле (2).

$$S = \frac{a\max}{Q\_\max}, \quad (2)$$

где $Q\_\max$ – максимальное представимое значение в целевом целочисленном типе.

Обратное преобразование, называемое деквантизацией, позволяет восстановить приблизительное значение с плавающей точкой. Вычисление производится по формуле (3).

$$r' \approx q * S, \quad (3)$$

Ключевым аспектом, определяющим точность и производительность квантизации, является гранулярность – то, для какого объема данных вычисляется один и тот же масштаб *S*. Существует несколько подходов (см. рисунок 2):

1) Квантизация на уровне тензора. Это самый простой подход, при котором для всего тензора (для всей матрицы) вычисляется единственный масштаб S. Этот метод требует минимального объема дополнительной памяти для хранения метаданных (всего одно число S на матрицу), но он очень чувствителен к выбросам. Если в матрице присутствует хотя бы одно значение с большой амплитудой, максимум будет большим, что приведет к большому значению S. В результате большинство «нормальных» значений после деления на S окажутся очень близкими к нулю, и после округления их точность будет почти полностью утеряна.

2) Квантизация на уровне канала. Чтобы решить проблему выбросов, применяется более гранулярный подход. Матрица рассматривается как набор векторов (строк или столбцов), и для каждого такого вектора вычисляется свой собственный, независимый масштаб S. Этот метод значительно повышает точность, так как выброс в одном канале не влияет на диапазоны квантизации в других. Однако он требует больше дополнительной памяти для хранения массива масштабов.

3) Квантизация на уровне групп. Этот метод еще лучше изолирует выбросы. Векторы внутри тензора дополнительно разбиваются на небольшие группы, и для каждой такой группы вычисляется свой масштаб. Это позволяет еще точнее локализовать диапазоны значений, обеспечивая высокую точность при еще больших накладных расходах на хранение метаданных (матрица масштабов). Важным аспектом этого метода является то, что математически невозможно восстановить результирующую матрицу в том случае, если отсутствует аппаратная поддержка данного типа квантизации.

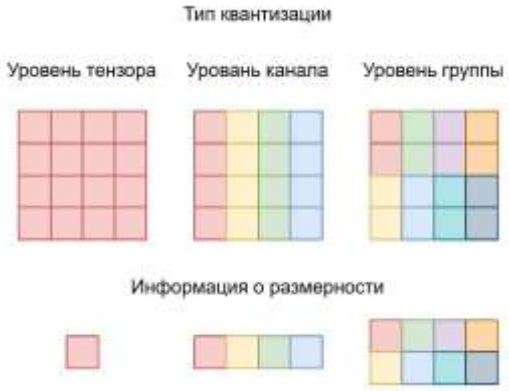

Рисунок 2 – Представление различных видов квантизации с дополнительной информацией для хранения

Выбор гранулярности является фундаментальным компромиссом между точностью модели, объемом памяти для хранения квантованных весов и масштабов, а также вычислительной сложностью, так как более гранулярные методы требуют более сложных операций при деквантизации [13].

F. Калибровка и математические преобразования

Точность симметричной квантизации напрямую зависит от выбора масштабного коэффициента. Наивный подход, использующий истинное максимальное значение в тензоре, часто приводит к неоптимальным результатам из-за наличия выбросов.

Калибровка – это процесс поиска оптимального абсолютного максимального значения, которое минимизирует общую ошибку квантизации. Вместо того чтобы охватывать 100% диапазона значений, калибровка позволяет пожертвовать точностью представления очень редких, но экстремальных значений, чтобы значительно повысить точность для основной массы данных. Существует несколько подходов:

1) Процентиль. Простой и эвристический метод, который заключается в выборе в качестве значения не абсолютного максимума, а значения, соответствующего высокому процентилю распределения абсолютных величин (например, 99.9%). Это позволяет эффективно игнорировать самые экстремальные выбросы, которые могут исказить масштаб.

2) Минимизация средней квадратичной ошибки. Более принципиальный подход, целью которого является поиск такого значения, который минимизирует среднюю квадратичную ошибку (MSE) между исходным и деквантизированным тензором. Алгоритм итеративно проверяет множество кандидатов в определенном диапазоне, для каждого из них выполняет симуляцию квантизации-деквантизации и вычисляет итоговую ошибку [3]. Выбирается то значения, которое дает наименьшую ошибку. Этот метод более вычислительно затратен, но часто дает лучшие результаты по точности.

3) Минимизация KL-дивергенции. Наиболее продвинутый метод, который рассматривает задачу с точки зрения теории информации. Он трактует гистограммы исходных и квантованных значений как два вероятностных распределения. Цель – найти такое

значение, при котором дивергенция Кульбака-Лейблера будет минимальной [3]. Этот метод стремится сохранить не столько числовую точность каждого значения, сколько общую информационную энтропию исходного распределения данных.

Однако многочисленные исследования больших языковых моделей выявили, что выбросы в матрицах весов и активаций несут в себе важнейшую часть информации и напрямую влияют на качество сгенерированных ответов. Простое отсечение выбросов зачастую несет за собой деградацию модели и полную потерю качества.

Более правильным подходом для устранения выбросов является не их отсечение, а сглаживание с помощью математических преобразований. Идея состоит в том, чтобы изменить само распределение данных, сделав его более «удобным» для квантизации. Одно исследование выявило, что для этого могут использоваться ортогональные преобразования [14].

Ортогональная матрица – это матрица, при умножении которой на транспонированную версию себя получается единичная матрица, т.е. матрица по главной диагонали которой стоят единицы, а все остальные значения – нули.

Такие преобразования представляют собой повороты и отражения в многомерном пространстве и, что критически важно, сохраняют скалярное произведение векторов.

Это свойство лежит в основе вычислительной инвариантности: результат умножения матрицы активаций на матрицу весов не изменится, если мы преобразуем их по формуле (4).

$$X'W' = (XH)(H'W) = X(HH^T)W = XIW = XW, \quad (4)$$

Таким образом, можно применить ортогональное преобразование к активациям, чтобы сгладить выбросы по всем измерениям, сделав распределение более гладким и близким к нормальному. Чтобы сохранить результат вычислений, к матрице весов необходимо применить обратное (транспонированное) преобразование.

Хотя в теории можно использовать любую ортогональную матрицу, на практике выбор ограничен вычислительной сложностью. Умножение на произвольную ортогональную матрицу размером $d \times d$ требует $O(d^2)$ операций, что является слишком затратным для выполнения на лету во время инференса. Идеальным решением этой проблемы является преобразование Адамара [14].

Матрица Адамара – это частный случай ортогональной матрицы, состоящей только из элементов {+1, -1}. Ее уникальное свойство заключается в существовании алгоритма быстрого преобразования Уолша-Адамара, который позволяет выполнить матрично-векторное умножение всего за $O(d \log d)$ операций. Это на порядок эффективнее стандартного матричного умножения и делает применение ортогональных преобразований в реальном времени практически осуществимым.

Преобразование Адамара предлагает мощный инструмент для устранения выбросов в активациях, сочетая теоретическую эффективность ортогональных преобразований для сглаживания выбросов без потери информации с реализуемой вычислительной сложностью для применения в режиме реального времени.

IV. Предлагаемый метод

Предлагаемый метод заключается в разработке и интеграции нового вычислительного бэкенда в Llama.cpp, который делегирует наиболее ресурсоемкую операцию – умножение матриц – на NPU Rockchip. Архитектура разработанного программного обеспечения, представленная на рисунке 3, построена по модульному принципу и включает три основных компонента:

1) Архитектура бэкенда, обеспечивающая его интеграцию в экосистему GGML и управление вычислительными ресурсами NPU.

2) Механизм предварительной обработки весов, который при загрузке модели преобразует их в формат, оптимальный для аппаратного ускорителя.

3) Конвейер выполнения инференса, который в реальном времени подготавливает активации, выполняет вычисления на NPU и обрабатывает результаты.

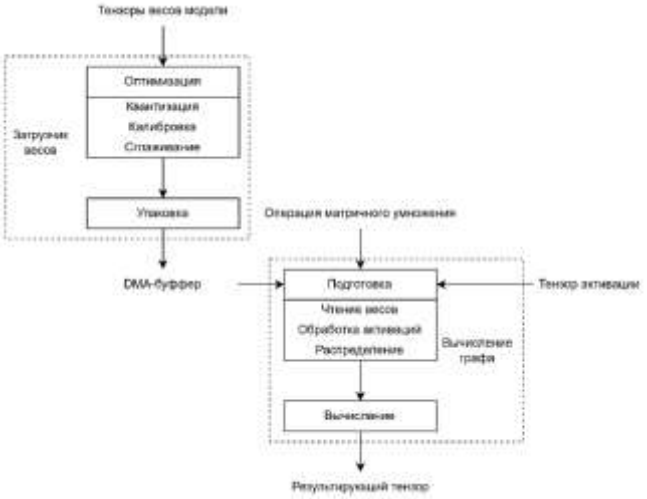

Рисунок 3 – Архитектура разработанного решения

A. Общая архитектура бэкенда

Бэкенд реализован в соответствии с модульной архитектурой GGML. Его основная задача – перехватывать в графе вычислений операцию матричного умножения (GGML_OP_MUL_MAT), которая является основной вычислительной нагрузкой в LLM.

При инициализации бэкенд регистрируется в системе GGML, сообщая о своей способности обрабатывать определенные типы операций. Он анализирует операцию и ее входные тензоры и определяет, может ли операция быть эффективно выполнена на NPU. Критериями являются тип операции (GGML_OP_MUL_MAT), типы данных (FP16, Q8_0, Q4_0) и соответствие размерностей тензоров требованиям NPU к выравниванию.

Для достижения максимальной производительности бэкенд использует собственный тип буфера. Этот буфер выделяет память через механизм DMA в операционной системе Linux. Это позволяет организовать zero-copy

передачу данных для весовых матриц, избегая дорогостоящих операций копирования из основной оперативной памяти в память, доступную NPU.

Во время выполнения инференса планировщик GGML вызывает основную функцию бэкенда – вычисление графа. Эта функция проходит по всем узлам графа вычислений, и если узел является поддерживаемой операцией умножения матриц, она инициирует конвейер ее выполнения на NPU.

B. Предварительная обработка весов

Особенностью разработанного бэкенда является этап предварительной обработки весовых матриц, который выполняется однократно при загрузке модели. Данный этап решает фундаментальную проблему архитектурной несовместимости форматов представления данных. С одной стороны, формат GGUF хранит веса с гранулярной квантизацией, что оптимизировано для гибких вычислений на CPU и графических процессорах, опирающихся на зрелые программные экосистемы (в частности, CUDA и Vulkan). С другой стороны, матричные блоки умножения-сложения в архитектуре Rockchip NPU жестко спроектированы для работы с непрерывными массивами данных и не имеют аппаратной поддержки динамической смены масштабных коэффициентов внутри одного вычисления тензора.

Математически корректное выполнение тензорных операций на данном ускорителе требует приведения данных к единому глобальному масштабу для матрицы весов и построчному масштабу для матрицы активаций. В рамках исследования была предпринята попытка программной эмуляции продвинутой гранулярности GGUF: операция умножения искусственно дробилась на множество независимых малых вычислений, размер которых соответствовал размеру группы квантизации. Однако экспериментальное профилирование показало, что накладные расходы на передачу данных, инициализацию ядер операционной системой и синхронизацию DMA-буферов полностью нивелируют вычислительную мощь NPU, снижая производительность бэкенда до значений, исключающих практическую применимость. Отказ от блочной квантизации в пользу глобального масштабирования является строгим аппаратным ограничением платформы. Процесс адаптации весовых матриц зависит от исходного типа данных.

Для весов в формате FP16, где квантизация отсутствует, основная задача заключается в оптимизации пространственного расположения данных. Выполняется сегментация и переупаковка матрицы в соответствии с требованиями параллельной архитектуры NPU. Это гарантирует, что каждое ядро нейропроцессора получит последовательный фрагмент данных в оптимальном для кэширования порядке.

Для 8-битных целочисленных форматов выполняется процедура переквантизации. Бэкенд деквантизирует блочные веса до формата FP32, вычисляет единый глобальный масштаб, охватывающий весь динамический диапазон тензора, и заново квантизирует данные в нативный формат INT8. Несмотря на сведение к единому масштабу, емкости 8-битного представления оказывается достаточно для сохранения высокого качества модели.

Наиболее остро проблема потери точности при сведении к глобальному масштабу стоит для экспериментального 4-битного конвейера, где экстремальные значения (выбросы) могут привести к существенной деградации ответов у компактных моделей. Чтобы предотвратить катастрофическое падение точности, на этапе переупаковки применяется комплекс математических преобразований. Сначала веса восстанавливаются до FP32, после чего к ним применяется ортогональное преобразование Адамара. Эта операция математически перераспределяет энергию локальных выбросов по всему вектору, сглаживая распределение значений. После этой нормализующей процедуры вычисляется оптимальный масштаб с использованием метода минимизации KL-дивергенции. Наконец, данные квантизируются и упаковываются в нативный для NPU формат INT4.

В результате этого этапа, независимо от исходной сложности и гранулярности данных в файле GGUF, веса преобразуются в унифицированный, аппаратно-оптимизированный формат. Это позволяет обойти архитектурные барьеры контроллера памяти NPU и обеспечить выполнение матричных умножений с максимально возможной пропускной способностью кристалла.

C. Конвейер выполнения инференса

После того как веса предварительно обработаны и находятся в DMA-памяти, основной вычислительный конвейер запускается для каждой операции умножения матриц во время инференса. Этот процесс можно разбить на четыре основных этапа.

1) Параллелизация вычислений. Для максимального задействования всех вычислительных ядер NPU бэкенд динамически разделяет задачу. Матрица весов W (размером $K \times N$) и выходная матрица C (размером $M \times N$) разрезаются на вертикальные сегменты по размерности N. Матрица активаций A (размером $M \times K$) остается целой и передается на каждое ядро. Таким образом, каждое ядро NPU получает задачу вычислить свою часть выходной матрицы: $C_i = A \times W_i$. Это позволяет выполнять матричное умножение параллельно и независимо на всех доступных ядрах.

2) Подготовка активаций. В отличие от весов, активации динамически меняются на каждом шаге генерации токена. Поэтому их подготовка выполняется в реальном времени. Входной тензор активаций, поступающий в формате FP32, квантизируется непосредственно перед отправкой на NPU. Применяется симметричная квантизация на уровне канала, то есть для каждой строки матрицы активаций вычисляется свой собственный масштаб. Это позволяет точно адаптироваться к динамическому диапазону входных данных. В случае использования преобразования Адамара, перед квантизацией каждая строка активаций сначала умножается на тот же самый знаковый вектор s, который использовался для весов, а затем к ней применяется быстрое преобразование Адамара. Подготовленный и квантованный тензор активаций копируется в общий буфер, доступный всем ядрам NPU.

3) Выполнение на нейросетевом процессоре. На

этом этапе бэкенд инициирует вычисления. С помощью OpenMP создается пул потоков, по одному на каждое ядро NPU. Каждый поток получает указатели на общую матрицу активаций и на свой сегмент весов. Так как веса находятся в DMA-памяти в нужном формате, вызов библиотеки позволяет NPU получить к ним доступ без копирования. Затем асинхронно запускается операция матричного умножения на каждом ядре.

4) Получение результата. После завершения параллельных вычислений на всех ядрах NPU, потоки синхронизируются. Результаты из выходных буферов каждого ядра копируются в итоговый тензор-результат, «собирая» его из отдельных сегментов. Если NPU возвращает результат в целочисленном формате, выполняется финальный шаг – деквантизация. Для этого используется комбинация ранее вычисленного масштаба для весов (глобального) и массива масштабов для активаций (построчного).

V. Экспериментальная оценка

Целью экспериментальной части исследования является подтверждение эффективности реализованного бэкенда для Rockchip NPU по трем основным критериям: производительность, энергопотребление и точность.

A. Тестирование разработанного решения

Все метрики были получены из встроенных в фреймворк Llama.cpp утилит. Результаты испытаний для моделей различного масштаба и различных архитектур сведены в таблицу 1.

Таблица 1 – Сравнение точности и производительности бэкендов

Модель	Тип	Бэкенд	Perplexity	PP, ток/с	TG, ток/с	Потребл., Вт
Gemma3 1B	F16	CPU	26.20	68.5	11.1	6.8
		NPU	26.18	249.6	10.8	2.8
	Q8_0	CPU	26.08	73.3	19.5	7.4
		NPU	29.15	378.6	16.5	3.0
Ministral3 3B	Q8_0	CPU	7.67	17.5	6.1	7.2
		NPU	7.84	120.9	6.9	3.2
Qwen3.5 9B	Q8_0	CPU	6.82	7.2	2.5	7.4
		NPU	7.14	44.6	3.2	3.2
	Q4_0	CPU	7.10	6.5	3.6	8.0
		NPU	14.67	7.3	4.4	4.0

На рисунке 4 приведено сравнение основных характеристик в виде столбчатых диаграмм.

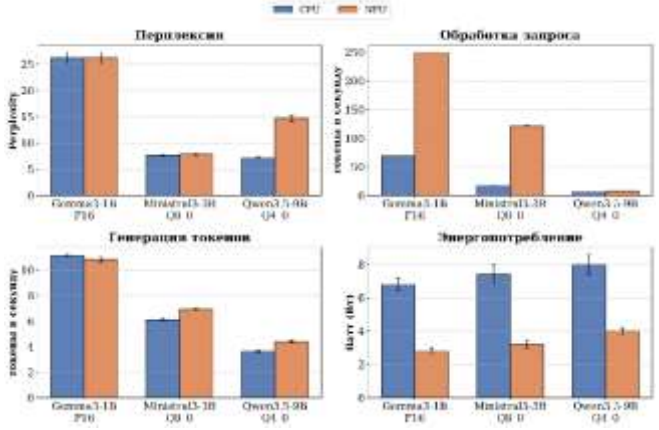

Рисунок 4 – Диаграммы сравнения точности, производительности и энергоэффективности решений

Для оценки эффективности моделей на разных аппаратных бэкендах (CPU и NPU) были выбраны следующие метрики:

1) Perplexity (Перплексия), безразмерная метрика характеризующая качество предсказания модели. Чем ниже значение перплексии, тем правильней модель воссоздает тестовый текст.

2) Prompt Processing (ток/с), скорость обработки входного текста. Высокий показатель особенно важен для длинных запросов.

3) Token Generation (ток/с), скорость генерации новых токенов. Значение того, как быстро нейросеть предсказывает следующий токен.

4) Энергопотребление (Вт), средняя мощность, потребляемая вычислительным блоком при инференсе. Позволяет оценить энергоэффективность.

B. Интерпретация полученных результатов

Анализ метрик показывает, что перенос вычислений на NPU в форматах FP16 и INT8 позволяет сохранить высокое качество генерации: значения перплексии для NPU-бэкенда практически идентичны эталонным результатам на CPU. При этом выполнение квантизации INT8 на нейропроцессоре обеспечивает уровень качества, сопоставимый с индустриальным стандартом Q4_0 для центральных процессоров, что делает этот подход оптимальным для развертывания LLM в реальных условиях без существенной деградации выходных данных. Следует отметить, что реализация низкобитного конвейера INT4xINT4 для формата Q4_0 в рамках данной работы носит экспериментальный характер (Proof-of-concept). Успешный запуск 4-битных вычислений доказывает гибкость предложенной модульной архитектуры и принципиальную возможность применения пользовательских оптимизаций на уровне графа вычислений, что недоступно в стандартных решениях вендора.

Основной прирост производительности зафиксирован на этапе обработки входного контекста (Prompt Processing), где бэкенд демонстрирует многократное превосходство по сравнению с многопоточным выполнением на CPU. В современных парадигмах использования больших языковых моделей, таких как генерация с дополненной выборкой, извлечение семантических эмбеддингов для векторных баз данных и суммаризация объемных документов, именно фаза предварительной обработки является главным вычислительным «бутылочным горлышком». Как показывают недавние исследования крупных индустриальных лабораторий [15], затраты на Prefill-фазу настолько высоки, что оптимизация и аппаратное ускорение именно этого этапа становятся критически важными для повышения общей пропускной способности систем инференса.

Кроме того, разработанное решение ускорило и фазу генерации токенов (Token Generation). Хотя для компактных моделей (Gemma3 1B) скорость Decode-фазы на NPU ограничивается архитектурными пропускными способностями подсистемы памяти и сопоставима с CPU, ситуация кардинально меняется при

масштабировании параметров. На моделях среднего и крупного размера (Ministral3 3B и Qwen3.5 9B) применение специализированного вычислителя обеспечивает прямое ускорение авторегрессионной генерации на 13-28%.

Важнейшим результатом внедрения разработанного решения является радикальное повышение энергоэффективности. Прирост скорости на всех этапах инференса сопровождается снижением энергопотребления более чем в 2 раза (с 7.4 Вт до 3.2 Вт). В контексте системного дизайна это нивелирует тепловые ограничения и открывает возможность интеграции современных LLM в автономные IoT-устройства, робототехнику и встраиваемые системы с пассивным охлаждением и строгими лимитами по питанию.

C. Практические выводы из полученных результатов

Главным практическим выводом из полученных результатов является фундаментальное изменение парадигмы оценки эффективности аппаратных ускорителей. Традиционно производительность LLM оценивается исключительно через призму скорости генерации (токен/с), однако в условиях глобального роста вычислительных масштабов и энергопотребления ИИ-индустрии на первый план выходит метрика удельной энергоэффективности – затраты энергии на генерацию одного токена (Джоуль/токен).

Анализ показывает, что специализированный NPU обладает большой эффективностью в долгосрочной перспективе по сравнению с традиционными дискретными графическими процессорами (GPU). Эмпирическое потребление нейропроцессора Rockchip при полной загрузке составляет около 3.0-3.2 Вт. Для генерации одного токена на моделях класса 2-3 млрд параметров NPU затрачивает в среднем 0.3 Дж. Для сравнения, традиционный десктопный или серверный GPU среднего сегмента, потребляя от 150 до 300 Вт, обеспечивает скорость около 60 ток/с, что эквивалентно затратам в 2.5-5.0 Дж на токен.

Чтобы сравняться с NPU по энергоэффективности, условный графический ускоритель с потреблением 150 Вт должен был бы генерировать не менее 500 токенов в секунду на единичном запросе, что практически недостижимо даже для флагманских решений из-за ограничений пропускной способности памяти. Кроме того, оценка потребления GPU часто игнорирует сопутствующие затраты базовой станции, которые в режиме простоя потребляют в десятки раз больше, чем вся SoC Rockchip при максимальной нагрузке.

Из этого следует, что NPU не призван конкурировать с серверными кластерами в задачах потоковой обработки тысяч одновременных запросов с минимальной задержкой. Его ниша – интеллектуальные системы непрерывного действия. Крайне низкое энергопотребление позволяет выполнять длительные фоновые задачи практически «бесплатно» с точки зрения затрат на электроэнергию и без влияния на отзывчивость основной операционной системы.

Высвобождение ресурсов центрального процессора открывает широкие возможности для асинхронных ИИ-агентов. К таким сценариям относятся:

1) Разметка больших синтетических наборов данных в фоне.
2) Извлечение эмбеддингов и семантический поиск.
3) Постоянная индексация локальных векторных баз данных.
4) Фоновая обработка естественного языка.
5) Вызов внешних инструментов, извлечение ключевых слов и анализ документов.

Современные компактные модели уже обладают достаточной когнитивной емкостью для решения подобных нишевых задач на уровне больших моделей предыдущих поколений.

Интеграция NPU-бэкенда в стек Llama.cpp переводит использование LLM из категории «ресурсоемких сеансов» в категорию «фоновых сервисов» операционной системы, многократно снижая совокупную стоимость владения интеллектуальным оборудованием.

VI. Заключение

В рамках проведенного исследования решена задача создания бэкенда для библиотеки Llama.cpp, обеспечивающего возможность эффективного использования больших языковых моделях на Rockchip NPU.

Разработанный программный интерфейс позволяет использовать ресурсы NPU для вычислений в форматах FP16, INT8 и INT4. Предложенный подход устраняет зависимость от закрытых высокоуровневых фреймворков: абстрагируясь от аппаратного драйвера через стандартный API производителя, решение полностью открывает для исследователей логику построения вычислительного графа, предварительной обработки весов и алгоритмов квантизации.

Экспериментально доказано, что использование NPU дает многократное ускорение (от 3 до 7 раз) на этапе предварительной обработки входного контекста (Prompt Processing), что является критически важным достижением для современных задач генерации с дополненной выборкой (RAG) и обработки длинных документов. При генерации токенов (Token Generation) на моделях свыше 2 млрд параметров бэкенд также демонстрирует превосходство над центральным процессором, обеспечивая ускорение вплоть до 28%.

Перенос вычислений на специализированный ускоритель снижает энергопотребление системы более чем в 2 раза. Высокая энергоэффективность решения открывает возможности для запуска как малых, так и крупных моделей различных архитектур на маломощных устройствах.

Показано, что применение операций FP16xFP16 и INT8xINT8 обеспечивает точность, сопоставимую с эталонными реализациями на центральных процессорах и предоставляет возможность развертывания в реальных условиях, а интеграция ортогональных преобразований Адамара минимизирует информационные потери при экспериментальных 4-битных вычислениях.

Библиография

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

[2] Samsi S. et al. From words to watts: Benchmarking the energy costs of large language model inference // 2023 IEEE high performance extreme computing conference (HPEC). – IEEE, 2023. – P. 1-9.
[3] Lang J., Guo Z., Huang S. A comprehensive study on quantization techniques for large language models // 2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC). – IEEE, 2024. – P. 224–231.
[4] Prieto P., Abad P. Edge Deployment of Small Language Models, a comprehensive comparison of CPU, GPU and NPU backends // arXiv preprint arXiv:2511.22334. – 2025.
[5] Intel® Distribution of OpenVINO™ Toolkit [Электронный ресурс] – URL: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (дата обращения: 14.01.2026).
[6] AMD Ryzen™ AI Software [Электронный ресурс] – URL: https://www.amd.com/en/developer/resources/ryzen-ai-software.html (дата обращения: 14.01.2026).
[7] Qualcomm AI Engine Direct SDK [Электронный ресурс] // Qualcomm Developer. – URL: https://www.qualcomm.com/developer/software/qualcomm-ai-engine-direct-sdk (дата обращения: 14.01.2026).
[8] LLM inference in C/C++ [Электронный ресурс] // GitHub. – URL: https://github.com/ggml-org/llama.cpp (дата обращения: 22.01.2026).
[9] RKLLM-Toolkit is a software development kit for users to perform model conversionand quantization on PC [Электронный ресурс] // GitHub. – URL: https://github.com/airockchip/rknn-llm (дата обращения: 22.01.2026).
[10] Vaswani A. et al. Attention is all you need // Advances in neural information processing systems. – 2017. – T. 30.
[11] Tensor library for machine learning [Электронный ресурс] // GitHub. – URL: https://github.com/ggml-org/ggml (дата обращения: 04.02.2026).
[12] RK3588 Brief Datasheet.pdf [Электронный ресурс] – URL:https://www.rock-chips.com/uploads/pdf/2022.8.26/192/RK3588%20Brief%20Datasheet.pdf (дата обращения: 05.02.2026).
[13] Nagel M. et al. A white paper on neural network quantization //arXiv preprint arXiv:2106.08295. – 2021.
[14] Ashkboos S. et al. Quarot: Outlier-free 4-bit inference in rotated llms // Advances in Neural Information Processing Systems. – 2024. – T. 37. – P. 100213–100240.
[15] Qin R. et al. Prefill-as-a-Service: KVCache of Next-Generation Models Could Go Cross-Datacenter //arXiv preprint arXiv:2604.15039. – 2026.

Антонянц Егор Николаевич. Новосибирский государственный технический университет, г. Новосибирск, Россия. Аспирант факультета автоматики и вычислительной техники. Количество печатных работ: 61. Область научных интересов: машинное обучение, системы компьютерного зрения, информационные сети. e-mail: bax201438@gmail.com (ответственный за переписку).

Каштанов Иван Александрович. Новосибирский государственный технический университет, г. Новосибирск, Россия. Студент бакалавриата факультета автоматики и вычислительной техники. Количество печатных работ: 1. Область научных интересов: большие языковые модели, глубокое обучение и системы искусственного интеллекта.

Волошин Ростислав Николаевич. Новосибирский государственный технический университет, г. Новосибирск, Россия. Студент бакалавриата факультета автоматики и вычислительной техники. Количество печатных работ: 1. Область научных интересов: нейронные сети, машинное обучение и информационные сети.

Development of a computing backend for Llama.cpp based on Rockchip NPU with support for low-bit computations

E.N. Antonyants, I.A. Kashtanov, R.N. Voloshin

***Abstract* — The article is devoted to solving the problem of energy-efficient inference of large language models (LLMs) on low-power devices. The paper examines the problem of high computational costs during text generation and presents an overview of existing hardware acceleration methods. Particular attention is paid to using the Neural Processing Unit (NPU) of the Rockchip platform to optimize resource-intensive tensor operations without significant quality loss.**

Within the scope of the research, a specialized computational backend for the Llama.cpp framework was developed, integrated into the modular GGML architecture. The proposed software delegates matrix multiplication operations to the NPU, supporting computations in FP16, INT8, and INT4 formats. Mechanisms for weight preprocessing with adaptation to the hardware format, algorithms for smoothing outliers in activations using orthogonal Hadamard transformations, as well as an execution pipeline with computation parallelization were implemented.

Experimental testing and comparative analysis of the developed solution against traditional execution on the Central Processing Unit (CPU) were performed. Experimental results on models ranging from 350 million to 8 billion parameters confirm the superiority of the proposed approach: a speedup of input context processing by more than 3 times compared to the CPU backend was recorded. Offloading computations to the specialized accelerator enabled reducing system energy consumption by more than 2 times with minimal accuracy degradation for models exceeding 1 billion parameters. The developed backend demonstrates high efficiency and makes the deployment of modern LLMs on devices with passive cooling and limited power capacity more realistic.

***Keywords* — large language models, inference, energy efficiency, neural processor, quantization, low-power devices.**

REFERENCES

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

[2] Samsi S. et al. From words to watts: Benchmarking the energy costs of large language model inference // 2023 IEEE high performance extreme computing conference (HPEC). – IEEE, 2023. – P. 1-9.

[3] Lang J., Guo Z., Huang S. A comprehensive study on quantization techniques for large language models // 2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC). – IEEE, 2024. – P. 224–231.

[4] Prieto P., Abad P. Edge Deployment of Small Language Models, a comprehensive comparison of CPU, GPU and NPU backends // arXiv preprint arXiv:2511.22334. – 2025.

[5] Intel® Distribution of OpenVINO™ Toolkit [Online] – URL: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (accessed: 14.01.2026).

[6] AMD Ryzen™ AI Software [Online] – URL: https://www.amd.com/en/developer/resources/ryzen-ai-software.html (accessed: 14.01.2026).

[7] Qualcomm AI Engine Direct SDK [Online] // Qualcomm Developer. – URL: https://www.qualcomm.com/developer/software/qualcomm-ai-engine-direct-sdk (accessed: 14.01.2026).

[8] LLM inference in C/C++ [Online] // GitHub. – URL: https://github.com/ggml-org/llama.cpp (accessed: 22.01.2026).

[9] RKLLM-Toolkit is a software development kit for users to perform model conversionand quantization on PC [Online] // GitHub. – URL: https://github.com/airockchip/rknn-llm (accessed: 22.01.2026).

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

[11] Tensor library for machine learning [Online] // GitHub. – URL: https://github.com/ggml-org/ggml (accessed: 04.02.2026).

[12] RK3588 Brief Datasheet.pdf [Online] – URL:https://www.rock-chips.com/uploads/pdf/2022.8.26/192/RK3588%20Brief%20Datasheet.pdf (accessed: 05.02.2026).

[13] Nagel M. et al. A white paper on neural network quantization //arXiv preprint arXiv:2106.08295. – 2021.

[14] Ashkboos S. et al. Quarot: Outlier-free 4-bit inference in rotated llms // Advances in Neural Information Processing Systems. – 2024. – T. 37. – P. 100213–100240.

[15] Qin R. et al. Prefill-as-a-Service: KVCache of Next-Generation Models Could Go Cross-Datacenter //arXiv preprint arXiv:2604.15039. – 2026.