Разработка платформы инференса больших языковых моделей с маршрутизацией запросов на потребительском оборудовании

Д.К. Свириденко, Е.В. Боброва, П.В. Анни

Аннотация - **В работе представлены проектирование, разработка и испытания прототипа платформы инференса больших языковых моделей (LLM), построенной по архитектуре «ведущий–ведомый» на потребительском оборудовании под управлением Windows + WSL2 + Docker с полностью открытым программным стеком. Платформа обеспечивает единую точку входа через веб-интерфейс и OpenAI-совместимый прикладной интерфейс, маршрутизацию запросов к вычислительным узлам по имени модели, аутентификацию и ограничение частоты запросов. Новые узлы подключаются без изменения клиентской части. Все узлы размещены на серверах МИФИ, поэтому весть трафик остается внутри периметра организации. На узлах развёрнуты модели разных классов, включая модель с архитектурой mixture-of-experts (MoE), распределённую между двумя видеокартами. Предложена аналитическая модель нагрузочной способности узла на основе бюджета видеопамяти и объёма KV-кэша. Практическая значимость: платформа даёт доступ к LLM на уже имеющемся оборудовании, устраняя зависимость от внешних облачных сервисов и гарантируя, что данные не покидают периметр организации.**

Ключевые слова - **инференс LLM, архитектура «ведущий–ведомый», горизонтальное масштабирование, Ollama, WSL2, Docker, mixture-of-experts, Qwen3, LiteLLM.**

I. ВВЕДЕНИЕ

За последние годы наблюдается экспоненциальный рост числа параметров больших языковых моделей - от полутора миллиардов у GPT-2 до сотен миллиардов у современных открытых моделей. При этом вычислительные ресурсы, доступные рядовому пользователю, не демонстрируют сопоставимой динамики, что порождает существенный разрыв между требованиями моделей и возможностями оборудования. Развёртывание LLM на потребительском оборудовании сопряжено с двумя фундаментальными ограничениями. Первое - объём видеопамяти: хранение параметров модели на 30 млрд параметров в формате FP16 требует порядка 61 ГБ видеопамяти [1].

Второе - скорость генерации: воспроизведение каждого токена предполагает последовательный проход через все слои сети. Таким образом, задача оптимизации инференса выходит за рамки сугубо теоретического интереса и становится необходимым условием практического применения современных языковых моделей в локальных средах, особенно при работе с конфиденциальными данными.

Стремительное развитие этой области отражено в обзорных работах 2024–2026 годов. Обзор [2] сводит все узкие места к трём причинам: большому размеру модели, квадратичному росту вычислений в механизме внимания при росте длины текста и пошаговому (авторегрессивному) порождению ответа, когда каждый следующий токен вычисляется отдельно. Соответственно и оптимизации авторы разносят по трём уровням: на уровне данных - сокращение числа обрабатываемых токенов (сжатие запроса, суммаризация, вынесение знаний во внешнюю базу через RAG); на уровне модели - уменьшение самой нейронной сети (4-битная квантизация весов, разрежённое внимание); на уровне системы - управление кэшем ключей и значений (KV-кэшем), планирование запросов и спекулятивное декодирование.

Для локального сценария особенно важно разобрать, где возникают задержки и что их порождает. Инференс состоит из двух фаз: обработка запроса, где вычислений много и узким местом становится производительность видеокарты, и генерация ответа, где на каждый шаг приходится заново читать все веса модели из видеопамяти, и потому скорость упирается уже не в вычисления, а в пропускную способность памяти [2]. При одиночных запросах генерация почти целиком ограничена памятью. Отсюда практический вывод: квантизация весов до 4 бит даёт выигрыш не только по памяти, но и по скорости, ведь она напрямую сокращает объём данных, читаемых на каждый токен.

Авторы [3] развивают ту же идею, но строже: они связывают каждый метод ускорения со своим типом аппаратного предела (модель Roofline). Квантизация, прунинг и дистилляция сжимают модель, чтобы уменьшить потребление памяти. Специальные алгоритмы генерации (спекулятивное декодирование, ранний выход) и оптимизация KV-кэша борются с ограничением по пропускной способности памяти - именно оно определяет скорость выдачи токенов. А эффективные механизмы внимания убирают вычислительный «потолок», который возникает при обработке промпта.

Другая ветвь обзоров смотрит на инференс не как на одну модель, а как на сервис, обслуживающий поток запросов от многих пользователей. Авторы работы [4] систематизируют приёмы на двух

масштабах. На уровне отдельного узла это размещение модели, планирование запросов (в том числе предсказание длины ответа, чтобы короткие запросы не застревали в очереди за длинными), управление хранением KV-кэша и разнесение стадий инференса, когда «тяжёлую по вычислениям» обработку запроса и «тяжёлую по памяти» генерацию выполняют разные ресурсы. На уровне кластера - развёртывание групп видеокарт и балансировка нагрузки. Главный практический вывод этой линии работ состоит в том, что в многопользовательском режиме пропускную способность поднимает не ускорение одиночного запроса, а непрерывная пакетная обработка (continuous batching) и разумное разделение KV-кэша между сеансами. Тот же компромисс между задержкой и пропускной способностью стоит в центре обзора [5], написанного с позиции систем машинного обучения.

Обзор [6] сравнивает 25 открытых и коммерческих решений и, что особенно ценно, сопоставляет их с типом нагрузки. Ollama и llama.cpp рассчитаны на потребительское железо и одиночные узлы; vLLM, TensorRT-LLM и SGLang - на масштабируемое обслуживание многих пользователей. По совокупности удобства, надёжности и скорости авторы выделяют vLLM, LMDeploy и TGI как сильные универсальные варианты: их пиковая скорость чуть ниже, чем у TensorRT-LLM, зато под высокой нагрузкой они заметно стабильнее - доля успешно обслуженных запросов держится выше 90% даже при множестве одновременных обращений. Для 4-битных моделей на одной видеокарте меньшую задержку до первого токена нередко показывает MLC LLM, а более высокую скорость генерации - llama.cpp. Ollama хорошо справляется с моделями среднего размера, но резко сдаёт на крупных (от 30 млрд параметров) при большом числе одновременных запросов.

Работа [7] сопоставляет дистилляцию, квантизацию и прунинг и отмечает, что 4-битная квантизация весов обычно удерживает качество в приемлемых рамках, тогда как заметное отсечение весов, как правило, требует дообучения. Специализированный обзор [8] разбирает оптимизацию моделей типа «смесь экспертов» (MoE) и выделяет их принципиальную особенность: хотя на каждый токен активируется лишь часть экспертов, в памяти приходится держать их все, а главными узкими местами становятся неравномерная загрузка экспертов и обмен данными между картами.

II. ПОСТАНОВКА ЗАДАЧИ

Непосредственным поводом к разработке послужила потребность университета в собственной системе инференса LLM для учебных целей - обучения студентов работе с языковыми моделями, построения на их основе проектов (RAG-системы, агентные системы) и повседневного использования модели через чат. Опора на внешние сервисы для этих задач оказывается неудобной или неприемлемой сразу по нескольким причинам. Доступ к официальным зарубежным API осложнён инфраструктурно: он требует VPN и обходных решений, что плохо совместимо с массовым использованием в учебном процессе. Отечественные облачные сервисы работают по платной модели с оплатой за токены, что при большом числе студентов и учебной нагрузке быстро становится затратным. Агрегаторы вроде OpenRouter также либо платны, либо предоставляют слишком узкий набор бесплатных моделей с малой квотой бесплатных токенов, недостаточной для полноценной работы группы. К этому добавляется требование локальной обработки данных: моделью пользуются и сотрудники университета, которым необходимо обрабатывать внутренние документы, а передача таких данных стороннему поставщику неприемлема. Совокупность этих ограничений и определяет актуальность работы: доступ к LLM для учебных и внутренних задач надёжнее и дешевле обеспечивается собственной платформой инференса, развёрнутой на имеющемся оборудовании, а описанного в литературе прикладного опыта построения таких систем из разнородного потребительского железа под Windows + WSL2 + Docker на открытом стеке нет.

Исходя из этого, цель работы — спроектировать, развернуть и экспериментально исследовать прототип платформы инференса LLM, обеспечивающей единую точку входа (веб-интерфейс и OpenAI-совместимый API) и горизонтальное масштабирование за счёт подключения новых вычислительных узлов без изменения клиентской части. Платформа должна скрывать от пользователя внутреннюю неоднородность оборудования, маршрутизировать запросы к нужному узлу по имени модели и поддерживать аутентификацию и ограничение частоты запросов.

В качестве аппаратной базы для выполнения проекта использовался Цифровой полигон Высшей инжиниринговой школы НИЯУ МИФИ [9].

В качестве целевого сценария эксплуатации принято одновременное обслуживание не менее 20 пользователей с контекстным окном до 32 тыс. токенов; критерий качества обслуживания — скорость генерации не ниже 20 токенов в секунду на пользователя по нижнему 5-му перцентилю распределения, то есть и для наименее удачливых запросов, а также метрика time to first token - TTFT (95-й перцентиль) не выше 2 с для коротких запросов (промпт до нескольких сотен токенов) и не выше 10 с для запросов, промпт которых превышает 10 % от сконфигурированной длины контекстного окна. Формулирование обоснованных требований к конфигурации, обеспечивающей целевой сценарий, является одним из результатов работы. Экспериментально проверяемым критерием жизнеспособности прототипа принята одновременная работа до 4-х пользователей с контекстом 8 тыс. токенов на одном ведомом узле.

III. ОПИСАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ

Архитектура построена по схеме «ведущий–ведомый» (master–worker) и разделяет ответственность между публичной точкой входа и вычислительными ресурсами. Она изображена на рисунке 1.

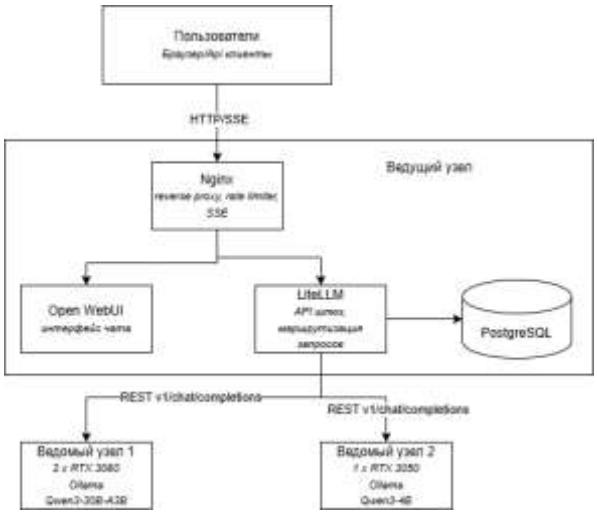

Рисунок 1 – Архитектура платформы

Ведущий узел (виртуальный сервер в облаке МИФИ)

- Оборудование: 16 ГБ ОЗУ, 8 виртуальных ядер, 50 ГБ SSD.
- Программное обеспечение: Ubuntu 22.04, Docker Compose.
- Сервисы:
  - Nginx - обратный прокси-сервер. Для потоковой выдачи ответов (Server-Sent Events, SSE) отключена буферизация (proxy_buffering off).
  - LiteLLM - OpenAI-совместимый шлюз API (порт 4000); отвечает за маршрутизацию запросов к ведомым узлам по имени модели, управление виртуальными ключами и ограничение частоты запросов на уровне пользователей.
  - PostgreSQL 15 - хранилище ключей и пользователей LiteLLM.
  - Open WebUI - веб-интерфейс чата (внутри на порту 8080, наружу - через Nginx).
- Все контейнеры объединены в мостовую сеть (bridge) master-net и управляются через Docker Compose.

Ведомые узлы (физические серверы с GPU)

Каждый ведомый узел - отдельный физический сервер, на котором развернут сервис инференса, принимающий запросы от ведущего узла по HTTP.

- Ведомый узел 1 (2× RTX 3060 12 ГБ, 64 ГБ ОЗУ, Intel Core i7-13700KF): используется Ollama [10] с послойным разделением модели (layer split) между двумя видеокартами. Основная модель - Qwen3-30B-A3B [11]. Дополнительно тестируется мультимодальная модель Gemma 4 26B A4B [12].
- Ведомый узел 2 (1× RTX 3050 8 ГБ, 32 ГБ ОЗУ, AMD Ryzen 9 5900X): средствами Ollama развёрнута лёгкая модель семейства Qwen3 (4 млрд параметров).

Обмен между ведущим и ведомыми узлами

Обмен реализован как обычный REST поверх HTTP: LiteLLM обращается к эндпоинту /v1/chat/completions ведомого узла, совместимому с OpenAI API. Конфигурация ведомых узлов и моделей задаётся в файле config.yaml LiteLLM; добавление нового узла требует внесения записи в этот файл и перезапуска только контейнера-шлюза, но не всей системы.

Такой подход даёт три ключевых преимущества:

1. Масштабируемость. Новый ведомый узел подключается добавлением записи в config.yaml LiteLLM с перезапуском одного лишь шлюза.
2. Безопасность. Узлы с GPU не имеют публичного IP-адреса, недоступны извне и вынесены в изолированную внутреннюю сеть.
3. Гибкость. Под разные задачи можно поднимать разные ведомые узлы и маршрутизировать запросы по имени модели.

IV. ВЫБОР ИНСТРУМЕНТОВ И ОПТИМИЗАЦИЯ ИНФЕРЕНСА НА ВЕДОМОМ УЗЛЕ

Для достижения целевой производительности были рассмотрены следующие методы оптимизации:

1. Квантизация весов - снижение разрядности с 16 до 4 бит сокращает объём видеопамяти под веса примерно в 3–4 раза [13].
2. Оптимизация механизма внимания - механизм FlashAttention [14] уменьшает накладные расходы по памяти и повышает эффективность использования видеопамяти, что особенно важно для кэша ключей и значений (KV-кэша) при длинных контекстах.
3. Смесь экспертов (MoE)- на каждый токен активируется лишь подмножество параметров, что обеспечивает качество, приближающееся к плотным моделям сопоставимого полного размера, при вычислительной стоимости, определяемой числом активных параметров [8].

На первом ведомом узле выбран Ollama. Выбор обоснован тем, что ведомые узлы работают под управлением Windows; Linux-окружение для контейнеров Docker обеспечивается подсистемой Windows Subsystem for Linux 2 (WSL2) [15] - средой виртуализации, в которой доступ контейнеров к GPU реализуется через паравиртуализированный драйвер. Экспериментальная проверка движка vLLM в среде WSL2 выявила невозможность стабильной работы в двухкарточной конфигурации: библиотека межкарточного обмена NCCL, используемая vLLM для тензорного параллелизма, завершалась

ошибками инициализации в паравиртуализированном GPU-окружении. Ollama, использующий послойное разделение без межкарточных коллективных операций, этой проблемы лишён. Кроме того, Ollama поддерживает FlashAttention совместно с квантизацией KV-кэша, что критически важно при работе с длинными контекстами. Модель распределяется между двумя видеокартами послойно (layer split). Механизм layer split означает, что при разделении по слоям и одиночном запросе (размер пакета равен единице) видеокарты работают последовательно: в каждый момент времени активна лишь одна из них, обрабатывающая свою группу слоёв. Следовательно, две RTX 3060 объединяются прежде всего в общий пул видеопамяти объёмом 24 ГБ, позволяющий разместить модель, не помещающуюся на одну карту, - но не дают удвоения скорости генерации.

V. МЕТОДИКА ТЕСТИРОВАНИЯ И ТЕКУЩИЕ РЕЗУЛЬТАТЫ

Испытания включали три части: сравнительное измерение производительности моделей, построение аналитической модели нагрузочной способности и тест производительности при нагрузке. Тестирование производительности выполнено на узле 2; узел 1 на период испытаний был выведен на обслуживание.

Сравнительное измерение производительности моделей

Назначение этого этапа - обоснование выбора моделей для узла 1. Скорость генерации фиксировалась через панель администратора LiteLLM (число токенов ответа, отнесённое к времени генерации).

Таблица 1 - Сравнение моделей-кандидатов для узла 1

Модель	Узел	GPU	Активных параметров	Скорость генерации, токен/с	Потребление VRAM, ГБ
Qwen3-30B-A3B	Узел 1	2× RTX 3060 12 ГБ	~3,3 млрд	~20	~17
Gemma 4 26B A4B	Узел 1	2× RTX 3060 12 ГБ	3,8 млрд	~15	~18

Перед серией замеров выполнялся прогрев - загрузка модели в видеопамять и инициализация CUDA-контекста, - чтобы исключить влияние первичной загрузки на результат. Приведённые значения - среднее по 10 последовательным измерениям на одном фиксированном запросе («Расскажи, что ты умеешь»), при контекстном окне 8 тыс. токенов. Замер характеризует скорость декодирования при одиночном запросе и незаполненном контексте.

Обе модели узла 1 успешно загрузились и распределились между картами по схеме послойного разделения - первые слои на GPU 0, последующие на GPU 1. Основной моделью узла выбрана Qwen3-30B-A3B: она оказалась примерно на треть быстрее Gemma 4 при меньшем потреблении видеопамяти, что согласуется с меньшим числом активных параметров на токен (3,3 млрд против 3,8 млрд).

Аналитическая модель нагрузочной способности

Веса модели занимают фиксированный объём видеопамяти, а основным переменным потребителем при росте числа пользователей и длины контекста становится кэш ключей и значений (KV-кэш), который хранит промежуточные представления всех уже обработанных токенов каждого слоя внимания и растёт линейно с суммарным числом удерживаемых токенов. Объём кэша на одну сессию оценивается по формуле [16, 17]:

$$M\_KV = 2 \cdot L\_attn \cdot H\_kv \cdot d\_head \cdot b \cdot N \quad (1),$$

где L_attn - число слоёв с полным механизмом внимания, H_kv - число голов ключей/значений, d_head - размерность головы, b - число байт на элемент кэша, N - длина контекста в токенах; множитель 2 отражает раздельное хранение ключей и значений.

На обоих узлах включены FlashAttention и 8-битная квантизация кэша (OLLAMA_FLASH_ATTENTION=1, OLLAMA_KV_CACHE_TYPE=q8_0); формат q8_0 хранит элементы блоками по 32 значения с общим масштабным коэффициентом, поэтому эффективный размер элемента составляет ≈1,06 байта, и далее принято b ≈ 1. При послойном разделении модели кэш распределяется вслед за слоями: каждая видеокарта хранит кэш размещённых на ней слоёв.

Для Qwen3-30B-A3B (48 слоёв полного внимания, H_kv = 4, d_head = 128 [11]) получаем ≈48 КБ на токен: сессия с контекстом 8 тыс. токенов занимает около 0,4 ГБ, с контекстом 32 тыс. - около 1,5 ГБ. Для Qwen3-4B (36 слоёв, H_kv = 8, d_head = 128 [11]) - ≈72 КБ на токен: около 0,6 ГБ на сессию 8 тыс. и 2,3 ГБ на сессию 32 тыс. токенов. Отметим, что у плотной модели кэш на токен больше, чем у существенно более крупной MoE-модели, - из-за большего произведения числа слоёв на число KV-голов.

Доступный под кэш объём - разность полной видеопамяти, весов модели и служебных расходов (CUDA-контексты, вычислительные буферы llama.cpp - порядка 1–1,5 ГБ на узел).

Таблица 2 - Оценка текущей нагрузочной способности (8-битный KV-кэш, приближённо)

Узел	Модель	Веса, ГБ	Служебные, ГБ	Доступно под KV, ГБ	KV на сессию 8K / 32K, ГБ	Сессий при 32K
Узел 1	Qwen3-30B-A3B	~17	~1,5	~5,5	0,4 / 1,5	3–4
Узел 2	Qwen3-4B	~2,8	~1	~4	0.6/2.3	1

Оценки таблицы 2 характеризуют предел по памяти; фактическая одновременность запросов может дополнительно ограничиваться вычислительной производительностью GPU, что проверяется нагрузочным тестированием.

Тестирование производительности при нагрузке узла 2

Тестирование выполнялось собственным сценарием на Python (asyncio + httpx) по схеме: N виртуальных пользователей одновременно и независимо выполняют по три последовательных потоковых запроса через полный путь системы (Nginx → LiteLLM → узел). Скрипт приведен в Приложении А. Для каждого запроса фиксировалась скорость генерации после первого токена и time to first token (TTFT) — время от отправки запроса до получения первого токена ответа. TTFT характеризует задержку обработки запроса на этапе prefill. По каждому режиму нагрузки вычислялся нижний 5-й перцентиль скорости (для оценки стабильности генерации у худших запросов) и 95-й перцентиль TTFT (показывает наихудшую задержку для 5 % самых медленных запросов, что важно для выявления аномалий при параллельной работе). Старты пользователей разносились случайной задержкой 0–2 с; число слотов Ollama (OLLAMA_NUM_PARALLEL) устанавливалось равным числу пользователей. Испытания включали две серии: измерения с короткими запросами и измерения на задаче суммаризации документа.

Измерения с короткими запросами. Первая серия измерений охватывала комбинации длины контекста (8, 16, 32 тыс. токенов) и числа слотов; результаты приведены в таблице 3. Критерию удовлетворяют режимы «8K × 5» (21,6 токен/с по нижнему перцентилю), «16K × 3» (22,3 токен/с) и «32K × 1» (57,3 токен/с). Увеличение числа слотов на одну ступень выводит режим за критерий: при «8K × 6» скорость падает до 14,6 токен/с.

Значения TTFT (p95) для всех рассмотренных режимов коротких запросов находятся в узком диапазоне 1,6–1,87 с, что указывает на то, что задержка обработки промпта (prefill) определяется в основном длиной входящего сообщения, которая одинакова для всех запросов, и практически не зависит от числа одновременно обрабатываемых слотов. Даже в режимах с превышением резерва KV-кэша («16K × 4» и «32K × 2») TTFT остаётся на уровне ~1,8 с для худших 5 % запросов, поскольку перегрузка памяти сказывается в первую очередь на скорости генерации (декодирования), а не на времени до первого токена – префилл всё ещё может выполняться последовательно, без дополнительных задержек. Таким образом, для коротких запросов узким местом является пропускная способность памяти при генерации, а не задержка обработки промпта.

Таблица 3 - Узел 2, короткие запросы (Qwen3-4B, Q4_K_M, кэш q8_0)

Режим (контекст × слоты)	Резерв KV, ГБ	Скорость на польз. p5, токен/с	Time to first token (TTFT) p95, с
32K × 1	2,4	57,3	1,6
16K × 3	3,5	22,3	1.87
8K × 5	2,9	21,6	1,85
8K × 6	3,5	14,6	1,81
16K × 4	4,7	15,7	1,86
32K × 2	4,7	13,9	1,8

Измерения на задаче суммаризации. Вторая серия проверяла работу с заполненным контекстом: на вход подавалась статья объёмом около половины контекстного окна с заданием составить структурированную суммаризацию; средняя длина ответа составляла ≈2 тыс. токенов. Результаты приведены в таблице 4. Заполнение контекста снижает допустимое число пользователей на одну ступень относительно коротких запросов: критерию удовлетворяют режимы «8K × 4» (21,1 токен/с), «16K × 2» (28,4 токен/с). Снижение скорости объясняется тем, что на каждом шаге декодирования читается KV-кэш всего накопленного контекста: для одиночного запроса скорость падает с 57,3 токен/с при незаполненном контексте до 33,4 токен/с при суммарном контексте ≈17 тыс. токенов.

В отличие от коротких запросов, значения TTFT (p95) для суммаризации демонстрируют существенный рост с увеличением длины контекста и числа одновременных слотов. Для режима «32K × 1» TTFT достигает 12,92 с, что обусловлено длительным вычислением механизма внимания на промпте из 15 тыс. токенов — эта фаза префилла упирается в вычислительную мощность GPU (тензорные ядра), а не в пропускную способность памяти. При добавлении параллельных пользователей TTFT дополнительно возрастает: для «16K × 3» p95 составляет 14,81 с, что почти втрое выше, чем для «16K × 2» (4,89 с).Таким образом, для длинных контекстов TTFT становится критическим показателем, ограничивающим интерактивность, и его рост служит ранним признаком достижения пределов нагрузочной способности узла.

Таблица 4 - Узел 2, суммаризация документа (Qwen3-4B)

Режим	Размер документа, токенов	Скорость на польз. p5, токен/с	Time to first token (TTFT) p95, с
8K × 4	3 965	21,1	6,99
8K × 5	3 965	17,9	9,09
16K × 2	7 657	28,4	4,89
16K × 3	7 657	14,0	14,81
32K × 1	15 141	33,4	12,92

Промежуточный ориентир работы (до 4-х одновременных пользователей с контекстом 8 тыс. токенов) на узле 2 достигнут и подтверждён экспериментально.

Требования к целевому состоянию

Суммарной ёмкости имеющихся узлов достаточно для промежуточной цели, однако целевое состояние (20 пользователей, контекст до 32 тыс. токенов) недостижимо простым распределением нагрузки.

Таблица 5 - Что требуется для достижения целевого состояния (20 пользователей, контекст до 32K)

Компонент	**Сейчас**	**Требуется**	**Обоснование**
Движок инференса	Ollama	vLLM (continuous batching, PagedAttention, кэширование префиксов)	Эффективное совместное использование KV-кэша между сессиями и высокая пропускная способность при пакетной обработке. Миграция предполагает перевод диалогового узла на нативный Linux и смену формата весов с GGUF на AWQ/GPTQ (объём сопоставим)
Видеопамять диалогового узла	24 ГБ	~40–48 ГБ	17 ГБ веса + 15–30 ГБ KV на 20 сессий × 32K (в зависимости от разрядности кэша и средней заполненности контекста) + служебные расходы
Вариант оборудования	2× RTX 3060	1× RTX A6000 48 ГБ, либо 2× RTX 3090 24 ГБ (с NVLink), либо несколько узлов по 24 ГБ	разные компромиссы «стоимость / сложность эксплуатации»; из потребительских карт NVLink для тензорного параллелизма поддерживает RTX 3090, но не RTX 4090
Квантизация KV-кэша	8 бит (q8_0)	8 бит и ниже (в vLLM - FP8)	снижение объёма кэша на сессию вдвое относительно 16 бит и далее
Балансировка	LiteLLM (маршрутизация по имени модели)	LiteLLM (распределение нагрузки между несколькими диалоговыми узлами)	горизонтальное масштабирование под растущее число пользователей

Оценим требуемую память основного диалогового узла по аналитической модели. В худшем случае, когда все 20 сессий одновременно заполнены до 32 тыс. токенов, для Qwen3-30B-A3B необходимо 17 ГБ на веса, 20 × 1,5 = 30 ГБ на кэш и 2–3 ГБ служебных - порядка 50 ГБ. Реалистичная оценка ниже: vLLM благодаря постраничному выделению памяти (PagedAttention [16]) резервирует кэш по фактической длине, а не под максимум - в отличие от предварительного резервирования Ollama; снижение разрядности кэша до 4 бит дополнительно уменьшает его объём вдвое. С учётом этих факторов потребность составляет 35–48 ГБ.

Приведённые требования опираются на аналитическую модель и подлежат уточнению при испытаниях целевой конфигурации.

Ограничения исследования. Нагрузочное тестирование выполнено на узле 2, нагрузочная способность узла 1 оценена только аналитически и по одиночным замерам, её экспериментальная проверка отнесена к дальнейшей работе. Число слотов Ollama устанавливалось равным числу пользователей, поэтому режим очереди (число пользователей превышает число слотов) не исследовался. При тестировании производительности под нагрузкой для каждого режима выполнялось N × 3 запросов (от 3 до 18 измерений на режим). В качестве консервативной оценки худшего случая использовался нижний 5-й перцентиль (линейная интерполяция), который на выборках такого объёма практически совпадает с минимумом наблюдений. Таким образом, критерий проверяется по оценке худшего наблюдения.

ЗАКЛЮЧЕНИЕ

В работе создан работоспособный прототип платформы инференса LLM, построенной по архитектуре «ведущий–ведомый» на потребительском оборудовании (GPU классов RTX 3050/3060, Windows + WSL2 + Docker) с полностью открытым программным стеком (Nginx, LiteLLM, PostgreSQL, Open WebUI, Ollama). Платформа обеспечивает единую точку входа через веб-интерфейс и OpenAI-совместимый API с маршрутизацией запросов по имени модели.

Основные результаты следующие. На узле с двумя GPU показана стабильная работа MoE-модели

Qwen3-30B-A3B при послойном разделении (около 20 токен/с при одиночном запросе, ~17 ГБ видеопамяти); сравнение с Gemma 4 26B A4B согласуется с ожидаемой связью скорости генерации с числом активных параметров. Построена аналитическая модель нагрузочной способности узла на основе бюджета видеопамяти и объёма KV-кэша. Тест производительности при нагрузке узла 2 (Qwen3-4B) определил допустимые режимы обслуживания по критерию «не ниже 20 токен/с на пользователя по нижнему 5-му перцентилю»: 4 пользователей при контексте 8 тыс. токенов, 2 при 16 тыс. и 1 при 32 тыс. для коротких запросов; при заполненном контексте (задача суммаризации документа) — 4, 2 и 1 соответственно. Измерения TTFT (95-й перцентиль) в этих допустимых режимах составили от 1,6 до 1,87 с для коротких запросов, что укладывается в целевой порог 2 с; для суммаризации значения TTFT лежат в диапазоне от 4,89 с (16K×2) до 12,92 с (32K×1), при этом режим 32K×1 превышает установленный порог 10 с, что требует дополнительной оптимизации обработки длинных контекстов. Промежуточный ориентир работы (до 4 одновременных пользователей с контекстом 8 тыс. токенов на одном узле) достигнут и подтверждён экспериментально.

Количественная оценка по аналитической модели показала, что целевое состояние (20 пользователей, контекст до 32 тыс. токенов) требует перехода на движок с непрерывной пакетной обработкой и постраничным выделением кэша (vLLM), увеличения видеопамяти диалогового узла до ~40–48 ГБ и горизонтального масштабирования средствами балансировки LiteLLM. Дальнейшая работа включает тестирование производительности узла 1 после его возвращения из обслуживания, миграцию диалогового узла на vLLM и экспериментальную проверку целевой конфигурации.

БЛАГОДАРНОСТИ

Авторы выражают благодарность Высшей инженерной школе НИЯУ МИФИ за предоставленную вычислительную инфраструктуру и поддержку при проведении экспериментальных исследований.

БИБЛИОГРАФИЯ

[1] Zhao W. X., Zhou K., Li J. et al. A Survey of Large Language Models // arXiv preprint arXiv:2303.18223. 2023.
[2] Zhou Z., Ning X., Hong K. et al. A Survey on Efficient Inference for Large Language Models // arXiv preprint arXiv:2404.14294. 2024.
[3] Yuan Z., Shang Y., Zhou Y. et al. LLM Inference Unveiled: Survey and Roofline Model Insights // arXiv preprint arXiv:2402.16363. 2024.
[4] Zhen R., Li J., Ji Y. et al. Taming the Titans: A Survey of Efficient LLM Inference Serving // Proceedings of the 18th International Natural Language Generation Conference (INLG). Hanoi, 2025. P. 522–541.
[5] Miao X. et al. Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems // ACM Computing Surveys. 2025.
[6] Lee J. et al. A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency // arXiv preprint arXiv:2505.01658. 2025.
[7] Girija S. S. et al. Optimizing LLMs for Resource-Constrained Environments: A Survey of Model Compression Techniques // 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC). 2025.
[8] Liu J., Tang P., Wang W. et al. A Survey on Inference Optimization Techniques for Mixture of Experts Models // ACM Computing Surveys. 2026. DOI: 10.1145/3794845.
[9] G.G. Hajilov, A.V. Khatunov, T.A. Voloshin, M.G. Zhabitsky MEPhI Higher Engineering School Digital polygon for educational and practical projects infrastructure support International Journal of Open Information Technologies ISSN: 2307-8162 vol. 13, no. 8, 2025
[10] [Электронный ресурс]. Ollama Project. Ollama: Get up and running with large language models locally. URL: https://ollama.com (дата обращения: 07.07.2026).
[11] Yang A., Li A., Yang B. et al. Qwen3 Technical Report // arXiv preprint arXiv:2505.09388. 2025.
[12] [Электронный ресурс]. Gemma Team (Google DeepMind). Gemma 4 Model Card. URL: https://ai.google.dev/gemma/docs/core/model_card_4 (дата обращения: 07.07.2026).
[13] Frantar E., Ashkboos S., Hoefler T., Alistarh D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers // ICLR. 2023.
[14] Dao T., Fu D. Y., Ermon S. et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness // NeurIPS. 2022.
[15] [Электронный ресурс]. Microsoft. GPU acceleration in WSL 2. URL: https://learn.microsoft.com/windows/wsl/gpu-acceleration (дата обращения: 07.07.2026).
[16] Kwon W., Li Z., Zhuang S. et al. Efficient Memory Management for Large Language Model Serving with PagedAttention // SOSP. 2023. P. 611–626.
[17] Ainslie J., Lee-Thorp J., de Jong M. et al. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints // Proceedings of EMNLP. 2023.

Статья получена 27 июня 2026 г.
Свириденко Д. К., НИЯУ МИФИ, магистрант (e-mail: dmitrii.sviridenko@yandex.ru)
Боброва Е.В., НИЯУ МИФИ, аспирант, (e-mail: EVBobrova@mephi.ru)
Анни Павел Витальевич, Red Hat, (e-mail: pavel.anni@gmail.com)

Development of a Large Language Model Inference Platform with Request Routing on Consumer-Grade Hardware

D.K. Sviridenko, E.V. Bobrova, P.V. Anni

***Abstract* - This paper presents the design, development, and testing of a prototype platform for large language model (LLM) inference built on a master–worker architecture using consumer-grade hardware running Windows + WSL2 + Docker with a fully open-source software stack. The platform provides a single point of entry via a web interface and an OpenAI-compatible API, routes requests to compute nodes by model name, and supports authentication and rate limiting. New nodes can be added without any changes to the client side. All nodes are hosted on MEPhI servers, so all traffic remains within the organization's perimeter. The nodes serve models of different classes, including a mixture-of-experts (MoE) model distributed across two GPUs. An analytical model of node serving capacity is proposed, based on the VRAM budget and KV-cache size. Practical significance: the platform provides access to LLMs on existing hardware, eliminating dependence on external cloud services and ensuring that data never leaves the organization's perimeter.**

***Keywords* - LLM inference, master–worker architecture, horizontal scaling, Ollama, WSL2, Docker, mixture-of-experts, Qwen3, LiteLLM.**

REFERENCES

БИБЛИОГРАФИЯ

[1] Zhao W. X., Zhou K., Li J. et al. A Survey of Large Language Models // arXiv preprint arXiv:2303.18223. 2023.

[2] Zhou Z., Ning X., Hong K. et al. A Survey on Efficient Inference for Large Language Models // arXiv preprint arXiv:2404.14294. 2024.

[3] Yuan Z., Shang Y., Zhou Y. et al. LLM Inference Unveiled: Survey and Roofline Model Insights // arXiv preprint arXiv:2402.16363. 2024.

[4] Zhen R., Li J., Ji Y. et al. Taming the Titans: A Survey of Efficient LLM Inference Serving // Proceedings of the 18th International Natural Language Generation Conference (INLG). Hanoi, 2025. P. 522–541.

[5] Miao X. et al. Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems // ACM Computing Surveys. 2025.

[6] Lee J. et al. A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency // arXiv preprint arXiv:2505.01658. 2025.

[7] Girija S. S. et al. Optimizing LLMs for Resource-Constrained Environments: A Survey of Model Compression Techniques // 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC). 2025.

[8] Liu J., Tang P., Wang W. et al. A Survey on Inference Optimization Techniques for Mixture of Experts Models // ACM Computing Surveys. 2026. DOI: 10.1145/3794845.

[9] G.G. Hajilov, A.V. Khatunov, T.A. Voloshin, M.G. Zhabitsky MEPhI Higher Engineering School Digital polygon for educational and practical projects infrastructure support International Journal of Open Information Technologies ISSN: 2307-8162 vol. 13, no. 8, 2025

[10] Ollama Project. Ollama: Get up and running with large language models locally. URL: https://ollama.com (дата обращения: 07.07.2026).

[11] Yang A., Li A., Yang B. et al. Qwen3 Technical Report // arXiv preprint arXiv:2505.09388. 2025.

[12] Gemma Team (Google DeepMind). Gemma 4 Model Card. URL: https://ai.google.dev/gemma/docs/core/model_card_4 (дата обращения: 07.07.2026).

[13] Frantar E., Ashkboos S., Hoefler T., Alistarh D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers // ICLR. 2023.

[14] Dao T., Fu D. Y., Ermon S. et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness // NeurIPS. 2022.

[15] Microsoft. GPU acceleration in WSL 2. URL: https://learn.microsoft.com/windows/wsl/gpu-acceleration (дата обращения: 07.07.2026).

[16] Kwon W., Li Z., Zhuang S. et al. Efficient Memory Management for Large Language Model Serving with PagedAttention // SOSP. 2023. P. 611–626.

[17] Ainslie J., Lee-Thorp J., de Jong M. et al. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints // Proceedings of EMNLP. 2023.