Бызов Степан Андреевич, магистрант
Амурский государственный университет
Byzov Stepan Andreevich, Amur State University

АРХИТЕКТУРА ПРОГРАММНОЙ СИСТЕМЫ ГЕНЕРАЦИИ И ПРОВЕРКИ УЧЕБНЫХ ЗАДАНИЙ С ЯЗЫКОВОЙ МОДЕЛЬЮ
SYSTEM ARCHITECTURE FOR LLM-BASED EXERCISE GENERATION AND ASSESSMENT

Аннотация. Описана архитектура генерации и проверки заданий с языковой моделью. Приём ответа отделён от инференса. Оценку связывают с версией рубрики и источниками контекста. Заданы метрики нагрузки.

Abstract. The proposed architecture separates answer submission from language model inference and links each assessment to a rubric version and its evidence sources. Load testing metrics are defined.

Ключевые слова: Программная архитектура, микросервисы, языковые модели, RAG, автоматическая проверка, нагрузочное тестирование.

Keywords: Software architecture, microservices, language models, RAG, automated assessment, load testing.

Границы архитектурной задачи. Результаты применения больших языковых моделей (LLM) к исправлению учебных текстов зависят от модели, формулировки запроса и способа оценки [1]. При встраивании модели в образовательную систему возникает отдельная задача: сохранить принятый ответ при отказе инференса, объяснить происхождение выставленной оценки и обрабатывать всплески нагрузки. Если запрос к LLM выполняется внутри операции приёма ответа, задержки модели удлиняют ожидание пользователя. Если версия рубрики не сохраняется, повторная проверка может опираться на другие критерии. Такие риски обусловлены связностью компонентов; общая идея раздельной оценки компонентов согласуется с подходом составных систем на основе ИИ [2].

Выбор способа развёртывания. Модульный монолит позволяет держать прикладные операции в одном процессе и упрощает согласование данных. Микросервисы дают возможность отдельно развёртывать сервисы с разными профилями нагрузки, но создают сетевые переходы и требуют согласования асинхронных операций. Экспериментальное сравнение показывает, что на одном узле монолит может выигрывать по производительности; перенос результата на другие конфигурации требует проверки [3]. Бессерверные обработчики можно масштабировать по событиям, однако запуск экземпляров зависит от распределения нагрузки и политики прогрева [4]. В таблице 1 указаны последствия выбора для рассматриваемой системы.

Таблица 1

Сравнение способов развёртывания

Подход	**Масштабирование**	**Издержки**
Монолит	Всё приложение	Совместное масштабирование
Микросервисы	Отдельный сервис	Сеть и согласование данных
Serverless	Функция по событию	Холодный запуск, лимиты среды

Для проектной архитектуры выбраны микросервисы с общим API Gateway и выделенным процессом инференса, который обновляется независимо от веб-приложения. Проверку и генерацию можно масштабировать раздельно до насыщения общего модельного узла. Управляемые функции уместны для нерегулярных задач, например выгрузки отчёта, и не входят в критический путь проверки. Небольшое число границ снижает число распределённых взаимодействий по сравнению с выделением отдельного сервиса на каждую операцию [3, 4]. Масштабирование не меняет внешний API.

Компоненты и маршрут запроса. Клиент Next.js отображает задания, отправляет ответы и запрашивает статус обработки [5]. Шлюз на FastAPI проверяет полномочия, валидирует сообщения, ограничивает частоту запросов и передаёт их сервису, не вызывая LLM напрямую [6]. Сервис пользователей владеет профилем и ролями, сервис генерации – версиями заданий и ключами, сервис проверки – попытками и вердиктами. Сервис аналитики строит агрегаты из опубликованных событий, не меняя данные проверки. На рисунке 1 показаны компоненты и их связи.

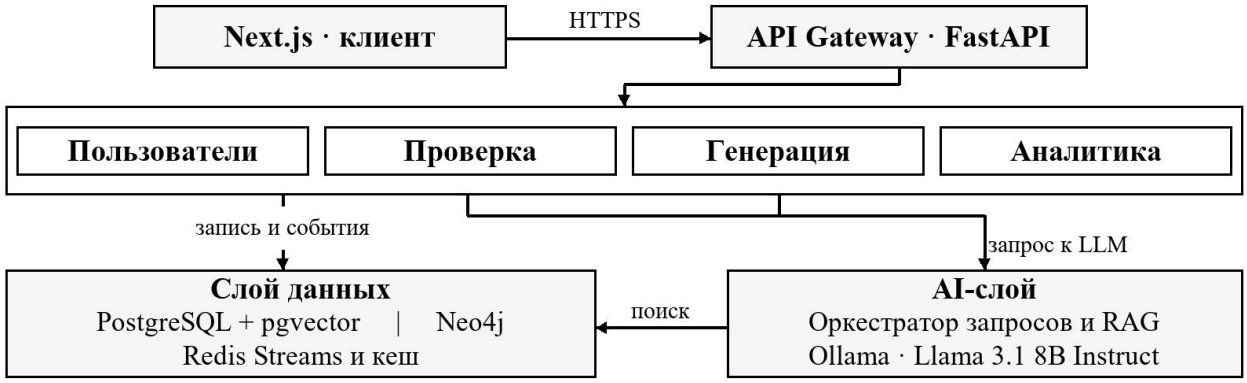

Рисунок 1. Компоненты системы и направления обмена

PostgreSQL хранит версии заданий, рубрик и журнал операций; владельцы данных используют отдельные схемы и роли [7]. Расширение pgvector поддерживает поиск по векторному представлению утверждённых фрагментов рубрик [8]. Neo4j хранит восстанавливаемую проекцию связей между заданием, правилом и попыткой для аналитических запросов [9]. Графовая проекция обновляется после записи результата и не нужна для приёма ответа. Redis Streams передаёт задания и события обработчикам; Redis также используется для временного кеша [10]. Источником статуса и результата остаётся PostgreSQL: потеря кеша не меняет принятую оценку.

При отправке ответа сервис проверки в одной транзакции сохраняет попытку и запись для публикации в очереди. Клиент получает идентификатор и статус «принято». Повтор с тем же ключом запроса возвращает прежний идентификатор; если при повторе изменилось содержимое, запрос отклоняется. Отдельный процесс публикует запись в Redis Streams. После сбоя брокера необработанные попытки можно поставить в очередь повторно по данным журнала. Повторную доставку допускает выбранная схема обмена [7, 10].

Обработчик захватывает попытку на ограниченное время. Результат принимается только от владельца актуального номера обработки; это исключает запись запоздавшего вердикта после повторного запуска. Оценка и событие для аналитики фиксируются одной транзакцией; подтверждение сообщения отправляется после фиксации. Уникальный ключ по попытке и версии проверки допускает не более одного принятого вердикта для этой пары. Вызов модели может повториться, поэтому гарантия относится к записи результата. Аналитика применяет события по уникальным идентификаторам, без повторного учёта при дублировании [7, 10].

Слой ИИ содержит общий адаптер к Ollama и отдельные шаблоны запросов для проверки и генерации. Для проекта выбрана модель Llama 3.1 8B Instruct семейства Llama 3 [11]. Фиксируются идентификатор модели, версия шаблона и параметры декодирования. Ollama поддерживает структурированный вывод по схеме JSON [12]; прикладной валидатор отдельно проверяет допустимость оценки и ссылок на рубрику. Генератор сохраняет новую версию задания после валидации; для проверки зарезервирована часть параллельных вызовов. При исчерпании бюджета повторов попытка передаётся на ручную проверку, а её статус можно запросить через шлюз по идентификатору попытки.

Контекст проверки и RAG. Для ответов с однозначным ключом используется детерминированная проверка. В открытых ответах RAG дополняет запрос к модели выдержками из утверждённых правил и рубрик [13]. Сервис сначала ограничивает поиск версией задания, языком и разрешённым набором материалов, затем ранжирует допустимые фрагменты по близости векторов [8]. Запрос строится из цели задания и критериев рубрики; сам ответ передаётся в отдельном поле для оценки. У ответов других учеников нет роли источника правил. Для аудита сохраняются идентификаторы переданных фрагментов, версия индекса и версия модели эмбеддингов.

Модель возвращает балл, код критерия и идентификатор фрагмента, на котором основано объяснение. Сервис проверяет диапазон балла и принадлежность ссылок переданному набору; смысловое соответствие объяснения источнику оценивается отдельно. Если доступных фрагментов недостаточно, выставление автоматической оценки откладывается. Для оценки вклада RAG сопоставляются два режима при одной модели и одной разметке: полная рубрика без поиска и рубрика с релевантными выдержками. Измеряются совпадение с экспертным баллом, доля ответов с опорой на источник и доля ручных проверок; качество поиска и ответа анализируется раздельно [14].

Архитектурные показатели. Из схемы следуют три показателя: до подтверждения приёма выполняется 0 вызовов LLM; приём и фиксация оценки оформлены двумя отдельными транзакциями; на пару «попытка, версия проверки» приходится не более одного принятого результата. Для нагрузочной проверки отдельно измеряются 95-е перцентили времени подтверждения приёма и получения результата, возраст старейшего сообщения, доля ручных проверок и число завершённых заданий в секунду [3, 4].

Протокол предполагает ступенчатое увеличение частоты подачи ответов при параллельной генерации и повтор сценария после отключения модельного узла. Конфигурация оборудования, размер контекста и доли заданий с ключом и открытым ответом фиксируются до прогона. В сценарии восстановления контролируются сохранность принятых попыток, повторная доставка и отсутствие дублей оценок. Варианты маршрута сравниваются при одинаковой модели и конфигурации данных. В данной работе нагрузочные прогоны на реальном оборудовании не проводились.

Предложенная архитектура отделяет приём ответа от вычисления оценки, фиксирует происхождение результата и задаёт границы, по которым можно измерить насыщение компонентов. Независимое масштабирование сервисов снимает ограничение прикладного уровня до насыщения модельного узла. Дальнейшая оценка должна установить, как при одной рубрике меняются качество проверки с RAG и время обработки при смешанном потоке задач и при разных долях открытых ответов

Список литературы:

1. Prompting open-source and commercial language models for grammatical error correction of English learner text / C. Davis et al. // Findings of ACL. 2024. P. 11952–11967. DOI: 10.18653/v1/2024.findings-acl.711.

2. The Shift from Models to Compound AI Systems / M. Zaharia et al. // Berkeley AI Research. 2024.

3. Blinowski G. J., Ojdowska A., Przybyłek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation // IEEE Access. 2022. Vol. 10. P. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803.

4. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider / M. Shahrad et al. // USENIX ATC. 2020. P. 205–218.

5. Vercel. Next.js 15. 2024.

6. FastAPI. Release 0.115.0. 2024.

7. PostgreSQL Global Development Group. PostgreSQL 17 Released. 2024.

8. pgvector. Release v0.8.0. 2024.

9. Neo4j. Neo4j v5 Long-Term Support and the Continued Evolution. 2025.
10. Redis. Release 7.2.0. 2023.
11. Grattafiori A. et al. The Llama 3 Herd of Models. arXiv, 2024.
12. Ollama. Structured Outputs. 2024.
13. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / P. Lewis et al. // NeurIPS. 2020. Vol. 33. P. 9459–9474.
14. Es S. et al. RAGAs: Automated Evaluation of Retrieval Augmented Generation // EACL (Demos). 2024. P. 150–158