

УДК 004.8:37.018.46

Эволюция архитектуры нейросетевого модуля: поддержка мультимодальных сервисов в цифровой образовательной платформе Мирера

А. С. Караваева¹, А. М. Куликов², Н. С. Мартынов³, К. А. Машченко⁴,
К. К. Пчелин⁵, Е. Д. Тарасюк⁶

¹НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, aleksandrakaravaeva@yandex.ru;

²НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, kulikrch@yandex.ru;

³НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, nikolai.martynov@math.msu.ru;

⁴НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, Государственный университет управления, Москва, Россия, kirill.mashchenko@niisi.ru;

⁵НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, pcelinkosta0@gmail.com;

⁶НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, ekaterina.tarasuk@math.msu.ru;

Аннотация. В работе рассматривается развитие архитектуры нейросетевого модуля цифровой образовательной платформы Мирера в связи с расширением круга применяемых моделей искусственного интеллекта. Показано, что исходное LLM-ядро оркестратора не позволяло без существенных переработок подключать модели иных модальностей. Описана универсальная мультимодальная архитектура, включающая абстракцию сервис-менеджера на основе единого контракта, конфигурируемые цепочки обработки запросов и унифицированный поток выходных артефактов. Приведены примеры интеграций. Показано, что предлагаемая архитектура обеспечивает подключение новых типов моделей без модификации ядра маршрутизатора и формирует основу для дальнейшего развития мультимодальных образовательных сценариев.

Ключевые слова: цифровая образовательная платформа, ЦОП Мирера, нейросетевой оркестратор, мультимодальные модели, микросервисная архитектура, LLM, OCR, RabbitMQ, AI-агенты, образовательные технологии

1. Введение

В последние годы большие языковые модели (LLM) стали одним из ключевых направлений практического внедрения искусственного интеллекта в образовательную сферу. Появление высокопроизводительных открытых архитектур, их доступность в форме готовых API-сервисов и снижение требований к аппаратной инфраструктуре благодаря методам квантизации сделали возможным переход от единичных экспериментов к систематическому встраиванию языковых моделей в цифровые образовательные платформы. Сегодня LLM применяются для автоматизированной генерации учебных заданий и тестовых вопросов, рецензирования решений обучающихся, построения интерактивных помощников по конкретным дисциплинам, персонализации образовательной траектории и снижения административной нагрузки

преподавателей [1]. Параллельно расширяется и круг сопутствующих моделей искусственного интеллекта: к языковым моделям подключаются системы оптического распознавания, обработки изображений, распознавания и синтеза речи, формируя инструменты для работы с мультимодальными сценариями в цифровых образовательных платформах.

В цифровой образовательной платформе Мирера [2] накоплен опыт интеграции моделей искусственного интеллекта в широкий круг учебных задач. В работе [3] обобщены критерии выбора больших языковых моделей под образовательные сценарии и приведена сравнительная характеристика применяемых в платформе архитектур, а также описаны конкретные сценарии их использования: от генерации заданий и тестовых вариантов до построения помощников студента и преподавателя. В работе [4] изложен опыт интеграции нейросетевого ассистента

обучающихся по программированию, способного анализировать решения студентов в реальном времени и формировать обучающие рекомендации непосредственно в среде платформы. В работе [5] представлена система автоматизированного формирования образовательных видеоматериалов на основе модульного цифрового двойника преподавателя, объединяющая модули автоматического распознавания речи (ASR), синтеза речи (TTS) и синхронизации движений губ (LipSync), указанная система демонстрирует существенное сокращение трудозатрат преподавателя на подготовку учебных видеоматериалов при сохранении их высокого качества. Помимо собственно нейросетевых сервисов, платформа развивает и смежные направления автоматизации учебного процесса, в частности, гибридный учёт посещаемости с QR-верификацией и видеоидентификацией [6], что в совокупности формирует целостную инфраструктуру интеллектуальной поддержки участников образовательного процесса.

Образовательный процесс по своей природе мультимодален: помимо текстовых формулировок задач и решений, в нём задействованы изображения и схемы из учебных материалов, аудиозаписи и видеозаписи лекций, рукописные конспекты, презентации преподавателей. По мере усложнения образовательных сценариев в цифровой платформе круг применяемых моделей искусственного интеллекта закономерно выходит за рамки чисто языковых: становятся востребованными системы оптического распознавания учебных материалов и рукописного текста, модели распознавания и синтеза речи и другие нейросетевые модули. Каждое из этих направлений требует моделей с собственной модальностью входных данных, специфической аппаратной конфигурацией (например, специализированных сетей для оптического распознавания и визуально-языковых моделей для документов сложной структуры) и собственной семантикой выходного результата – структурированного текста, аудиосигнала, графического представления. Реализация подобных сценариев в рамках единой образовательной платформы требует, чтобы ядро нейросетевого модуля было способно подключать модели разной природы без существенных переработок маршрутизации, контрактов взаимодействия с платформой и инфраструктуры обработки запросов.

Реализация описанных в работах [3; 4] сценариев потребовала создания самостоятельного нейросетевого модуля, взаимодействующего с ядром платформы

Мирера через очереди сообщений и выполняющего функции маршрутизации запросов, управления пулом моделей и сборки ответов по контракту с платформой. Изначально этот модуль проектировался под единственный класс задач – обработку текстовых запросов с применением LLM, что определило его архитектурную особенность: все цепочки обработки запросов жёстко предполагали, что в конце сценария работы стоит языковая модель, обработка вложений (документов, изображений) выполнялась локально внутри агента с применением библиотеки распознавания, привнося в монолит существенные аппаратные зависимости.

С появлением задач иной модальности обнаружилась несовместимость подобной архитектуры с расширенным кругом моделей. Подключение каждого нового типа сервиса в её рамках потребовало бы либо дублирования инфраструктуры маршрутизации, либо точечных модификаций ядра под специфику конкретного сервиса, оба пути ведут к нарастанию сложности и ухудшению поддерживаемости. Кроме того, размещение зависимостей распознавания внутри монолита не позволяло независимо масштабировать ресурсы под задачи распознавания и под LLM-инференс, имеющие существенно различающиеся профили нагрузки и аппаратные требования. (LLM-инференсом мы называем этап, на котором уже обученная LLM-модель обрабатывает пользовательский запрос и генерирует ответ без изменения своих внутренних параметров).

Данная статья описывает архитектурную эволюцию нейросетевого модуля платформы Мирера, направленную на универсальную поддержку мультимодальных сервисов. В разделе 2 рассматривается исходная LLM-ориентированная архитектура и её ограничения. В разделе 3 описывается переход к универсальной мультимодальной архитектуре с единым контрактом сервис-менеджера, конфигурируемыми цепочками и унифицированным потоком выходных артефактов. В разделе 4 приведены практические реализации возможностей новой архитектуры.

2. Исходная LLM-ориентированная архитектура

2.1 Двухфазная обработка запросов

Нейросетевой модуль платформы Мирера построен на принципе асинхронной обработки запросов через брокер сообщений RabbitMQ, использующий протокол AMQP [7]. Запросы от

платформы поступают в виде структурированных сообщений-конвертов в очередь приёма, ответы помещаются в очередь уведомлений и доставляются обратно платформе для отображения результата участникам учебного процесса. Между этими крайними точками обработка запроса разделена на две фазы, каждая из которых реализуется собственным циклом-потребителем сообщений.

В фазе приёма (Intake loop) маршрутизатор выполняет валидацию структуры входящего сообщения-конверта, загружает описание цепочки обработки соответствующей команды из конфигурационного файла, проверяет полезную нагрузку на соответствие контракту команды и помещает сообщение в очередь первого сервиса в цепочке. Эта фаза не выполняет вычислительной работы и обеспечивает строгий входной контроль: некорректные запросы с нарушенной структурой сообщения, неизвестным именем команды или несоответствующей контракту полезной нагрузкой отсекаются до того, как они достигнут моделей, чувствительных к нагрузке. Благодаря отсутствию тяжёлых зависимостей цикл приёма может масштабироваться независимо от циклов исполнения, что особенно важно при пиковых нагрузках на платформу в начале учебных занятий, при массовых отправках решений студентами и в периоды контрольных мероприятий.

В фазе исполнения (Service loop) сервис-менеджер извлекает сообщение из очереди соответствующего сервиса, выполняет преобразование входных полей в соответствии с описанием шага цепочки, валидирует запрос по строгой схеме конкретного пайплайна, выполняет инференс модели и затем преобразование выходных полей в артефакты. Если цепочка содержит следующий шаг, сообщение помещается в очередь следующего сервиса, если шаг последний, ответ собирается по контракту команды и отправляется в очередь уведомлений платформы Мирера. Каждый сервис-менеджер удерживает в памяти собственный пул моделей в состоянии готовности и подтверждает обработку очередного сообщения брокеру только после успешного завершения соответствующего шага, что обеспечивает атомарность доставки и устойчивость системы к сбоям отдельных компонентов: при аварийном завершении обработки сообщение возвращается в очередь и поднимается другим экземпляром сервис-менеджера.

2.2 LLMServiceManager как единая точка работы с моделями

Ранее в реестре маршрутизатора был зарегистрирован единственный класс сервис-менеджера – LLMServiceManager. Этот компонент реализовывал несколько обязанностей: управление пулом экземпляров языковых моделей, разрешение конфликтов одновременного доступа через механизм захвата и освобождения моделей, выбор конкретного пайплайна по имени и непосредственное взаимодействие с моделью через подсистему LLMManager. Пул моделей размещался в режиме горячего ожидания: при старте сервиса все используемые модели загружались в видеопамять графических ускорителей, что устраняло задержки повторной инициализации, от десятков секунд до нескольких минут на одну модель в зависимости от её размера и метода загрузки, и обеспечивало низкую задержку отклика при поступлении запроса от платформы.

Помимо поддержки моделей, сервис-менеджер выполнял функции примитивного балансировщика нагрузки между экземплярами одной и той же модели: при поступлении запроса он искал свободный экземпляр среди занятых и, только при отсутствии такового, возвращал признак временной недоступности, побуждая брокер повторно поставить сообщение в очередь после небольшой задержки. В LLMServiceManager были также инкапсулированы логика опроса состояния моделей и сбор прикладных метрик – счётчиков обработанных запросов, гистограмм времени отклика, отметок ошибок инференса.

Для каждой образовательной задачи определялся отдельный пайплайн – самодостаточный модуль, содержащий строгую схему запроса и ответа, шаблон формирующей запрос инструкции и логику выполнения. Точкой входа в пайплайн служила асинхронная функция, принимающая валидированный запрос, экземпляр модели и ссылку на сервис-менеджер, возвращаемое значение представляло собой словарь именованных артефактов. Поиск пайплайнов был организован двухуровневым: сначала проверялась частная папка с реализациями, специфичными для платформы Мирера, затем – общая папка с обобщёнными пайплайнами. Такая организация позволяла переиспользовать общие пайплайны и при необходимости перекрывать их частными версиями без изменений в ядре маршрутизатора, что упрощало развитие функциональности и обеспечивало преемственность между поколениями реализаций.

2.3 Локальная обработка вложений

Образовательные задачи в платформе Мирера предполагают работу с разнообразными прикрепленными материалами: текстами условий заданий и решений студентов в форматах PDF и DOCX, презентациями преподавателей в PPTX, изображениями с рукописными решениями математических и инженерных задач, скан-копиями конспектов. В исходной архитектуре обработка этих вложений выполнялась локально внутри агента: компонент ContentProcessor объединял несколько библиотек парсинга документов и предоставлял пайплайнам уже подготовленный текст в унифицированном виде. Центральное место среди них занимала библиотека Docling [8] – специализированная система IBM, сочетающая нейросетевую модель сегментации макета страницы PP-DocLayoutV3 и сетевой компонент извлечения структурированного текста, способная обрабатывать сложные документы с таблицами, формулами и многоколоночной разметкой.

Подобный подход обеспечивал простоту первоначальной реализации и единый интерфейс для пайплайнов, но быстро выявил ряд недостатков. Дело в том, что сетевая составляющая библиотеки Docling требовала поддержки графического ускорителя, что существенно увеличивало время инициализации агента – на стандартной конфигурации старт занимал несколько десятков секунд из-за необходимости загрузки моделей сегментации в видеопамять. Это снижало гибкость развёртывания: даже в сценариях, в которых обработка вложений не предполагалась, агент по-прежнему резервировал ресурсы графического ускорителя для библиотеки распознавания. Любая модификация или обновление парсера требовали полной пересборки и перезапуска монолита, тогда как смена версии языковой модели в общем случае не должна была затрагивать обработку вложений и наоборот. Наконец, было невозможно независимо масштабировать ресурсы под задачи распознавания и под инференс языковых моделей: при росте потока вложений приходилось добавлять графические ускорители именно агенту, при том что нагрузки на языковые модели в это время могло не быть, и наоборот.

2.4 Ограничения исходной архитектуры

Анализ эксплуатации описанной архитектуры позволил выделить три ключевых ограничения, препятствующих подключению моделей иной модальности и развитию системы

в целом.

Во-первых, цепочки обработки и формирование ответа жёстко предполагали, что в конце пайплайна стоит языковая модель, а её выходное сообщение – текст определённой структуры. Для каждой команды в коде маршрутизатора присутствовала специфичная обёртка, извлекавшая отдельные поля результата работы LLM и собиравшая из них ответ платформе. Это нарушало принцип разделения обязанностей: код ядра маршрутизатора оказывался зависим от деталей конкретных пайплайнов, и любое изменение формата ответа конкретной образовательной задачи приводило к необходимости править ядро. Модель иной модальности, например, система оптического распознавания, возвращающая разметку документа, потребовала бы добавления собственного варианта обёртки и условной логики выбора способа сборки ответа в зависимости от типа сервиса, что лишь усугубило бы существующую связанность кода.

Во-вторых, в реестре сервис-фабрик был зарегистрирован единственный сервис, и сам интерфейс между маршрутизатором и сервис-менеджером не был отделён от специфики языковых моделей, в его сигнатурах фигурировали понятия, осмысленные только для LLM: параметры генерации, длина контекстного окна, обработка системного промпта. Добавление нового типа сервиса потребовало бы либо повторения этой специфики в новом классе с фиктивными значениями полей, не имеющих смысла для моделей иной модальности, либо более глубокой переработки маршрутизатора и сопутствующих интерфейсов с распространением изменений на все существующие пайплайны.

В-третьих, инкорпорация обработки документов в код самого агента создавала жёсткую связь между планированием ресурсов под нейросетевую модуль и под распознавание документов, описанную в разделе 2.3. К этому добавлялась организационная сложность: обновление компонентов распознавания требовало координации между разработчиками нейросетевого модуля и специалистами, поддерживающими инфраструктуру распознавания, при том, что эти подсистемы концептуально решают независимые задачи.

Совокупность перечисленных ограничений делала развитие платформы в сторону мультимодальной обработки запросов нецелесообразным без существенной архитектурной переработки. В следующем разделе описано, какие принципы были положены в основу новой архитектуры, устраняющей эти ограничения.

3. Универсальная мультимодальная архитектура

3.1 Принцип разделения маршрутизации и работы с моделью

В основу переработанной архитектуры положен принцип строгого разделения обязанностей между маршрутизатором запросов и сервис-менеджерами конкретных моделей. Маршрутизатор отвечает только за приём сообщений, валидацию их структуры, маршрутизацию по очередям сервисов и сборку ответа по контракту команды, он не содержит информации о том, какие именно типы моделей подключены к платформе и каким образом устроена работа с ними. Сервис-менеджеры, в свою очередь, инкапсулируют всю специфику работы с моделями своего типа: от управления набором экземпляров до особенностей форматирования входных и выходных данных.

Связующим элементом этих компонентов является класс `BaseServiceManager`, определяющий единый контракт сервис-менеджера через четыре обязательных асинхронных метода: метод поиска свободного экземпляра модели для указанного пайплайна, метод резервирования модели для эксклюзивной обработки запроса, метод освобождения модели после завершения обработки, метод выполнения полной цепочки «препроцессинг, инференс, постпроцессинг» в рамках конкретного пайплайна. Дополнительно класс предоставляет реализацию метода сбора прикладных метрик по умолчанию, что обеспечивает однородный мониторинг любого подключённого сервиса без необходимости дублирования телеметрической инфраструктуры в каждом производном классе. Сигнатуры методов сформулированы в терминах, нейтральных по отношению к модальности модели: вместо специфичных для языковых моделей параметров (длина контекстного окна, параметры генерации, обработка системного промпта) в них фигурируют обобщённые понятия – идентификатор пайплайна, словарь полей запроса, словарь именованных артефактов на выходе.

Подключение нового типа модели сводится к реализации этого контракта в производном классе и регистрации фабрики сервиса в реестре маршрутизатора. Никакие изменения в логике приёма запросов, маршрутизации, обработки ошибок или сборки ответов при этом не требуются. Реестр сервис-фабрик представляет собой словарь, отображающий имя сервиса в соответствующий класс, маршрутизатор при старте инициализирует по

экземпляру каждого зарегистрированного сервиса и запускает для них отдельные циклы-потребители сообщений на соответствующих очередях. Дополнительный практический эффект подобного разделения – упрощение модульного тестирования: каждый сервис-менеджер может проверяться в изоляции от остальных компонентов системы с использованием макетных реализаций моделей.

3.2 Конфигурируемые цепочки обработки

Описание того, через какие сервисы и пайплайны проходит конкретная команда, вынесено из кода в конфигурационные файлы. Для каждой команды цифровой образовательной платформы определены два файла: описание цепочки шагов и контракт взаимодействия с платформой Мирера.

Файл цепочки задаёт последовательность шагов с явным указанием сервиса, имени пайплайна и правил преобразования полей между соседними шагами. Преобразование может включать переименование переменных, чтение значений из объектного хранилища S3 по указанному пути и обратное сохранение результатов в S3 – это особенно важно для крупных артефактов (текстов транскрипций, образов документов, аудиофайлов), передавать которые целиком в сообщениях нецелесообразно из-за ограничений брокера на размер сообщения. Пути в хранилище задаются относительно общего префикса, привязанного к идентификатору запроса, что обеспечивает изоляцию данных разных запросов друг от друга и упрощает их утилизацию после завершения обработки. Пример сложной цепочки – команда транскрипции и резюмирования аудиозаписи занятия: первый шаг обрабатывает в сервисе распознавания речи и формирует выходное поле с транскрибированным текстом, второй шаг – в сервисе языковых моделей с пайплайном суммаризации, для которого это поле автоматически переименовывается во входное и подаётся как контекст для генерации обобщения.

Файл контракта описывает, какие поля платформа Мирера обязана передать при запросе и каким образом из артефактов цепочки собирается успешный ответ либо ответ об ошибке. Маршрутизатор не содержит зашитых в код шаблонов ответов: вся семантика выходного сообщения определяется контрактом, что позволяет добавлять новые команды и менять формат ответа без модификации кода ядра. Это особенно важно для согласования версий протокола с платформой Мирера:

эволюция формата сообщений происходит на уровне конфигурации и допускает одновременную поддержку нескольких версий контракта одной и той же команды, что критически важно для обеспечения совместимости при поэтапном развёртывании обновлений.

3.3 Унифицированный поток выходных артефактов

Одной из ключевых переработок при переходе к универсальной архитектуре стало устранение специфичных для каждого пайплайна обёрток выходного потока. В исходной реализации каждый пайплайн самостоятельно формировал структуру выходного сообщения для платформы Мирера, что приводило к дублированию логики сборки ответа: одни и те же преобразования полей повторялись в коде разных пайплайнов, а маршрутизатор оказывался привязан к деталям именования и формата результатов конкретных пайплайнов. При появлении нового пайплайна разработчику требовалось не только реализовать собственно логику обработки, но и продублировать инфраструктурный код сборки сообщения.

В новой архитектуре любой сервис, будь то языковая модель, система оптического распознавания или перспективные сервисы распознавания и синтеза речи, возвращает результат в виде единообразного словаря с именованными полями (артефактами). Преобразование этих артефактов в сообщение для следующего шага цепочки или в финальный ответ платформе выполняется единым компонентом – конфигурируемым трансформером сообщений, который читает правила преобразования из описания цепочки. Трансформер поддерживает переименование полей, чтение и запись артефактов в объектное хранилище S3, простые преобразования типов и подстановку значений по умолчанию. Это обеспечивает однотипность поведения маршрутизатора для всех типов сервисов и позволяет, в частности, использовать в одной цепочке артефакты как от пайплайна языковой модели, так и от пайплайна распознавания изображений, не прибегая к специальным склейкам кода. Дополнительный практический эффект – упрощение модульного тестирования пайплайнов: их корректность проверяется по соответствию словаря возвращаемых артефактов схеме, без необходимости в каждом тесте конструировать целевой формат выходного сообщения для конкретной команды.

3.4 Пайплайн как самодостаточный модуль

Каждый пайплайн оформляется в виде отдельной директории, содержащей минимально необходимый набор файлов: строгую схему запроса и ответа на основе декларативного описания структур данных, реализацию логики выполнения, шаблоны формирующих запрос инструкций (для пайплайнов языковых моделей) или конфигурацию обращений к сервису (для пайплайнов иной модальности). Точкой входа в пайплайн служит асинхронная функция, принимающая валидированный запрос, экземпляр модели и ссылку на сервис-менеджер, возвращаемое значение представляет собой словарь именованных артефактов, который в дальнейшем обрабатывается трансформером сообщений в соответствии с описанием шага цепочки. Использование строгой типизации схем входа и выхода исключает возможность неявных ошибок интеграции между шагами и обеспечивает быстрое обнаружение несоответствий ещё на этапе валидации запроса.

Подобная организация поддерживает иерархический поиск пайплайнов «частное – общее»: при наличии в системе реализации, специфичной для платформы Мирера, она имеет приоритет, при её отсутствии используется обобщённая реализация из общего каталога. Это позволяет переиспользовать общие пайплайны (например, базовый пайплайн транскрипции аудиозаписи) в специфичных для Мирера цепочках без копирования кода и одновременно изолировать частную бизнес-логику – особенности форматирования заданий для конкретных курсов, специфичные ограничения на длину ответа и дополнительные постфильтры в рамках частных пайплайнов, не загромождая ими общий каталог.

С организационной точки зрения такая структура существенно упрощает развитие функциональности: добавление новой образовательной задачи сводится к созданию новой директории пайплайна и регистрации соответствующей команды в реестре цепочек и контрактов. Никакие изменения в коде маршрутизатора или сервис-менеджеров для этого не требуются, что снижает риск нежелательных побочных эффектов при релизе новой функциональности и сокращает срок ввода новых образовательных сценариев в эксплуатацию.

4. Дополнительные особенности новой архитектуры

4.1 Поддержка специализированных типов вложений

Унификация работы с вложениями в рамках компонента `ContentProcessor` сделала возможным быстрое добавление поддержки новых типов учебных материалов. В частности, реализована обработка вложений в формате `LaTeX`, что особенно важно для образовательных курсов математического, физического и инженерного профиля, где условия задач, решения студентов и методические материалы преподавателей зачастую записываются на этом языке разметки. Поскольку файл `LaTeX` представляет собой структурированный текстовый документ с фиксированной кодировкой, его обработка не требует обращения к подсервису распознавания: содержимое декодируется сразу и передаётся в пайплайн напрямую. Аналогичный быстрый путь предусмотрен и для простых текстовых вложений с расширениями `.txt` и т.д.

Интерфейс работы со всеми типами вложений остаётся единым: пайплайн получает уже подготовленное текстовое представление документа независимо от того, был ли исходный файл обработан удалённым сабсервисом распознавания, локальным резервным парсером или прямым декодированием. Это упрощает написание пайплайнов, способных принимать материалы в произвольных форматах, и снимает с разработчика пайплайна необходимость учитывать особенности каждого формата отдельно. Добавление поддержки нового типа вложений сводится к описанию способа извлечения текста и регистрации новой ветви в карте обработчиков расширений в составе `ContentProcessor`, что упрощает развитие системы по мере появления новых типов учебных материалов, например, исходного кода в архивах различных языков программирования, формы интерактивных заданий или вложений со специализированной разметкой.

4.2 Классификация и обработка ошибок

В исходной архитектуре каждый пайплайн самостоятельно отвечал за перехват и интерпретацию ошибок инференса, что приводило к дублированию кода и неоднородной семантике ответов: одни и те же по природе ошибки могли быть классифицированы по-разному в разных пайплайнах, что усложняло наблюдаемость и поддержку системы. В новой архитектуре принят принцип «пайплайны без перехвата исключений»: возникающие в ходе

исполнения исключения прокидываются до маршрутизатора, который классифицирует их через специальный компонент-обработчик ошибок и принимает решение о реакции системы.

Ошибки делятся на две основные категории. Транзиентные ошибки (превышение времени ожидания при обращении к модели, потеря соединения с моделью, временная недоступность сабсервиса, отсутствие свободного экземпляра модели в пуле) приводят к возврату сообщения в очередь для повторной обработки – с учётом ограничения общего числа попыток и нарастающей задержки между ними. Постоянные ошибки (нарушение схемы запроса, отсутствие требуемого пайплайна, фатальная ошибка инференса, неверная конфигурация цепочки) переводят запрос в состояние неуспешной обработки: маршрутизатор формирует ответ об ошибке по контракту команды, обогащая его кодом из каталога постоянных ошибок. Каталог постоянных ошибок ведётся как часть конфигурации и описывает каждую категорию ошибки с её кодом, кратким описанием для целей логирования и пользовательским сообщением, отображаемым в платформе Мирера. Подобная классификация унифицирована для всех типов сервисов: сервис языковых моделей, сабсервис распознавания и перспективные сервисы распознавания и синтеза речи используют один и тот же механизм категоризации, что обеспечивает однородность реакции системы на ошибочные ситуации независимо от модальности обрабатываемых данных и существенно упрощает работу команды эксплуатации по диагностике инцидентов.

4.3. Единые метрики и наблюдаемость

Реализация метода сбора метрик непосредственно в базовом классе `BaseServiceManager` обеспечивает однородный сбор телеметрических данных любого подключаемого сервиса. Метрики экспортируются в систему мониторинга Prometheus [9] через стандартный эндпоинт публикации и включают: счётчики принятых и завершённых запросов с разбивкой по пайплайнам, гистограммы времени обработки с настраиваемыми границами интервалов для гранулярного анализа задержек, индикаторы текущего состояния пула моделей (число занятых и свободных экземпляров каждой модели), счётчики ошибок с категоризацией по типу. На стороне сервиса перечисленные метрики обновляются автоматически в рамках стандартных операций

сервис-менеджера и не требуют от разработчика конкретного производного класса какого-либо дополнительного кода.

Поскольку набор метрик и их семантика идентичны для всех сервисов, мониторинг и алертинг настраиваются однократно и распространяются на любые будущие сервисы – это снимает с разработчика нового сервис-менеджера задачу проектирования собственной телеметрии и существенно упрощает работу команды эксплуатации: однотипные информационные панели Grafana используются для всех сервисов, а пороговые значения алертов формулируются в терминах, не зависящих от типа модели. Дополнительный практический эффект – возможность сквозного отслеживания обработки запроса по всей цепочке шагов, в том числе пересекающих границы сервисов: единые идентификаторы запроса и согласованные семантики временных метрик позволяют выстраивать целостные временные диаграммы обработки и быстро локализовывать узкие места в составных цепочках обработки.

4.4 Готовность архитектуры к подключению сервисов речи

В рамках переработки заложены каркасы для будущих сервисов распознавания и синтеза речи, потребность в которых обусловлена развитием в платформе Мирера лекционных и семинарских форматов обучения, а также параллельной работой над цифровым двойником преподавателя [4], уже использующим компоненты распознавания и синтеза речи в составе самостоятельной системы. В реестре сервис-фабрик предусмотрены соответствующие точки подключения для сервисов распознавания (ASR) и синтеза (TTS), в конфигурации цепочек обработки уже описаны примеры сложных мультимодальных цепочек, в частности, цепочка «транскрипция аудиозаписи занятия - суммаризация полученного текста языковой моделью» и более сложная цепочка «транскрипция - суммаризация - синтез речи на основе обобщённого текста», объединяющая три сервиса разных модальностей в единой последовательности обработки. Соответствующие правила преобразования полей между шагами и работы с объектным хранилищем для крупных аудио и текстовых артефактов уже сформулированы, согласованы и протестированы на макетных реализациях. На текущем этапе сервис распознавания речи и сервис синтеза речи реализованы в виде интерфейсных заглушек, корректно выполняющих контракт BaseServiceManager для целей интеграционного тестирования и проверки целостности цепочек, но возвращающих маркер нереализованности

при попытке фактической обработки запроса. После замены заглушек на полноценные реализации соответствующих моделей подключение нового типа сервиса произойдёт без изменений в коде маршрутизатора и сервис-менеджеров других модальностей, что и является основным практическим результатом проведённой архитектурной эволюции и подтверждением жизнеспособности предложенного подхода.

5. Заключение

В работе описана эволюция архитектуры нейросетевого модуля платформы Мирера, обеспечивающая универсальную поддержку мультимодальных сервисов ИИ. Показано, что исходное LLM-ориентированное ядро имело принципиальные ограничения.

Предложенная архитектура устраняет эти ограничения за счёт взаимосвязанных решений: абстракции с единым контрактом для сервисов любой модальности, вынесения цепочек обработки и контракта взаимодействия с платформой в конфигурационные файлы, унификации потока выходных артефактов с их преобразованием через конфигурируемый трансформер.

В дальнейшем планируется расширение поддерживаемых модальностей и набора сценариев с новыми моделями, а также улучшение текущей архитектуры и ее оптимизация.

Работа выполнена в рамках государственного задания НИЦ «Курчатовский институт» - НИИСИ по теме № FNEF-2024-0001 (1023032100070-3-1.2. 1).

Evolution of the neural network module architecture: supporting multimodal services in the Mirera digital educational platform

A. S. Karavaeva¹, A. M. Kulikov², N. S. Martynov³, K. A. Mashchenko⁴,
K. K. Pchelin⁵, E. D. Tarasuk⁶

¹National Research Center «Kurchatov Institute»- SRISA, Moscow, Russia, aleksandrakaravaeva@yandex.ru;

²National Research Center «Kurchatov Institute»- SRISA, Moscow, Russia, kulikrch@yandex.ru;

³National Research Center «Kurchatov Institute» - SRISA, Moscow, Russia, nikolai.martynov@math.msu.ru;

⁴National Research Center «Kurchatov Institute» - SRISA, Moscow, Russia; Lomonosov Moscow State University, Moscow, Russia;
State University of Management, Moscow, Russia, kirill.mashchenko@niisi.ru;

⁵National Research Center «Kurchatov Institute» - SRISA, Moscow, Russia, pcelinkosta0@gmail.com;

⁶National Research Center «Kurchatov Institute» - SRISA, Moscow, Russia, ekaterina.tarasuk@math.msu.ru;

Abstract. This paper examines the evolution of the neural network module architecture of the Mirera Digital Educational Platform in connection with the expanding range of applied artificial intelligence models. It is shown that the original LLM-oriented orchestrator core did not allow models of other modalities to be integrated without substantial reworking. A universal multimodal architecture is described, including a service manager abstraction based on a unified contract, configurable request-processing chains, and a unified stream of output artifacts. Examples of integrations are presented. It is shown that the proposed architecture enables the integration of new model types without modifying the router core and provides a basis for the further development of multimodal educational scenarios.

Keywords: digital educational platform, DEP Mirera, neural network orchestrator, multimodal models, microservice architecture, LLM, OCR, RabbitMQ, AI agents, educational technologies

Литература

1. Chu Z., Wang S., Xie J., Zhu T., Yan Y., Ye J., Zhong A., Hu X., Liang J., Yu P. S., Wen Q. LLM agents for education: Advances and applications // arXiv preprint arXiv:2503.11733. - 2025. URL: <https://arxiv.org/pdf/2503.11733> (дата обращения: 02.07.2026).
2. Платформа Мирера [Электронный ресурс]. - URL: <https://www.mirera.ru> (дата обращения: 02.07.2026).
3. Леонов А. Г., Мартынов Н. С., Машенко К. А., Паремузов М. С., Пчелин К. К., Шляхов А. В. Области применения больших языковых моделей для цифровых образовательных платформ // Труды НИИСИ. - 2025. - Т. 15, № 2. - С. 9-15. URL: <https://www.t-niisi.ru/jour/article/view/72/120> (дата обращения: 02.07.2026).
4. Кушниренко А. Г., Леонов А. Г., Машенко К. А., Мартынов Н. С., Пчелин К. К., Шляхов А. В. Интеграция нейросетевого ассистента обучающихся по программированию в цифровую образовательную платформу // Информатика и образование. - 2026. - Т. 41, № 1. - С. 23-36. URL: <https://info.infojournal.ru/jour/article/view/1426/783> (дата обращения: 02.07.2026).
5. Леонов А. Г., Машенко К. А., Мартынов Н. С., Пчелин К. К., Шляхов А. В. Автоматизированное формирование образовательных видеоматериалов на основе модульной системы цифрового двойника преподавателя // Программные продукты и системы. - 2025. - Т. 38, № 4. - С. 694-703. URL: <https://swsys.ru/files/2025-4/694-703.pdf> (дата обращения: 02.07.2026).
6. Гриднев Н. В., Караваева А. С., Леонов А. Г., Машенко К. А., Пчелин К. К., Тарасюк Е. Д. Гибридная модель обучения: новации в учёте посещаемости в цифровой образовательной платформе Мирера // Труды НИИСИ. - 2025. - Т. 15, № 1. - С. 58-64. URL: <https://www.t-niisi.ru/jour/article/view/10/20> (дата обращения: 02.07.2026).
7. Vinoski S. Advanced Message Queuing Protocol // IEEE Internet Computing. - 2006. - Vol. 10, No. 6. - P. 87-89. - DOI: 10.1109/MIC.2006.116.
8. Auer C., Lysak M., Nassar A., Dolfi M., Livathinos N., Vagenas P., Berrospi C., Omenetti M.,

Lindlbauer F., Dinkla K., Mishra L., Kim Y., Gupta S., de Lima R. T., Weber V., Morin L., Meijer I., Kuropiatnyk V., Staar P. W. Docling Technical Report // arXiv preprint arXiv:2408.09869. - 2024. URL: <https://arxiv.org/pdf/2408.09869> (дата обращения: 02.07.2026).

9. Brazil B. Prometheus: Up & Running: Infrastructure and Application Performance Monitoring. - 2nd ed. - O'Reilly Media, 2024. - 506 p.

Поступила в редакцию / Received: 25.06.2026.

Поступила после рецензирования / Revised: 30.06.2026.

Принята к печати / Accepted: 03.07.2026.