СТАТЬИ

УДК 004.42
DOI 10.17513/snt.40384

КОНТЕКСТНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА ИНТЕГРАЦИИ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА НА ОСНОВЕ СЛОЯ АБСТРАКТНЫХ ОПЕРАТОРОВ

Алпатов А.Н.

ФГБОУ ВО «МИРЭА – Российский технологический университет», Москва, e-mail: aleksej01-91@mail.ru

Быстрое развитие технологий искусственного интеллекта, а также широкое распространение облачных вычислений стимулируют внедрение интеллектуальных модулей в различные сферы. Однако включение систем искусственного интеллекта в состав распределённых программных систем сопровождается существенными затруднениями, связанными с отсутствием унифицированных подходов к их интеграции. Целью данной работы является разработка контекстно-ориентированной архитектуры интеграции систем искусственного интеллекта, обеспечивающей снижение сложности сопряжения интеллектуальных модулей с прикладной логикой распределённых программных систем. В работе проведён анализ существующих архитектурных решений, формализована модель интеграции с использованием отображений между задачами, операциями и моделями, а также предложен механизм описания задач в виде операторного графа. Также предложен подход к построению интеграционной архитектуры для систем искусственного интеллекта, основанный на использовании программного шлюза с поддержкой контекстно-ориентированного взаимодействия. Основным ключевым элементом решения является абстрактный слой операторов, который позволяет отделить описание задачи от конкретных моделей и тем самым существенно снизить связность системы. Предложенное решение ориентировано на поддержку моделей различных типов (языковых, визуальных и других) и обеспечивает прозрачную маршрутизацию запросов и возможность масштабирования в условиях распределённой вычислительной среды. В работе отдельно отмечено, что достижение соответствующих характеристик подхода возможно при реализации стандартизированного набора операторов. Представленный подход обеспечивает требуемую масштабируемость архитектуры интеграции систем искусственного интеллекта и может служить основой для построения универсальных интеграционных решений.

Ключевые слова: искусственный интеллект, программный шлюз, контекстное взаимодействие, интеграция систем, распределённая архитектура, семантическая маршрутизация

CONTEXT-ORIENTED ARCHITECTURE FOR THE INTEGRATION OF ARTIFICIAL INTELLIGENCE SYSTEMS BASED ON AN ABSTRACT OPERATOR LAYER

Alpatov A.N.

MIREA – Russian Technological University, Moscow, e-mail: aleksej01-91@mail.ru

The rapid development of artificial intelligence technologies and the widespread use of cloud computing stimulate the introduction of intelligent modules in various spheres. However, the inclusion of artificial intelligence systems in distributed software systems is accompanied by significant difficulties associated with the lack of unified approaches to their integration. The purpose of this paper is to develop a context-oriented architecture for integration of artificial intelligence systems, which reduces the complexity of interfacing intelligent modules with the application logic of distributed software systems. The paper analyzes existing architectural solutions, formalizes the integration model using mappings between tasks, operations and models, and proposes a mechanism for describing tasks in the form of an operator graph. We also propose an approach to building an integration architecture for artificial intelligence systems based on the use of a software gateway with support for context-oriented interaction. The main key element of the solution is an abstract layer of operators, which allows to separate the task description from concrete models and thus significantly reduce the coherence of the system. The proposed solution is oriented to support models of different types (linguistic, visual and others) and provides transparent request routing and scalability in a distributed computing environment. In the paper it is separately noted that the achievement of the corresponding characteristics of the approach is possible with the implementation of a standardized set of operators. The presented approach provides the required scalability of the integration architecture of artificial intelligence systems and can serve as a basis for the construction of universal integration solutions.

Keywords: artificial intelligence, systems integration, software gateway, contextual interaction, distributed architecture, semantic routing

Введение

Бурное развитие технологий обработки данных и широкое распространение облачных вычислений привели к активному внедрению систем искусственного интеллекта в различные сферы деятельности от промышленного производства до образования и медицины. Интеллектуальные модули, реализующие функции классификации, распознавания образов, прогнозирования, становятся неотъемлемой частью современных программных систем. Однако их практическое использование требует эффективной интеграции с другими

компонентами программной архитектуры, включая хранилища данных, интерфейсы прикладного программирования и модули принятия решений. На сегодняшний день отсутствует единый общепринятый подход к интеграции систем искусственного интеллекта в распределённые программные комплексы, особенно в условиях гетерогенной программной среды и высокой степени распределённости компонентов. Это приводит к необходимости разработки собственных промежуточных решений, обеспечивающих связность между интеллектуальными подсистемами и остальной инфраструктурой.

Интеграция систем искусственного интеллекта (ИИ) в существующие архитектуры программного обеспечения представляет собой ряд технических проблем, особенно при работе с распределёнными средами и разнообразными компонентами системы. Например, типичным сценарием, при котором возникает необходимость в интеграции интеллектуальных моделей, является добавление ранее разработанной функциональности, основанной на модели искусственного интеллекта, в новую функциональность, написанную совершенно на другом стеке технологий. На практике, примером такой задачи может быть такая ситуация, когда модель рекомендаций, обученная в PyTorch, должна быть интегрирована в платформу электронной коммерции на базе Java с использованием REST API и последовательной сериализации данных. Поэтому одним из основных препятствий в этой области является проблема совместимости данных между гетерогенными системами. Существующая инфраструктура часто использует разные форматы данных, структуры и методологии хранения, которые должны быть согласованы с моделями искусственного интеллекта, что подтверждается рядом работ. Так авторами S. Sinha и Y.M. Lee подчёркивается проблематика интеграции искусственного интеллекта в существующую инфраструктуру, включая необходимость гармонизации различных форматов данных и структур хранения [1]. В работе P.M.A. van Ooijen и соавторов рассмотрены технические аспекты построения ИИ-пайплайнов, включая вопросы хранения данных, использования облачных инфраструктур и структурирования процессов обработки [2]. Авторы подчёркивают важность чёткого проектирования ИИ-конвейеров и отмечают, что в распределённых условиях высокая степень связности между компонентами может стать узким местом при масштабировании. Однако представленный ими подход ориентирован преимущественно на линейные сценарии исполнения и не предусматривает явного механизма абстрагирования задач от конкретных моделей.

Проблема интеграции становится особенно острой, когда организациям необходимо подключить несколько источников данных. Так в последних исследованиях и опросах, проведённых компанией Tray.ai показано, что примерно около 42% предприятий нуждаются в доступе к восьми или более источникам данных для успешного развёртывания агентов искусственного интеллекта. В работе M. Zhao и соавторов рассмотрен вопрос реализации конвейера хранения и обработки данных, состоящего из центрального хранилища данных, построенного на распределенном хранении, который позволяет сотням моделям обучаться совместно через территориально-распределённые дата-центры [3]. В рамках необходимости соблюдения стандартов представления данных, возникают сложности с тем, что многие организации работают с устаревшими системами, построенными на устаревших технологических стеках, языках программирования или операционных системах, которые несовместимы с современными фреймворками и инструментами в области искусственного интеллекта. Это несоответствие между устоявшейся инфраструктурой и передовыми приложениями систем искусственного интеллекта добавляет значительную сложность интеграции

Но помимо проблемы совместимости данных между системами, существенным препятствием к интеграции является также качество самих данных. Как известно, модели искусственного интеллекта в значительной степени зависят от высококачественных данных для процессов обучения и формирования вывода и основными сложностями в данной части являются несоответствия, неточности и пропущенные значения в наборах данных, что может существенно снизить производительность систем, основанных на искусственном интеллекте [4,5]. Поэтому при решении проблемы интеграции должны быть учтены и разработаны соответствующие политики управления данными и реализованы общие протоколы очистки данных, для того чтобы гарантировать целостность информации, поступающей в системы [6].

Таким образом, можно констатировать, что существенным препятствием для интеграции систем искусственного интеллекта является отсутствие общепринятых стандартов интеграции, что приводит к непоследовательным подходам в организациях,

даже даже в пределах одного предприятия или даже проекта. Без общих стандартов организации должны разрабатывать собственные подходы, что часто приводит к потенциальным пробелам в реализации, что говорит об актуальности данной задачи.

Цель исследования – разработка контекстно-ориентированной архитектуры интеграции систем искусственного интеллекта, направленной на снижение сложности сопряжения интеллектуальных модулей с прикладной логикой распределённых программных систем. В основе предлагаемого подхода лежит использование промежуточного слоя абстрактных операторов и стандартизированных механизмов отображения задач на модели, независимых от конкретных поставщиков решений.

Материалы и методы исследования

Исследование выполнено в форме теоретико-прикладного анализа существующих подходов к интеграции систем искусственного интеллекта. В качестве материала использованы архитектурные схемы микросервисных систем, документы форматов YAML/JSON, описания моделей различной природы (языковые, визуальные и др.). Методологической основой послужили принципы декомпозиции, архитектурного моделирования и абстракции, а также положения, изложенные в работах из списка литературы.

Архитектуру и механизмы отображения задач в модели планируется реализуются в рамках модульного фреймворка, использующего стек технологий REST/gRPC, а также унифицированные контракты взаимодействия на основе JSON Schema и Protocol Buffers. В качестве целевой платформы рассматривается реализация на языке Python с использованием библиотек FastAPI, Pydantic и Docker-контейнеризации для изоляции моделей. Особое внимание уделяется возможности масштабирования и повторного использования операторных блоков при формировании задач в виде графов обработки.

Результаты исследования и их обсуждение

Прежде чем перейдём к рассмотрению решения обозначенной выше проблемы, сформулируем и формализуем задачу интеграции систем искусственного интеллекта. Пусть имеется множество моделей искусственного интеллекта, которые могут описаны как $M = \{M_1 M_2, ..., M_n\}$, а также множество типов задач (или прикладных целей использования моделей) $T = \{T_1, T_2, ..., T_M\}$. Каждая модель из множества M реализует отображение вида

$$M_i : X_i \to Y_i, \quad (1)$$

где X_i – множество допустимых входных данных (изображения, текст, аудио),

Y_i – множество возможных выходных данных (текст, классы, эмбеддинги и т.д.).

А каждая задача из множества T, в свою очередь, требует выполнения некоторого преобразования вида

$$T_j : D_j \to R_j, \quad (2)$$

где D_j – множество исходных данных задачи,

R_j – множество желаемых результатов.

Следовательно, для того чтобы модель M_i могла участвовать в решении задачи T_j, необходимо определить связку между моделью и задачей. Это связь и будет являться интеграцией модели в новую задачу. Все возможные такие связки (пары M_i и T_j) можно представить как отношение интеграции, которое можно записать в виде

$$I = M \times T = \{M_i, T_j \mid M_i \in M, T_i \in T\}. \quad (3)$$

Каждая модель M_i представляется как расширенный конечный автомат, который может быть описан кортежем

$$A_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_{0_i}, F_i, D_i), \quad (4)$$

где Q_i – множество состояний

Σ_i – входной алфавит (события, запросы)

Γ_i – выходной алфавит

δ_i – функция перехода в зависимости от входа D_i

q_{0i} – начальное состояние автомата (выполнение модели)

F_i – множество конечных состояний

D_i – набор внутренних переменных (память, флаги, контекст и т.д.).

Тогда для каждой задачи T_j и модели M_i, логика перехода δ_i должна включать индивидуальные правила, определённые как

$$\delta_i : (q, \sigma, D_i) \xrightarrow{T_j} q'.$$

Таким образом модель должна иметь в своей логике переходов δ_i отдельные правила для каждой задачи T_j. В свою очередь, для каждой пары отображения (M_i,T_j), чтобы осуществить взаимодействие, также необходимо определить композицию преобразований вида

$$\varphi_{j \to i} : D_j \to X_i, \quad (5)$$

$$\psi_{\mathrm{i} \to \mathrm{j}} : Y_i \to R_j. \quad (6)$$

Таким образом, полная интеграция пары требует создания отображения

$$F_{ij} = \psi_{i \to j} \circ M_i \circ \varphi_{j \to i} : D_j \to R_j, \quad (7)$$

Формула (7) описывает, как задача из пространства D_j (данные задачи j) преобразуется в результат R_j (ожидаемый формат результата для задачи j) с помощью модели M_i, предназначенной для решения задач другого типа. Так как модель M_i напрямую не совместима с форматом задачи j, то используются два дополнительных преобразования, а именно транслятор φ и интерпретатор ψ. Иными словами, сначала к данным $d \in D_j$ применяется транслятор φ, с целью преобразования входа задачи в формат, понятный модели, далее результат подаётся на вход модели M_i, после чего результат модели интерпретируется соответствующим интерпретатором, то есть происходит преобразование результата модели в формат, ожидаемый задачей. Таким образом, одна универсальная модель может применяться к задачам разных типов с помощью обёрток-преобразователей, тем самым определена прямая интеграция каждой модели с каждой задачей. Однако такая прямая интеграция модели M_i с задачей T_j требует, чтобы либо сама модель изначально была обучена на задаче данного типа, либо чтобы внешние преобразования $\psi_{i \to j}$ и $\varphi_{j \to i}$ были достаточно сложны, чтобы компенсировать отсутствие у модели знаний о специфике задачи. Это означает, что в случае, если модель M_i не была заранее обучена учитывать особенности задачи, то все требования к адаптации полностью ложатся на внешнюю инфраструктуру интеграции, что в обязательном порядке требуется учитывать при реализации систем и протоколов интеграции систем искусственного интеллекта. Особенно это критично при повторном использовании одной и той же модели M_i в новой задаче T_{j+1}, с которой она ранее не взаимодействовала. Здесь же стоит отметить, что в данной работе, по умолчанию предполагается, что интеграция всегда требует явных адаптационных преобразований. В частном случае, если модель M_i изначально была обучена на задаче T_j, и входные и выходные форматы полностью совпадают с требованиями задачи, отображения φ_j^i и ψ_i^j могут быть реализованы как тождественные функции. В этом случае интеграция упрощается до прямого вызова модели, то есть $F_{ij} = M_i$.

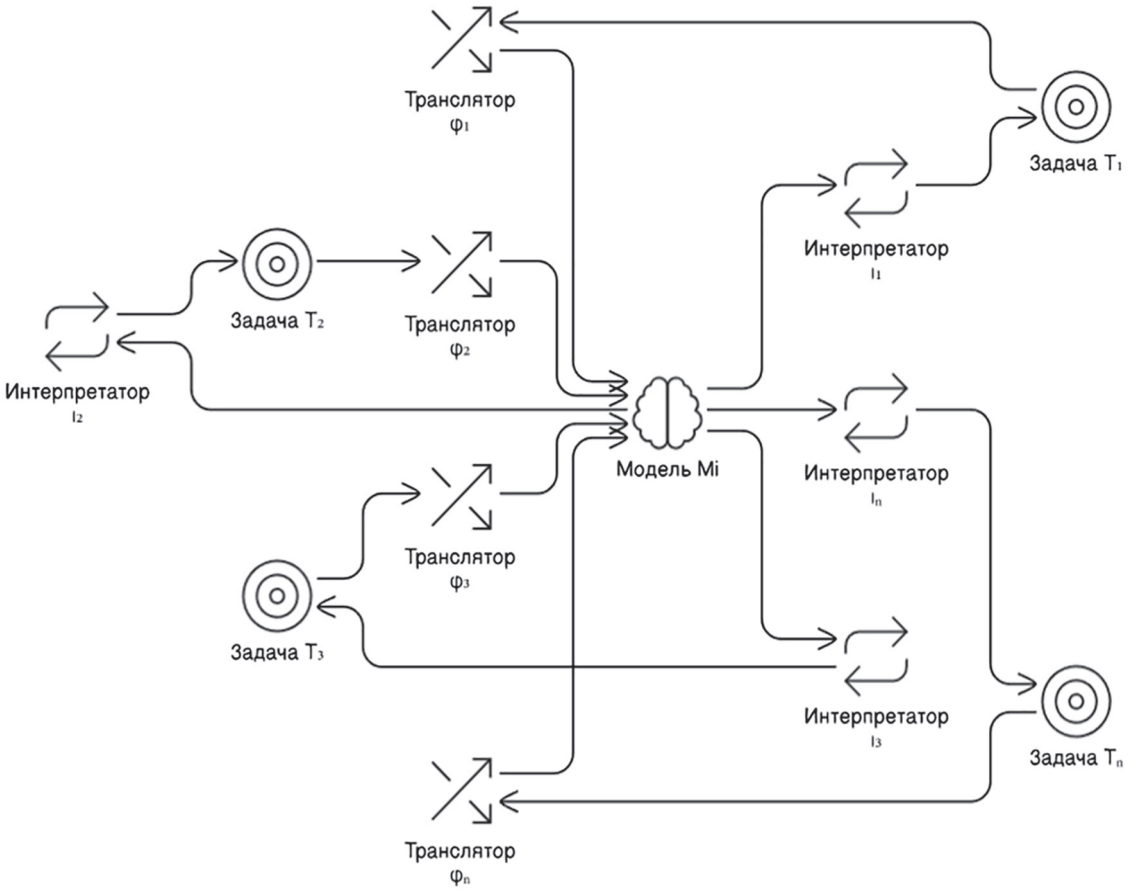

Рис. 1. Интеграция универсальной модели ИИ с множеством задач через индивидуальные преобразователи
Источник: составлен автором

На рисунке 1 показана схема использования модели M_i для новых классов задач при описанном выше подходе к интеграции.

Даже если сама модель остаётся неизменной, для корректного включения её в новую прикладную цель, фактически, необходимо спроектировать новую пару транслятор-интерпретатор, что при большом количестве используемых моделей усложняет систему. Таким образом, возникает необходимость в архитектурных подходах и протоколах, которые могли бы централизованно управлять взаимодействием моделей.

В последние годы разработано множество таких решений. Одним из наиболее распространённых подходов к интеграции компонентов в системах искусственного интеллекта является использование микросервисной архитектуры, в которой каждая модель развёртывается как отдельный сервис с внешним API, а задачи реализуются как отдельные оркестраторы, направляющие данные в нужные компоненты, посредством использования таких протоколов и архитектурных стилей, как, например, REST (HTTP/JSON) или gRPC (Protocol Buffers). В самой простой реализации такого подхода каждая модель M_i обёрнута в микросервис с endpoint *POST /predict*, который на вход принимает сериализованные данные, а каждая задача T_j реализует собственный клиент, который формирует запросы к нужным микросервисам. Каждый клиент реализует прикладную логику задачи T_j, связанную со сбором и преобразованием исходных данных, а также выбором нужных моделей и дальнейшей агрегацией результатов в единое решение, таким образом, обеспечивается оркестрация и выполнение прикладного сценария. Благодаря такому подходу, в рамках одного вызова задача может последовательно активировать несколько моделей. Типичный цикл обработки прикладной задачи с поступления HTTP запроса на вход оркестратора, который анализирует полученные данные и определяет необходимую последовательность действий, вызывая соответствующую модель. Если для выполнения текущего шага требуется дополнительная модель, то задача формирует запрос к сервису модели M_i. После получения запроса сервис модели выполняет преобразование данных во внутренний формат, запускает выполнение и получает ответ, содержащий результат выполнения и дополнительную информацию. Данный ответ возвращается задаче в виде JSON формата или protobuf-сообщения. Важно отметить, что здесь могут использоваться дополнительные промежуточные слои, отвечающие за трансформацию входных и выходных данных, а также согласования форматов между задачей и моделью. Эти компоненты являются реализацией адаптационных отображений $\psi_{i\to j}$ и $\varphi_{j\to i}$, которые были описаны выше. После всех этих действий, оркестратор задачи принимает результат, выполняет его постобработку и решает, завершена ли задача или требуются дополнительные шаги. На рисунке 2 показана возможная схема реализации данного подхода.

Несмотря на то, что архитектура, показанная на рисунке 2, является модульной и обладает требуемым уровнем прозрачности, такое решение остаётся жёстко связанным, нарушая принцип низкой связанности и высокого зацепления (англ. low coupling – high cohesion) [7]. Это проявляется в том, что для каждой задачи необходимо явно указывать, какие модели вызывать, в каком порядке, с какими параметрами, и как обрабатывать результат, а чтобы интегрировать все возможные взаимодействия между M_i и T_j, необходимо реализовать $S = M \times N$ уникальных интерфейсов, сохраняя при этом локальные контексты внутри каждой модели. Тогда трудоёмкость одной интеграции модели может быть определена как

$$C_{i,j} = c_1 + c_2 + c_3 + c_4, \qquad (8)$$

где c_1 – реализация маршрута вызова,
c_2 – преобразование входа ($\varphi_{j\to i}$),
c_3 – интерпретация выхода ($\psi_{i\to j}$),
c_4 – логика ошибок, логгирование, тестирование и прочее.

А общая трудоёмкость для всей системы может быть рассчитана, как

$$C_{sum} = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{i,j} = \left(c_1 + c_2 + c_3 + c_4\right)\cdot m \cdot n. \qquad (9)$$

На рисунке 3 показана тепловая карта роста сложности системы (количество точек интеграции) по формуле второго порядка $S(m,n) = O(mn)$, полученная с использованием языка Python и Jupyter Notebook.

Как видно из рисунка 3 уже при, например, 5 моделях и 4 задачах необходимо реализовать 20 связей, что на практике будет приводить к возникновению дублированного кода в реализации, а при 20 моделях и 15 задачах необходимо реализовать 300 связей, что приводит к значительной сложности такого решения, которое проявляется в том, что необходимо реализовать и поддерживать большое количество маршрутов, адаптеров, обработчиков ошибок и прочей функциональности, а при попытке добавления новой модели или задачи нужно модифицировать десятки компонентов.

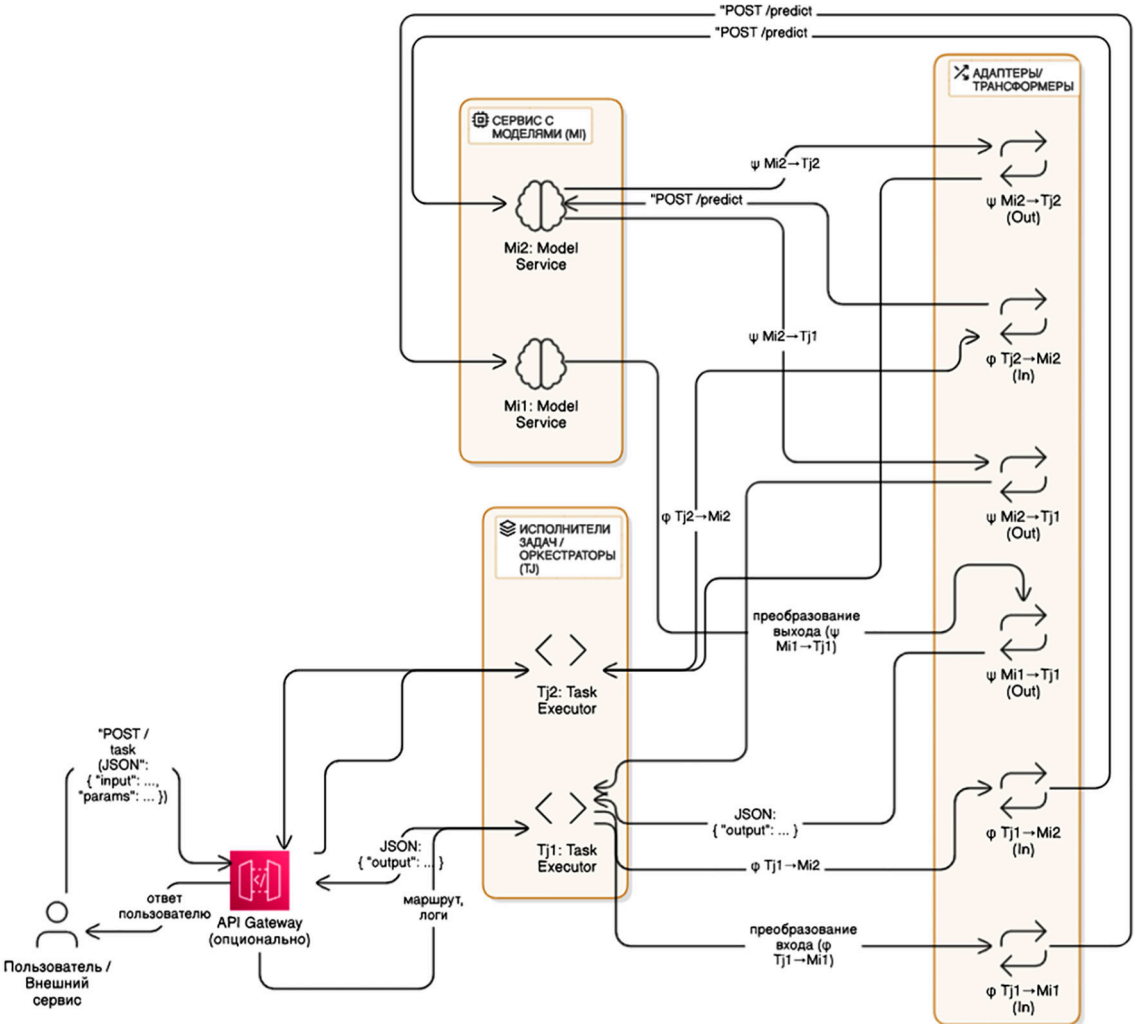

Рис. 2. Микросервисная архитектура с RPC-интеграцией моделей искусственного интеллекта с задачами
Источник: составлен автором

Для решения данной проблемы можно в существующую архитектуру добавить дополнительный слой операторов, по некоторой аналогии, как предложено в работах [8-10], которые будут поддерживать стандартные операции. То есть, вместо прямой интеграции вида «модель – задача», можно разделить взаимодействие на уровень реализации базовых операций [11], которые реализуют модели и уровень описания задачи, как последовательность операторов. Для этого необходимо усложнить описанную выше модель путём определения множества абстрактных операторов $K=\{O_1,O_2,...,O_k\}$, реализующих набор стандартных операций вида «генерация текста», «эмоциональный анализ», «извлечение сущностей», «поиск», «ответ на вопрос», «перевод» и т.д. Тогда каждая модель M_i реализует один или несколько операторов из K, то есть $M_i : Op_{M_i} \subseteq K$. Тогда задача T_j может быть определена как композиция операторов

$$T_j = O_1 \circ O_2 \circ \ldots \circ O_{j_r}, \qquad (10)$$

где $O_{j_r} \in K, r \le k.$

При таком подходе любая задача становится простым потоком исполнения операторов, не привязанных к конкретным моделям. Для каждого оператора $O_k \in K$ существует отображение $\mu : K \to P(M)$ такое, что $\mu(O_k) = \{M_i \in M \mid M_i \vDash O_k\}$. Это множество моделей, реализующих соответствующую операцию. В таком случае для исполнения задачи необходимо выбрать

конкретную модель ($M_k^* \in \mu(O_k)$), которая будет вызвана при выполнении соответствующего шага пайплайна. Чтобы формализовать этот выбор, вводится функция σ, сопоставляющая паре (оператор, контекст задачи) конкретную модель

$$\sigma : (O_k, C) \mapsto M_k^* \in \mu(O_k), \qquad (11)$$

где C – это контекст исполнения задачи.

Полная интеграция задачи T_j теперь выглядит как композиция

$$T_j = \psi_{k_r \to j} \circ O_{j_r} \circ \ldots \circ O_{j_2} \circ O_{j_1} \circ \varphi_{j \to k_1}, \qquad (12)$$

где $\varphi_{k_r \to j} : D_j \to A_{k_1}$ – преобразование входа задачи в формат первого оператора,

$\psi_{k_r \to j} : B_{k_r} \to R_j$ – интерпретация результата последнего оператора под формат задачи.

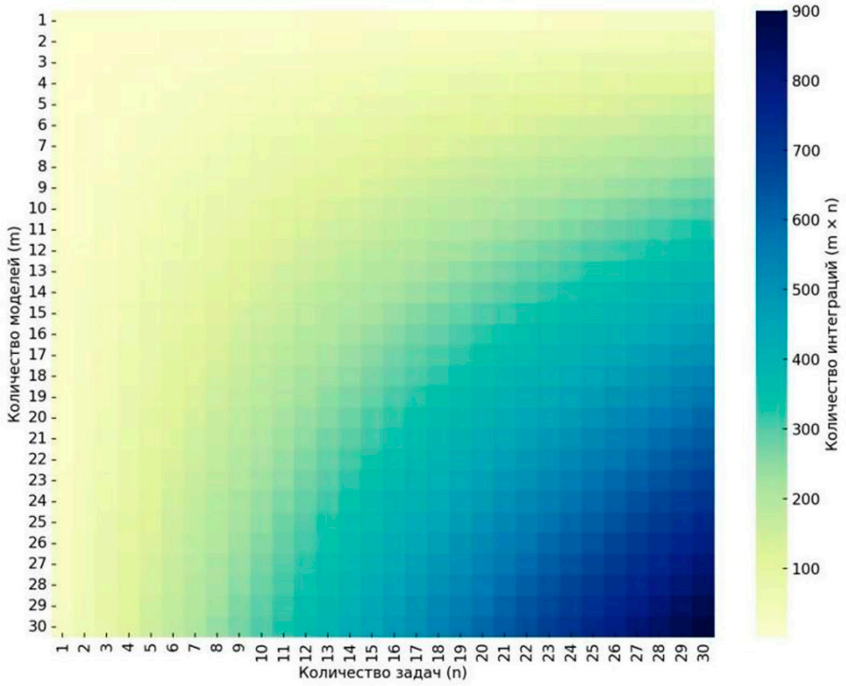

Рис. 3. Рост количества интеграций при фиксированном числе задач
Источник: составлен автором

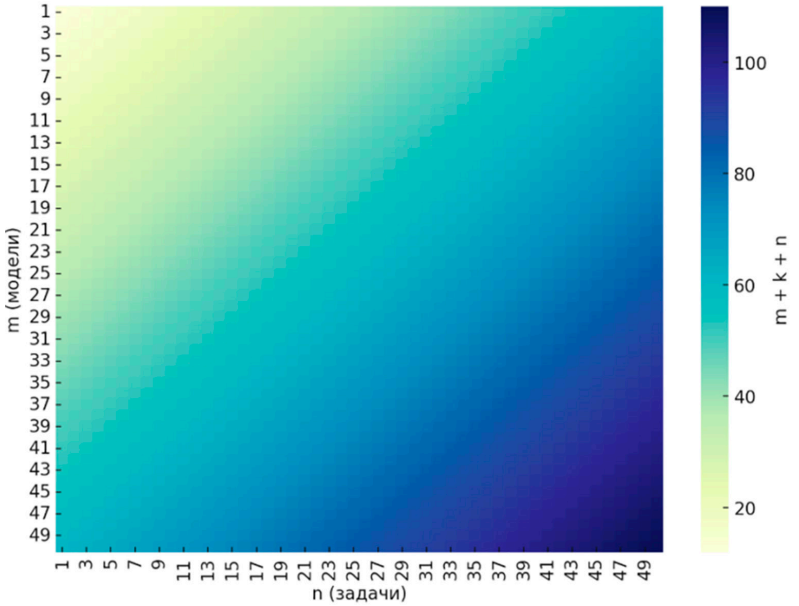

Рис. 4. Сложность интеграции при использовании операторного слоя
Источник: составлен автором

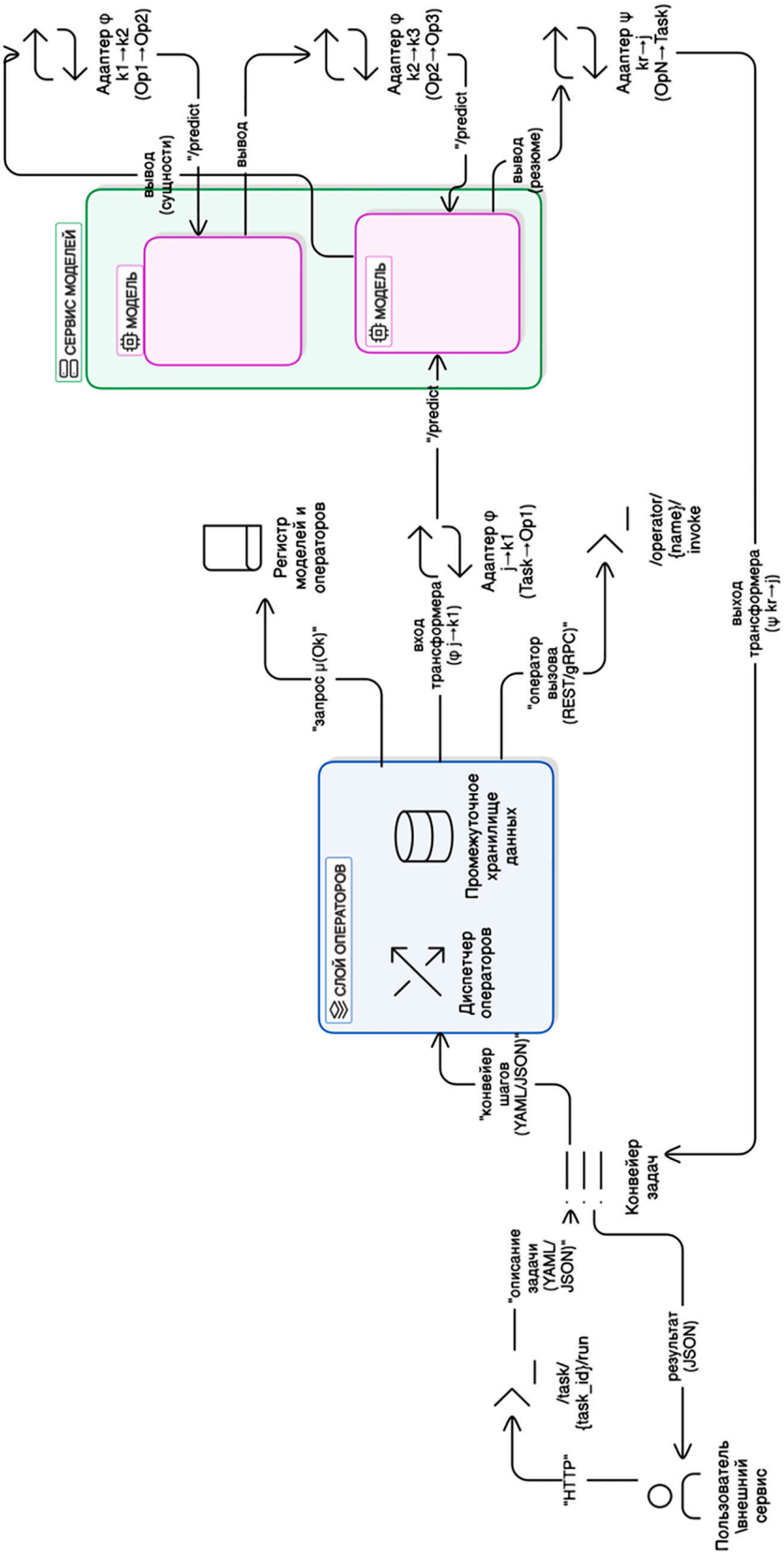

Рис. 5. Архитектура системы интеграции моделей ИИ с промежуточным слоем абстрактных операторов (составлен автором)

Тогда итоговая сложность для способа интеграции моделей искусственного интеллекта с использованием микросервисной архитектуры и дополнительного слоя операторов определяется как $O(m+k+n)$ где $k \ll m,n$, что приводит к тому, что количество необходимых интеграций в системе растёт линейно по числу моделей, задач и операторов, так как множество операторов относительно стабильно и компактно. То есть, если даже количество моделей m=200, а задач n=300, набор таких базовых операций останется равным (примерно) 10-20. На рисунке 4 показана сложность интеграции при использовании операторного слоя, полученная с использованием языка Python и Jupyter Notebook.

С технической стороны вопроса, реализация такого подхода может быть произведена за счёт введения дополнительных архитектурных компонентов, как показано на рисунке 5.

Как видно из рисунка 5, пользователь или внешняя система инициирует выполнение задачи через HTTP-запрос к API системы (/task/{task_id}/run). В запросе передаётся описание задачи в виде сценария обработки (pipeline), выраженного, например, в формате YAML или JSON. На рисунке 6 показан возможный формат задачи на языке yaml.

Sample

```yaml
task: review_analysis
steps:
  - id: extract
    operator: extract_entities
    input: $input
  - id: emotion
    operator: emotion_analysis
    input: extract
  - id: summary
    operator: summarize
    input: emotion
```

Рис. 6. Декларативное описание задачи
Источник: составлен автором

Хотя в декларативном описании задачи отображения $\varphi_{k_r \to j}$ и $\psi_{k_r \to j}$ не указываются явно, они реализуются в интеграционном слое системы. Система выполняет проверку согласования типов вида $\tau_{\text{out}}(O_a) = \tau_{\text{in}}(O_b)$ и если типы не согласуются, пайплайн считается некорректным. Такой формат превращает задачу в граф исполнения операторов, а не в просто в набор шагов, что позволяет системе динамически подставить модели, реализующие каждый оператор и как видно из рисунка 6 пайплайн не содержит указаний на конкретные модели. Тогда направленный ациклический граф операторов может быть представлен, как

$$G_T = (V_T, E_T), V_T \subseteq O, E_T \subseteq V_T \times V_T, \quad (13)$$

в котором узлы будут являться операторами, а рёбра графа задают порядок исполнения операций. При этом формальное условие согласования типов, для любого корректного графа G_T, будет представлено как

$$\forall (O_a \to O_b) \in E_T : \tau_{\text{out}}(O_a) = \tau_{\text{in}}(O_b), \quad (14)$$

что позволяет реализовать механизм валидации декларативного описания задачи.

Далее задача поступает на вход центрального исполнительного компонента системы, в котором реализован компонент «Диспетчер операторов», который отвечает за управление выполнением задач и распределением вызовов операторов, и временное хранилище данных между шагами пайплайна. То есть, он анализирует пайплайн и вызывает нужные модели по операторам, используя адаптеры. Именно компонент «Диспетчер операторов» отвечает за формирование функционального графа исполнения G_M по описанному задачей операторному графу G_T. Каждому оператору $O_i \in V_T$ диспетчер сопоставляет множество допустимых моделей $\mu(O_i) \subseteq M$, где каждая модель реализует соответствующий оператор. Из этого множества выбирается конкретная модель $M_i^* \in \mu(O_i)$, исходя из требуемого контекста, как описано в (11) и дополнительных эвристик, например производительности модели на текущей момент времени, в случае дублирования моделей. Таким образом, формируется граф исполнения моделей (механизм роутинга)

$$G_M = (V_M, E_M), V_M \subseteq M, \text{ такое что}$$

$$\forall O_i \in V_T, \exists M_i^* \in \mu(O_i), M_i^* \in V_M, \quad (15)$$

а структура рёбер графа согласована с операторным графом

$$\forall (O_a \to O_b) \in E_T \Rightarrow (M_a^* \to M_b^*) \in E_M, (16)$$

что позволяет выбирать конкретные реализации операторов на этапе исполнения.

Формат возможного запроса к модели, на примере автоматического мультимодального анализ пользовательского фотоотзыва, показан на рисунке 7.

Данный слой также связан с реестром моделей.

```
POST /predict
{
  "input": {
    "text": "На фотографии изображена счастливая собака
      в парке, который я посетил вчера. Мне понравилась
      атмосфера и природа данного места и я вернусь
      туда ещё раз!",
    "image": "
      /4AAQSkZJRgABAQAAAQABAAD...",
    "metadata": {
      "user_id": "u2398",
      "language": "ru",
      "submitted_at": "2025-05-01T09:34:12Z",
      "source": "mobile_app"
    }
  },
  "context": {
    "task": "user_feedback_analysis",
    "purpose": "generate_public_emotional_summary",
    "domain": "pet_experience",
    "target_audience": "website_visitors",
    "output_tone": "friendly",
    "output_length": "short",
    "include_key_entities": true,
    "multioperator_chain": [
      "extract_entities",
      "emotion_analysis",
      "summarization"
    ]
  },
  "parameters": {
    "temperature": 0.5,
    "top_k": 40,
    "max_tokens": 120,
    "language": "ru"
  }
```

Рис. 7. Пример запроса к модели
Источник: составлен автором

На основании запроса вида $\mu(O_k)$, диспетчер получает список всех моделей, реализующих указанный оператор, что позволяет динамически маршрутизировать вызовы моделей без жёсткой привязки задач к моделям. Далее слой адаптеров обеспечивают преобразование входа задачи к формату всех операторов из пайплайна задачи, а результат последнего оператора интерпретируют в формат, пригодный для возврата пользователю.

Данная архитектура обеспечивает слабую связность за счёт того, что благодаря введению абстрактного слоя операторов, новые задачи могут быть собраны как сценарии без модификации существующих моделей, а новые модели, в свою очередь, могут подключаться без пересборки задач.

Подход к интеграции моделей искусственного интеллекта с использованием дополнительных слоёв абстракции или схожих механизмов, набирает всё большую популярность на практике. Так, GitLab, в рамках проекта AI Abstraction Layer, ведёт работы по созданию унифицированной архитектуры для интеграции и управления AI/ML-функциями для своей платформы. Согласно официальной документации проекта, абстрактный слой функционирует как промежуточный слой между интерфейсами GitLab (веб-интерфейс, IDE, API) и различными AI-провайдерами. Для этого взаимодействия предлагается использовать расширяемый GraphQL API. Однако данное решение находится ещё на стадии концепции и конечный вид данной технологии ещё не определён. Возможно, что данное решение, пока что, опирается на более технически фиксированный маршрут вызова AI-функций (через aiAction), по сравнению с предложенным выше решением. Помимо этого, уже появились инструменты, такие как TensorFlow Serving, TorchServe и т.д., которые представляют собой системы сервинга моделей [12]. Данные инструменты обеспечивают интеграцию на уровне вынесения модели в отдельный сервис, что позволяет вызывать конкретную модель по её API-эндпоинту [13]. Каждая задача знает, какую именно модель (или сервис) ей вы-

звать для получения результата, и формирует запросы под требования этой модели. Однако данное решение обладает существенным недостатком, а именно, чтобы переключить задачу на другую модель, нужно изменить её конфигурацию или код вызова. Появились и более сложные системы инструменты оркестровки для моделей искусственного интеллекта [14], такие как Seldon, KServe, BentoML и т.д., которые позволяют выстраивать более сложные пайплайны обработки. Все эти инструменты упрощают процесс развёртывания и позволяют разворачивать модели как Kubernetes-сервисы, автоматически предоставляя им соответствующие endpoint, но проблема маршрутизации между множеством моделей и задач этими средствами не решается в полном объёме. В предложенном же решении контекстно-ориентированность и динамический выбор модели позволяют устранить данный недостаток.

В современных публикациях подчёркивается, что микросервисный подход хорошо сочетается с требованиями ИИ-систем к масштабированию и обновляемости моделей. Однако одной лишь декомпозиции на сервисы будет недостаточно для решения всех проблем интеграции, поэтому необходимы дополнительные уровни абстракции или стандарты взаимодействия. Предложенный в работе подход по реализации контекстно-ориентированного шлюза с абстрактным слоем операторов позволяет решить эти задачи и уйти от жёсткой привязанности когда, например, путь */vision/modelA* вызывает модель A, а путь */nlp/modelB* вызывает модель B. Предложенный в работе способ описывать задачу в виде графа операций (операторного графа), а не привязывать её жёстко к одной модели, позволяет в дальнейшем использовать контекстно-ориентированную маршрутизацию, чтобы при выборе того, как обработать запрос, система учитывала дополнительный контекст поставленной задачи. Но для успешной работы модели контекстно-ориентированной маршрутизации необходимо, чтобы множество операторов, составляющих абстрактный слой, было достаточно стабильным и стандартизированным. В предложенной текущей итерации реализации отсутствует единый словарь операций, общепринятый протокол или какой-то иной механизм описывающий допустимые типы входов, выходов и контекстов. Это может приводить к тому, что при росте числа задач и моделей система может столкнуться с тем, что каждая новая задача требует уникального набора операторов. На практике это означает, что мощность множества операторов K может увеличиваться до размеров сопоставимых с количеством задач и моделей, а сама сложность интеграции при использовании операторного слоя при таких условиях уже не будет демонстрировать умеренный линейный рост сложности.

В связи с этим, в дальнейших работах планируется разработать механизм стандартизации описания операторов, включая формальную спецификацию типов, контекстов и семантики операций. Для этих целей предполагается использовать открытый стандарт Model Context Protocol (протокол контекста модели) [15], разработанный в декабре 2024 года компанией Anthropic и поддержанный многими крупными технологическими компаниями. Использование данного решения позволит стандартизировать интерфейс взаимодействия с моделями таким образом, чтобы контекст задачи передавался модели, позволяя ей адаптировать своё поведение. В текущей итерации это уже частично реализуется через отображение μ и абстрактные операторы, но в случае использования протокола MCP все модели будут объединены под единым интерфейсом, что должно также понизить сложность интеграции.

Заключение

Проведённое исследование позволило разработать контекстно-ориентированную архитектуру интеграции систем искусственного интеллекта с использованием промежуточного слоя абстрактных операторов, ориентированную на снижение сложности сопряжения интеллектуальных модулей с прикладной логикой распределённых программных систем. В рамках работы разработана также архитектура программного шлюза, поддерживающего контекстно-ориентированное взаимодействие и декомпозицию задач на абстрактные операторы, что обеспечило отделение описания задачи от конкретных реализаций моделей. Показано, что использование операционного слоя позволяет перейти от жёсткой маршрутизации запросов к более гибкой модели динамического связывания, в которой выбор модели осуществляется с учётом контекста задачи и текущих условий исполнения. Предложенное решение ориентировано на микросервисную среду и применимо для решения задачи интеграции разрозненных систем искусственного интеллекта. Также в работе выявлены ограничения, связанные с отсутствием стандартизации операторного слоя, что в перспективе может повлиять на масштабируемость подхода, а именно повышение уровня сложности интеграции. Для решения данной задачи пред-

полагается формализировать операционные онтологии, что можно реализовать за счёт использования нового перспективного протокола Model Context Protocol.

Список литературы

1. Sinha S., Lee Y.M. Challenges with developing and deploying AI models and applications in industrial systems // Discover Artificial Intelligence. 2024. Vol. 4, Is. 1. P. 55. DOI: 10.1007/s44163-024-00151-2.

2. van Ooijen P.M.A., Darzi E., Dekker A. Data Storage, Cloud Usage and Artificial Intelligence Pipeline // Artificial Intelligence in Cardiothoracic Imaging. Cham: Springer International Publishing. 2022. C. 45-55.

3. Zhao M., Agarwal N., Basant A., Gedik B., Pan S., Ozdal M., Komuravelli R., Pan J., Bao T., Lu H., Narayanan S., Langman J., Wilfong K., Rastogi H., Wu C.-J., Kozyrakis C., Pol P. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product // Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA). 2022. P. 1042–1057. DOI: 10.1145/3470496.3533044.

4. Hiniduma K., Byna S., Bez J.L. Data readiness for AI: A 360-degree survey // ACM Computing Surveys. 2025. Vol. 57, Is. 9. P. 1–39. DOI: 10.1145/3722214.

5. Enshaei N., Naderkhani F. The Role of Data Quality for Reliable AI Performance in Medical Applications // IEEE Reliability Magazine. 2024. Vol. 1, Is. 3. P. 24–28. DOI: 10.1109/MRL.2024.3430192.

6. Madhavan D. Enterprise Data Governance: A Comprehensive Framework for Ensuring Data Integrity, Security, and Compliance in Modern Organizations // International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2024. Vol. 10, Is. 5. P. 731–743. DOI: 10.32628/CSEIT241051062.

7. Zhong C., Zhang H., Li C., Huang H., Feitosa D. On measuring coupling between microservices // Journal of Systems and Software. 2023. Vol. 200. Article ID: 111670. DOI: 10.1016/j.jss.2023.111670.

8. Nguyen P., Hilario M., Kalousis A. Using meta-mining to support data mining workflow planning and optimization // Journal of Artificial Intelligence Research. 2014. Vol. 51. P. 605–644. DOI: 10.1613/jair.4377.

9. Murphy A.C., Moreland J.D. Integrating AI microservices into hard-real-time SoS to ensure trustworthiness of digital enterprise using mission engineering // Journal of Integrated Design and Process Science. 2021. Vol. 25, Is. 1. P. 1–18. DOI: 10.3233/JID-210013.

10. Maddukuri N. The Transformative Impact of AI on Modern System Integration // International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2025. Vol. 11, Is. 1. DOI: 10.32628/CSEIT25111218.

11. Lopez R., Lourenço R., Rampin R., Castelo S., Santos A.S.R., Piazentin Ono J. H., Silva C., Freire J. AlphaD3M: An Open-Source AutoML Library for Multiple ML Tasks // Proceedings of the AutoML Conference 2023 (ABCD Track). 2023. Vol. 224. P. 1–22. URL: https://proceedings.mlr.press/v224/lopez23a.html (дата обращения: 02.04.2025).

12. Luu H., Pumperla M., Zhang Z. Model serving infrastructure // MLOps with Ray: Best Practices and Strategies for Adopting Machine Learning Operations. Berkeley, CA: Apress, 2024. P. 167–215. DOI: 10.1007/978-1-4842-8888-2_7.

13. García Á.L. DEEPaaS API: A REST API for machine learning and deep learning models // Journal of Open Source Software. 2019. Vol. 4, Is. 42. P. 1517. DOI: 10.21105/joss.01517.

14. Bogacka K., Sowiński P., Danilenka A., Biot F. M., Wasielewska-Michniewska K., Ganzha M., Paprzycki M., Palau C. E. Flexible deployment of machine learning inference pipelines in the cloud–edge–IoT continuum // Electronics. 2024. Vol. 13, Is. 10. Article 1888. DOI: 10.3390/electronics13101888.

15. Patil M., Lokhande V. Model Context Protocol (MCP): Enabling scalable AI data integration // International Journal for Multidisciplinary Research. 2025. Vol. 7, Is. 2. DOI: 10.36948/ijfmr.2025.v07i02.43583.