Аналитическая статья
УДК 004.89; 004.75; DOI: 10.61260/2218-130X-2026-2-111-119

АНАЛИЗ АРХИТЕКТУРНЫХ КОМПОНЕНТОВ МУЛЬТИАГЕНТНЫХ СИСТЕМ

Саратова Татьяна Евгеньевна;
✉Апухтин Александр Михайлович.
МИРЭА – Российский технологический университет, Москва, Россия
✉gtmaleksa@gmail.com

Аннотация. Целью работы является анализ архитектурных компонентов мультиагентных систем на основе больших языковых моделей. Метод исследования – сравнительный анализ архитектур MetaGPT, Generative Agents, AutoGen, OrgAgent, MIRIX и ChatDev по единому набору критериев (организация ролей, память, координация, структурированные результаты, взаимодействие, проверка качества) с последующей интеграцией компонентов в единую систему и опытной проверкой на задачах двух типов в одноагентном и мультиагентном режимах. Установлено, что на рассмотренных задачах мультиагентный режим уступает одноагентному по качеству (оценки 3,0 и 4,5 против 9,0 по десятибалльной шкале). Выявлена архитектурная причина: несоответствие типа задачи набору действий в конфигурации ролей. Обнаружена закономерность: при интеграции трех контуров проверки – детектора отказов, проверки соответствия и контура принятия решений – возникает способность к самодиагностике, не наблюдаемая ни в одной из рассмотренных систем по отдельности. Три контура согласованно определили ненадежность результата и отказались его выдать. Данное свойство отличается от известных подходов к диагностике мультиагентных систем, где анализ выполняется после завершения работы. Результаты могут быть использованы при проектировании мультиагентных систем для задач с высокой ценой ошибки.

Ключевые слова: мультиагентные системы, большие языковые модели, архитектура агентов, память агентов, координация, самодиагностика

Для цитирования: Саратова Т.Е., Апухтин А.М. Анализ архитектурных компонентов мультиагентных систем // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2026. № 2. С. 111–119. DOI: 10.61260/2218-130X-2026-2-111-119

Analytical article

ANALYSIS OF ARCHITECTURAL COMPONENTS OF MULTI-AGENT SYSTEMS

Saratova Tatiana E.;
✉Apuhtin Alexander M.
MIREA – Russian technological university, Moscow, Russia
✉gtmaleksa@gmail.com

Abstract. The aim of this work is to analyze the architectural components of LLM-based multi-agent systems. The research method involves a comparative analysis of MetaGPT, Generative Agents, AutoGen, OrgAgent, MIRIX, and ChatDev architectures against a uniform set of criteria (role organization, memory, coordination, structured outputs, communication, quality control), followed by integration of the components into a single system and experimental evaluation on two task types in single-agent and multi-agent modes. It is found that on the examined tasks the multi-agent mode underperforms the single-agent baseline in quality (scores of 3,0 and 4,5 versus 9,0 on a ten-point scale). The architectural cause is identified: a mismatch between task type and the preconfigured action chain.

A pattern is discovered: the integration of three verification loops – a failure detector, a compliance check, and a decision-making loop – produces a self-diagnosis capability not observed in any of the examined systems individually. The three loops jointly identified result unreliability and refused to deliver it. This property differs from known multi-agent systems diagnostic approaches where analysis is performed after task completion. The results may be applied when designing multi-agent systems for tasks where the cost of error is high.

Keywords: multi-agent systems, large language models, agent architecture, agent memory, coordination, self-diagnosis

For citation: Saratova T.E., Apuhtin A.M. Analysis of architectural components of multi-agent systems // Scientific and analytical journal «Vestnik Saint-Petersburg university of State fire service of EMERCOM of Russia». 2026. № 2. P. 111–119. DOI: 10.61260/2218-130X-2026-2-111-119

Введение

Мультиагентные системы на основе больших языковых моделей активно развиваются с 2023 г. Существующие обзоры систематизируют подходы к построению таких систем: команды LLM-агентов способны совместно разрабатывать программное обеспечение, моделировать социальное поведение и решать многошаговые задачи. Каждая из опубликованных систем вносит самостоятельный архитектурный вклад, однако ни одна не объединяет все описанные элементы: одни реализуют роли и типизированные результаты, но лишены долговременной памяти; другие развивают когнитивную память, но не производят файловых результатов; третьи координируют агентов иерархически, но обходятся без памяти и артефактов. Это порождает вопрос: дает ли сочетание лучших элементов из нескольких систем результат, превосходящий каждую из них?

Отдельную задачу составляет диагностика провалов. Существующие эмпирические исследования выделяют десятки режимов отказа в мультиагентных системах, однако предлагают только описательные классификации без средств обнаружения в ходе работы. Известные подходы к диагностике анализируют протоколы выполнения после завершения работы при участии человека. Вопрос о том, способна ли мультиагентная система распознавать собственные провалы в реальном времени, остается открытым.

Цель настоящей работы – анализ архитектурных компонентов мультиагентных систем (МАС) на основе больших языковых моделей (LLM) и исследование свойств, возникающих при совместной работе. Предложенный подход отличается тем, что каждый компонент взят из конкретной публикации с указанием степени соответствия оригиналу, а проверка выполнена с измерением количественных показателей.

Модели и методы исследования

Мультиагентные системы как направление исследований относятся к работам по распределенному искусственному интеллекту [1] и организационной теории [2]. Обзоры [3, 4] систематизируют современные подходы к разработке МАС на основе больших языковых моделей. С появлением генеративных моделей роли агентов, память и координация реализуются новыми средствами. Ниже рассмотренные системы по архитектурным направлениям, что позволяет сопоставить подходы разных авторов к одной и той же задаче.

Роли и распределение обязанностей формализованы в трех из рассмотренных систем, причем подходы заметно различаются. В работе [5] закодированы этапы конвейера: менеджер продукта формирует техническое задание, архитектор проектирует систему, инженер пишет код и тесты, тестировщик проверяет; каждое действие принимает структурированный вход и порождает структурированный выход, проверяемый по JSON-схеме. В исследовании [6] участвуют в свободной беседе на естественном языке до достижения консенсуса. В работе [7] разнесены по трем уровням: управление

(планирование и назначение), исполнение (генерация и ревью) и соответствие (проверка формата); координация идет сверху вниз с фиксированным лимитом раундов [7]. Остальные системы [5, 6, 8] ролевую структуру либо не формализуют, либо задают текстовыми описаниями без ограничений.

Память агентов наиболее развита в двух работах. В исследовании [8] реализован непрерывный поток воспоминаний с оценкой важности, извлечение по формуле с тремя факторами (давность, важность, релевантность), периодическая рефлексия, обобщающая эпизоды в высокоуровневые суждения, и автономное планирование; ablation подтверждает, что отключение рефлексии снижает качество поведения [8]. В работе [9] память разделена на шесть хранилищ (Core, Episodic, Semantic, Procedural, Resource, Knowledge Vault) с LLM-маршрутизатором, определяющим, к какому хранилищу обращаться; на бенчмарке LOCOMO – 85,4 % [9]. Остальные системы [2, 3, 5] долговременной памятью не располагают.

Способы взаимодействия между агентами представлены двумя подходами. В исследовании [9] предложены четыре паттерна: парный диалог, последовательная цепочка, групповое обсуждение с выбором говорящего и вложенные диалоги; предусмотрены также пользовательские функции ответа, позволяющие обрабатывать случаи без обращения к модели [9]. В работе [8] агенты общаются через свободный диалог и передают знания друг другу (социальная диффузия). В остальных системах [2, 3, 5, 6] взаимодействие либо жестко встроено в конвейер, либо не структурировано, либо не предусмотрено.

Работа со структурированными результатами реализована полноценно только в источнике [5], где каждое действие порождает типизированный выход (техническое задание, UML-диаграмма, код, тесты), а инженер использует результаты тестов для итеративной отладки; ablation показывает прирост 4,2 % на HumanEval от этого цикла [5]. Частично результаты создаются в работе [9] (исполнение кода в изолированной среде) и работе [6] (код как итог беседы). В исследования [7–9] файловые результаты не порождаются.

Координация и иерархия наиболее развиты в работе [7]: трехуровневая архитектура дает на SQuAD 2.0 прирост F1-score на 102,73 % при снижении потребления токенов на 74,52 % для GPT-OSS-120B по сравнению с плоской организацией [7, табл. 1]. В исследовании [9] координация гибкая, но без формальной иерархии. В работах [5, 6] – линейный конвейер. В исследовании [8] координация отсутствует: каждый агент автономен. В источнике [9] координация выходит за рамки работы.

Диагностика провалов изучена в одном исследовании [10]: на материале более 1 600 протоколов выполнения семи МАС с участием шести аннотаторов (согласие k = 0,88) предложена таксономия MAST из 14 режимов отказа в трех категориях – проблемы проектирования, рассогласование между агентами, провалы проверки. Средств обнаружения провалов в ходе работы авторы не предлагают; подход носит описательный характер. Близкое по направлению исследование [11] также сосредоточено на классификации причин отказов, не вводя механизмов онлайн-диагностики. Ни одна из остальных систем не содержит встроенного механизма самодиагностики.

Таблица 1

Сравнение архитектурных компонентов рассмотренных МАС

Система	Роли	Память	Разд. пам.	Связь	Результ.	Координ.
MetaGPT [5]	Да	Нет	Нет	Нет	Да	Частично
ChatDev [6]	Да	Нет	Нет	Частично	Да	Частично
Gen. Agents [8]	Частично	Да	Да	Да	Нет	Нет
AutoGen [9]	Частично	Нет	Нет	Да	Частично	Да
OrgAgent [7]	Да	Нет	Нет	Частично	Нет	Да
MIRIX [9]	Нет	Да	Да	Нет	Нет	Нет

Как видно из табл. 1, каждая из рассмотренных систем покрывает два-три компонента, но ни одна не охватывает все. Роли и результаты развиты в работах по разработке программного обеспечения [2, 5], память – в работах по моделированию поведения [6, 8], координация – в работе по организационной иерархии [7]. Это и определяет задачу интеграции, описанную далее.

Разработанная система

Для каждого из компонентов были выбраны элементы, показавшие наиболее полную его реализацию. Структурированные результаты с проверкой по JSON-схеме воспроизведены по модели [5] – единственной среди рассмотренных, где ввод и вывод каждого действия строго типизированы; ablation в той же работе показывает прирост 4,2 % от цикла обратной связи [5]. Трехуровневая иерархия построена по схеме [7], где координация разделена на три уровня с фиксированным лимитом раундов; на MuSiQue и SQuAD 2.0 она превосходит плоскую по F1-score [7]. Память воспроизводит подход [9] с шестью раздельными хранилищами, обоснованными когнитивными моделями структуры памяти [9]. Рефлексия и извлечение следуют [8], где предложена формула оценки по трем факторам и обобщение эпизодов; ablation подтверждает падение качества без рефлексии [8]. Способы взаимодействия опираются на [9] с четырьмя паттернами и динамическим выбором говорящего. Для диагностики использована таксономия [10] – единственное среди известных авторам исследование отказов MAC на 1 600+ протоколах выполнения с согласием аннотаторов $k = 0,88$.

Каждый элемент переключается настройкой в конфигурационном файле. Все прогоны записывают полный протокол выполнения в JSONL – каждое обращение к модели, каждое сообщение и каждое действие зафиксированы.

Таблица 2

Компоненты разработанной системы и их источники

Элемент	Откуда	Степень
Действия с типизированными результатами и JSON-схемой	[5]	Совпадает
Цикл исполнения кода: sandbox, ошибка, исправление	[5]	Совпадает
Способы взаимодействия агентов	[9]	Совпадает
Стратегии выбора говорящего	[9]	Добавлена role_priority
Участие человека (утверждение, замечания, отмена)	[9]	Упрощено
Кэш вызовов LLM (SHA-ключ, TTL, обход)	[9]	Совпадает
Трехуровневая иерархия	[7]	Совпадает
Эскалация через общего руководителя	[7]	Совпадает
Контур принятия решений: выпустить / доработать	[7]	Расширен до LLM-цикла
Проверка соответствия: 5 аспектов	[7]	Расширена
Память из 6 частей	[9]	Совпадает
LLM-маршрутизатор памяти	[9]	Совпадает
Рефлексия: эпизоды в обобщения	[8]	Расширена

Элемент	Откуда	Степень
Формула оценки: давность * важность * релевантность	[8]	Совпадает
Передача знаний между агентами с разграничением	[8]	Расширена
Детектор отказов (5 из 14 режимов)	[10]	Расширен
Разграничение доступа по ролям (3 слоя)	[12]	Расширено
Гибридный поиск (dense + BM25 + RRF)	Ориг.	–

При интеграции компонентов приняты осознанные упрощения по сравнению с оригиналами, чтобы покрыть большее количество доменов, с которыми можно работать. Раздельные наборы действий для разных типов задач, предусмотренные в работе [5], заменены единой цепочкой. Пользовательские функции ответа из исследования [9], позволяющие обходиться без вызова модели, не реализованы: каждый шаг проходит через модель. Миграция записей между хранилищами памяти, описанная в работе [9], и автономное планирование агентов из исследования [8] опущены. Совместная генерация кода и тестов из работы [5] упрощена до исполнения только точки входа.

Разработанная система объединяет три контура качества. Сначала проверка по пяти аспектам: корректность ролевых границ, целостность результатов, трассируемость, покрытие требований и качество рассуждений; построена по схеме [7] с двумя дополнительными аспектами. Потом детектор отказов: LLM-классификатор на основе таксономии [10]; в отличие от оригинала, где таксономия описательная, здесь она применяется в ходе работы, анализируя журнал решений и последние результаты. Третий – контур принятия решений, где агент уровня управления оценивает итог и выносит решение «выпустить» или «доработать»; построен по модели [7] и расширен до итеративного LLM-цикла.

Все три контура работают в ходе выполнения задачи и способны отказать в выдаче результата. Этим они отличаются от подходов [13, 14], где анализ выполняется после завершения работы.

Результаты исследования

Проведена серия прогонов с рассуждениями объемом до 4 086 токенов на каждое обращение к модели. Ниже подробно разобраны два характерных случая: задача на генерацию программного кода (REST API на Python для TODO-приложения с тремя конечными точками, хранением в памяти и запуском через python main.py) и аналитическая задача (оценка рисков срыва квартального релиза – 5 разработчиков, 6 недель, 32 задачи в бэклоге, CI без performance-тестов – с выработкой рекомендаций). Эти задачи предъявляют разные требования к архитектуре: первая задействует конвейер создания и проверки кода, вторая – аналитические рассуждения нескольких ролей.

Каждая задача выполнена в одноагентном и мультиагентном режимах. В мультиагентном режиме задействованы четыре роли – технический руководитель, менеджер проекта, разработчик и тестировщик – с иерархией подчинения и разграничением доступа к инструментам. Качество оценено LLM-судьей по шкале от 1 до 10 при температуре 0,5 с трехкратным повторением (стандартное отклонение 0,0). Использована бюджетная модель; LLM-судья не заменяет экспертную оценку.

В табл. 3 приведены сводные показатели. Качество – по шкале от 1 до 10, покрытие – доля выполненных требований, pass@1 – запуск с первой попытки. Контур решений – итог управленческого цикла, проверка – результат по пяти аспектам, отказы – число зафиксированных режимов отказа по таксономии [10].

Таблица 3

Сводные показатели

Показатель	API (один)	API (команда)	Риски (один)	Риски (команда)
Качество (1–10)	9,0	3,0	9,0	4,5
Покрытие требований	1,0	0,5	0,95	0,45
pass@1	1,0	0,0	1,0	0,0
Длительность, с	11	469	30	1162
Число вызовов LLM	2	125	2	97
Контур решений	–	выпустить	–	доработать
Проверка пройдена	–	нет (3 зам.)	–	нет (3 зам.)
Режимов отказа	–	4 из 5	–	5 из 5

На задаче генерации API команда подготовила код на FastAPI и приложила файл зависимостей (requirements.txt). Судья снизил оценку: по условию задачи приложение должно запускаться командой python main.py, а код требует установки сторонней библиотеки. Одноагентный режим дал аналогичный код, но без явного списка зависимостей, что не привлекло внимания судьи. Этот случай указывает на особенность методики оценки: более прозрачный результат оценивается строже. Цикл исполнения в sandbox сработал по плану – обнаружена ошибка, инженерный агент её исправил. Проверка по пяти аспектам зафиксировала три замечания, однако они не были переданы в решающий контур. Это выявленный пробел реализации: решение выносилось без информации о замечаниях. Пробел устраним и не связан с архитектурой.

На аналитической задаче набор ролей был рассчитан на генерацию кода. Задача анализа рисков требует другой цепочки: оценки ситуации и выработки рекомендаций. Координатор верно определил тип задачи, однако при построении цепочки действий это поле не учитывается – пробел, устранимый введением раздельных наборов действий, как это сделано в работе [5]. Показательно, что система самостоятельно распознала несоответствие: инженерный агент четырежды отказался генерировать программный результат для непрограммистской задачи, контур принятия решений не подписал итог, а детектор зафиксировал все пять режимов. При наличии архитектурного пробела контуры проверки сработали согласованно и предотвратили выдачу ненадежного результата.

Таблица 4

Показатели мультиагентных прогонов

Показатель	Генерация API	Анализ рисков
Шагов конвейера	8	8
Раундов обсуждения	1	2
Эскалаций	0	1
Раундов контура решений	1	2
Кэш: попадания / промахи	19/106	19/78
Обращений к маршрутизатору [9]	0	0
Зафиксировано режимов отказа	4	5 (все)

Маршрутизатор памяти из работы [9] показал нулевое число обращений: все запросы оказались короче порога обхода (8 токенов). Заявленный в исследовании [9] способ динамического выбора хранилища в данных условиях не активировался. Кэш вызовов дал долю попаданий 15–20 %, сокращая число обращений к модели на повторных циклах.

Несмотря на расхождение в оценках качества, мультиагентный режим продемонстрировал свойство, отсутствующее в одноагентном: способность распознавать собственную неготовность и отказываться от выдачи результата. На аналитической задаче все три контура сработали согласованно: детектор зафиксировал пять из пяти режимов отказа (два серьезных), проверка по пяти аспектам не пройдена с тремя замечаниями, а завершающий контур отказал в выдаче. Одноагентный режим дал ответ с оценкой 9,0, но не содержит никакого средства самооценки – пользователь не знает, насколько ответ надежен. Мультиагентный режим с оценкой 4,5 сам указал на ненадежность. Это свойство не описано ни в одной из рассмотренных работ и не возникает при использовании их элементов порознь.

На рассмотренных задачах с бюджетной моделью мультиагентный режим не оправдал накладных расходов по сравнению с одноагентным. Однако установленная причина носит конфигурационный характер: отсутствие раздельных наборов действий по типам задач привело к тому, что аналитическая задача обрабатывалась конвейером, рассчитанным на код. Разделение, аналогичное реализованному в работе [5], устранит это расхождение.

Часть элементов, взятых из рассмотренных работ, в условиях прогонов не активировалась. Маршрутизатор [9] не получил обращений, хранилище общих знаний осталось пустым. При выполнении отдельных задач (в отличие от непрерывного моделирования [8]) потребность в них не возникает. Для проектов с длительным горизонтом эти элементы могут оказаться востребованы.

Наиболее заметный итог серии прогонов – согласованная работа трех контуров. На аналитической задаче они независимо определили, что результат ненадежен, и не допустили его выдачи. Для задач, где цена ошибки высока, способность системы распознать собственную неготовность может оказаться важнее абсолютного качества ответа.

Заключение

Проведен анализ архитектурных компонентов мультиагентных систем на основе больших языковых моделей; для каждой определены сильные стороны и границы. На основе анализа разработана система из 19 компонентов, каждый из которых воспроизведен по конкретной публикации.

Опытные прогоны выявили конфигурационный пробел: единая цепочка действий не подходит для задач разного типа. Пробел не затрагивает архитектуру и устраним введением раздельных наборов действий, как реализовано в работе [5]. Цикл исполнения кода и кэш вызовов сработали в соответствии с описаниями в источниках.

При совместной работе элементов из разных систем проявилось свойство, не описанное ни в одной из них: три контура проверки независимо и согласованно распознали ненадежность результата и не допустили его выдачи. Одноагентный режим таким свойством не располагает. Дальнейшая работа предполагает введение раздельных наборов действий по типам задач, проверку с более мощными моделями и применение самодиагностики в областях с высокой ценой ошибки.

Список источников

1. Wooldridge M. An Introduction to MultiAgent Systems. Wiley, 2009. 484 p.

2. Минцберг Г. Структура в кулаке: создание эффективной организации / пер. с англ. СПб.: Питер, 2004. 512 с.

3. Large Language Model based Multi-Agents: A Survey of Progress and Challenges / T. Guo [et al.] // arXiv:2402.01680. 2024.

4. Multi-Agent Collaboration Mechanisms: A Survey of LLMs / K.-T. Tran [et al.] // arXiv:2501.06322. 2025.

5. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework / S. Hong [et al.] // Proc. ICLR. 2024. arXiv:2308.00352.

6. ChatDev: Communicative Agents for Software Development / C. Qian [et al.] // Proc. ACL. arXiv:2307.07924. 2024.

7. OrgAgent: Organize Your Multi-Agent System like a Company / Wang [et al.] // arXiv:2604.01020. 2026.

8. Generative Agents: Interactive Simulacra of Human Behavior / J.S. Park [et al.] // Proc. ACM UIST. arXiv:2304.03442. 2023.

9. MIRIX: Multi-Agent Memory System for LLM-Based Agents / Wang [et al.] // arXiv:2507.07957. 2025.

10. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation / Q. Wu [et al.] // arXiv:2308.08155. 2023.

11. Why Do Multi-Agent LLM Systems Fail? / M. Cemri [et al.] // Proc. NeurIPS (Datasets and Benchmarks Track). arXiv:2503.13657. 2025.

12. Role-Based Access Control Models / R. Sandhu [et al.] // IEEE Computer. 1996. Vol. 29. № 2. P. 38–47.

13. Towards Self-Improving Error Diagnosis in Multi-Agent Systems / Zhang [et al.] // arXiv:2604.17658. 2026.

14. DiLLS: Interactive Diagnosis of LLM-based Multi-agent Systems via Layered Summary / R. Sheng [et al.] // Proc. CHI. arXiv:2602.05446. 2026.

References

1. Wooldridge M. An Introduction to MultiAgent Systems. Wiley, 2009. 484 p.

2. Mincberg G. Struktura v kulake: sozdanie effektivnoj organizacii / per. s angl. SPb.: Piter, 2004. 512 s.

3. Large Language Model based Multi-Agents: A Survey of Progress and Challenges / T. Guo [et al.] // arXiv:2402.01680. 2024.

4. Multi-Agent Collaboration Mechanisms: A Survey of LLMs / K.-T. Tran [et al.] // arXiv:2501.06322. 2025.

5. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework / S. Hong [et al.] // Proc. ICLR. 2024. arXiv:2308.00352.

6. ChatDev: Communicative Agents for Software Development / C. Qian [et al.] // Proc. ACL. arXiv:2307.07924. 2024.

7. OrgAgent: Organize Your Multi-Agent System like a Company / Wang [et al.] // arXiv:2604.01020. 2026.

8. Generative Agents: Interactive Simulacra of Human Behavior / J.S. Park [et al.] // Proc. ACM UIST. arXiv:2304.03442. 2023.

9. MIRIX: Multi-Agent Memory System for LLM-Based Agents / Wang [et al.] // arXiv:2507.07957. 2025.

10. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation / Q. Wu [et al.] // arXiv:2308.08155. 2023.

11. Why Do Multi-Agent LLM Systems Fail? / M. Cemri [et al.] // Proc. NeurIPS (Datasets and Benchmarks Track). arXiv:2503.13657. 2025.

12. Role-Based Access Control Models / R. Sandhu [et al.] // IEEE Computer. 1996. Vol. 29. № 2. P. 38–47.

13. Towards Self-Improving Error Diagnosis in Multi-Agent Systems / Zhang [et al.] // arXiv:2604.17658. 2026.

14. DiLLS: Interactive Diagnosis of LLM-based Multi-agent Systems via Layered Summary / R. Sheng [et al.] // Proc. CHI. arXiv:2602.05446. 2026.

Информация о статье:
Статья поступила в редакцию: 26.05.2026; одобрена после рецензирования: 27.06.2026;
принята к публикации: 30.06.2026

Information about the article:
The article was submitted to the editorial office: 26.05.2026; approved after review: 27.06.2026;
accepted for publication: 30.06.2026

Информация об авторах:
Саратова Татьяна Евгеньевна, заведующий кафедрой прикладной математики Института информационных технологий МИРЭА – Российского технологического университета (119454, Москва, пр. Вернадского, д. 78), e-mail: saratova@mirea.ru, https://orcid.org/0000-0003-4810-8734
Апухтин Александр Михайлович, МИРЭА – Российский технологический университет (119454, Москва, пр. Вернадского, д. 78), e-mail: gtmaleksa@gmail.com

Information about authors:
Saratova Tatyana E., head of the department of applied mathematics at the institute of information technologies of the MIREA – Russian university of technology (119454, Moscow, Vernadsky ave., 78), e-mail: saratova@mirea.ru, https://orcid.org/0000-0003-4810-8734
Apuhtin Alexander M., MIREA – Russian university of technology (119454, Moscow, Vernadsky ave., 78), e-mail: gtmaleksa@gmail.com