

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»**

Журнал практики

Институт	Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра	ЦОП ВО «ТОП-ИТ»
Учебная группа	М8О-105БВ-25
ФИО обучающегося	Путиловский Михаил Вячеславович
Направление подготовки / специальность	Фундаментальная информатика и информационные технологии
Вид практики	[указать вид практики]
Оценка за практику	_____ [ФИО руководителя]

Москва 2026

1. Место и сроки проведения практики

Наименование организации: ИИ-лаборатория lambda

Сроки проведения практики:

Дата начала практики: 09.02.2026

Дата окончания практики: 25.06.2026

2. Инструктаж по технике безопасности

Проведен инструктаж по технике безопасности.

Подпись проводившего: _____ / [ФИО проводившего
инструктаж] / 9 февраля 2026 г.

подпись, расшифровка подписи, дата

3. Индивидуальное задание обучающегося

Разработка и сопровождение поверхностей взаимодействия пользователей с AI-агентами платформы Lambda. В рамках задания необходимо исследовать пользовательские сценарии Telegram и Matrix, спроектировать общий surface-слой, реализовать простой Telegram-прототип, разработать и довести до рабочего deployment-состояния Matrix-поверхность, подготовить эксплуатационную документацию и координировать командную работу по дальнейшему развитию поверхностей, включая старт направления МАХ.

Поверхность МАХ разрабатывалась другими участниками команды. В данном отчете она учитывается только как зона тимлидской координации, без включения ее реализации в мой личный технический вклад.

4. План выполнения индивидуального задания обучающегося

№ п/п	Место проведения	Тема	Период выполнения
1	ИИ-лаборатория lambda	Инструктаж по технике безопасности. Знакомство с задачами лаборатории Lambda, агентной платформой и направлением пользовательских поверхностей.	09.02.2026-09.02.2026
2	ИИ-лаборатория lambda	Исследование Telegram Bot API, Matrix API, вариантов организации пользовательских чатов, команд и передачи вложений.	10.02.2026-19.03.2026
3	ИИ-лаборатория lambda	Проектирование общего ядра surface-слоя: протокол событий, диспетчеризация, хранение состояния, абстракция платформенного клиента.	20.03.2026-30.03.2026
4	ИИ-лаборатория lambda	Реализация простого Telegram-прототипа и проверка режима Telegram Forum Topics.	31.03.2026-08.04.2026
5	ИИ-лаборатория lambda	Реализация Matrix-поверхности: onboarding, рабочие комнаты, команды управления чатами, маршрутизация сообщений и состояние чатов.	02.04.2026-19.04.2026
6	ИИ-лаборатория lambda	Интеграция Matrix-поверхности с реальным agent API, поддержка потоковых ответов и разделение контекстов по Matrix room.	19.04.2026-24.04.2026

№ п/п	Место проведения	Тема	Период выполнения
7	ИИ-лаборатория lambda	Реализация передачи файлов через shared workspace, поддержки нескольких агентов и маршрутизации пользователей к агентам.	20.04.2026-28.04. 2026
8	ИИ-лаборатория lambda	Подготовка Docker- и production deployment-артефактов, документации для передачи Matrix-поверхности в эксплуатацию.	27.04.2026-08.05. 2026
9	ИИ-лаборатория lambda	Координация работ команды по дальнейшим поверхностям, включая старт направления МАХ.	09.05.2026-18.05. 2026
10	ИИ-лаборатория lambda	Подготовка материалов отчета, описание архитектуры проекта, результатов разработки и итогов практики.	19.05.2026-07.06. 2026

Утверждаю: _____ / [ФИО руководителя от МАИ] / 9
февраля 2026 г.

подпись, расшифровка подписи, дата

Утверждаю: _____ / [ФИО руководителя от
организации] / 9 февраля 2026 г.

подпись, расшифровка подписи, дата

Ознакомлен: _____ / [Путиловский М.] / 9 февраля 2026
г.

подпись, расшифровка подписи, дата

5. Отзыв руководителя практики от организации/предприятия

Обучающийся группы [указать группу] [ФИО] проходил практику в ИИ-лаборатории lambda.

В ходе практики обучающийся был назначен тимлидом команды поверхностей взаимодействия с агентами и принимал участие в разработке surface-слоя для платформы Lambda. Основное внимание было уделено проектированию транспортно-независимой архитектуры, реализации простого Telegram-прототипа, разработке полноценной Matrix-поверхности, интеграции с

agent API, передаче файлов через shared workspace и подготовке deployment-артефактов.

Обучающийся выполнил исследование ограничений Telegram и Matrix, подготовил архитектурные спецификации, реализовал ключевые backend-компоненты, настроил маршрутизацию пользователей к агентам и оформил эксплуатационную документацию. Matrix-поверхность была доведена до рабочего состояния, задеплоена и используется для взаимодействия пользователей с агентами Lambda.

Как тимлид обучающийся координировал дальнейшую работу команды по развитию новых поверхностей, включая старт направления МАХ, обеспечивал передачу контекста и поддерживал единый архитектурный подход к разработке адаптеров.

За время прохождения практики обучающийся продемонстрировал высокий уровень самостоятельности, системного мышления и инженерной ответственности, уверенные знания Python, асинхронного программирования, Docker, API мессенджеров, проектирования интеграционных сервисов и командной разработки.

Материалы, изложенные в отчете обучающегося, полностью соответствуют индивидуальному заданию, рекомендуемая оценка «отлично».

Подпись _____ руководителя _____ от _____ организации/предприятия:

Расшифровка подписи: [ФИО руководителя]

Дата: _____ 2026 г.

6. Отчет обучающегося по практике

Цель и задачи практики

Целью моей практики была разработка пользовательских поверхностей для взаимодействия с AI-агентами платформы Lambda. Под поверхностью в проекте понимался транспортный интерфейс, через который пользователь может создавать отдельные контексты общения, отправлять сообщения и файлы, получать потоковые ответы агента и работать с несколькими агентными сессиями без прямого доступа к внутренней инфраструктуре платформы.

В рамках практики были поставлены следующие задачи:

- изучить ограничения Telegram Bot API и Matrix API для сценариев агентного общения;
- спроектировать общий surface-слой, не зависящий от конкретного мессенджера;
- реализовать простой Telegram-прототип для проверки ранних UX-гипотез;
- реализовать полноценную Matrix-поверхность для рабочих агентных чатов;
- подключить Matrix-поверхность к реальному agent API;
- обеспечить разделение контекстов по комнатам и маршрутизацию пользователей к агентам;
- реализовать передачу файлов через shared workspace;
- подготовить Docker- и deployment-артефакты;
- оформить техническую документацию;
- координировать командную работу по новым поверхностям, включая направление МАХ.

Общий контекст проекта

Практика проходила в ИИ-лаборатории lambda и была связана с развитием агентной платформы Lambda. Одной из задач платформы является предоставление пользователю удобного интерфейса для работы с AI-агентом. Пользователь не должен напрямую взаимодействовать с внутренними

сервисами платформы: ему нужен привычный канал общения, например мессенджер, в котором можно создать отдельный чат, отправить запрос, приложить файл и получить ответ агента.

В ходе практики я был назначен тимлидом команды поверхностей взаимодействия с агентами. Основным техническим результатом стала Matrix-поверхность, которая была доведена до рабочего состояния, задеплоена и используется для взаимодействия с агентами Lambda. Дополнительно был реализован простой Telegram-прототип, использованный для проверки ранних архитектурных и UX-решений. После стабилизации Matrix-направления часть команды начала работу над поверхностью MAX; эту работу я координировал как тимлид, но не выполнял ее реализацию лично.

Разработка велась в репозитории surfaces-bot. По истории Git мой личный вклад включает 197 авторских коммитов, сделанных в период с 26 марта по 8 мая 2026 года. Основные зоны изменений: core, sdk, adapter/telegram, adapter/matrix, config, Docker Compose-артефакты, тесты и проектная документация.

Анализ задачи и исследование вариантов реализации

На первом этапе была изучена предметная область проекта. Для платформы было важно иметь изолированный и расширяемый слой коммуникации, который не привязан к одному мессенджеру. Поэтому в основу проекта была положена идея общего ядра и тонких транспортных адаптеров.

Для Telegram были исследованы варианты организации нескольких чатов: обычные личные сообщения, группы, супергруппы и Forum Topics. Telegram Bot API не позволяет боту самостоятельно создавать группы для пользователя, поэтому автоматическое создание полноценного рабочего пространства оказалось невозможным. В результате Telegram был использован как быстрый прототип, а Forum Topics рассматривались как дополнительный режим для пользователей, которые готовы вручную подключить супергруппу.

Для Matrix были исследованы события комнат, приглашения, организация пространств, работа с отдельными room-чатами, ограничения шифрования и способы хранения метаданных. Matrix лучше подходил для целевой задачи, поскольку позволяет создавать отдельные комнаты, приглашать пользователей, хранить структуру общения на уровне транспорта и строить управляемую модель room-per-chat.

Также на раннем этапе был зафиксирован платформенный контракт. Поскольку реальный SDK платформы находился в развитии, для независимой разработки surface-слоя был создан собственный интерфейс клиента платформы и mock-реализация. Это позволило разрабатывать пользовательский сценарий, обработчики, хранилище и адаптеры без ожидания финального состояния агентной платформы.

Проектирование общего ядра surface-слоя

Для исключения дублирования логики между Telegram и Matrix было реализовано общее ядро в директории core. В него вошли единый протокол событий и ответов, диспетчер входящих событий, менеджер чатов, менеджер авторизации, менеджер настроек и абстракции хранилища состояния.

Ключевое архитектурное решение состояло в разделении проекта на три слоя: транспортный адаптер, который знает особенности конкретного мессенджера; общее ядро, которое работает с унифицированными событиями и не зависит от Telegram или Matrix; платформенный клиент, через который surface-слой обращается к агенту.

Такой подход позволил проверять бизнес-логику через тесты без запуска реальных мессенджеров. Он также подготовил проект к добавлению новых поверхностей: для новой платформы требуется написать адаптер, конвертер входящих событий и рендеринг ответов, не меняя базовые правила работы с чатами и сообщениями.

В ходе разработки также была устранена архитектурная проблема с именованием: ранний каталог platform был переименован в sdk, чтобы избежать

конфликта с одноименным модулем стандартной библиотеки Python и точнее выразить назначение слоя.

Реализация Telegram-прототипа

Telegram-направление использовалось как быстрый способ проверить UX взаимодействия пользователя с агентом. В рамках личного вклада был реализован адаптер на базе aiogram 3.x, базовая обработка команд, конвертация сообщений в общий протокол, хранение соответствий Telegram-пользователей и внутренних чатов, а также прототип режима Forum Topics.

В простом Telegram-прототипе были реализованы стартовый сценарий через /start, создание и выбор чатов, обработка пользовательских сообщений, меню команд бота, базовая работа с настройками и подтверждениями, привязка Telegram thread/topic к внутреннему контексту и первичная поддержка передачи файлов в общий формат вложений.

Отдельно была проработана модель Telegram Forum Topics. Она позволяла использовать темы супергруппы как визуальное представление отдельных чатов. При этом из-за ограничений Telegram Bot API пользователь должен был создать группу самостоятельно и выдать боту права администратора. В ходе проверки были исправлены проблемы обработки команд, архивирования тем, маршрутизации сообщений и поведения /new в topic-контексте.

Telegram-прототип не стал основной production-поверхностью, но выполнил важную роль: помог проверить общий surface-протокол, выявить ограничения транспортов, уточнить требования к хранению контекста и подготовить архитектуру для Matrix.

Реализация Matrix-поверхности

Основной результат практики - полноценная Matrix-поверхность, которая была доведена до рабочего состояния, задеплоена и используется для взаимодействия с агентами Lambda.

В Matrix-адаптере были реализованы endpoint бота на базе matrix-nio, обработка приглашений пользователя, создание персонального пространства и

рабочих комнат, модель отдельной Matrix room для отдельного агентного чата, конвертация Matrix-событий во внутренний протокол, маршрутизация входящих сообщений, команды управления чатами, обработка подтверждений через текстовые команды, защита от обработки собственных сообщений бота, хранение метаданных комнат, восстановление состояния после рестарта и документация по запуску.

В процессе реализации Matrix-направления были пересмотрены некоторые ранние UX-гипотезы. Изначально рассматривалась модель Space-first с отдельной комнатой настроек. На практике более устойчивым оказался DM-first onboarding с последующим созданием реальных рабочих комнат. Пользователь приглашает бота, бот создает рабочее пространство и комнату первого чата, а дальнейшая работа ведется в отдельных комнатах. Это решение лучше соответствует модели Matrix и делает чаты видимыми сущностями самого транспорта.

Существенная часть работы была связана с надежностью. Были исправлены сценарии повторного invite, дублирующего приветствия, случайной обработки собственных сообщений, потери room metadata, неправильного соответствия между Matrix room и платформенным chat id. Также была реализована startup reconciliation: при рестарте бот восстанавливает локальные связи между пользователями, комнатами и агентными контекстами.

Интеграция с агентной платформой

После стабилизации базового Matrix-адаптера была выполнена интеграция с реальным agent API. Поверхность стала обращаться к агенту через тонкий transport adapter, использующий AgentApi. Для каждого Matrix room поддерживается отдельный platform_chat_id, что позволяет разделять контексты разных комнат и не смешивать историю общения.

Была реализована маршрутизация пользователей к агентам через статический реестр. Конфигурация matrix-agents.yaml задает соответствие Matrix-пользователей и агентных контейнеров, а также base_url и

workspace_path каждого агента. Такой подход подходит для MVP-топологии, где один инстанс Matrix-бота обслуживает нескольких пользователей, а каждый пользователь связан со своим агентным контейнером.

Отдельное внимание было уделено потоковым ответам. Поверхность должна не просто ждать финального ответа агента, а корректно принимать события, обновлять пользовательский интерфейс и передавать сообщения в Matrix. В результате Matrix-поверхность стала пригодна для реального диалога с агентом, а не только для mock-сценария.

Передача файлов и shared workspace

Для практического использования агентной поверхности требовалась передача файлов от пользователя агенту и обратно. В Matrix файлы и текстовые сообщения приходят отдельными событиями, поэтому была реализована staged-модель вложений: пользователь может отправить файл, посмотреть список вложений, удалить лишнее и затем отправить текстовую инструкцию, с которой файлы будут переданы агенту.

Для production-сценария была выбрана схема shared workspace. Matrix-бот сохраняет входящие файлы в рабочую директорию конкретного агента, а агент видит эту же директорию как свой /workspace. Если имя файла уже существует, бот выбирает безопасное имя с суффиксом вида file (1).ext. Исходящие файлы работают в обратную сторону: агент пишет файл в workspace, бот читает его и отправляет пользователю как Matrix file message.

Такой подход упростил интеграцию: не потребовался отдельный HTTP-сервис для upload/download, а контракт между поверхностью и агентом остался понятным и воспроизводимым в Docker Compose.

Deployment и эксплуатационная документация

Для передачи Matrix-поверхности в эксплуатацию были подготовлены Docker-артефакты и документация. Production-сценарий оформлен как bot-only deployment: платформа запускает свои агентные контейнеры отдельно, а surface-контейнер подключается к ним через base_url и общий volume.

Были подготовлены Dockerfile с production target, docker-compose.prod.yml для production handoff, docker-compose.fullstack.yml для внутреннего fullstack E2E-сценария, пример реестра агентов config/matrix-agents.example.yaml, документация docs/deploy-architecture.md и обновленный README.md с инструкциями по запуску, конфигурации и ограничениям.

В README зафиксировано, что production target - один контейнер Matrix-поверхности на 25-30 внешних agent containers/services. Также описаны обязательные переменные окружения, формат matrix-agents.yaml, схема shared volume и известные ограничения.

Тестирование и контроль качества

Разработка сопровождалась тестами и документированием принятых решений. В репозитории были добавлены тесты для общего ядра, mock- и real-platform слоев, Telegram-адаптера, Matrix-адаптера, передачи файлов, agent registry, восстановления состояния, per-room маршрутизации и deployment handoff.

Проверялись сценарии создания и маршрутизации чатов, преобразования сообщений разных транспортов во внутренний протокол, обработки команд пользователя, подтверждения действий, соответствия Matrix room внутреннему chat id и платформенному chat id, загрузки agent registry, передачи файлов через workspace, восстановления состояния после рестарта и корректности Docker/deployment-артефактов.

Кроме автоматических проверок, Matrix-поверхность была доведена до фактического deployment-состояния и работает в инфраструктуре проекта. Это является ключевой практической проверкой результата: поверхность не ограничилась локальным прототипом, а была подготовлена к реальному использованию.

Тимлидская работа и координация команды

Помимо личной разработки, в ходе практики я выполнял функции тимлида команды поверхностей. Эта роль включала декомпозицию задач,

фиксацию архитектурных решений, подготовку спецификаций, ревью планов, оформление known limitations и передачу контекста другим участникам.

В репозитории это отражено большим количеством документации и planning-артефактов: спецификации Telegram и Matrix, планы реализации, исследования API, deployment-документация, инструкции по созданию новой поверхности и отчеты о ходе работ.

После стабилизации Matrix-направления часть команды начала заниматься поверхностью MAX. Я не включаю реализацию MAX в свой личный технический вклад, однако как тимлид участвовал в постановке направления: объяснял общую архитектуру surface-слоя, границы ответственности адаптеров и требования к совместимости с ядром и агентной платформой.

Полученные результаты

По итогам практики мной были выполнены следующие работы:

- спроектирован общий surface-слой для разных пользовательских транспортов;
- реализовано общее ядро с унифицированным протоколом событий;
- создан простой Telegram-прототип и проверен режим Forum Topics;
- реализована полноценная Matrix-поверхность;
- выполнена интеграция Matrix-поверхности с agent API;
- реализовано разделение контекстов по Matrix room;
- добавлена маршрутизация пользователей к агентам через agent registry;
- реализована передача файлов через shared workspace;
- подготовлены Docker- и deployment-артефакты;
- оформлена эксплуатационная документация;
- выполнена тимлидская координация команды поверхностей.

Вывод

В ходе практики был создан и доведен до рабочего состояния surface-слой для взаимодействия пользователей с агентами Lambda. Личным основным

результатом стала Matrix-поверхность: она поддерживает отдельные комнаты для чатов, команды управления контекстом, маршрутизацию к агентам, передачу файлов через shared workspace, восстановление состояния после рестарта и production deployment-сценарий.

Также был реализован простой Telegram-прототип, который позволил проверить ранние UX-решения и общий протокол взаимодействия. Несмотря на ограничения Telegram Bot API, этот прототип оказался полезен для архитектурной проверки и исследования альтернативных пользовательских сценариев.

В ходе практики я получил практический опыт проектирования транспортно-независимых backend-слоев, интеграции мессенджеров с агентной платформой, работы с асинхронными Python-сервисами, Docker deployment, тестирования, документирования архитектуры и технического руководства небольшой командой разработки.

Материалы, изложенные в отчете обучающегося, полностью соответствуют индивидуальному заданию.

Подпись обучающегося: _____ / [Путиловский М.] / 7
июня 2026 г.