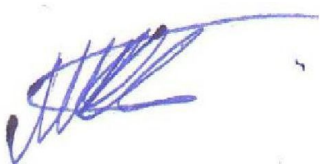
НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК ТАДЖИКИСТАНА

На правах рукописи

УДК 51:681.3+004.8
ББК 22.1+73
Р-27



Рахимов Манучехр Фарходович

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ДОКУМЕНТООБОРОТА НА ОСНОВЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ И НЕЙРОСЕТЕВЫХ МЕХАНИЗМОВ BERT

ДИССЕРТАЦИЯ

на соискание ученой степени доктора философии (PhD)
по специальности 6D060200 – Информатика (6D060204 – Теоретические основы информатики)

Научный руководитель:
Доктор физико-математических наук,
профессор Комилиён Ф.С.

Душанбе-2026

СОДЕРЖАНИЕ

СЛОВАРЬ СИСТЕМНЫХ ТЕРМИНОВ ДИССЕРТАЦИИ ……. 4
ВВЕДЕНИЕ ……. 14
ОБЩАЯ ХАРАКТЕРИСТИКА ИССЛЕДОВАНИЯ ……..…… 19
ГЛАВА 1. ОБЗОР ЛИТЕРАТУРЫ: ВВЕДЕНИЕ В МИКРОСЕРВИСНУЮ АРХИТЕКТУРУ И ИНТЕЛЛЕКТУАЛЬНЫЙ ДОКУМЕНТООБОРОТ НА ОСНОВЕ BERT ……..…… 30
1.1. Обзор литературы по микросервисной архитектуре и технологиям NLP ……..…… 30
1.2 Преимущества и недостатки микросервисов в интеллектуальных системах 38
1.3 Основные принципы построения микросервисных систем документооборота с BERT 41
1.4 Методы расширения и масштабирования интеллектуальных микросервисов 44
Выводы по первой главе ……..…..… 50
ГЛАВА 2. АРХИТЕКТУРНЫЕ РЕШЕНИЯ ДЛЯ МАСШТАБИРОВАНИЯ МИКРОСЕРВИСОВ 53
2.1 Горизонтальное и вертикальное масштабирование в микросервисных архитектурах 53
2.2 Использование контейнеров для управления микросервисами 57
2.3 Автоматическое масштабирование с помощью оркестраторов 62
Выводы по второй главе ……..……..…..… 67
ГЛАВА 3. СРАВНИТЕЛЬНЫЙ АНАЛИЗ МОНОЛИТНОЙ И МИКРОСЕРВИСНОЙ АРХИТЕКТУР В КОНТЕКСТЕ ТРАНСФОРМАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ И ОПТИМИЗАЦИИ ИТ-РЕСУРСОВ 71
3.1. Микросервисная архитектура: от монолита к гибким распределенным системам ……..……..……..… 71
3.2. Анализ характеристик монолитных и микросервисных архитектур 83
3.3. Внедрение микросервисной архитектуры для оптимизации

распределения информационных ресурсов …….…..……..…… 91
Выводы по третьей главе ……..……..……..…..…… 103
ГЛАВА 4. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ МИКРОСЕРВИСНОЙ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ДОКУМЕНТООБОРОТА С ИСПОЛЬЗОВАНИЕМ МОДЕЛЕЙ BERT 105
4.1. Интеллектуальная система документооборота на основе микросервисной архитектуры и моделей BERT …….……...… 105
4.2. Разработка математической модели системы документооборота: микросервисы и нейросетевые технологии BERT ....……...… 113
4.3. Микросервисная архитектура и BERT-модели в интеллектуальном документообороте профессионального образования 128
4.4. Алгоритмизация микросервисной модели интеллектуального документооборота в профобразовании на базе BERT 141
Выводы по четвёртой главе ……..…… 156
ВЫВОДЫ ……..……..……...……...……. 159
1. Основные научные результаты диссертации ……..……..… 159
2. Рекомендации по практическому использованию результатов ……. 160
ЛИТЕРАТУРА ……..…..… 163
1. Список использованных источников ……...…….…… 163
2. Список научных публикаций соискателя ……...…….…… 179
ПРИЛОЖЕНИЯ ……...……...……..…… 182

СЛОВАРЬ СИСТЕМНЫХ ТЕРМИНОВ ДИССЕРТАЦИИ

English Term	Русский термин и определение
Activation Function	**Функция активации** – математическая функция, определяющая выходное значение нейрона на основе входного сигнала.
Adaptive System	**Адаптивная система** – система, способная автоматически изменять параметры функционирования в зависимости от изменений внешней среды и текущего состояния.
Aggregation Mechanism	**Механизм агрегации** – способ объединения признаков или результатов вычислений из нескольких источников.
AI Service	**Сервис искусственного интеллекта** – программный сервис, реализующий функции машинного обучения и интеллектуальной обработки информации.
Amazon ECS	**Amazon Elastic Container Service (ECS)** – облачная платформа компании Amazon для управления контейнеризированными приложениями.
Amdahl's Law	**Закон Амдала** – принцип оценки предельного ускорения вычислений при параллельном выполнении задач.
AMQP	**Протокол очередей сообщений AMQP** – стандарт взаимодействия распределённых приложений посредством обмена сообщениями.
Apache Kafka	**Apache Kafka** – распределённая платформа потоковой обработки и передачи событий в режиме реального времени.
API (Application Programming Interface)	**Интерфейс прикладного программирования** – набор правил и средств взаимодействия между программными компонентами.
API Gateway	**API-шлюз** – централизованный компонент, обеспечивающий маршрутизацию, безопасность и управление запросами к микросервисам.
Asynchronous System (Asyncio)	**Асинхронная система (Asyncio)** – программная система, выполняющая несколько операций параллельно без блокирования основного потока исполнения.
Attention Mechanism	**Механизм внимания** – алгоритм выделения наиболее значимых элементов входных данных для последующей обработки.
Backend	**Серверная часть приложения** – программные компоненты, выполняющие обработку данных и бизнес-логику на стороне сервера.
Backend & AI	**Серверная платформа и искусственный интеллект** – совокупность серверных сервисов и интеллектуальных модулей обработки данных.
Batching	**Пакетная обработка данных** – объединение нескольких запросов или объектов данных в один пакет для эффективной обработки.
BERT (Bidirectional Encoder Representations from Transformers)	**BERT** – двунаправленная трансформерная языковая модель, предназначенная для извлечения контекстных представлений текста и решения задач обработки естественного языка.
BERT Models	**Модели семейства BERT** – совокупность языковых моделей, основанных на архитектуре BERT и адаптированных для различных задач анализа текста.

BiLSTM-CRF	**Двунаправленная модель BiLSTM-CRF** – архитектура для последовательной разметки текста, сочетающая рекуррентные сети и условные случайные поля.
BIO Scheme	**Схема BIO** – формализованный способ обозначения границ сущностей в текстовых данных.
BIO Tagging	**BIO-разметка** – схема разметки текста для выделения именованных сущностей с использованием меток Begin, Inside и Outside.
Business Domain	**Бизнес-домен** – предметная область деятельности организации, в рамках которой функционирует информационная система.
Business Logic	**Бизнес-логика** – совокупность правил, алгоритмов и процессов, реализующих функциональные требования предметной области.
Business Opportunity	**Бизнес-возможность** – потенциальное направление развития или улучшения деятельности организации за счет внедрения новых технологий или решений.
Business Task	**Бизнес-задача** – прикладная задача, направленная на достижение конкретной цели организации.
Caching	**Кэширование** – временное хранение часто используемых данных для ускорения доступа к ним.
Chatbot	**Чат-бот** – программный агент, осуществляющий автоматизированное взаимодействие с пользователем посредством текстового или голосового интерфейса.
CI/CD	**CI/CD-конвейер** – совокупность процессов непрерывной интеграции, тестирования и развертывания программного обеспечения.
CI/CD Pipeline	**Конвейер CI/CD** – автоматизированная последовательность этапов сборки, тестирования и публикации программного продукта.
CircleCI	**CircleCI** – облачная система автоматизации процессов непрерывной интеграции и развертывания.
Circuit Breaker	**Предохранитель отказов** – механизм предотвращения каскадного распространения ошибок между сервисами.
Cloud Infrastructure	**Облачная инфраструктура** – совокупность вычислительных, сетевых и программных ресурсов, предоставляемых через облачные платформы.
Cluster	**Кластер** – группа взаимосвязанных вычислительных узлов, функционирующих как единая система.
Cluster Autoscaling	**Автоматическое масштабирование кластера** – динамическое изменение количества вычислительных узлов в зависимости от нагрузки.
Confidence Score	**Оценка уверенности модели** – числовая характеристика достоверности предсказания модели.
Container Runtime	**Среда выполнения контейнеров** – программное обеспечение для запуска и управления контейнерами.
Containerization	**Контейнеризация** – технология упаковки приложения вместе со всеми зависимостями в изолированный контейнер.
Continuous Deployment (CD)	**Непрерывное развертывание** – автоматизированный процесс доставки изменений программного обеспечения в рабочую среду.

Continuous Integration (CI)	**Непрерывная интеграция** – практика регулярного объединения изменений программного кода в общий репозиторий с автоматической проверкой корректности.
Control Flow	**Управляющий поток** – последовательность выполнения операций и передачи управления внутри системы.
Conway's Law	**Закон Конвея** – принцип соответствия архитектуры программной системы структуре коммуникаций организации-разработчика.
CQRS	**Разделение команд и запросов (CQRS)** – архитектурный паттерн, разделяющий операции чтения и изменения данных.
Cross-Entropy	**Кросс-энтропия** – функция потерь, измеряющая различие между истинным и прогнозируемым распределениями вероятностей.
Cross-Functional Team	**Кросс-функциональная команда** – группа специалистов различных направлений, совместно участвующих в разработке программного продукта.
Crowdsourcing	**Краудсорсинг** – привлечение широкого круга пользователей к решению задач посредством цифровых платформ.
Data Augmentation	**Аугментация данных** – искусственное расширение обучающего набора данных для повышения качества модели.
Data Flow	**Поток данных** – направленное движение данных между компонентами системы.
Dataset	**Набор данных (датасет)** – структурированная совокупность данных, используемая для обучения, тестирования или валидации моделей.
Declarative Infrastructure Management	**Декларативное управление инфраструктурой** – способ управления инфраструктурой через описание желаемого состояния системы.
DevOps	**DevOps** – методология организации разработки и эксплуатации программного обеспечения, направленная на ускорение выпуска и повышение качества программных продуктов.
DevOps Approach	**DevOps-подход** – совокупность организационных и технологических практик, обеспечивающих непрерывную интеграцию разработки и эксплуатации.
DevOps Engineer	**DevOps-инженер** – специалист, отвечающий за автоматизацию процессов разработки, тестирования, развертывания и сопровождения программных систем.
DevOps Practitioner	**Практик DevOps** – специалист, применяющий DevOps-методологию в процессах разработки и эксплуатации программного обеспечения.
DevOps Specialist	**DevOps-специалист** – специалист по внедрению инструментов автоматизации и управлению жизненным циклом программных продуктов.
Distributed Computing	**Распределённые вычисления** – вычисления, выполняемые одновременно на нескольких взаимосвязанных узлах.
Distributed Information System	**Распределённая информационная система** – информационная система, компоненты которой размещены на нескольких вычислительных узлах.
Docker	**Docker** – платформа контейнеризации приложений и сервисов.

Docker Compose	**Docker Compose** – инструмент оркестрации многоконтейнерных приложений.
Docker Swarm	**Docker Swarm** – встроенная система кластеризации и оркестрации контейнеров Docker.
Document Classification	**Классификация документов** – процесс автоматического распределения документов по заранее заданным категориям.
Document-Oriented NoSQL Database	**Документо-ориентированная NoSQL база данных** – система хранения данных, использующая документы вместо таблиц реляционной модели.
DOM (Document Object Model)	**Объектная модель документа (DOM)** – программное представление структуры веб-документа в виде дерева объектов.
Elasticsearch	**Elasticsearch** – распределённая поисковая и аналитическая система для работы с большими объёмами данных.
ELK Stack	**Стек ELK** – комплекс Elasticsearch, Logstash и Kibana для мониторинга и анализа данных.
Embedding Matrix	**Матрица эмбеддингов** – матрица векторных представлений слов или токенов, используемая в нейросетевых моделях.
Event Bus	**Шина событий** – инфраструктурный компонент для передачи событий между сервисами системы.
Event Sourcing	**Хранение на основе событий** – подход, при котором состояние системы восстанавливается по журналу событий.
Event-Driven Architecture (EDA)	**Событийно-ориентированная архитектура** – архитектура, в которой взаимодействие компонентов осуществляется посредством генерации и обработки событий.
$\mathcal{F}_1$-Score	**$\mathcal{F}_1$-мера** – гармоническое среднее между точностью и полнотой классификации.
Fallback Mechanism	**Резервный механизм обработки** – механизм переключения на альтернативный сценарий выполнения при отказе основной функции.
False Negatives (FN)	**Ложноотрицательные результаты** – объекты, ошибочно отнесённые к отрицательному классу.
False Positives (FP)	**Ложноположительные результаты** – объекты, ошибочно отнесённые к положительному классу.
FastAPI	**FastAPI** – высокопроизводительный фреймворк для разработки веб-сервисов на языке Python.
FastAPI + PyTorch	**Интеграция FastAPI и PyTorch** – программное решение для предоставления нейросетевых моделей через веб-интерфейсы.
Fault Tolerance	**Отказоустойчивость** – способность системы продолжать функционирование при отказе отдельных компонентов.
Fine-Tuned Model	**Дообученная модель** – модель, адаптированная к конкретной предметной области после дополнительного обучения.
Fine-Tuning	**Дообучение модели** – дополнительное обучение предварительно обученной модели на специализированном наборе данных.
GitLab CI/CD	**GitLab CI/CD** – встроенная система автоматизации процессов разработки и развертывания в платформе GitLab.
GloVe (Global Vectors)	**GloVe** – метод построения векторных представлений слов на основе статистики совместной встречаемости слов в корпусе текстов.

Grafana	**Grafana** – платформа визуализации метрик и мониторинга информационных систем.
gRPC	**gRPC** – высокопроизводительная технология удалённого вызова процедур, основанная на Protocol Buffers и HTTP/2.
Health Check	**Проверка работоспособности** – механизм контроля состояния приложения или сервиса.
High Availability	**Высокая доступность** – способность системы обеспечивать непрерывное предоставление сервисов пользователям.
Horizontal Pod Autoscaling (HPA)	**Горизонтальное масштабирование подов** – автоматическое изменение числа экземпляров приложения в Kubernetes.
Horizontal Scaling	**Горизонтальное масштабирование** – увеличение производительности системы путем добавления новых вычислительных узлов.
HTTP API	**HTTP-интерфейс** – программный интерфейс, функционирующий поверх протокола HTTP.
HTTP Request	**HTTP-запрос** – сообщение клиента серверу для получения или изменения данных.
HTTP/2	**Протокол HTTP/2** – усовершенствованная версия HTTP с поддержкой мультиплексирования и высокой производительности передачи данных.
ICT (Information and Communication Technologies)	**Информационно-коммуникационные технологии (ИКТ)** – совокупность методов и средств обработки, передачи и хранения информации.
Image Registry	**Реестр контейнерных образов** – централизованное хранилище Docker-образов и других контейнерных пакетов.
Inference	**Инференс** – этап практического применения обученной модели для получения прогнозов на новых данных.
Information System (IS)	**Информационная система (ИС)** – организационно-техническая система для хранения, обработки и передачи информации.
Infrastructure as Code (IaC)	**Инфраструктура как код** – подход к управлению инфраструктурой посредством программного описания её конфигурации.
Instance	**Экземпляр (инстанс)** – отдельный запущенный экземпляр приложения, сервиса или виртуальной машины.
Intelligent Document Management System	**Интеллектуальная система документооборота** – информационная система, использующая методы искусственного интеллекта для обработки, анализа и маршрутизации документов.
Istio	**Istio** – платформа Service Mesh для управления безопасностью, мониторингом и маршрутизацией сервисов.
IT Architecture	**ИТ-архитектура** – совокупность принципов, компонентов и взаимосвязей информационной системы.
IT Community	**ИТ-сообщество** – профессиональное сообщество специалистов в области информационных технологий.
IT Industry	**ИТ-сфера** – совокупность отраслей, связанных с разработкой, внедрением и эксплуатацией информационных технологий.

IT Infrastructure	**ИТ-инфраструктура** – комплекс аппаратных, программных и сетевых средств обеспечения функционирования информационной системы.
IT Resource	**ИТ-ресурс** – вычислительный, программный, сетевой или информационный ресурс системы.
IT Startup	**ИТ-стартап** – инновационный технологический проект, ориентированный на создание нового цифрового продукта или сервиса.
Jenkins	**Jenkins** – программная платформа автоматизации процессов сборки, тестирования и доставки программного обеспечения.
JSON (JavaScript Object Notation)	**JSON** – текстовый формат обмена структурированными данными между программными компонентами.
JSON Web Token (JWT)	**JWT-токен** – компактный формат передачи данных для аутентификации и авторизации пользователей.
JWT Authentication	**JWT-аутентификация** – механизм подтверждения личности пользователя посредством JWT-токенов.
JWT Authorization	**JWT-авторизация** – механизм предоставления прав доступа на основе JWT-токенов.
JWT Session	**JWT-сессия** – механизм хранения информации о сеансе пользователя посредством токенов.
Kibana	**Kibana** – система визуализации и анализа данных Elasticsearch.
Kubernetes	**Kubernetes** – платформа автоматического управления контейнеризированными приложениями.
Kubernetes Dashboard	**Панель управления Kubernetes** – веб-интерфейс для мониторинга и администрирования Kubernetes-кластера.
Latency	**Задержка обработки** – время между поступлением запроса и получением результата.
Linkerd	**Linkerd** – лёгкая сервисная сетка для управления взаимодействием микросервисов.
Load Balancing	**Балансировка нагрузки** – распределение вычислительных задач между узлами системы для повышения производительности и устойчивости.
Location	**Локация** – логическое или физическое местоположение объекта, сервиса либо пользователя в информационной системе.
Logging	**Логирование** – процесс регистрации событий и действий, происходящих в системе.
Logit	**Логит** – значение выхода нейронной сети до применения функции нормализации (например, Softmax).
Logstash	**Logstash** – инструмент сбора, преобразования и передачи журналов событий.
Low-Resource Language	**Низкоресурсный язык** – язык, для которого доступно ограниченное количество цифровых текстовых ресурсов и размеченных данных.
Machine Learning	**Машинное обучение** – раздел искусственного интеллекта, изучающий методы автоматического построения моделей на основе данных.
Message Broker	**Брокер сообщений** – программный компонент, обеспечивающий передачу сообщений между сервисами.

Microservice Architecture (MSA)	**Микросервисная архитектура** – архитектурный стиль построения программных систем в виде набора независимых сервисов, взаимодействующих через стандартизированные интерфейсы.
ML Library	**Библиотека машинного обучения** – программный пакет для построения, обучения и применения моделей машинного обучения.
ML Model	**Модель машинного обучения** – математическая модель, обученная на данных для решения задач прогнозирования или классификации.
ML Module	**Модуль машинного обучения** – программный компонент, реализующий функции машинного обучения.
ML Technology	**Технология машинного обучения** – совокупность методов, алгоритмов и инструментов машинного обучения.
Monolithic System	**Монолитная система** – программная система, в которой все функциональные компоненты реализованы в едином приложении и развёртываются как единое целое.
Multi-Head Attention	**Механизм многоголового внимания** – расширение механизма внимания, позволяющее одновременно анализировать различные типы зависимостей в данных.
Named Entity Extraction (NER)	**Извлечение сущностей** – процесс автоматического обнаружения и выделения именованных объектов из текстовой информации.
Natural Language Processing (NLP)	**Обработка естественного языка (NLP)** – область искусственного интеллекта, связанная с анализом, пониманием и генерацией текстовой информации.
Neural Network Inference	**Инференс нейросетевой модели** – процесс применения обученной нейронной сети для получения прогнозов или классификаций на новых данных.
NLP Module	**NLP-модуль** – программный компонент, реализующий методы обработки естественного языка.
NLP Service	**NLP-сервис** – специализированный сервис для автоматического анализа текстовой информации.
NLP Solution	**NLP-решение** – комплекс программных средств для решения задач обработки естественного языка.
NLP Technology	**NLP-технология** – совокупность методов и инструментов обработки текстовых данных.
Node.js	**Node.js** – серверная среда выполнения JavaScript-приложений.
OAuth2	**OAuth 2.0** – протокол авторизации, обеспечивающий безопасный доступ к ресурсам от имени пользователя.
OAuth2 + JWT	**Комбинированная схема OAuth2 и JWT** – современный механизм безопасной аутентификации и авторизации в распределённых системах.
Observability	**Наблюдаемость системы** – способность оценивать внутреннее состояние системы на основе её внешних метрик, журналов и трассировок.
OCR (Optical Character Recognition)	**Оптическое распознавание символов** – технология автоматического преобразования изображений текста в машинно-читаемый формат.
One-Hot Encoding	**One-Hot кодирование** – способ представления категориальных признаков в виде бинарных векторов.

One-Hot Label	**Истинная метка (One-Hot Encoding)** – способ представления категориальных данных в виде бинарного вектора.
Optimization Mechanism	**Механизм оптимизации** – алгоритм настройки параметров модели для минимизации функции ошибки.
Parsing	**Парсинг** – процесс синтаксического анализа и структурирования данных.
Precision	**Точность (Precision)** – доля правильно определённых положительных объектов среди всех объектов, отнесённых моделью к положительному классу.
Production Deployment	**Промышленное развертывание** – процесс внедрения программной системы в рабочую эксплуатационную среду.
Prometheus	**Prometheus** – система мониторинга и сбора метрик распределённых приложений.
Prometheus + Grafana	**Комплекс мониторинга Prometheus-Grafana** – решение для сбора, анализа и визуализации эксплуатационных показателей системы.
Protocol Buffers	**Protocol Buffers** – механизм сериализации структурированных данных, разработанный компанией Google.
PyTorch	**PyTorch** – программная библиотека глубокого обучения для создания и обучения нейронных сетей.
RabbitMQ	**RabbitMQ** – брокер сообщений с поддержкой протокола AMQP для организации асинхронного взаимодействия сервисов.
React	**React** – библиотека JavaScript для разработки пользовательских интерфейсов.
Recall	**Полнота (Recall)** – доля правильно найденных положительных объектов среди всех реально существующих положительных объектов.
Redis	**Redis** – высокопроизводительная система хранения данных в оперативной памяти, часто используемая для кэширования.
Regularization	**Регуляризация** – методы предотвращения переобучения модели за счет ограничения её сложности.
Reliability	**Надёжность системы** – способность системы корректно выполнять функции в течение заданного периода времени.
Resilience	**Устойчивость к сбоям** – способность системы восстанавливаться после отказов и поддерживать работоспособность.
REST API	**REST API** – программный интерфейс, реализующий архитектурный стиль REST для взаимодействия распределённых систем.
REST Approach	**REST-подход** – архитектурная концепция взаимодействия распределённых компонентов через ресурсы и HTTP-методы.
REST Interaction	**REST-взаимодействие** – обмен данными между сервисами по протоколу HTTP с использованием REST-подхода.
RESTful API	**RESTful API** – программный интерфейс, полностью соответствующий принципам REST-архитектуры.
Retry Mechanism	**Механизм повторных попыток** – механизм автоматического повторного выполнения операции при возникновении временных ошибок.

Rolling Updates	**Пошаговое обновление сервисов** – обновление приложения без остановки системы.
Rule-Based Algorithm	**Алгоритм, основанный на правилах** – алгоритм принятия решений на основе заранее заданного набора логических правил и условий.
Rule-Based Method	**Правилоориентированный метод** – подход к обработке данных, основанный на заранее определённых логических правилах.
Saga Pattern	**Паттерн Saga** – механизм управления распределёнными транзакциями в микросервисной архитектуре.
Scalability	**Масштабируемость** – способность системы сохранять эффективность при увеличении нагрузки.
Scalable Cluster	**Масштабируемый кластер** – кластер, способный автоматически увеличивать или уменьшать вычислительные ресурсы в зависимости от нагрузки.
Self-Attention	**Механизм самовнимания** – механизм, позволяющий модели определять важность различных элементов входной последовательности относительно друг друга.
Self-Healing	**Самовосстановление системы** – автоматическое восстановление работоспособности после сбоев.
Semantic Analysis	**Семантический анализ** – процесс определения смыслового содержания текста и взаимосвязей между его элементами.
Semi-Supervised Learning	**Полуконтролируемое обучение** – метод обучения моделей на сочетании размеченных и неразмеченных данных.
Service Discovery	**Обнаружение сервисов** – автоматическое определение местоположения и параметров сервисов в распределённой системе.
Service Mesh	**Сервисная сетка** – инфраструктурный слой управления взаимодействием между микросервисами.
Service-Oriented Architecture (SOA)	**Сервис-ориентированная архитектура** – архитектурный подход, основанный на организации функциональности в виде взаимосвязанных сервисов.
Smart Assistant	**Интеллектуальный помощник** – программная система, использующая методы искусственного интеллекта для поддержки пользователя при выполнении различных задач.
Starlette	**Starlette** – асинхронный программный фреймворк, лежащий в основе FastAPI.
Stress Testing	**Стресс-тестирование** – метод оценки устойчивости системы при экстремальных нагрузках.
Subtoken	**Субтокен** – часть токена, образованная алгоритмами разбиения слов на подсловные единицы.
Subword Tokenization	**Подсловная токенизация** – метод разбиения слов на более мелкие смысловые единицы для повышения качества обработки текста.
Synthetic Data Generation	**Генерация обучающих данных** – процесс создания искусственных данных для обучения моделей машинного обучения.
Thrashing	**Эффект трешинга** – резкое снижение производительности системы вследствие чрезмерного переключения ресурсов между задачами.

Throughput	**Пропускная способность системы** – количество запросов или операций, обрабатываемых системой за единицу времени.
Token	**Токен** – минимальная единица текста, обрабатываемая языковой моделью.
Transformer Architecture	**Трансформерная архитектура** – архитектура нейронных сетей, основанная на механизме внимания (Attention) и предназначенная для обработки последовательностей данных.
Transformer Model	**Трансформерная модель** – нейросетевая модель, использующая механизм внимания для обработки последовательных данных без рекуррентных связей.
Transformer-based BERT Architecture	**Трансформерная архитектура BERT** – архитектура глубокого обучения, построенная на двунаправленных блоках самовнимания (Self-Attention).
True Positives (TP)	**Истинно положительные результаты** – объекты, корректно отнесённые к положительному классу.
Validation	**Валидация** – процедура проверки соответствия модели, системы или данных установленным требованиям.
Vertical Pod Autoscaling (VPA)	**Вертикальное масштабирование подов** – автоматическое изменение ресурсов процессора и памяти контейнеров.
Vertical Scaling	**Вертикальное масштабирование** – увеличение вычислительных ресурсов отдельного узла путем наращивания мощности процессора, памяти или дискового пространства.
WebSocket	**WebSocket** – протокол двустороннего обмена данными между клиентом и сервером в режиме реального времени.
Word2Vec	**Word2Vec** – алгоритм преобразования слов в плотные числовые векторы, отражающие их семантическую близость.

ВВЕДЕНИЕ

Актуальность темы исследования. Рост объёмов данных, усложнение внутренних коммуникаций и постоянное давление на скорость управленческих решений привели к тому, что традиционные модели документооборота всё чаще упираются в ограничения по производительности и по устойчивости обработки информационных потоков. В организациях с сильным документооборотом крупные объёмы информации, различающейся по структуре и по назначению, накапливаются постоянно. Указанные проблемы в этих условиях заметны сильнее. Большую часть процедур всё ещё либо выполняют вручную, либо регулярно привлекают к ним ручное участие. Вероятность ошибок из-за этого растёт, контроль усложняется. Дополнительное время уходит на принятие решений.

Под влиянием распределённых вычислительных технологий изменились и требования к современным системам. Приоритетными считают масштабируемость и устойчивость к сбоям. От систем также ждут способности держать производительность при меняющейся нагрузке. Архитектуры, разделяющие систему на независимые сервисы, выбирают всё чаще. Обработка данных при микросервисной архитектуре становится гибкой. Развитие программной системы упрощается за счёт декомпозиции на сервисы. Стабильность работы сохраняется даже при значительных нагрузках.

Цифровизацию государственного управления в Республике Таджикистан относят к приоритетным направлениям развития. Стратегические и нормативные акты фиксируют этот приоритет; среди них Национальная стратегия развития Республики Таджикистан на период до 2030 года, Концепция формирования электронного правительства в Республике Таджикистан, Программа реализации концепции государственной информационной политики Республики Таджикистан. В названных

документах ставят задачи развития цифровой инфраструктуры и более широкого применения интеллектуальных методов обработки информации.

Указанные приоритеты увеличивают спрос на интеллектуальные системы документооборота. Им нужно хранить документы и передавать их между подразделениями. Параллельно требуется автоматический анализ содержания с извлечением важных данных, определением категории документа и поддержкой принятия решений. Чтобы такие системы появились, методы математического моделирования объединяют с технологиями вычислений и с современными алгоритмами обработки текста.

В последние годы быстро продвинулись методы обработки естественного языка. Трансформерные модели BERT (Bidirectional Encoder Representations from Transformers - Двунаправленные кодировщики-представления из трансформеров) расширили средства интеллектуального анализа текстов. С их помощью решают семантическую классификацию, извлечение сущностей, контекстную интерпретацию документов и автоматизированный анализ содержания точнее, чем при традиционных подходах. Однако внедрение этих моделей в системы документооборота сталкивается с практическими ограничениями. Особенно остро ограничения проявляются для низкоресурсных языков, где нет крупных размеченных корпусов и не хватает адаптированных языковых моделей.

Отдельная трудность состоит в том, чтобы встроить интеллектуальные алгоритмы в распределённую вычислительную среду. Во многих работах архитектурное проектирование и интеллектуальный анализ документов изучают разобщенно. Одни авторы сосредоточены на микросервисной организации программных систем, другие на отдельных алгоритмах NLP или на нейросетевых моделях обработки текста. При этом проблемы комплексного объединения микросервисной архитектуры и математических моделей интеллектуального анализа документов изучены ещё недостаточно.

Важность темы темы связана также с адаптацией интеллектуальных систем к условиям реального государственного управления. В качестве среды в настоящем исследовании рассматривается Комитет по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан. Такой выбор позволяет учесть действующие регламенты обработки документов, как устроено учреждение и информационные потоки, которые возникают в ходе его работы. Реальные условия делает разработанные решения более подходящими для дальнейшего внедрения.

Актуальность диссертации связана с некоторыми обстоятельствами. Требуется повысить эффективность документооборота, также возрастает потребность в масштабируемых и отказоустойчивых архитектурах, рассчитанных на интенсивную обработку данных. Инструментарий интеллектуального анализа текстовой информации тоже расширяется. Существующие NLP-решения слабо приспособлены к низкоресурсным языковым средам, и это ограничивает их практическое применение. Сохраняется задача построения математически обоснованных моделей интеллектуальной обработки документов.

Степень исследования научной темы. Научная тема «Математическое моделирование интеллектуальной системы документооборота на основе микросервисной архитектуры и нейросетевых механизмов BERT» исследована на уровне фундаментальных и прикладных аспектов, включая разработку математических моделей, проектирование архитектуры системы, методы адаптации трансформерных моделей к низкоресурсным языкам, а также их практическую реализацию и апробацию в условиях реальной информационной инфраструктуры.

При разработке комплекса компьютерных моделей использованы результаты исследований в области обработки естественного языка. Существенный вклад внесли Ashish Vaswani [3; 165], Jacob Devlin [13; 130] и

Thomas Wolf [166], заложившие основы трансформерной архитектуры и модели BERT, обеспечивающей высокую точность анализа текстовых данных.

Архитектурные аспекты опираются на работы Sam Newman [85; 146-148], Martin Fowler [109; 134-136] и Thomas Erl [119; 132], в которых микросервисная архитектура рассматривается как эффективный подход к построению масштабируемых и отказоустойчивых систем. Развитие данных идей отражено в исследованиях Chris Richardson [97; 157], формализовавшего ключевые паттерны распределённых приложений.

Особое значение в контексте интеллектуальных систем приобретают технологии контейнеризации и оркестрации. Использование платформ Docker (платформа контейнеризации приложений и сервисов) и Kubernetes (платформа автоматического управления контейнеризированными приложениями), как показано в работах Chris Richardson [97; 157], а также Brendan Burns и Joe Beda [126] обеспечивает изоляцию сервисов, автоматизацию развёртывания и эффективное управление масштабированием, что позволяет интегрировать ресурсоёмкие компоненты машинного обучения в распределённую инфраструктуру. Это позволяет интегрировать вычислительно сложные компоненты, включая модели машинного обучения, в распределённую инфраструктуру с возможностью динамического перераспределения ресурсов.

В отечественных исследованиях рассматривались вопросы проектирования распределённых систем и оптимизации обработки больших объёмов текстовой информации в условиях ограниченных вычислительных ресурсов [4-6; 8; 16-18; 24-66; 80; 98-102; 111; 1-А–12-А].

Проведённый анализ показал, что интеграция микросервисной архитектуры с моделями BERT (модели семейства BERT – совокупность языковых моделей, основанных на архитектуре BERT и адаптированных для различных задач анализа текста) формирует основу интеллектуальных систем

документооборота, обеспечивающих автоматическое извлечение и структурирование данных. Вместе с тем недостаточно исследованы вопросы адаптации моделей к низкоресурсным языкам и их эффективной интеграции в распределённые среды.

Методологическую основу исследования составили работы отечественных и зарубежных учёных в области математического моделирования, распределённых систем и машинного обучения: Ф.С. Комилиён, З.Дж. Усманов, М.К. Юнуси, М.Р. Ёров, Х.А. Худойбердиев, В.В. Меншуткин, M. Fowler, S. Newman, L. Bass, I. Weber, L. Zhu, B. Burns, J. Beda, K. Hightower, N. Dragoni, C. Pahl, Q. Zhang, M. Chen, J. Devlin, M.W. Chang, K. Lee, K. Toutanova, A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, A. Rogers, T. Wolf, C. Richardson, T. Brown, P. Jamshidi, D. Taibi, T. Erl, J. Thönes, N. Alshuqayran, H. Zhang, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, J. Lewis и другие.

В рамках диссертации получили развитие следующие направления: формализация процессов обработки документов, разработка микросервисной архитектуры с выделенным интеллектуальным компонентом, адаптация моделей BERT для низкоресурсного языка (язык, для которого доступно ограниченное количество цифровых текстовых ресурсов и размеченных данных), а также создание практического алгоритма реализации системы, ориентированного на условия функционирования государственных структур. Особое внимание в исследовании уделено задачам масштабируемости, отказоустойчивости и эффективного использования вычислительных ресурсов.

Несмотря на достигнутые результаты, остаются открытыми задачи повышения производительности при больших объёмах данных, расширения мультиязычной поддержки и интеграции с государственными системами электронного документооборота.

Таким образом, проведённое исследование занимает промежуточное положение между фундаментальными разработками трансформерных моделей и их прикладным внедрением в интеллектуальные системы документооборота, закрывая ряд ранее недостаточно изученных аспектов, связанных с адаптацией моделей BERT к низкоресурсным языкам и комбинированным масштабированием микросервисных архитектур.

Связь исследований с программами и научными темами. Диссертационное исследование выполнено в соответствии с приоритетными направлениями научно-технического и цифрового развития Республики Таджикистан и направлено на развитие методов математического моделирования, интеллектуальных информационных систем и технологий обработки данных.

Тематика диссертации непосредственно связана с реализацией государственных стратегий и программ в области цифровизации, включая Национальной стратегии развития Республики Таджикистан до 2030 года [81], Концепции формирования электронного правительства в Республике Таджикистан [71] и Программы цифровой экономики Республики Таджикистан, в рамках которых предусмотрено внедрение современных информационных технологий, систем электронного документооборота и методов искусственного интеллекта в процессы государственного управления.

Диссертация выполнена в рамках научного направления кафедры информатики Таджикского национального университета «Разработка и исследование математико-компьютерных моделей экосистем водоёмов, заповедников и системы образования», зарегистрированного в Национальном патентно-информационным центром Республики Таджикистан (№ 0116TJ00657), и расширяет его за счёт применения методов математического моделирования и искусственного интеллекта к задачам автоматизации документооборота.

Результаты исследования ориентированы на разработку и внедрение интеллектуальных систем документооборота, основанных на микросервисной архитектуре и трансформерных моделях типа BERT, что соответствует современным тенденциям цифровой трансформации и повышению эффективности управления информационными ресурсами.

ОБЩАЯ ХАРАКТЕРИСТИКА ИССЛЕДОВАНИЯ

Диссертационное исследование посвящено разработке и математическому обоснованию интеллектуальной системы документооборота, реализованной на основе микросервисной архитектуры и трансформерных моделей типа BERT, обеспечивающей повышение эффективности обработки документов, а также обладающей свойствами масштабируемости и адаптивности к изменяющимся условиям эксплуатации.

Цель исследования заключается в разработке и теоретико-прикладном обосновании математической модели интеллектуальной системы документооборота, построенной с использованием микросервисной архитектуры и нейросетевых механизмов семейства BERT, обеспечивающей повышение точности обработки документов, устойчивость функционирования системы и возможность её масштабирования (способность системы сохранять эффективность при увеличении нагрузки) в условиях распределённой вычислительной среды.

Поставленная цель обусловила необходимость решения комплекса взаимосвязанных исследовательских и прикладных задач, охватывающих как архитектурные, так и алгоритмические аспекты построения интеллектуальной системы документооборота.

Задачи исследования:

1. Провести анализ современных подходов к построению распределённых информационных систем, включая исследование

особенностей микросервисной архитектуры, её преимуществ, ограничений и областей практического применения.

2. Исследовать существующие методы масштабирования микросервисных приложений и механизмы обеспечения отказоустойчивости в распределённых программных средах с высокой интенсивностью обработки запросов.

3. Разработать архитектурную модель интеллектуальной системы документооборота с учётом требований к модульности, расширяемости, устойчивости и взаимодействию отдельных сервисов.

4. Выполнить адаптацию трансформерных моделей типа BERT для решения задач извлечения сущностей, семантического анализа и классификации документов в условиях ограниченности языковых ресурсов.

5. Разработать методы формирования и генерации обучающих данных, ориентированные на повышение эффективности обучения нейросетевых моделей обработки текстовой информации.

6. Реализовать программный прототип интеллектуальной системы документооборота с последующей экспериментальной оценкой эффективности предложенных моделей, алгоритмов и архитектурных решений на основе совокупности количественных показателей.

Объект исследования – интеллектуальные системы документооборота, функционирующие в условиях цифровой трансформации и построенные на основе распределённых микросервисных архитектур.

Предмет исследования – методы и модели математического обеспечения, алгоритмы и архитектурные решения построения и масштабирования интеллектуальной системы документооборота с

использованием трансформерных моделей типа BERT в микросервисной среде.

Теоретическую основу исследования составляют концепции математического и компьютерного моделирования информационных систем, микросервисная архитектура и интеллектуальный подход к процессам автоматизации документооборота. Связь темы с существующими теориями проявляется в критическом анализе трансформерных моделей типа BERT и решении проблем адаптации многоязычных моделей в условиях ограниченности ресурсов малоресурсных языков. Разработка комплекса математических моделей, гибридных методов обработки текста и архитектурных решений обеспечила теоретическую базу для проведения компьютерных экспериментов и научного управления документооборотом.

Научная новизна диссертационной работы связана с разработкой теоретических и прикладных подходов к построению интеллектуальных систем документооборота, основанных на совместном использовании микросервисной архитектуры, методов математического моделирования и трансформерных моделей обработки естественного языка. В процессе исследования получены результаты, обладающие как элементами научной новизны, так и практической значимостью.

1. Разработана математическая модель интеллектуальной системы документооборота, в которой объединены принципы микросервисной организации вычислительной среды и трансформерные механизмы обработки текстовых данных. Предложенная модель формализует процессы извлечения, структурирования и интерпретации информации в условиях распределённой обработки и высокой интенсивности документных потоков [6-А; 7-А].

2. Предложено архитектурное решение распределённой системы документооборота с выделением специализированного компонента интеллектуального анализа – AI Service (сервис искусственного интеллекта). В отличие от традиционных монолитных систем, предложенный подход

обеспечивает независимое масштабирование ресурсоёмких вычислительных модулей, что повышает производительность системы и упрощает её дальнейшее развитие [3-А].

3. Разработан метод адаптации многоязычных моделей BERT к условиям ограниченности размеченных языковых данных. Метод основан на использовании синтетически сформированных обучающих выборок и последующем дообучении модели; это позволяет повысить качество извлечения именованных сущностей в низкоресурсных языках [5-А].

4. Предложен гибридный метод обработки документов, сочетающий трансформерные модели машинного обучения с алгоритмами, основанными на правилах. Такое сочетание рассматривается как способ повысить устойчивость системы при неполных и неоднородных данных, а также сделать результаты автоматизированного анализа более интерпретируемыми [3-А].

5. Систематизированы методы масштабирования микросервисных систем, включая горизонтальное (увеличение производительности системы путем добавления новых вычислительных узлов) и вертикальное (увеличение вычислительных ресурсов отдельного узла путем наращивания мощности процессора, памяти или дискового пространства) распределение вычислительной нагрузки. В рамках исследования показана целесообразность их комбинированного применения для повышения производительности и отказоустойчивости распределённых приложений, включающих компоненты интеллектуальной обработки данных [8-А; 10-А].

6. Теоретически обоснована роль контейнеризации и механизмов оркестрации в построении распределённых интеллектуальных систем [3-А]. Показано, что данные технологии создают основу для автоматического масштабирования сервисов, балансировки нагрузки, повышения отказоустойчивости и более рационального распределения вычислительных ресурсов [11-А].

7. На основе разработанной математической модели интеллектуальной системы документооборота, ориентированной на применение

микросервисной архитектуры и моделей BERT, предложен практический алгоритм реализации системы с учётом особенностей организационной структуры Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан. Разработанный подход адаптирован к условиям существующей информационной инфраструктуры и ориентирован на возможность практического внедрения [1-А–5-А].

8. Проведена экспериментальная апробация предложенных моделей и алгоритмов, результаты которой подтвердили повышение точности обработки документов и сокращение времени выполнения операций по сравнению с традиционными подходами к документообороту. Полученные результаты свидетельствуют о прикладной эффективности разработанных решений и возможности их использования в распределённых информационных системах различного назначения [1-А].

Совокупность представленных результатов формирует научно обоснованную базу для дальнейшего развития интеллектуальных систем документооборота нового поколения, ориентированных на функционирование в условиях цифровой экономики и распределённой вычислительной инфраструктуры.

Положения, выносимые на защиту:

1. Разработана и обоснована математическая модель и архитектурное решение интеллектуальной системы документооборота, которые построены на базе микросервисной архитектуры с использованием трансформерных моделей типа BERT и обеспечивают формализацию процессов извлечения, структурирования и обработки данных в условиях реальной информационной инфраструктуры. При этом за счет выделения специального компонента интеллектуального анализа и применения совместной горизонтальной и вертикальной масштабируемости микросервисов гарантируются высокая производительность и устойчивость системы к изменяющимся нагрузкам.

2. Доказано, что метод адаптации мультиязычных трансформерных моделей BERT на основе синтетической генерации обучающих данных и переобучения является эффективным для условий малоресурсных языков. На этой основе разработан гибридный метод обработки документов, объединяющий алгоритмы машинного обучения и подходы на основе правил, позволяющий существенно повысить качество и точность обработки текстов в условиях ограниченности языковых ресурсов.

3. Установлено, что внедрение практического алгоритма реализации интеллектуальной системы документооборота в структуру Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан обеспечивает сокращение времени обработки документов и повышение точности извлечения информации, что подтверждает высокую практическую значимость результатов исследования и их экспериментальную устойчивость.

Теоретическое и практическое значения исследования. Теоретическая значимость диссертационной работы заключается в развитии научных основ математического моделирования интеллектуальных систем документооборота в условиях распределённых вычислительных сред. Полученные результаты расширяют существующие представления о принципах построения микросервисных архитектур за счёт их интеграции с современными методами обработки естественного языка на основе трансформерных моделей.

Разработанная математическая модель интеллектуальной системы документооборота формализует процессы извлечения и структурирования данных, что вносит вклад в развитие методов анализа текстовой информации и теории распределённых информационных систем. Существенным является теоретическое обоснование гибридного подхода, объединяющего методы машинного обучения и алгоритмы на основе правил, что позволяет повысить устойчивость и интерпретируемость интеллектуальных систем.

Важным теоретическим результатом является систематизация и обоснование методов горизонтального и вертикального масштабирования микросервисных систем, а также доказательство эффективности их комбинированного применения в задачах обработки данных. Кроме того, разработан метод адаптации трансформерных моделей к условиям низкоресурсных языков, что расширяет возможности применения современных NLP-технологий (совокупность методов и инструментов обработки текстовых данных) в многоязычных средах.

В совокупности полученные результаты формируют теоретическую базу для дальнейших исследований в области интеллектуальных распределённых систем, цифрового документооборота и прикладного использования методов искусственного интеллекта.

Практическая значимость исследования определяется возможностью непосредственного применения разработанных моделей, методов и архитектурных решений при создании и внедрении интеллектуальных систем документооборота в различных отраслях.

На основе предложенной математической модели и архитектурного подхода разработана интеллектуальная система, обеспечивающая автоматизацию процессов обработки документов, включая извлечение данных, классификацию и формирование структурированной информации. Внедрение системы позволяет существенно сократить время обработки документов, снизить количество ошибок ручного ввода и повысить эффективность управленческих процессов.

Особую практическую ценность представляет разработанный инженерно-прикладной алгоритм реализации системы, адаптированный под структуру Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан, что подтверждает применимость результатов исследования в реальных условиях государственной информационной инфраструктуры.

Предложенные решения также могут быть использованы:

– в государственном секторе при автоматизации документооборота и оказании электронных услуг;

– в коммерческих организациях для обработки договорной и финансовой документации;

– в финансовых учреждениях при реализации процедур контроля и анализа данных.

Дополнительно практическая значимость проявляется в возможности масштабирования системы, её интеграции с существующими информационными платформами и адаптации к различным предметным областям.

Степень достоверности результатов диссертации обоснована реализацией комплексного исследования научно обоснованных запланированных задач, использованием современных технологий математического и компьютерного моделирования, а также трансформерных моделей BERT. Достоверность результатов также подтверждается проведением компьютерных экспериментов на основе реальных данных информационной инфраструктуры, синтетической генерацией обучающих данных и экспериментальным внедрением разработанных алгоритмов в структуру Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан.

Соответствие диссертации паспорту научной специальности. Диссертация и ее содержание охватывают области научных исследований паспорта специальности доктора философии (PhD) 6D060204 – Теоретические основы информатики и соответствует следующим направлениям (пунктам) специальности:

Пункт 4. Комплексные исследования научно-технических и фундаментально-прикладных проблем с применением современной технологии математического моделирования и вычислительного эксперимента.

Пункт 11. Машинное обучение и искусственный интеллект: теоретические основы обучающихся систем, методы обнаружения новых знаний (Data Mining), логический вывод.

Пункт 15. Модели информационных процессов и структур.

Пункт 18. Новые средства приобретения знаний и создания онтологии, средства интеллектуализации бизнес-процессов.

Пункт 25. Распределённые многопользовательские системы.

Пункт 26. Реализация эффективных численных методов и алгоритмов в виде комплексов проблемно-ориентированных программ для проведения вычислительного эксперимента.

Личный вклад соискателя ученой степени. Соискатель участвовал в формировании темы диссертации, самостоятельно провел анализ литературы, разработал математическую модель интеллектуальной системы документооборота на основе микросервисной архитектуры и BERT, реализовал практический алгоритм, провел эксперименты, обработку и интерпретацию результатов, а также подготовил методические рекомендации и текст диссертации.

Апробация и применение. Результаты диссертационного исследования обсуждались на научных семинарах Института математики имени академика А. Джураева Национальной академии наук Таджикистана (НАНТ), на семинаре кафедры информатики Таджикского национального университета (ТНУ), а также были представлены на республиканских и международных конференциях 2022-2026 гг. в НАНТ, ТНУ, Академии государственного управления при Президенте Республики Таджикистан, Филиале Московского государственного университета им. М.В. Ломоносова в г. Душанбе, Бохтарском государственном университете имени Носира Хусрава, Таджикском международном университете туризма и предпринимательства, Технологическом университете Таджикистана и других профильных учреждениях.

Публикации по теме диссертации. Основные результаты диссертационного исследования опубликованы в 12 работах [1-А–12-А], включая 7 статей в рецензируемых журналах Республики Таджикистан, 4 тезиса докладов, представленных на международных и республиканских конференциях, и 1 статью в зарубежном журнале.

Структура и объем диссертации. Диссертация включает «Введение», «Общую характеристику исследования», четыре главы, раздел «Выводы» с подразделами «Основные научные результаты диссертации» и «Рекомендации по практическому использованию результатов», раздел «Литература» с подразделами «Список использованных источников» и «Список публикаций соискателя», а также «Приложения». Работа выполнена на 197 страницах в Microsoft Word 2016, содержит 9 таблиц и 15 рисунков, нумерация которых осуществляется в соответствии с главами диссертации. Список литературы с перечнем публикаций соискателя включает 182 наименований.

ГЛАВА 1. ОБЗОР ЛИТЕРАТУРЫ: ВВЕДЕНИЕ В МИКРОСЕРВИСНУЮ АРХИТЕКТУРУ И ИНТЕЛЛЕКТУАЛЬНЫЙ ДОКУМЕНТООБОРОТ НА ОСНОВЕ BERT

1.1. Обзор литературы по микросервисной архитектуре и технологиям NLP

Эволюция архитектур систем документооборота. Эволюция архитектур систем. «Микросервисная архитектура хорошо согласуется с практиками непрерывной интеграции и поставки в крупных организациях» [126]. Информационные технологии смещают акцент с монолитных архитектур на распределённые системы, среди которых выделяется микросервисная архитектура.

На рисунке 1 видна смена подходов к построению систем документооборота, где монолитные решения уступают место сервисно-ориентированным моделям, затем микросервисной архитектуре и далее распределённым микросервисным платформам.

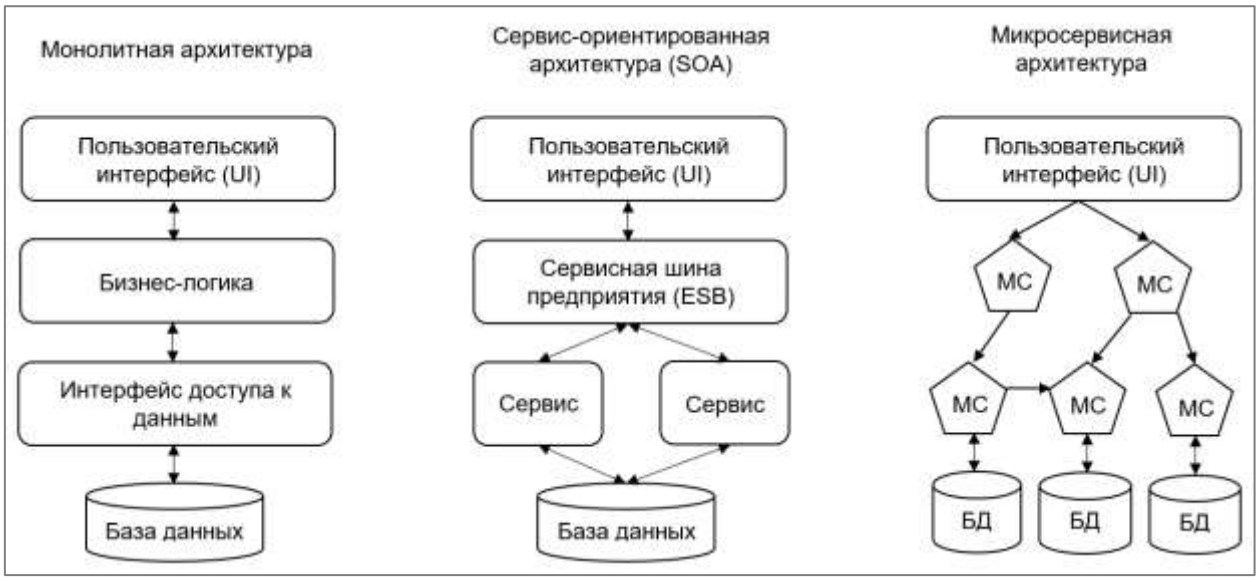


Рисунок 1.1. Эволюция архитектур программных систем

Рассматриваемое решение опирается на Service-Oriented Architecture SOA, но дробит систему сильнее, так независимые компоненты получают заметно большую автономность. Анализ публикаций показывает, что применение микросервисной архитектуры делает комплекс живучее при отказах и расширяет обособленность отдельных сервисов.

Сэм Ньюман [85; 148-150] трактует микросервисы так «способ построения программных систем, при котором приложение формируется из совокупности небольших, слабо связанных между собой сервисов, взаимодействующих через лёгкие сетевые протоколы, преимущественно HTTP/REST» [85; 148-150]. У Томаса Эрла [119; 132] и Мартина Фаулера [109; 134-136] внимание идет к развёртыванию сервисов, их масштабированию, и автономный жизненный цикл компонентов. «Границы сервисов целесообразно проводить по предметным областям, а не только по техническим слоям приложения» [85].

Становление этих подходов объясняется разными факторами. Среди них - рост запросов к масштабируемости и отказоустойчивости ИС, сжатие срока вывода продукта на рынок (time-to-market) и то, что бизнес-логика современных приложений стала сложнее

У Фаулера прямо сказано: «переход от монолитной модели к микросервисной архитектуре приводит к снижению связанности между компонентами системы и делает её менее чувствительной к отказам отдельных модулей»[109; 134-136].

Контейнеры усилили этот сдвиг. Крис Ричардсон подчёркивает: «использование платформ Docker (платформа контейнеризации приложений и сервисов) и Kubernetes (платформа автоматического управления контейнеризированными приложениями) обеспечивает изоляцию сервисов, упрощает процессы развёртывания и делает управление масштабированием системы более гибким» [97; 157].

Основные этапы эволюции архитектур программных систем приведены в таблице 1.

Таблица 1.1. Эволюция архитектур программных систем

Этап	Характеристика	Ограничения
Монолитная архитектура	Единое приложение	Низкая гибкость
SOA	Сервисная декомпозиция	Сложность интеграции
Микросервисы	Независимые сервисы	Сложность управления

Современные источники относят микросервисную архитектуру к ведущим линиям развития информационных систем. «В распределённых системах необходимо явно учитывать частичные отказы, сетевые задержки и согласование данных» [159].

Микросервисная архитектура системы документооборота. На таких сервисах удобно собирать распределённый комплекс, так как модуль закрывает узкий набор функций и почти не зависит от соседних узлов. Целевая схема на рисунке 2 объединяет API Gateway, независимые микросервисы и общее хранилище данных. «Чётко определённый API снижает связанность сервисов и упрощает независимую эволюцию программных интерфейсов» [106]. Под API (Application Programming Interface) – понимают интерфейс прикладного программирования, набор правил и средств, по которым компоненты обмениваются данными.

Вход пользователя (рисунок 2)Далее по конвейеру следуют: клиент; API Gateway; сервис аутентификации; сервис загрузки документов, сервис NLP/BERT, автоматически разбирающий текст; очередь сообщений; файловое хранилище



Рисунок 1.2. Микросервисная архитектура системы интеллектуального документооборота

Ключевой состав включает аутентификацию, обработку документов, интеллектуальный анализ на NLP/BERT, а также подсистемы хранения и обмена сообщениями. «Доменная адаптация трансформеров, как в BioBERT, повышает качество анализа специализированных текстов» [75]. Поток между узлами начинается с API Gateway, затем идут сервисы обработки документов, извлечения сущностей средствами NER и BERT, классификации и маршрутизации; дальше база данных и очередь на Kafka либо RabbitMQ.

В работе Apache Kafka берётся как средство потоковой доставки и обработки событий в реальном времени на распределённых узлах; RabbitMQ выступает брокером AMQP и связывает сервисы асинхронно. «Гибридные контуры, сочетающие нейросетевые модели и правила, повышают интерпретируемость результатов анализа» [112].

В условиях цифровой трансформации экономики Республики В Таджикистане всё острее нужны информационные системы, которые остаются гибкими, масштабируемыми и живучими при сбоях. В приоритете автоматизация документооборота и методы интеллектуальной обработки данных [1-А–12-А] [1-А–12-А]. «Классификация документов необходима для маршрутизации потоков и поддержки принятия управленческих решений» [13].

Микросервисы задают каркас распределённых интеллектуальных комплексов: каждый узел выполняет одну узкую роль. Роли типичны: обработка документов, извлечение сущностей, классификация, маршрутизация данных.

NLP-методы, прежде всего модели Bidirectional Encoder Representations from Transformers, разбирают текст сразу на нескольких языках, включая таджикский и русский. «Качество извлечения информации определяется полнотой корпуса и согласованностью схемы аннотации данных» [155]. Государственным ИС с многоязычным документооборотом без этого не обойтись [5-А].

Монолит проигрывает, так как NLP-сервисы на микросервисах масштабируются отдельно, ресурсы выделяют под задачу, модели меняют без остановки всей системы. «Автоматизация обработки текста снижает долю ручных операций в цепочках делопроизводства организации» [109].

Интеллектуальная обработка документов. В актуальных системах документооборота анализ текста всё чаще автоматизируют средствами обработки естественного языка.

Рисунок 3 показывает конвейер разбора документов на моделях BERT для классификации и извлечения сущностей. Процесс идёт от загрузки документа через предобработку и извлечение текста к классификации, затем к NER, то есть к автоматической разметке в тексте персон, организаций, адресов, дат и сходных объектов, и к извлечению сущностей. После этого следуют проверка, запись в базу и выдача итога.

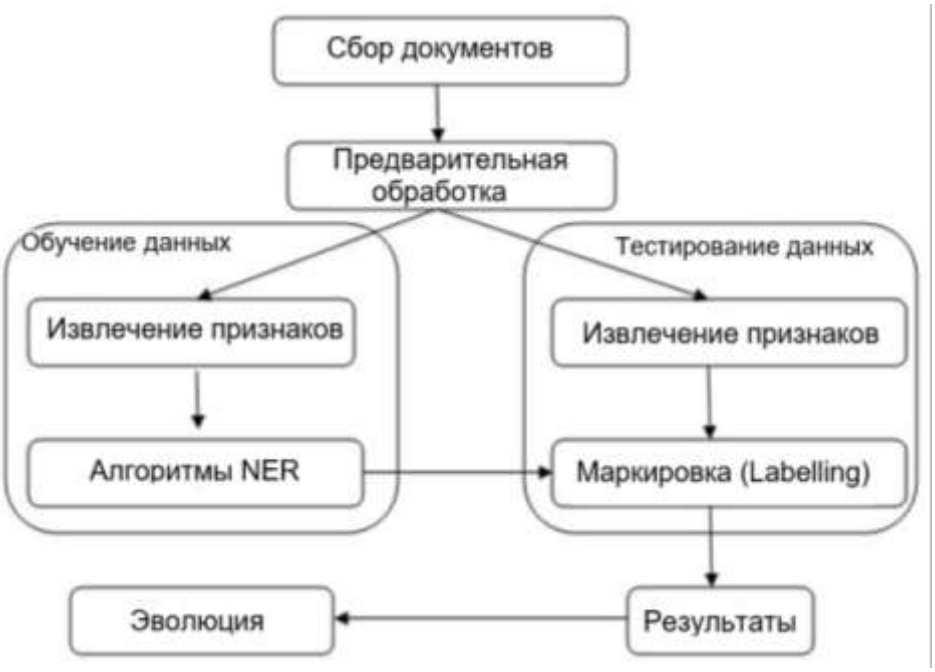


Рисунок 1.3. Конвейер интеллектуальной обработки документа

Рисунок 3 задаёт показывает, что сначала загружают данные и готовят их, затем извлекают текст, классифицируют его моделями семейства BERT, выделяют именованные сущности методом NER, затема проводят валидацию и сохраняют результат в базе данных.

Эволюция методов обработки документов. «Горизонтальное масштабирование увеличивает пропускную способность за счёт добавления однотипных экземпляров сервиса» [85]. Первые системы управления документами разрабатывались преимущественно на базе монолитной архитектуры, для которой была характерна высокая связанность компонентов [85]. При умеренной нагрузке такие решения ещё держали приемлемую производительность, но объёмы данных росли, а функциональность усложнялась, масштабирование и сопровождение системы становились проблемными. «Вертикальное масштабирование ограничено физическими пределами узла и хуже подходит для эластичных облачных нагрузок» [96].

Появление сервис-ориентированной архитектуры позволило разнести функции программной системы по отдельным сервисам [117]. Практика показала оборотную сторону: SOAP и WS, и централизованные ESB-шины часто поднимали задержки и одновременно создавали единую точку отказа. «Межсервисное взаимодействие следует проектировать как явный версионируемый контракт, а не как скрытую зависимость» [96].

Микросервисная архитектура [109; 1-A-12-A] стала следующим шагом. Её распространение шло вместе с децентрализацией данных, легковесными протоколами взаимодействия и автоматизацией CI/CD. Изоляция сервисов подняла отказоустойчивость.

У Криса Ричардсона [97; 157] названы «три паттерна, без которых сегодня редко обходится микросервисная архитектура: API Gateway, Service Discovery и Circuit Breaker» [97; 157]. Именно эта тройка закрепилась как типовая для распределённых комплексов.

«Оптимизация распределения информационных ресурсов в микросервисной схеме опирается на чётко определённый программный интерфейс» [2]. Через API Gateway централизуют маршрутизацию запросов, управление трафиком к микросервисам и контроль безопасности. Service

Discovery сам находит, где живут сервисы и с какими параметрами. Circuit Breaker обрывает каскад ошибок между узлами. «Структурирование данных документов создаёт основу для аналитики и межведомственного информационного обмена» [71].

Методы машинного обучения в обработке документов. Ранний разбор документов держался главным образом на rule-based схемах (обработка по заранее заданным логическим правилам), регулярные выражения, словари, морфологические анализаторы [116]. «Условные случайные поля остаются классическим инструментом вероятностной разметки последовательностей» [73]. Их настройка требовала много экспертного труда, а вариативность текстов давалась плохо.

Дальнейшее развитие получили вероятностные модели, в том числе CRF («Conditional Random Fields – Условные случайные поля – класс статистических моделей машинного обучения, которые используются для задач структурированного предсказания» [73]), при обработке текста принимающие во внимание окружение соседних токенов. «Слабая программная разметка ускоряет подготовку обучающих выборок для задач извлечения информации» [95]. Практика показывает: чтобы применять их, признаки обычно задают вручную, и это усложняет разработку. Распределённые представления слов – прежде всего системы Word2Vec («эффективная оценка представлений слов в векторном пространстве» [79]) и GloVe («глобальные векторы для представления слов» [87]) существенно улучшили качество текстовых моделей. Опираясь на эти представления, схемы вида BiLSTM-CRF (двунаправленные модели для разметки последовательностей) [112] достигли высокой точности при извлечении сущностей. «Правила и нейросетевые модели целесообразно сочетать в гибридных контурах анализа документов» [112]. Word2Vec отображает слова в плотные числовые векторы, которые фиксируют семантическую близость. Иной путь у GloVe (Global Vectors): векторы строятся по статистике совместной встречаемости слов в корпусе. Архитектура BiLSTM-CRF

(Bidirectional Long Short-Term Memory with Conditional Random Fields) соединяет двунаправленные рекуррентные сети и условные случайные поля. «Для низкоресурсных языков критична адаптация мультиязычных моделей и наращивание локальных текстовых корпусов» [87]. Это позволяет последовательной разметки текста выполняться точнее.

Механизм внимания (Attention) [3] и построенная на нём, изменили сам способ обработки текста. BERT [13] строит контекстуальные представления через двунаправленное самовнимание (Self-Attention) и закрепился как основной инструмент для задач NER. «Двунаправленные модели LSTM-CRF долгое время задавали базовый уровень качества разметки последовательностей» [108].

Появление мультиязычного BERT [88] усилило перенос обучения между языками, это напрямую важно для низкоресурсных языковых сред, в том числе для таджикского языка.

Межсервисное взаимодействие: REST vs gRPC. Основная схема обмена между микросервисами опирается на систему REST API с использованием формата JSON [110]. «Сравнение gRPC и REST показывает практический компромисс между производительностью и универсальностью интеграции» [74]. Такой способ обмена распространён широко, но у него есть существенные ограничения. В основном слабая производительность и отсутствие строгой типизации данных

Технология gRPC [139], опирается на Protocol Buffers (механизм сериализации структурированных данных, разработанный компанией Google) и HTTP/2, Она применяет бинарную сериализацию и задаёт строгую типизацию контрактов и повышает производительность межсервисного взаимодействия. «Асинхронный обмен сообщениями повышает устойчивость распределённых контуров к пиковым нагрузкам» [137]. Так же поддерживается потоковая передача данных.

Сравнительные исследования [74] показывают, что относительно REST задержки запросов при gRPC ниже на 30–40%, что заметно для систем с интенсивной обработкой данных.

Синтез обучающих данных. «Эволюция микросервисной архитектуры требует согласованных практик наблюдения, конфигурации и управления релизами» [119]. Для низкоресурсных языков главной проблемой по-прежнему остаётся дефицит размеченных данных [115]. Её обычно закрывают четырьмя путями. При Semi-Supervised Learning (полуконтролируемом обучении) модель обучают на сочетании размеченных и неразмеченных данных, а аугментация данных увеличивает обучающую выборку преобразованием существующих примеров. Краудсорсинг привлекает пользователей к разметке через цифровые платформы; для синтетических наборов данных примеры генерируют с заранее заданной разметкой. У последнего варианта есть практическое удобство, так как разметка возникает вместе с генерацией, а не дописывается вручную. В работе [96] описан фреймворк Snorkel, рассчитанный на программную разметку данных. СочетаниеМетоды машинного обучения, в том числе модели семейства BERT, подключают к микросервисной архитектуре и оценивают это соединение как перспективное для развития систем документооборота. У низкоресурсных языков синтетические данные лишь частично восполняют недостаток обучающих выборок, поэтому для них соединение особенно важно.

1.2. Преимущества и недостатки микросервисов в интеллектуальных системах

Чем быстрее развиваются цифровые технологии и чем выше требования к интеллектуальной обработке данных, тем сильнее выбор архитектуры влияет на проектирование систем. Микросервисную архитектуру применяют как один из этапов в системах, где востребованы масштабируемость, адаптивность и интеграция методов машинного обучения.

Объясняют практический интерес к микросервисам несколько преимуществ. Когда отдельные компоненты масштабируют независимо,

вычислительные ресурсы распределяют рациональнее, с учётом текущей нагрузки; для ресурсоёмких модулей машинного обучения это особенно существенно. Разработку и развёртывание сервисов ведут параллельно, в том числе силами разных команд, поэтому создание, обновление и сопровождение программного продукта ускоряются. Разные языки программирования, библиотеки и фреймворки допустимо сочетать в одной системе, подбирая инструменты под конкретные задачи, включая задачи искусственного интеллекта. Отказоустойчивость учитывают отдельно: сбой одного микросервиса, как правило, не останавливает всю систему, поэтому растёт её надёжность.

У микросервисной архитектуры есть и ограничения. Обычно называют сложность управления распределённой системой, необходимость DevOps-практик и специализированной инфраструктуры, проблемы согласованности данных между сервисами, а также рост сетевых задержек при межсервисном взаимодействии.

Монолитную архитектуру с микросервисной сравнивает таблица 1.2.

Таблица 1.2. Сравнение монолитной и микросервисной архитектур

Критерий	Монолит	Микросервисы
Масштабируемость	Ограниченная	Высокая
Разработка	Централизованная	Распределённая
Надёжность	Низкая	Высокая
Сложность	Низкая	Высокая

Архитектура интеллектуальной системы документооборота.

Разрабатываемую систему строят по микросервисному архитектурному подходу. Функциональность разнесена по независимым и слабо связанным компонентам. Каждый микросервис закреплён за отдельным бизнес-доменом и развивается независимо от других частей системы.

Архитектура делится на логические слои.

1. За взаимодействие пользователя с системой и за отображение результатов обработки данных отвечает презентационный слой.

2. Слой интеграции принимает входящие запросы в единой точке входа, маршрутизирует их, агрегирует ответы и преобразует используемые протоколы.

3. К слою бизнес-логики относятся сервис управления шаблонами, сервис интеллектуального анализа и сервис справочных данных. 4. Слой опирается на NoSQL базу данных, у которой гибкое хранение информации и поддержка горизонтального масштабирования.

Взаимодействие пользователя с системой организовано через API Gateway направляет запросы к микросервисам.

Архитектурные принципы построения системы. Обработку текстовой информации ведёт сервис интеллектуального анализа; в нём применяют модели машинного обучения. Между компонентами данные передают по API и через асинхронный обмен сообщениями.

Архитектурные принципы построения системы. Их учитывали на этапе проектирования.

1. Принцип единственной ответственности. Область задачи у микросервиса одна.

2. Принцип изоляции данных. Свои данные микросервис ведёт сам, команды принимает через API.

Недоступность компонента отрабатывают деградацией: система переключается на упрощённые алгоритмы обработки данных.

Функциональные компоненты системы:

основную бизнес-логику обработки документов несёт компонент шаблонов. Он принимает документ, извлекает переменные, классифицирует данные, сохраняет метаданные и формирует итоговый документ. Связь с сервисом интеллектуального анализа снабжена механизмами отказоустойчивости, поэтому отказ отдельных компонентов не подрывает

работу системы. «Современные библиотеки трансформеров стандартизируют доступ к предобученным языковым моделям и сценариям дообучения» [161].

Сервис интеллектуального анализа анализа обрабатывает текстовые данные средствами. В круг его задач входят анализ документов, извлечение сущностей (NER), дообучение моделей и генерация синтетических обучающих данных. Применение трансформерной архитектуры улучшает извлечение структурированной информации из неструктурированных текстов. «Выделение отдельного сервиса оправдано, если модуль меняется и масштабируется иначе, чем остальная часть системы» [145].

Для интеллектуальных систем микросервисная архитектура может служить основой благодаря гибкости, масштабируемости и устойчивости к отказам. Ограничения связаны со сложностью управления распределённой средой и потребностью в развитой инфраструктуре, но для систем с элементами искусственного интеллекта такой выбор по-прежнему оправдан. «Повторное использование логики через сеть обычно дороже локальных вызовов внутри монолитного процесса» [96].

Интеграция микросервисной архитектуры с методами машинного обучения, улучшает автоматизацию и повышает качество извлекаемой информации. Направление все еще актуально для цифровой трансформации экономики и разработки информационных систем. «Независимое масштабирование сервисов позволяет направлять ресурсы туда, где фактическая нагрузка максимальна» [85].

1.3. Основные принципы построения микросервисных систем документооборота с BERT

Принципы микросервисной архитектуры. В системах документооборота функции разносят по автономным сервисам: каждый закрывает отдельный участок обработки данных или бизнес-логики. Изменения остаются локальными, а компоненты обновляют, не перерабатывая всю архитектуру [5-А].

Через стандартизированные API добиваются слабой связанности сервисов: компоненты взаимодействуют, не опираясь напрямую на внутреннюю реализацию соседних модулей. Для систем с BERT-обработкой документов [3-А] независимость NLP-модулей принципиальна: их дообучают, обновляют и масштабируют независимо от остального приложения.

Автономный сервис разрабатывают, развёртывают и масштабируют независимо от остальных. В данной работе автономность сервисов принимают за техническое условие гибкого включения моделей машинного обучения в контур документооборота. Децентрализация управления снижает зависимость всей системы от одного управляющего компонента и повышает устойчивость архитектуры при отказах отдельных сервисов [6-А].

Практическая реализация указанных принципов связана с применением DevOps-подходов и механизмов CI/CD. «Практики DevOps объединяют разработку и эксплуатацию для ускорения поставки изменений при сохранении стабильности» [119]. Их роль состоит в автоматизации сборки, тестирования и доставки программных компонентов; это даёт возможность поддерживать непрерывное обновление микросервисной системы, не нарушая её общей работоспособности [2-А].

Обращение клиента к системе идёт через API Gateway, обеспечивающий маршрутизацию запросов между соответствующими микросервисами. «Микросервисы упрощают параллельную разработку, но увеличивают операционную сложность распределённой среды» [96]. Специализированную функцию реализует каждый сервис, работая со своей подсистемой хранения данных и тем самым соблюдая принцип изоляции данных.

Сравнение принципов микросервисной архитектуры. Основные принципы микросервисной архитектуры систематизированы в таблице 1.3. «Микросервисная архитектура позволяет разрабатывать, развёртывать и масштабировать отдельные компоненты системы независимо друг от друга» [85].

Таблица 1.3. Принципы микросервисной архитектуры

Принцип	**Описание**	**Эффект**
Декомпозиция	Разделение системы на независимые сервисы	Повышение гибкости
Автономность	Независимость сервисов	Повышение надёжности
API-first	Взаимодействие через API	Улучшение масштабируемости

По источникам [116], внедрение микросервисной архитектуры заметно чаще оказывается успешным именно тогда, когда перечисленные принципы реально соблюдаются. «Автономные микросервисы совместно реализуют бизнес-возможности организации при сохранении слабой связанности между компонентами» [145].

Алгоритм обработки документа. Интеллектуальный разбор документа идёт по цепочке шагов: из бинарного файла извлекается текст; текст токенизируется так, как того требует выбранная модель; нейросеть классифицирует токены; токены собираются в именованные сущности; сущностям назначаются семантические типы; для классификации вычисляется оценка уверенности. «Автоматическая обработка таджикского текста требует учёта морфологии и ограниченных объёмов размеченных данных» [109].

На один документ в среднем уходит от 2 до 5 секунд: время растёт вместе с объёмом текста. Пик вычислительных затрат приходится на Inference (инференс – этап практического применения обученной модели для получения прогнозов на новых данных) нейросетевой модели. «Работа Attention Is All You Need заложила архитектурную основу современных языковых моделей на трансформерах» [160]. Используемая модель по размеру достигает примерно 700 и максимальное потребление ОЗУ составляет около 1,5 ГБ [1-А].

Режимы работы системы. Извлечение данных ведётся в двух режимах.

1. *Паттерн-ориентированный режим* опирается на регулярные выражения: обработка занимает порядка 100 мс, обучение модели не нужно, но точность остаётся сравнительно ограниченной, составляющей около 70% [3-А]. «Возможности few-shot обучения крупных языковых моделей расширяют сценарии прикладной обработки текста» [120].

2. *Режим машинного обучения* строится на модели BERT. Точность обработки здесь в диапазоне 90-95%. В этом режиме на один документ уходит 2-5 секунд, зато модель нужно заранее обучить [4-А].

Если сервис машинного обучения недоступен, система сама уходит в паттерн-ориентированный режим обработки и тем самым не прерывает функционирование. «К ключевым свойствам микросервисов относят независимое развёртывание, слабую связанность и ориентацию на бизнес-возможности» [131].

Слой персистентности. Данные хранятся в NoSQL базе данных. За каждым микросервисом закреплён изолированный набор данных, поэтому принцип Database Per Service, где каждый микросервис использует свою БД выдерживается на концептуальном уровне даже при использовании единого экземпляра СУБД. «Переход от монолита к микросервисам требует последовательной декомпозиции и осознанного управления распределённой сложностью» [96].

Фактически это компромисс между полной изоляцией данных и упрощением эксплуатации системы. Для систем со средней нагрузкой до 100 000 документов такая схема даёт приемлемую производительность при меньшей сложности сопровождения и администрирования [5-А].

Полнотекстовые индексы и индексы по временным меткам ускоряют выборку. Поиск по персональным данным опирается на составные индексы.

1.4. Методы расширения и масштабирования интеллектуальных микросервисов

Обеспечить масштабируемость и устойчивость ИС нужно при росте объёмов данных, усложнении бизнес-логики и ужесточении требований к

производительности. Для интеллектуальных систем документооборота с машинным обучением это требование жёстче: нагрузка на вычисления высока, а входящий поток запросов неравномерен. Методы расширения микросервисной архитектуры применяют как рабочий инструмент для распределённых систем [6-А].

Методы расширения микросервисной архитектуры. Расширение микросервисных систем входит в проектирование архитектуры по мере роста нагрузки и усложнения функциональности решений. Нужный уровень масштабируемости обычно даёт связка инфраструктурных и архитектурных приёмов, а не один механизм. Чаще всего применяют:

1. *Горизонтальное масштабирование (Horizontal Scaling). «Распределённые системы требуют явного учёта частичных отказов, задержек сети и согласования реплицируемых данных» [159]. Горизонтальное масштабирование добавляет реплики сервисов и распределяет нагрузку между ними: система остаётся доступной даже при отказе узлов.*

2. *Применение API-шлюз (API Gateway).* Применение API-шлюза (API Gateway). Маршрутизацию запросов, управление потоками данных, агрегацию ответов сервисов, механизмы безопасности и контроль доступа проводят через единую точку входа.

3. *Событийно-ориентированная архитектура (EOA).* Помогают системе держаться при неравномерной или резко нарастающей нагрузке.

4. *Сервисная сетка (Service Mesh).* Мониторинг, балансировка нагрузки, трассировка запросов и механизмы безопасности работают вне бизнес-логики микросервисов. Сетевые взаимодействия для этого выносят на инфраструктурный уровень.

5. *Кэширование (Caching).* В промежуточные хранилища кладут часто запрашиваемые данные, разгружая основные сервисы и уменьшая время ответа.

Таблица 1.4 сводит основные методы расширения микросервисной архитектуры.

Таблица 1.4 – Методы расширения микросервисов

Метод	Назначение	Преимущества
Масштабирование	Увеличение производительности	Гибкость
Очереди сообщений	Асинхронность	Надёжность
Кэширование	Ускорение обработки	Экономия ресурсов

При высокой нагрузке совместное использование методов масштабирования усиливает производительность и устойчивость распределённых систем [73].

Архитектура масштабируемой системы. Масштабируемость задают четыре механизма: горизонтальное масштабирование, балансировка нагрузки, очереди сообщений и кэширование. За их счёт система подстраивается под меняющуюся интенсивность запросов и перераспределяет ресурсы между компонентами [6-А]. Контейнерная оркестрация при изменении состояния системы разворачивает сервисы и сдвигает нагрузку между узлами [7-А].

Ресурсные запросы микросервисов машинного обучения выше, специфика исполнения моделей у них своя. При интеграции под них выделяют специализированные ресурсы, ускоряют вычисления и настраивают пайплайны обработки данных: автоматизированные конвейеры, где выход предыдущего этапа становится входом следующего [96; 107].

Облачные технологии и контейнеризация Облачная инфраструктура во многом задаёт масштабируемость архитектуры. Ресурсы перераспределяют динамически. Систему подстраивают под меняющуюся нагрузку и при росте числа запросов удерживают доступность сервисов. Без расчётов или ссылки на источник тезис о снижении затрат на сопровождение оставлять нельзя [164; 8-А].

Развёртывают через контейнеризацию: компоненты изолируют, программное окружение и зависимости фиксируют вместе. Для переноса сервисов между средами исполнения такая фиксация нужна. Отдельные контейнеры на микросервисы снижают риск конфликтов библиотек и версий относительно общего окружения [2-А].

Сервисы через оркестрацию контейнеров запускают и останавливают, связывают между собой, масштабируют при росте нагрузки и держат в сетевой изоляции.

Для сервиса интеллектуального анализа задан потолок в 3 ГБ оперативной памяти [2-А]. Так один тяжёлый по вычислениям компонент не «съедает» ресурсы и не подрывает стабильность всей системы.

Масштабируемость и отказоустойчивость. Между запросами экземпляр микросервиса состояние не хранит: без этого горизонтальное масштабирование (stateless-подход) не работает. Несколько копий компонента поднимают и разводят нагрузку балансировщиками, снижая риск перегрузки отдельных узлов [7-А].

Отказоустойчивость держат на нескольких уровнях. Когда сервис машинного обучения недоступен, система уходит на упрощённый алгоритм и сохраняет базовую функциональность. Обращения к нестабильному сервису Circuit Breaker при повторяющихся сбоях или перегрузке на время блокирует. Кратковременные сбои отрабатывает Retry Mechanism: короткие задержки или нестабильное соединение между компонентами.

Регулярная проверка доступности при мониторинге состояния сервисов выявляет проблемные экземпляры и до восстановления выводит их из обработки запросов; это ещё один уровень [4-А].

Анализ производительности системы. Узкое место, сервис интеллектуального анализа: около 10 запросов в секунду на один экземпляр [5-А]. Причина, вычислительная сложность нейросетевого инференса. Остальные компоненты пропускают заметно больше запросов. Маршрутизацию и первичную обработку трафика API-шлюз выполняет при 50–100 запросах в секунду. Сервис управления шаблонами держит около 50 запросов в секунду; в текущей конфигурации узкого места в нём нет [1-А].

Остальные компоненты пропускают заметно больше запросов. API-шлюз обрабатывает 50–100 запросов в секунду, выполняя маршрутизацию и первичную обработку трафика. Сервис управления шаблонами, около 50

запросов в секунду; в текущей конфигурации он не является узким местом [1-А].

Запросы при росте нагрузки разносят по нескольким экземплярам сервисов: включают горизонтальное масштабирование. Компонентам машинного обучения ресурсы наращивают ещё вертикально, с GPU-ускорением, и применяют батчирование запросов, объединяя входные данные, чтобы поднять эффективность обработки и пропускную способность.

Машинное обучение и архитектура BERT для NER. При распознавании именованных сущностей токены текста выделяют и классифицируют по заданным семантическим классам. PERSON, DATE и MONEY здесь типичны. Решение берёт трансформерную архитектуру BERT: двунаправленный учёт контекста, самовнимание и дообучение под конкретную предметную область.

Мультиязычный BERT покрывает более сотни языков. Знания переносят между языковыми корпусами; низкоресурсным языкам с малым объёмом размеченных данных такой перенос особенно полезен.

Цепочку Fine-tuning держат неизменной: загрузка размеченного корпуса, токенизация, выравнивание меток с субтокенами, обучение, валидация и сохранение итоговой конфигурации. Эксперименты при таком порядке воспроизводимы.

Разметку последовательностей часто ведут субворд-токенизацией (subword tokenization): слова дробят на корни, приставки, суффиксы и другие мелкие части. Целевую метку ставят только на первый субворд-токен слова, а остальным частям дают тег отсечения (например, TNU или -2026). Обучающие сигналы остаются согласованными, метки не привязываются неверно к фрагментам расщеплённого слова [11-А].

Дефицит размеченных данных закрывают синтетическим расширением корпуса: строят шаблоны документов, подставляют значения, имена, даты, денежные суммы и автоматически расставляют метки сущностей по заранее

заданным правилам. Обучающая выборка растёт, а затраты на ручную разметку не растут пропорционально [12-А].

Гибридный подход к обработке данных. Машинное обучение в связке с rule-based-компонентами обычно устойчивее чисто нейросетевого варианта. Модель обобщает по обучающим данным; rule-based-компоненты дают предсказуемый результат, когда вход имеет нестандартную структуру или слабо формализован [4-А].

Если модель временно недоступна из-за перегрузки или ошибки выполнения, обработка уходит на паттерн-ориентированные алгоритмы.

Одного инфраструктурного приёма для масштабирования интеллектуальных микросервисных систем недостаточно. В данной архитектуре масштабирование складывают на стыке архитектурного, инфраструктурного и алгоритмического уровней. Горизонтальное масштабирование, контейнеризация, облачные платформы и асинхронное взаимодействие компонентов на практике удерживают устойчивость и производительность при росте нагрузки [1-А].

Модули на моделях семейства BERT и других методах машинного обучения требуют отдельной оптимизации. В распределённой среде в эту работу входят аппаратные ускорители и расширение обучающих выборок синтетической генерацией данных. Интеллектуальные модули стыкуются с требованиями масштабируемой системы документооборота и сохраняют устойчивость при эксплуатационной нагрузке [130; 166].

В экспериментах проявились и ограничения. Система чувствительна к вычислительной нагрузке, особенно при глубоком обучении, и не всегда стабильно обрабатывает неструктурированные или рукописные документы. Перенос на новые предметные области или расширение языковой поддержки требует повторной адаптации моделей.

Дальше оптимизируют текущие решения квантованием и дистилляцией моделей, переходом к мультимодальным подходам и повышением интерпретируемости результатов. Устойчивость и управляемость системы на

длительном горизонте усиливают интеграцией с инструментами мониторинга и управления жизненным циклом моделей.

Направление развития систем документооборота в первой главе связывают с микросервисной архитектурой и методами искусственного интеллекта; у этого направления есть теоретическая база и прикладная ценность.

Выводы по первой главе

В обзоре работ по архитектуре программных систем и интеллектуальным решениям для документооборота несколько исследовательских линий выстроены рядом. Эволюция технологий сопровождается сдвигом к распределённым вычислительным моделям, которые в разных отраслях всё чаще берут за основу цифровой трансформации.

Микросервисный подход продолжает развитие программных архитектур как устойчивую практику проектирования. Переход от монолитных приложений к набору независимых сервисов повышает гибкость: системы легче подстраиваются под меняющиеся условия эксплуатации. Проектирование опирается на модульность, автономность компонентов, независимое масштабирование и обновление. Распределённая среда при этом усложняется, требуется внедрение DevOps-практик (методологий DevOps – набор подходов, инструментов и культурных принципов для интеграции процессов разработки и системного администрирования), острее встают синхронизация и согласованность данных в децентрализованных системах.

Слабая связанность сервисов, изоляция данных на уровне компонентов, автономный жизненный цикл каждого модуля и автоматизация разработки и развёртывания поддерживают устойчивость в эксплуатации. На инфраструктурном уровне к ним добавляют контейнеризацию, оркестрацию, сервисные сетки, асинхронное взаимодействие и облачные платформы. На этой основе держатся масштабируемость, отказоустойчивость и

управляемость микросервисных систем. Масштабирование интеллектуальных микросервисов разобрано отдельно.

Горизонтальное масштабирование, событийно-ориентированная архитектура, балансировка нагрузки и кэширование дают большую отдачу в связке, чем по отдельности. Сервисам с высокой вычислительной нагрузкой, в первую очередь модулям машинного обучения, дополнительно нужны батчирование запросов и аппаратные ускорители. Полученные результаты согласуются со стратегическими направлениями развития Республики Таджикистан и применимы на практике.

«Национальная стратегия развития Республики Таджикистан до 2030 года» [81], Концепция формирования электронного правительства и Программа цифровой экономики Республики Таджикистан связывают развитие страны с цифровизацией государственного управления, развитием информационной инфраструктуры и внедрением технологий искусственного интеллекта. Спрос на интеллектуальные системы документооборота, способные автоматизировать обработку данных и повышать эффективность управленческих процессов, растёт вместе с этими приоритетами.

В исследовании спроектирована и реализована интеллектуальная система документооборота на микросервисной основе с отдельным компонентом интеллектуального анализа. Функции разнесены по сервисам: тяжёлые вычисления вынесены в самостоятельный модуль, поэтому масштабирование упрощается, а взаимная зависимость частей системы ослабевает. За счёт современных механизмов межсервисного взаимодействия систему можно развивать дальше без радикальной перестройки всей структуры. Эффективность модели BERT оценена отдельно.

При извлечении именованных сущностей модель дала точность около 93% по метрике F1 и превзошла классические rule-based решения (решения на основе правил), типичные для базовых сценариев обработки текста. Многоязычную версию адаптировали к таджикскому языку как низкоресурсному: синтетически сформированные обучающие данные и

дополнительное дообучение частично закрыли дефицит размеченных корпусов.

Новизну главы даёт связка решений: микросервисная архитектура с выделенным ИИ-компонентом и асинхронным обучением; гибрид машинного обучения и rule-based логики (программная логика, построенная на строго заданных человеком правилах, условиях и инструкциях, без использования машинного обучения); методика генерации синтетических данных для задач NER (распознавание именованных сущностей); адаптация трансформерных моделей к низкоресурсным языковым условиям. Элементы рассчитаны на совместную работу.

Тестирование и внедрение прототипа подтверждают практическую сторону исследования: сократилось время обработки документов и снизилось число ошибок, связанных с ручным вводом данных. В текущем виде систему допустимо применять в государственных структурах, а также в финансовых и коммерческих организациях, где нужна автоматизация документооборота и более высокое качество обработки информации.

ГЛАВА 2. АРХИТЕКТУРНЫЕ РЕШЕНИЯ ДЛЯ МАСШТАБИРОВАНИЯ МИКРОСЕРВИСОВ

2.1 Горизонтальное и вертикальное масштабирование в микросервисных архитектурах

Современные информационные системы сталкиваются с ростом объёмов обрабатываемых данных и с ужесточением требований к производительности вычислительной инфраструктуры. Устойчивая работа программных систем, особенно микросервисных и занятых интеллектуальными задачами (обработка документов, анализ текста), требует масштабируемости [16-17; 28; 34-61; 11-А]. Масштабируемость означает способность системы подстраиваться под рост нагрузки так, чтобы производительность, надёжность и качество сервисов не снижались.

В микросервисной модели применяют две стратегии: горизонтальное и вертикальное масштабирование. Архитектура, профиль вычислительной нагрузки и требования к отказоустойчивости задают выбор между ними. Раздел разбирает оба подхода: плюсы, минусы и области применения в интеллектуальных микросервисных системах [3-А; 4-А; 5-А]. Теоретические основы масштабирования. Масштабированием называют наращивание вычислительных ресурсов ради роста пропускной способности.

В распределённых средах выделяют горизонтальное масштабирование (scale-out) и вертикальное масштабирование (scale-up). Механизмы повышения производительности и требования к архитектуре у этих вариантов различны [122-170].

Теоретические основы масштабирования. Горизонтальное масштабирование: сущность и принципы. Горизонтальный вариант увеличивает число параллельных экземпляров (реплик) сервисов. Система в микросервисной архитектуре уже разбита на независимые компоненты, и горизонтальный вариант с этой схемой совместим [15; 85]. Входящую нагрузку по узлам распределяют балансировщики; ресурсы используются равномернее, а отдельные компоненты реже уходят в перегрузку.

- Балансировщики раскидывают входящую нагрузку по узлам, ресурсы используются равномернее, отдельные компоненты реже уходят в перегрузку.

Горизонтальное масштабирование:

Сущность и принципы: При горизонтальном масштабировании растёт число параллельно работающих экземпляров (реплик) сервисов. Декомпозиция системы на независимые компоненты уже заложена в микросервисной архитектуре; с этой моделью горизонтальное наращивание совместимо [15; 85].

Балансировщики распределяют входящую нагрузку между несколькими узлами. Вычислительные ресурсы загружаются равномернее, отдельные компоненты реже уходят в перегрузку.

Преимущества: Избыточность сервисов даёт высокую отказоустойчивость, а масштабируемость потенциально не ограничена. Управление нагрузкой остаётся гибким, интеграция с облачными платформами естественна.

Ограничения: Усложняется управление распределённой системой; отдельно приходится обеспечивать согласованность данных. Растут сетевые задержки. Нужна развитая инфраструктура (оркестрация, мониторинг, сервис-дискавери).

Вертикальное масштабирование:

Сущность и принципы: Вертикальное масштабирование увеличивает вычислительные ресурсы отдельного узла: CPU (Central Processing Unit, центральный процессор), RAM (Random Access Memory, оперативная память), GPU (Graphics Processing Unit, графический специализированный микропроцессор). В отличие от горизонтального подхода система сохраняет централизованную структуру, а производительность повышают усилением аппаратной конфигурации [85; 97].

Преимущества. Реализация относительно проста. Архитектуру системы менять не нужно. Накладные расходы на управление остаются низкими.

Ограничения. Физические пределы сдерживают масштабирование оборудования. Единая точка отказа остаётся. Увеличение ресурсов ведёт к высокой стоимости. Гибкость в условиях переменной нагрузки ограничена.

Сравнительный анализ подходов приведён в таблице 2.1.

Таблица 2.1. Сравнительный анализ подходов

Критерий	**Горизонтальное масштабирование**	**Вертикальное масштабирование**
Принцип	Увеличение числа узлов	Увеличение ресурсов узла
Отказоустойчивость	Высокая	Низкая
Масштабируемость	Практически неограниченная	Ограниченная
Сложность реализации	Высокая	Низкая
Стоимость	Оптимизируемая	Высокая при росте

В распределённых системах при высокой динамике нагрузки предпочтительнее горизонтальное масштабирование. Вертикальное оставляют для специализированных задач с высокой плотностью ресурсов.

Применение в интеллектуальных микросервисных системах.

В интеллектуальных системах на методах машинного обучения требования к ресурсам у компонентов неоднородны [13; 15; 21]: сервисы обработки пользовательских запросов масштабируют горизонтально, а компонентам машинного обучения подключают GPU и наращивают объём памяти. Совмещают оба подхода.

Лучше всего работает гибридная схема: горизонтально расширяют

Комбинированная стратегия масштабирования API и бизнес-логику совмещают. Ресурсоёмкие ML-модули усиливают в основном вертикально. Сервисы связывают через очереди сообщений, и оставляют один главный API Gateway.

Практические аспекты реализации.

Для разворачивания в основном используют Docker, а для его управления и балансировки используют Kubernetes. Состояние компонентов

отслеживают системы мониторинга и логирования, сбои в эксплуатации фиксируют там же.

Горизонтальное масштабирование расширяет систему автоматическим запуском дополнительных реплик. Вертикальный путь другой: контейнерам выделяют дополнительные вычислительные ресурсы, а для самых ресурсоёмких операций подключают специализированные аппаратные ускорители [2-А].

Для систем документооборота оправдано комбинировать оба подхода, закрепляя каждый за соответствующими функциями компонентов. Так уравновешиваются скорость обработки, устойчивость при сбоях и возможность расширения архитектуры [7-А]. Полученные результаты пригодны при проектировании масштабируемых распределённых систем, рассчитанных на большие объёмы данных и интеграцию методов искусственного интеллекта [2-А]. «Микросервисы остаются практическим ответом на потребность в эволюционирующих и независимо поставляемых распределённых системах» [15].

В разделе разобраны методы горизонтального и вертикального масштабирования в микросервисных архитектурах как средства обеспечения производительности и отказоустойчивости распределённых систем. Выполнен сравнительный анализ подходов, зафиксированы преимущества и ограничения, сформулированы рекомендации для интеллектуальных систем документооборота с элементами машинного обучения. «Паттерны микросервисной разработки фиксируют повторяемые решения для декомпозиции, взаимодействия служб и повышения устойчивости» [96]. Комбинированное применение методов масштабирования при рациональном распределении вычислительных ресурсов может повысить эффективность обработки данных [6-А].

2.2 Использование контейнеров для управления микросервисами

По мере распространения распределённых информационных систем и микросервисной архитектуры востребованы технологии, которые управляют жизненным циклом сервисов, их развёртыванием и масштабированием. «Сервис-ориентированная архитектура заложила принципы повторного использования служб и контрактного межкомпонентного взаимодействия» [115]. Контейнеризация входит в число основных инструментов: она даёт изоляцию, переносимость и воспроизводимость программных компонентов.

Контейнерные технологии упаковывают приложения вместе с зависимостями в изолированные среды – контейнеры, работающие поверх общей операционной системы; это один из современных вариантов виртуализации. «Микросервисный подход отличается от классической SOA более мелкой гранулярностью сервисов и акцентом на независимое развёртывание» [151]. По сравнению с традиционной виртуализацией у контейнеров ниже накладные расходы и выше плотность размещения сервисов [13]. Роль контейнеризации в микросервисной архитектуре.

Роль контейнеризации в микросервисной архитектуре. «Архитектурный стиль микросервисов поддерживает автономность продуктовых команд и практики непрерывной поставки изменений» [105]. Микросервисная архитектура делит систему на самостоятельные сервисы, работающие независимо. Для каждого из них отдельно ведутся разработка, развёртывание и масштабирование, что меняет требования к программной среде. Контейнеризация поэтому фактически стала технологической базой такой модели. «Эволюционный переход к микросервисам снижает риски за счёт поэтапного выделения ограниченных предметных контекстов» [145].

Контейнеры изолируют микросервисы в собственной среде выполнения, снимают конфликты зависимостей и удерживают одинаковое окружение на этапах разработки, тестирования и эксплуатации. Перенос приложений между инфраструктурными платформами при этом становится проще и предсказуемее. «Оркестрация контейнеров автоматизирует развёртывание,

восстановление после сбоев и распределение нагрузки между сервисами» [121].

На практике контейнеризацию чаще реализуют средствами Docker (платформа контейнеризации приложений): платформа даёт инструменты сборки, хранения и запуска контейнеров. Управление изолированными средами упрощается, развёртывание сервисов в разных инфраструктурных условиях становится стабильнее. Архитектура контейнеризированных микросервисов.

Архитектура контейнеризированных микросервисов. «Контейнерная организация микросервисной системы строится на принципе изолированного размещения каждого сервиса в собственном контейнере, где уже присутствуют необходимые библиотеки, зависимости и компоненты среды выполнения. «Слабая связанность сервисов облегчает замену технологий и позволяет локализовать отказы в пределах одного компонента» [85]. За счёт этого обеспечивается автономность отдельных модулей и упрощается их независимое функционирование» [3; 13; 15]. Типичная структура включает контейнеры с микросервисами, реестр образов (image registry), механизмы оркестрации, а также средства мониторинга и ведения журналов событий. «Микросервисная архитектура хорошо согласуется с практиками непрерывной интеграции и поставки в крупных организациях» [126].

Контейнеры обмениваются данными через сетевые интерфейсы; жизненный цикл контролируют средства оркестрации, среди которых чаще всего используют Kubernetes (платформа автоматического управления контейнеризированными приложениями). «Декларативная модель Kubernetes задаёт целевое состояние кластера и непрерывно стремится поддерживать его» [121]. Оркестрация контейнеров.

Оркестрация контейнеров. «Контейнеризация упаковывает приложение вместе с зависимостями и обеспечивает воспроизводимость среды выполнения» [146]. С ростом числа микросервисных приложений

сопровождение контейнерной инфраструктуры усложнилось. «Контейнеры разделяют ядро операционной системы и поэтому имеют меньшие накладные расходы по сравнению с виртуальными машинами» [147]. Когда сервисов много, ручное администрирование даёт сбой, и на смену ему приходят системы оркестрации, автоматизирующие управление контейнерами [5-А]. Автоматическое развёртывание и масштабирование под нагрузку выполняют они. Между экземплярами запросы перераспределяются, а отказоустойчивость они же поддерживают. Отдельно встаёт работа с конфигурациями и секретами: при большом числе микросервисов без централизованного управления этими параметрами обойтись трудно. Оркестраторы используют декларативную модель. Конфигурационными описаниями задают целевое состояние среды; фактическое соответствие ему контролируют автоматически.

Если состояние сервисов или компонентов отклоняется, платформа возвращает инфраструктуру к заданной конфигурации. Постоянное ручное вмешательство для удержания согласованности среды при этом не требуется [6-А].

Преимущества контейнеризации. Развёртывание в микросервисной архитектуре контейнеризация упрощает; меняется и эксплуатация распределённых систем.

Преимущества контейнеризации. При совместимом container runtime контейнер переносится между вычислительными средами. Зависимость сервисов от конкретной инфраструктуры при этом снижается [7-А]. При изоляции компонентов каждый сервис работает в собственной среде, поэтому сбой одного на остальные не распространяется [7-А].

Контейнеры делят общую операционную систему, виртуальные машины её не делят. Накладные расходы ниже. Запуск занимает меньше времени.

С CI/CD контейнеры стыкуются напрямую: автоматизируются сборка, тестирование и доставка сервисов. «При росте нагрузки контейнеры могут

быстро реплицироваться, что облегчает горизонтальное масштабирование системы и снижает административную сложность этого процесса» [7-А].

Жёстче становятся требования к разграничению доступа, управлению образами и контролю уязвимостей. Сетевое взаимодействие в распределённой системе тоже усложняется: нужна стабильная маршрутизация и устойчивый обмен данными между компонентами. Мониторинг и логирование предъявляют дополнительные требования — без централизованного сбора и анализа технической информации контролировать большое число контейнеров нельзя. Для stateful-сервисов (сервисы с состоянием) отдельно встают хранение данных и согласованность между экземплярами приложений [7-А]. «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этих организациях» [131]. Контейнеризация и интеллектуальные системы.

Контейнеризация и интеллектуальные системы. В интеллектуальных микросервисных системах контейнеризация изолирует приложения и одновременно управляет вычислительно сложными компонентами, в том числе сервисами машинного обучения [4-А]. «Границы сервисов целесообразно проводить по предметным областям, а не только по техническим слоям приложения» [85]. Модели и вспомогательные модули проще разносить по контейнерам и вести с независимыми окружениями. Это критично при разных версиях ML-библиотек (библиотеки машинного обучения – Machine Learning Libraries) и зависимостей, где иначе возникают конфликты [3-А]. «В распределённых системах необходимо явно учитывать частичные отказы, сетевые задержки и согласование данных» [159].

Фиксированное окружение контейнера важно и для воспроизводимости экспериментов, и для переноса моделей с этапов разработки в production-развёртывание (развёртывание в производство – финальный этап жизненного цикла ПО, на котором готовое, протестированное приложение публикуется на серверах и становится доступным для реальных бизнес-клиентов). «Непрерывная поставка требует автоматизации тестирования, сборки

артефактов и контролируемого выката релизов» [118]. Различия в поведении моделей на разных узлах при этом менее вероятны [6-А]. Ресурсоёмкие компоненты, включая inference-сервисы (сервисы исполнения/вывода моделей), обычно выделяют в отдельные контейнеры с лимитами по CPU (Central Processing Unit – центральный процессор), оперативной памяти и GPU (Graphics Processing Unit – графический специализированный микропроцессор). «Наблюдаемость распределённой системы строится на согласованных метриках, логах и трассировках запросов» [121]. Так снижается риск, что рост вычислительной нагрузки ухудшит работу остальных частей системы [7-А].

Практические аспекты реализации: Современные контейнерные платформы, как правило, работают вместе с дополнительными инфраструктурными средствами [3-А]. «Скользящие обновления снижают риск простоя при выводе новой версии сервиса в эксплуатационную среду» [121]. На практике контейнеризация стыкуется с CI/CD (Continuous Integration/Continuous Deployment) для автоматизации сборки и доставки и с хранилищами образов, включая Docker Registry. Состояние среды контролируют средства мониторинга — Prometheus (открытая система мониторинга и оповещения с открытым исходным кодом, оптимизированная для сбора метрик в динамических микросервисных средах) и Grafana (ведущая открытая платформа для визуализации, анализа и мониторинга метрик, логов и трассировок) [2-А]. Согласованность параметров между сервисами поддерживают централизованным управлением конфигурацией. Во многих микросервисных системах контейнеры разворачивают как stateless-компоненты (без сохранения состояния).

После сбоев восстановление проходит проще. Затраты на расширение снижаются. Данные хранят вне контейнеров: во внешних хранилищах или в распределённых базах, которые обеспечивают сохранность и согласованность информации [6-А]. Контейнеризацию берут за базовый механизм построения и сопровождения микросервисных систем.

Развёртывание сервисов, их восстановление и управление состоянием автоматизирует контейнеризация совместно с платформами оркестрации [6-А]. В системах с компонентами машинного обучения вычислительно сложные сервисы стыкуют с остальной архитектурой через контейнеры.

Если среда выполнения стандартизирована, воспроизводимость вычислений выше. Различия инфраструктуры слабее влияют на работу моделей.

Контейнеризацию в данном исследовании берут за технологическую основу микросервисной инфраструктуры при соблюдении требований к управлению, мониторингу и защите; универсального решения из неё не выводят. Задачи администрирования, безопасности и сопровождения распределённых сред остаются.

2.3 Автоматическое масштабирование с помощью оркестраторов

Особую остроту масштабированию вычислительных ресурсов в распределённых микросервисных системах придаёт неравномерная и быстро меняющаяся нагрузка.номерной и быстро меняющейся нагрузке. По мере усложнения инфраструктуры ручное управление сервисами уже не даёт нужной скорости реакции и требует больших административных затрат. Поэтому шире применяют интеллектуальные механизмы управления на базе контейнерной оркестрации, которые автоматически перераспределяют ресурсы между компонентами. «Межсервисное взаимодействие следует проектировать как явный версионируемый контракт, а не как скрытую зависимость» [96].

Автоматическое масштабирование — это динамическое изменение числа экземпляров сервисов или корректировка параметров их работы по текущей нагрузке и заданным политикам управления. Такая модель поддерживает нужный уровень производительности без постоянного участия администратора и одновременно сокращает избыточное потребление вычислительных ресурсов [7-А]. «Управление Docker-кластерами упрощает

жизненный цикл микросервисов в облачной и гибридной инфраструктуре» [147].

Роль оркестраторов в масштабировании микросервисов. Системы оркестрации контейнеров задают основной контур управления жизненным циклом микросервисных приложений. Им поручены развёртывание сервисов, обновление, восстановление после отказов и динамическое масштабирование. «Автоматическое масштабирование опирается на метрики нагрузки и заранее заданные пороги реакции инфраструктуры» [121]. Среди платформ такого типа чаще всего выбирают Kubernetes (платформа автоматического управления контейнеризированными приложениями): в ней есть встроенные средства автоматического масштабирования и перераспределения ресурсов [159]. «Балансировка нагрузки распределяет запросы между экземплярами и повышает равномерность использования ресурсов» [1].

На практике оркестраторы прежде всего автоматизируют управление инфраструктурой и сокращают ручные операции при сопровождении распределённых систем. Они же адаптируют сервисы к меняющимся условиям эксплуатации, балансируют нагрузку между экземплярами и снижают влияние человеческого фактора на административные операции. «В распределённых системах масштабируемость достигается сочетанием репликации сервисов и управления состоянием данных» [159]. Масштабирование на платформах оркестрации идёт сразу на нескольких уровнях — в зависимости от характера нагрузки и доступных ресурсов.

Горизонтальное масштабирование меняет число реплик сервиса по метрикам нагрузки: загрузке CPU (Central Processing Unit), использованию памяти или числу входящих запросов. «Эластичность инфраструктуры позволяет наращивать и сокращать ресурсы вслед за изменением профиля нагрузки» [165].

Горизонтальное масштабирование При росте нагрузки появляются новые экземпляры; при снижении лишние освобождаются. Вертикальное масштабирование динамически меняет ресурсы отдельного контейнера,

прежде всего CPU и RAM (оперативная память). «Кластерное автомасштабирование дополняет масштабирование подов на уровне вычислительных узлов платформы» [121]. Для ресурсоёмких сервисов, включая компоненты машинного обучения, этот механизм особенно востребован.

Вертикальное масштабирование Кластерное масштабирование (Cluster Autoscaling) автоматически меняет число вычислительных узлов кластера. «Для ресурсоёмких модулей интеллектуального анализа независимое масштабирование критично по стоимости и задержке» [85]. При нехватке ресурсов добавляются новые узлы; при спаде нагрузки избыточные удаляются, что сокращает инфраструктурные затраты. Метрики и политики масштабирования.

Кластерное масштабирование (Cluster Autoscaling) От выбора метрик и стратегий управления во многом зависит эффективность автоматического масштабирования. «Для оценки состояния системы обычно применяются показатели загрузки процессора и памяти, интенсивность входящих запросов, время отклика сервисов, длина очередей сообщений, а также пользовательские бизнес-метрики» [2].

Метрики и политики масштабирования. «При высокой динамике нагрузки горизонтальная схема обычно предпочтительнее наращивания мощности отдельного узла» [85]. Стратегии масштабирования строятся по-разному. Реактивные подходы смотрят на текущее состояние инфраструктуры; проактивные опираются на прогноз нагрузки. На практике часто берут гибридные схемы, совмещающие оба варианта. «Метрики загрузки процессора, памяти и длины очереди составляют типичную основу политик автомасштабирования» [121]. Стабильность системы сильно зависит от корректных пороговых значений и временных интервалов в механизмах масштабирования.

Ошибочные параметры ведут либо к избыточному потреблению ресурсов, либо к нестабильной работе сервисов при смене нагрузки.

«Горизонтальное масштабирование увеличивает пропускную способность за счёт добавления однотипных экземпляров сервиса» [85]. Преимущества автоматического масштабирования. Оркестраторы подстраивают систему под меняющуюся нагрузку и перераспределяют ресурсы без ручного вмешательства.

Эксплуатационные затраты снижает автоматическое восстановление. Отказоустойчивость сервисов оно же повышает. «Автоматизация инфраструктуры сокращает время внедрения изменений за счёт формализованных сборки, развёртывания и проверки состояния компонентов» [7-А]. Автоматическое масштабирование приводит к конкретным техническим сложностям.

Преимущества автоматического масштабирования. Ошибки конфигурации при настройке метрик и порогов замедляют реакцию на резкие скачки нагрузки. Они же вызывают thrashing: частое и необоснованное переключение между состояниями масштабирования. Инфраструктура получает лишнюю нагрузку. Предсказуемость системы падает [6-А].

Ограничения и сложности реализации В распределённой инфраструктуре усложняются мониторинг и диагностика. Сопровождение среды требует более высокой квалификации персонала и дополнительных вычислительных ресурсов. Масштабирование stateful-сервисов (сервисы с сохранением состояния), то есть компонентов, сохраняющих состояние между запросами, составляет отдельную задачу. Жёсткие требования к согласованности данных между компонентами распределённой среды обостряют эту проблему [2-А].

Применение в интеллектуальных системах. Особенности компонентов машинного обучения (высокая вычислительная сложность и неравномерное потребление ресурсов) учитывают при настройке автоматического масштабирования в интеллектуальных микросервисных системах [1-А; 3-А]. Нагрузка на такие сервисы зависит от интенсивности входящих запросов, параметров моделей и объёма обрабатываемых данных. Нужную

эффективность универсальные схемы распределения ресурсов дают не всегда и должны подстраиваться под конкретную модель и сценарий обработки [3-А].

Применение в интеллектуальных системах. Горизонтальное масштабирование (увеличение числа экземпляров) чаще берут для сервисов обработки запросов. Вертикальное масштабирование с дополнительными ресурсами, включая GPU, нередко нужно ML-компонентам. Групповая обработка при батчировании запросов снижает нагрузку на inference-сервисы. Асинхронные очереди сглаживают пики и уменьшают риск перегрузки отдельных компонентов [12-А]. Избыточное потребление ресурсов при inference и обучении моделей сокращает автоматическое масштабирование. Ожидаемый баланс между производительностью и затратами на инфраструктуру ещё предстоит подтвердить экспериментальными данными [12-А].

Практические аспекты реализации. Ряд инфраструктурных компонентов на практике работает вместе с механизмами автоматического масштабирования [4-А; 5-А]. Состояние распределённой среды снимают системы мониторинга, включая Prometheus (открытая система мониторинга и оповещения с открытым исходным кодом); анализ и визуализацию метрик выполняют средствами Grafana (открытая платформа для визуализации, анализа и мониторинга метрик, логов и трассировок).

Распределение запросов и обработку событий в микросервисной инфраструктуре контролируют балансировщики нагрузки и централизованные системы логирования [4-А].

По проверкам состояния сервисов (Health-Check – проверки работоспособности) находят отказавшие компоненты.

Если отказ подтверждён, включается самовосстановление (Self-Healing – самовосстановление): проблемные экземпляры приложений перезапускаются без ручного вмешательства. Контейнеры при Rolling Updates

(скользящие обновления) подменяют по очереди, оставляя систему доступной. Вероятность простоя при такой подмене уменьшается [2-А].

Устойчивость микросервисных систем держат автоматическим масштабированием, встроенным в оркестрацию. Параметры инфраструктуры под фактическую нагрузку и запас ресурсов подстраивают механизмы HPA, VPA и Cluster Autoscaling. Вероятность перегрузки отдельных компонентов уменьшается [6-А]. В микросервисной архитектуре сервисы машинного обучения различаются вычислительной сложностью и потреблением ресурсов, поэтому компоненты приходится масштабировать раздельно.
Автоматическое масштабирование перераспределяет вычислительные мощности между компонентами без постоянного ручного вмешательства. Перераспределение помогает удерживать нужную производительность при inference-задачах (задачи инференса/исполнения моделей) и при обучении моделей [3-А].

Отказоустойчивость, производительность и экономическая эффективность инфраструктуры по опыту эксплуатации распределённых систем зависят от корректной настройки оркестрации и масштабирования [7-А]. Сам эффект, однако, определяют качество метрик, выбранная стратегия масштабирования и свойства сервисов. Автоматическое масштабирование универсальным решением считать нельзя: это инструмент, который нужно подстраивать под конкретную архитектуру системы.

Выводы по второй главе

Подходы, которыми пользуются при создании современных распределённых информационных систем, описывает разбор архитектурных решений для масштабирования микросервисных систем. Разобранные механизмы рассчитаны на производительность и отказоустойчивость инфраструктуры. Менять параметры системы по мере работы с большими объёмами данных и с вычислительно тяжёлыми компонентами, связанными с методами искусственного интеллекта, те же механизмы допускают.

Совместным применением горизонтального и вертикального масштабирования как связанных стратегий управления ресурсами удерживают масштабируемость микросервисной архитектуры. При горизонтальном масштабировании растёт эластичность, а нагрузка распределяется по экземплярам сервисов. Устойчивость к отказам усиливает репликация компонентов. К вычислительно сложным задачам, типичным для сервисов машинного обучения и обработки данных, чаще применяют вертикальное масштабирование.

Показано, что масштабируемость микросервисной архитектуры удерживается совместным применением горизонтального и вертикального масштабирования как связанных стратегий управления ресурсами. Горизонтальное масштабирование повышает эластичность, распределяет нагрузку по экземплярам сервисов и через репликацию компонентов усиливает устойчивость к отказам. Вертикальное масштабирование чаще применяют к вычислительно сложным задачам, типичным для сервисов машинного обучения и обработки данных. По итогам разбора наибольший эффект даёт именно сочетание обоих подходов: так учитываются и разный характер нагрузки, и специфика работы отдельных сервисов в распределённой системе. Для интеллектуальных микросервисных систем с методами машинного обучения задачи масштабирования имеют собственную специфику [7-А]. «Вертикальное масштабирование ограничено физическими пределами узла и хуже подходит для эластичных облачных нагрузок» [96].

Сервисы, обслуживающие пользовательские запросы, обычно наращивают число экземпляров по горизонтали, а вычислительно сложные части, в том числе inference-модули моделей (модули исполнения/вывода моделей), нуждаются в вертикальном расширении ресурсов и, при необходимости, в специализированном оборудовании [12-А]. «Практика Infrastructure as Code повышает воспроизводимость конфигурации среды и снижает конфигурационный дрейф» [119]. Отсюда потребность в комбинированной схеме: разные механизмы распределения нагрузки,

асинхронное взаимодействие сервисов, кэширование и пакетная обработка данных. Существенный итог исследования : уточнение роли оркестрации в управлении распределёнными системами. «Мониторинг распределённых приложений должен учитывать зависимости между сервисами и сетевые задержки» [159].

Оркестраторы (системы оркестрации) автоматизируют развёртывание сервисов, распределение нагрузки между компонентами, восстановление приложений после сбоев и обновления без полной остановки системы. Декларативная модель управления инфраструктурой (подход к администрированию ИТ-систем, при котором в конфигурационных файлах описывается конечный желаемый результат) повышает управляемость программного комплекса и помогает сохранять более предсказуемое состояние распределённой среды при смене нагрузки или отказе отдельных компонентов [11-А]. «Практики DevOps объединяют разработку и эксплуатацию для ускорения поставки изменений при сохранении стабильности» [119]. Наряду с преимуществами выявлены и ограничения эксплуатации распределённой инфраструктуры.

По мере усложнения микросервисной среды усложняется управление сервисами, появляется необходимость во внедрении DevOps practices (DevOps-практик), растут требования к мониторингу, безопасности и сопровождению. «Системы понимания документов учитывают как текстовое содержимое, так и элементы визуальной разметки страницы» [103]. Отдельные трудности связаны с масштабированием stateful-компонентов (компоненты с сохранением состояния) и с поддержанием согласованности данных там, где к стабильности хранения и обработки предъявляются высокие требования. «Надёжная категоризация входящих материалов обеспечивает корректную маршрутизацию документных потоков» [71].

Практическая значимость результатов состоит в том, что их можно применять при разработке и внедрении масштабируемых интеллектуальных систем, включая системы автоматизированного документооборота.

Рассмотренные архитектурные решения повышают производительность обработки данных, удерживают устойчивость к отказам и позволяют рациональнее распределять вычислительные ресурсы при меняющейся нагрузке. «Оптимизация распределения информационных ресурсов в микросервисной схеме опирается на чётко определённый программный интерфейс» [2].

Итоги второй главы сводятся к тому, что горизонтальное и вертикальное масштабирование вместе с контейнеризацией и автоматической оркестрацией составляют технологическую основу современных микросервисных систем. Именно эти подходы позволяют получать адаптивные, устойчивые и масштабируемые программные решения, рассчитанные на условия цифровой экономики и на дальнейшее развитие интеллектуальных информационных технологий.

ГЛАВА 3. СРАВНИТЕЛЬНЫЙ АНАЛИЗ МОНОЛИТНОЙ И МИКРОСЕРВИСНОЙ АРХИТЕКТУР В КОНТЕКСТЕ ТРАНСФОРМАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ И ОПТИМИЗАЦИИ ИТ-РЕСУРСОВ

3.1. Микросервисная архитектура: от монолита к гибким распределенным системам

На нынешнем этапе развитие информационных технологий заметно меняет отрасль программного обеспечения: растут требования к гибкости систем, к скорости разработки и к возможностям масштабирования. «Выделение отдельного сервиса оправдано, если модуль меняется и масштабируется иначе, чем остальная часть системы» [145]. Традиционные монолитные архитектуры, то есть единые и тесно связанные приложения, всё чаще не справляются с адаптацией к новым условиям. Для ряда регионов, в том числе Республики Таджикистан, переход к микросервисной архитектуре имеет уже не только практический, но и стратегический смысл, поскольку связан с необходимостью модернизации существующих информационных систем [3; 13; 15-17]. «Повторное использование логики через сеть обычно дороже локальных вызовов внутри монолитного процесса» [96].

Монолитные системы (МС) держат весь функционал в одной кодовой базе и удобны главным образом на старте разработки. Когда система растёт и структура усложняется, проявляются серьёзные ограничения. При росте нагрузки и объёмов данных масштабировать монолит становится трудно. «Независимое масштабирование сервисов позволяет направлять ресурсы туда, где фактическая нагрузка максимальна» [85]. Изменения и расширение функций тоже затруднены из-за плотной связности компонентов внутри общей архитектуры. В итоге замедляется развитие системы, усложняется внедрение новых решений, а сама модель всё хуже соответствует текущим требованиям рынка.

С учётом требований к масштабируемости, сопровождению и обновлению программных систем переход к микросервисной архитектуре выглядит более обоснованным. «Микросервисы упрощают параллельную

разработку, но увеличивают операционную сложность распределённой среды» [96]. Её основа это разбиение приложения на самостоятельные сервисы с ограниченным набором функций у каждого. Тогда разработку и сопровождение компонентов можно вести автономно, не меняя всю систему сразу. Масштабирование упрощается, а изменения внедряются оперативнее. «Микросервисная архитектура позволяет разрабатывать, развёртывать и масштабировать отдельные компоненты системы независимо друг от друга» [85]. Кроме того, для отдельных сервисов допустимы разные технологии и языки программирования без перестройки архитектуры приложения в целом [8-А].

В Республике Таджикистан переход от монолитных решений к микросервисной архитектуре означает не только обновление технической базы информационных систем, но и смену подходов к разработке программных продуктов. «Автономные микросервисы совместно реализуют бизнес-возможности организации при сохранении слабой связанности между компонентами» [145].

Здесь на первый план выходят масштабируемость, гибкость сопровождения и способность системы подстраиваться под изменение бизнес-процессов [3; 13; 15-17; 28; 2-А; 6-А; 7- Применение микросервисной архитектуры ослабляет зависимость работоспособности всей системы от роста нагрузки на отдельные компоненты и упрощает расширение цифровых сервисов. «К ключевым свойствам микросервисов относят независимое развёртывание, слабую связанность и ориентацию на бизнес-возможности» [131].

Степень изученности задачи. Для организаций региона такая трансформация может считаться одним из условий устойчивого развития ИТ-инфраструктуры, однако вывод о влиянии на конкурентные позиции нуждается в отдельном подтверждении [10-А]. Степень изученности задачи. «Непрерывная поставка требует автоматизации тестирования, сборки артефактов и контролируемого выката релизов» [118]. История внедрения МА

в МС отражает постепенную смену подходов к разработке программного обеспечения. На ранних этапах большинство приложений строили по монолитной модели: базы данных, бизнес-логика, пользовательский интерфейс и вспомогательные модули объединялись в одно приложение. «Наблюдаемость распределённой системы строится на согласованных метриках, логах и трассировках запросов» [121]. Долгое время такая схема считалась стандартной и широко применялась в корпоративных информационных системах. По мере роста функционала и усложнения решений ограничения этого подхода стали заметнее.

Сопровождение крупных монолитных приложений делалось трудоёмким, а любое изменение всё чаще грозило сбоем в других компонентах. «Переход от монолита к микросервисам требует последовательной декомпозиции и осознанного управления распределённой сложностью» [96]. Концепция микросервисов изменила представление о построении программных систем и постепенно сместила развитие от монолита к более гибкой архитектуре. Приложение разделяется на отдельные сервисы: каждый выполняет свою функцию и работает относительно независимо. «Микросервисы остаются практическим ответом на потребность в эволюционирующих и независимо поставляемых распределённых системах» [15]. За счёт этого упрощаются разработка, тестирование и последующее развёртывание компонентов. Правка в одном сервисе не требует полного обновления всего приложения, что снижает сложность сопровождения и повышает устойчивость системы при расширении функционала. Дополнительно можно подбирать разные технологии и инструменты для отдельных сервисов с учётом задач конкретного программного модуля. «Паттерны микросервисной разработки фиксируют повторяемые решения для декомпозиции, взаимодействия служб и повышения устойчивости» [96].

На распространение идей микросервисной архитектуры заметно повлияли специалисты в области разработки программного обеспечения, среди них Martin Fowler [20], Sam Newman [85], Edson Yanaga [168] и другие

исследователи и практики. Их работы во многом сформировали современное понимание принципов построения микросервисных систем. «Сервис-ориентированная архитектура заложила принципы повторного использования служб и контрактного межкомпонентного взаимодействия» [115]. Через публикации, профессиональные конференции, технические руководства и образовательные программы они обобщали опыт применения МА, разбирая масштабируемость, сопровождение и ограничения монолитных приложений.

Переход от монолитной архитектуры к микросервисной модели обычно связан с существенной переработкой программной части и организационных процессов [6-А]. «Микросервисный подход отличается от классической SOA более мелкой гранулярностью сервисов и акцентом на независимое развёртывание» [151]. Меняются структура кода, способы взаимодействия компонентов, инфраструктурная среда и порядок сопровождения программных решений. Иногда приходится пересматривать распределение задач внутри команд и подходы к разработке, тестированию и развёртыванию сервисов. В таких преобразованиях участвуют разработчики, архитекторы программного обеспечения, DevOps-специалистыы (ИТ-профессионалы, которые обеспечивают техническую реализацию DevOps-практик) наряду с прочими участниками проекта, адаптирующими информационную систему к новой архитектурной модели. «Архитектурный стиль микросервисов поддерживает автономность продуктовых команд и практики непрерывной поставки изменений» [105].

Микросервисную архитектуру в разрабатываемый программный комплекс внедряют ради большей гибкости, устойчивости к нагрузкам и масштабирования без чрезмерного усложнения сопровождения. Приложения такого типа проще подстраивать под смену требований и быстрее отвечать на запросы пользователей, что особенно важно при развитии цифровых сервисов. «Эволюционный переход к микросервисам снижает риски за счёт поэтапного выделения ограниченных предметных контекстов» [145]. Изменения касаются и технической архитектуры, и организации разработки, взаимодействия

команд, управления проектами и сопровождения программного обеспечения [3; 13; 15-17]. Задачи внедрения микросервисной архитектуры.

Задачи внедрения микросервисной архитектуры. «Слабая связанность сервисов облегчает замену технологий и позволяет локализовать отказы в пределах одного компонента» [85]. На разных этапах разработки и сопровождения программного обеспечения (рисунок 3.1) возникает широкий набор задач. Одни касаются перестройки архитектуры приложений и взаимодействия сервисов, другие , инфраструктуры, развёртывания, мониторинга и координации команд. «Микросервисная архитектура хорошо согласуется с практиками непрерывной интеграции и поставки в крупных организациях» [126]. Объём и содержание задач зависят от особенностей организации, технологической среды и структуры программной системы [28; 2-А–10-А]. Стратегически переход к микросервисной архитектуре предполагает выбор модели внедрения и порядка преобразований.

Архитекторы и руководство сопоставляют технические возможности, риски и сроки, а также оценивают влияние изменений на бизнес-процессы и на скорость вывода продуктов на рынок [3-А].

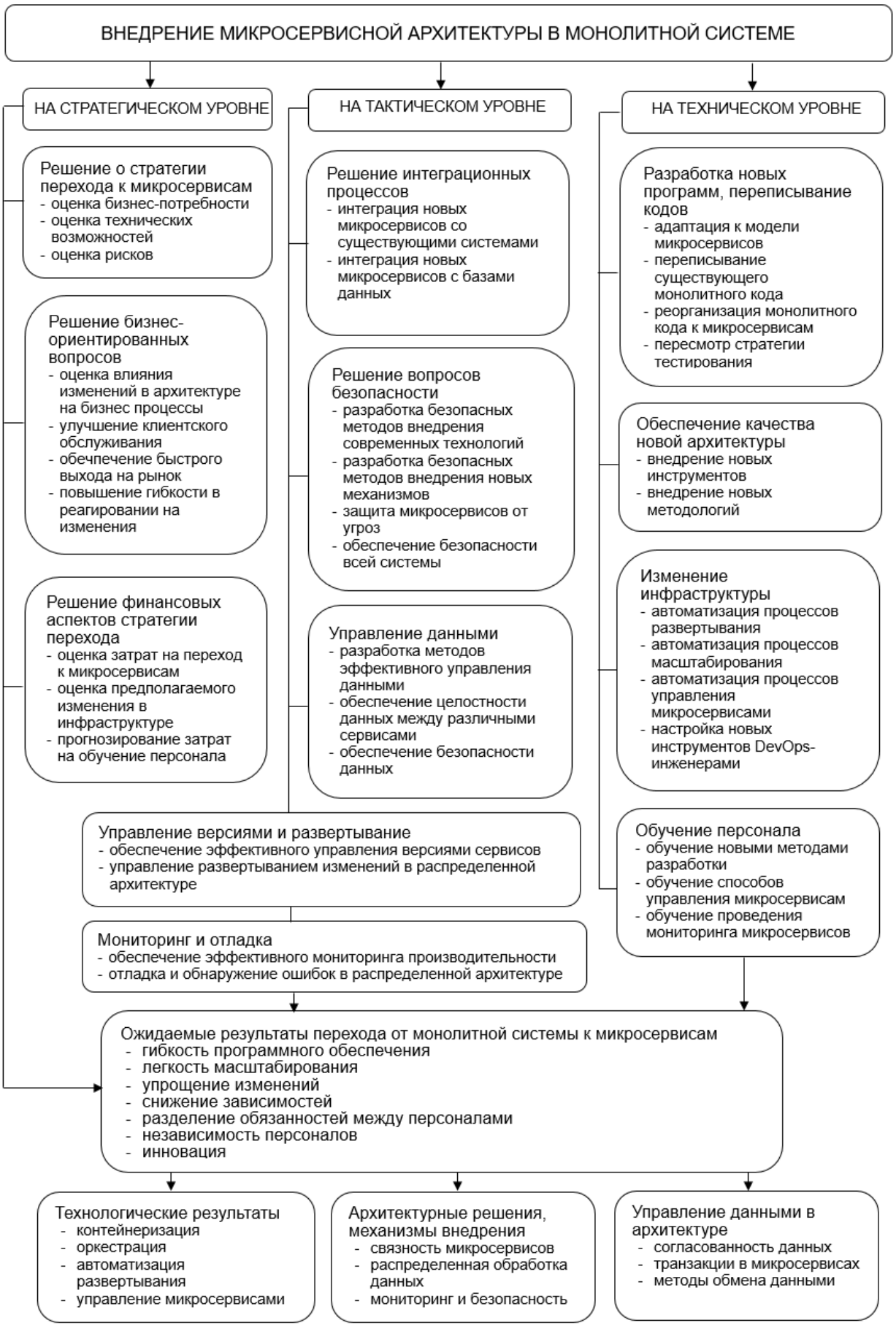


Рисунок 3.1. Решаемые задачи внедрения МА в МС

С финансовой стороны переход тоже требует ресурсов: модернизация инфраструктуры, обучение персонала и приобретение новых инструментов разработки [3-А]. На техническом уровне выделяют четыре направления работ.

На техническом уровне код монолита разработчики переписывают и реорганизуют под микросервисную модель. «Скользящие обновления снижают риск простоя при выводе новой версии сервиса в эксплуатационную среду» [121]. В QA пересматривают стратегии тестирования и подключают новые инструменты. DevOps-инженеры выстраивают автоматизацию развёртывания, масштабирования и управления сервисами. Отдельно организуют обучение команд новым методам разработки и мониторинга.

Открытые вопросы: интеграция новых сервисов с существующими системами и базами данных; управление целостностью и безопасностью данных между несколькими сервисами; мониторинг и отладка в распределённой среде; защита каждого сервиса при разнородном стеке технологий; управление версиями и координация развёртывания изменений. «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этих организациях» [131].

Переход затрагивает сразу техническую, организационную и финансовую стороны; если их не согласовать, ожидаемого результата не будет. «Концепция формирования электронного правительства ориентирует органы власти на развитие цифровых услуг» [71]. Определение стратегии перехода к микросервисной архитектуре.

Определение стратегии перехода к микросервисной архитектуре. Сначала анализируют монолит: фиксируют функциональные блоки, зависимости между ними и проблемные участки. «Границы сервисов целесообразно проводить по предметным областям, а не только по техническим слоям приложения» [85]. По этим данным определяют границы и состав будущих микросервисов [8-А]. На практике используют два основных подхода.

Функциональный закрепляет за каждым микросервисом отдельный бизнес-процесс и тем самым упрощает управление и разработку. Подход по данным выносит в отдельные сервисы компоненты, работающие с разными наборами данных, и обеспечивает их автономность. «В распределённых системах необходимо явно учитывать частичные отказы, сетевые задержки и согласование данных» [159]. Какой вариант выбрать, зависит от структуры системы, бизнес-логики и требований к интеграции; в обоих случаях оценивают риски совместимости.

В итоге нужна система с чёткими API между сервисами, допускающая покомпонентное масштабирование и сопровождение. Выделение функционала из монолита идёт итерациями, и после каждого шага проверяют работоспособность.

Технические и технологические аспекты перехода. «Переход к микросервисной архитектуре строится на трёх группах инструментов: контейнеризации, оркестрации и автоматизации развёртывания» [73]. Контейнеры упаковывают приложение вместе с зависимостями в изолированную единицу. «Автоматическое масштабирование опирается на метрики нагрузки и заранее заданные пороги реакции инфраструктуры» [121].

Технические и технологические аспекты перехода. Наиболее распространённый инструмент здесь : Docker: он даёт одинаковое поведение сервиса в разных средах и упрощает развёртывание. Оркестрационные платформы берут на себя автоматическое развёртывание, балансировку нагрузки и масштабирование контейнеров. «Платформы оркестрации обеспечивают изоляцию сбоев и политики размещения контейнеров по узлам кластера» [1].

Kubernetes управляет кластером и сам восстанавливает упавшие сервисы [79].

Альтернативы — Docker Swarm и Amazon ECS. CI/CD-инструменты — Сборку, тестирование и доставку микросервисов автоматизируют средства Jenkins, GitLab CI/CD и CircleCI.. Это обеспечивает быстрое внедрение

изменений и обновлений в производственную среду без прерывания работы системы [2-А];

Эффективный мониторинг является неотъемлемой частью работы с микросервисами.Состояние и производительность сервисов контролируют стеки вроде Prometheus (открытая система мониторинга и оповещения с открытым исходным кодом), Grafana (открытая платформа для визуализации, анализа и мониторинга метрик, логов и трассировок) и ELK Stack (комплекс инструментов с открытым исходным кодом, предназначенный для централизованного сбора, глубокого анализа и хранения текстовых логов в распределенных системах): метрики собирают, разбирают и выводят на экран, чтобы быстрее находить сбои и устранять их.;

Управление взаимодействием микросервисов часто выносят в слой Service Mesh (сервисная сетка - инфраструктурный слой управления взаимодействием между микросервисами); типичные реализации: Istio (платформа Service Mesh для управления безопасностью, мониторингом и маршрутизацией сервисов) и Linkerd (лёгкая сервисная сетка для управления взаимодействием микросервисов). «Практика Infrastructure as Code повышает воспроизводимость конфигурации среды и снижает конфигурационный дрейф» [119]. Через этот слой задают контроль, наблюдение, политики безопасности и маршрутизацию трафика, из-за чего растёт устойчивость и отказоустойчивость всей системы..

Жизненный цикл микросервисов (развёртывание, масштабирование, обновление) оператор ведёт через веб-интерфейс кластера Kubernetes Dashboard [79; 85]. «Мониторинг распределённых приложений должен учитывать зависимости между сервисами и сетевые задержки» [159].

На практике мониторинг и журналирование снова опираются на Prometheus (сбор метрик и оповещения), Grafana (визуализация метрик и трассировок) и ELK Stack, где Elasticsearch отвечает за поиск и аналитику по журналам, Logstash занимается сбором и преобразованием событий, а Kibana отвечает за их отображение. «Межсервисное взаимодействие следует

проектировать как явный версионируемый контракт, а не как скрытую зависимость» [96].

Перечисленные технологии закрывают лишь фрагмент того набора, которым пользуются при переходе к микросервисной архитектуре. Состав стека подбирают под устройство конкретной системы, требования к безопасности и масштабируемости и прочие ограничения, сверяя его с возможностями и нуждами организации. «Оптимизация распределения информационных ресурсов в микросервисной схеме опирается на чётко определённый программный интерфейс» [2]. Реализация в Таджикистане.

Реализация в Таджикистане. «Концепция цифровой экономики в Республике Таджикистан задаёт рамки внедрения информационно-коммуникационных технологий» [72]. Перевод монолитных систем на микросервисы для ИТ-сферы Таджикистана является заметным этапом развития [1-А–12-А]. Достичь цели можно несколькими параллельными линиями.

Опорой служит обучение и поддержка ИТ-сообщества (IT Communitys – профессиональные сообщества специалистов в области информационных технологий): курсы, тренинги и мероприятия по микросервисной архитектуре готовят специалистов, способных внедрять новые технологии [3; 17]. «Эволюция микросервисной архитектуры требует согласованных практик наблюдения, конфигурации и управления релизами» [119].

Параллельно усиливают ИТ-стартапы (IT Startup – инновационная компания, ориентированная на разработку и коммерциализацию новых информационных технологий) и предпринимательство. «Предоставление финансовой поддержки и инфраструктуры для компаний, желающих внедрить инновационные решения на основе микросервисов, способствует развитию технологического предпринимательства в стране» [13; 15-16].

Центры разработки ПО и инновационные лаборатории формируют среду, в которой молодые специалисты берутся за проекты на базе МА. «Выделение отдельного сервиса оправдано, если модуль меняется и

масштабируется иначе, чем остальная часть системы» [145]. Обмен опытом с зарубежными компаниями и экспертами, партнёрства и привлечение внешних ресурсов дополнительно поддерживают ИТ-сферу (IT Industry – совокупность отраслей, связанных с разработкой, внедрением и сопровождением информационных технологий) в Таджикистане [3; 13; 15-17]. «Повторное использование логики через сеть обычно дороже локальных вызовов внутри монолитного процесса» [96].

Государственная поддержка в виде инвестиций в ИТ-инфраструктуру (IT Infrastructure – совокупность аппаратных средств, программного обеспечения, сетей связи и сервисов, обеспечивающих функционирование информационной системы), а также технологические парки и налоговые стимулы ускоряют рост индустрии информационных технологий в стране. «Независимое масштабирование сервисов позволяет направлять ресурсы туда, где фактическая нагрузка максимальна» [85]. Если государственные и коммерческие проекты в Таджикистане перейдут на микросервисную архитектуру, системы станут гибче, а качество услуг вырастет [13; 15–16]. Архитектурные решения и механизмы внедрения микросервисов.

На этапе проектирования закрывают четыре блока: связность, распределённую обработку данных, мониторинг и безопасность [28; 73; 79; 2-А; 6-А; 8-А]. «Контракты взаимодействия должны версионироваться, чтобы не нарушать работу потребителей сервиса» [96].

Межсервисную связность строят на RESTful API, gRPC либо брокерах сообщений вроде Kafka; конкретный вариант диктует сценарий взаимодействия. Обработку ошибок и устойчивость к сбоям выделяют в отдельную проработку [28; 2-А].

Когда монолит дробят на микросервисы, данные между ними обычно гоняют асинхронными сообщениями либо через событийную архитектуру. «Микросервисы упрощают параллельную разработку, но увеличивают операционную сложность распределённой среды» [96]. Для работы с данными в распределённой среде применяют паттерны CQRS (разделение команд и

запросов) и Event Sourcing (состояние как неизменяемый лог событий) [87]. Производительность и состояние сервисов наблюдают средствами Prometheus и Grafana [73; 79].

Для каждого сервиса отдельно настраивают безопасность: аутентификацию, авторизацию, контроль доступа и шифрование данных при передаче [2-А]. «Микросервисная архитектура позволяет разрабатывать, развёртывать и масштабировать отдельные компоненты системы независимо друг от друга» [85].

Управление данными в микросервисной архитектуре. В управлении данными выделяют согласованность, транзакции и способы обмена между сервисами. У каждого сервиса нередко своя база данных.

Управление данными в микросервисной архитектуре. Протащить ACID-согласованность (атомарность, согласованность, изоляция, устойчивость) через несколько независимых сервисов без единой точки управления данными почти нереально. «Автономные микросервисы совместно реализуют бизнес-возможности организации при сохранении слабой связанности между компонентами» [145]. Классические ACID-транзакции в распределённой среде дают сбой.

Поэтому на практике берут Saga Pattern: длинную транзакцию режут на короткие компенсирующие шаги и так удерживают согласованность при ошибках.

Для обмена данными между сервисами используют RESTful API, gRPC (Protocol Buffers + HTTP/2) либо асинхронную доставку через брокеры Kafka или RabbitMQ. «Выбор формата сериализации данных влияет на задержку и пропускную способность межсервисных вызовов» [76]. Решение зависит от того, какая задержка и какая надёжность нужны конкретному сервису. Анализ и результаты исследования показали, что: 1.

По мере ускорения смены ИТ-ландшафта растут требования к гибкости, масштабируемости и скорости обновления программного обеспечения.

1. «Балансировка нагрузки распределяет запросы между экземплярами и повышает равномерность использования ресурсов» [1].

2. Внедрение микросервисной архитектуры в учреждениях Таджикистана требует одновременно технических и стратегических сдвигов. Тогда повышаются эффективность работы, адаптивность и гибкость разработки, а ответы на рыночные требования становятся своевременнее.

3.2. Анализ характеристик монолитных и микросервисных архитектур

При разработке программного обеспечения сопоставление монолитной и микросервисной архитектур входит в число ключевых проектных задач. «К ключевым свойствам микросервисов относят независимое развёртывание, слабую связанность и ориентацию на бизнес-возможности» [131]. Монолитная модель держится на привычном принципе: весь функционал приложения собран в одном программном блоке. Микросервисная схема действует иначе и дробит систему на независимые сервисы, которые общаются через API (Application Programming Interface – интерфейс прикладного программирования). «Переход от монолита к микросервисам требует последовательной декомпозиции и осознанного управления распределённой сложностью» [96].

В сопоставлении смотрят на масштабируемость, гибкость, управляемость, производительность и надёжность обеих моделей. «Программа реализации концепции государственной информационной политики предусматривает развитие инфраструктуры и сервисов» [95]. Отдельно считают затраты на разработку, развёртывание и сопровождение систем, построенных по монолитному либо микросервисному пути.

Для Республики Таджикистан такой разбор имеет прикладной вес при проектировании программного обеспечения для финансовой сферы, здравоохранения, государственного управления, науки, образования и смежных направлений [13; 15-17; 2-А–9-А]. «Микросервисы остаются практическим ответом на потребность в эволюционирующих и независимо

поставляемых распределённых системах» [15]. Архитектурную модель выбирают с оглядкой на региональные условия: доступность интернета, уровень технической поддержки и наличие специалистов, способных сопровождать выбранную инфраструктуру.

Монолитная архитектура оправдана в проектах, где на первом плане быстрое развёртывание, простота управления и ограниченные ресурсы сопровождения. «Паттерны микросервисной разработки фиксируют повторяемые решения для декомпозиции, взаимодействия служб и повышения устойчивости» [96]. Микросервисная модель уместна для систем, которым нужны гибкое масштабирование, независимое развитие компонентов и устойчивость при росте нагрузки.

С учётом экономических и технических условий, а также отраслевой специфики архитектуру выбирают не по общей симпатии к одной из моделей, а по требованиям конкретного проекта и доступным ресурсам в условиях Республики Таджикистан [16- Задача

настоящего исследования состоит в выявлении преимуществ и ограничений монолитной и микросервисной архитектур для обоснованного выбора проектного решения с учётом особенностей конкретной организации, отрасли и эксплуатационной среды. «Сервис-ориентированная архитектура заложила принципы повторного использования служб и контрактного межкомпонентного взаимодействия» [115]. Анализ особенностей монолитных систем.

Анализ особенностей монолитных систем. Развитие технологий и программного обеспечения усиливает требования к информационным системам. Монолитные решения, долгое время существовавшие как единые приложения, всё чаще упираются в ограничения масштабируемости, гибкости и поддержки новых технологий. «В распределённых системах масштабируемость достигается сочетанием репликации сервисов и управления состоянием данных» [159].

Монолитная архитектура как способ построения ПО объединяет все компоненты и функции приложения в одну систему, чаще всего в виде крупного модуля. На старте разработки такая организация удобна: функционал лежит в одном блоке кода. По мере роста системы и усложнения её структуры ограничения проступают резче. «Эластичность инфраструктуры позволяет наращивать и сокращать ресурсы вслед за изменением профиля нагрузки» [165]. Масштабировать монолитные системы при росте нагрузки и объёма данных становится трудно.

Компоненты тесно связаны между собой, поэтому любое изменение или добавление функционала требует заметных усилий и несёт повышенные риски. Гибкость системы при этом падает. Замедляется и внедрение изменений. Разбор монолитной системы выявляет ограничения, с которыми разработчики сталкиваются при её развитии и сопровождении.

Микросервисная архитектура является одним из ответов: приложение режут на независимые сервисы, и каждый отвечает за ограниченный набор функций. Независимо масштабируют и обновляют отдельные компоненты. Для каждого сервиса допустим собственный технологический стек. Не только технический стек, но и подходы к разработке системы меняет переход к микросервисной архитектуре. Масштабируемость после такой трансформации выше. Гибче становится разработка. К меняющимся требованиям рынка система адаптируется быстрее.

Характерные особенности и наиболее распространённые проблемы монолитных систем, которые часто становятся основанием для перехода к микросервисной архитектуре, приведены на рисунке 3.2 [73; 79; 85; 96; 7-А]. Монолитные системы сочетают преимущества и ограничения.

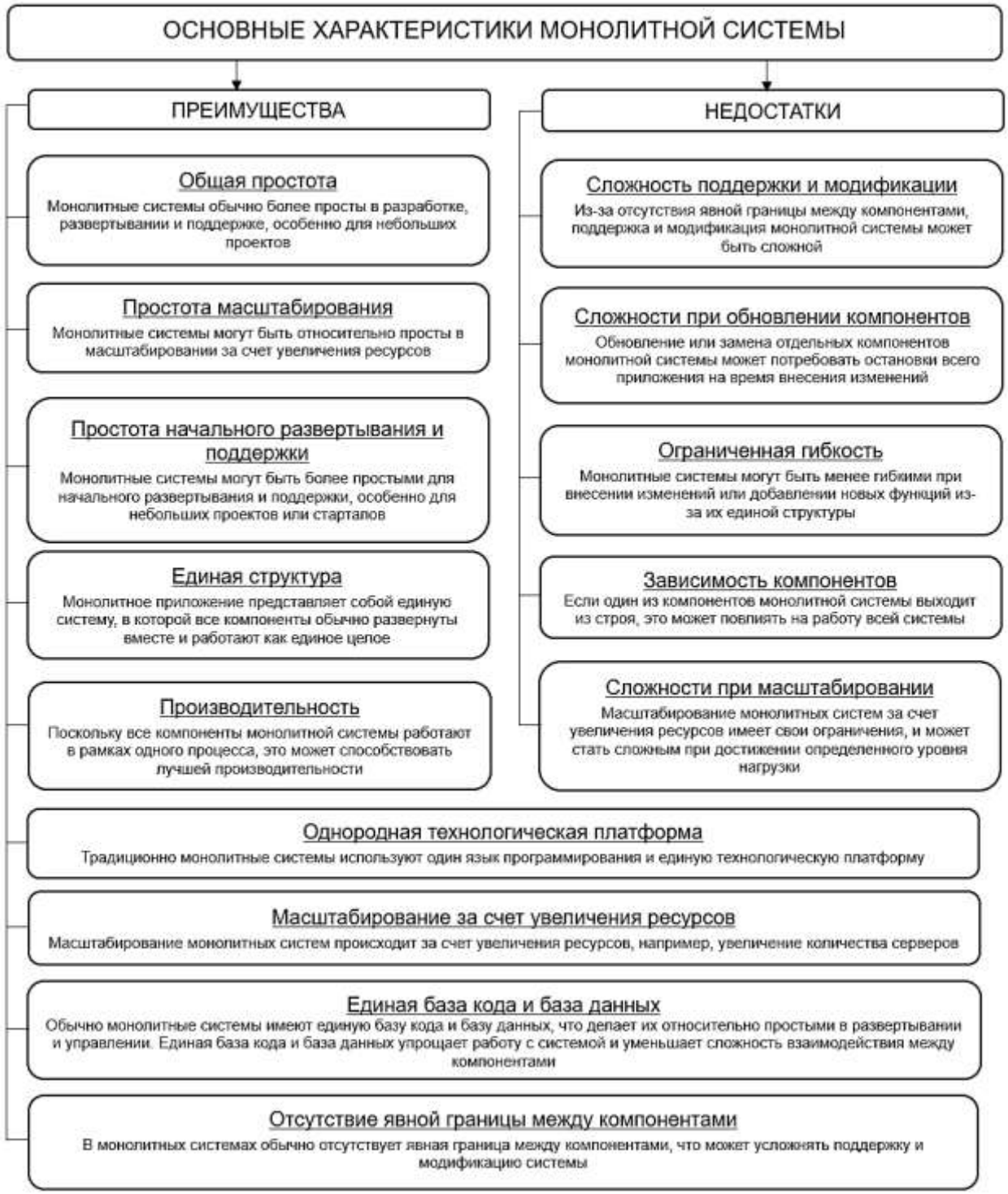


Рисунок 3.2. Графическое представление особенностей и типичных проблем монолитной системы

Обусловлены тесной взаимосвязью компонентов и сложность, и риск изменений монолитной системы. Правка функционала или добавление новых возможностей затрагивает общий блок кода. Вероятность сбоев при этом растёт, сопровождение утяжеляется. Всю систему, а не отдельные компоненты, приходится масштабировать, когда приложение растёт. Гибкость

разработки ограничена: внедрять новые технологии мешает единый технологический стек; обновления тормозит ломка сложившейся структуры.

Монолитная модель на ранних этапах удобна. По мере усложнения системы возникают трудности с масштабированием и сопровождением [7-А]. Микросервисная архитектура выглядит обоснованной альтернативой.

Анализ особенностей микросервисных систем.

Анализ особенностей микросервисных систем. В основе микросервисной архитектуры лежит разделение крупного приложения на независимые сервисы, каждый из которых выполняет отдельную функцию. Такая организация повышает гибкость развития системы и упрощает масштабирование отдельных компонентов, однако требует продуманного управления, настройки инфраструктуры и корректного взаимодействия между сервисами (рисунок 3.3) [6-А]. «Микросервисный подход отличается от классической SOA более мелкой гранулярностью сервисов и акцентом на независимое развёртывание» [151]. Микросервисная архитектура допускает независимое масштабирование отдельных сервисов.

Производительность наращивают не у всей системы сразу, а только у компонентов с основной нагрузкой, и тем самым точнее распределяют вычислительные ресурсы [3; 73; 97; 8- А]. Разделение функционала снижает связанность частей приложения, облегчает локальные правки и расширение возможностей системы. «Кластерное автомасштабирование дополняет масштабирование подов на уровне вычислительных узлов платформы» [121]. Сервисы разрабатывают, разворачивают и обновляют отдельно друг от друга; отказ одного сервиса, как правило, не останавливает приложение целиком. Дополнительный плюс: для отдельных компонентов можно брать разные технологии и языки программирования без переработки всей архитектуры.



Рисунок 3.3. Графическое представление преимуществ и недостатков микросервисной архитектуры

Управлять программной системой при микросервисном подходе становится сложнее. Чем больше сервисов появляется, тем жёстче требования к мониторингу, координации, конфигурации и сопровождению инфраструктуры. «Архитектурный стиль микросервисов поддерживает автономность продуктовых команд и практики непрерывной поставки изменений» [105]. Когда архитектура распределена и нет единого центра управления, сетевое взаимодействие и согласованность данных между сервисами даются особенно тяжело [3-А].

Для управления распределёнными транзакциями применяют Saga Pattern. Инфраструктура мониторинга, оркестрации, управления конфигурацией и проверки состояния сервисов необходима, иначе стабильная работа микросервисов не обеспечивается [88]. «Эволюционный переход к микросервисам снижает риски за счёт поэтапного выделения ограниченных предметных контекстов» [145].

Перевод монолита на микросервисную архитектуру выполняют по нескольким стратегиям. Сначала поэтапно дробят функционал: фиксируют границы сервисов и последовательно выносят отдельные компоненты. Путь декомпозиции выбирают исходя из бизнес-функций, структуры приложения и технологических особенностей системы [73; 87; 97]. «Слабая связанность сервисов облегчает замену технологий и позволяет локализовать отказы в пределах одного компонента» [85].

При декомпозиции монолита может применяться подход Domain-Driven Design (предметно-ориентированное проектирование программных систем) выделяет ключевые бизнес-домены и дробит функциональность на относительно самостоятельные модули. Согласованное взаимодействие сервисов поддерживают за счёт интерфейсов и контрактов обмена данными. «Микросервисная архитектура хорошо согласуется с практиками непрерывной интеграции и поставки в крупных организациях» [126]. Если нужен событийный обмен, берут Event-Driven Architecture (событийно-ориентированная архитектура): сервисы связываются через события [7-А]. Переход к микросервисной архитектуре опирается на инструменты и практики, которые упрощают управление распределённой системой. «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этих организациях» [131].

Контейнеризация и оркестрация средствами Docker (платформа контейнеризации приложений) и Kubernetes (платформа автоматического управления контейнеризированными приложениями) облегчают развёртывание, сопровождение и масштабирование микросервисов.

Состояние сервисов контролируют средствами мониторинга, в том числе Prometheus (открытая система мониторинга и оповещения с открытым исходным кодом) и Grafana (открытая платформа для визуализации, анализа и мониторинга метрик, логов и трассировок). «Стандартизация контейнерных образов снижает расхождения между средами разработки, тестирования и эксплуатации» [146]. Практики DevOps CI (непрерывная интеграция) и CD (непрерывное развёртывание) автоматизируют сборку, тестирование и доставку сервисов и тем самым сохраняют автономность компонентов при переходе к микросервисной архитектуре [96]. Сам переход затрагивает и организацию работы: Agile и DevOps перестраивают взаимодействие разработчиков, операционных команд и службы сопровождения.

«Service mesh дополняет оркестрацию управлением трафиком, наблюдаемостью и политиками безопасности между сервисами» [124]. За счёт DevOps-практик автоматизируются разработка, тестирование и развёртывание, обеспечиваются регулярная поставка изменений и непрерывный контроль состояния системы. На высоконагруженной платформе обработки таджикского текста на базе трансформеров DevOps-практики связывают программный код, математические модели и аппаратные ресурсы кластера в непрерывно функционирующий цикл [6-А]. «Практики DevOps объединяют разработку и эксплуатацию для ускорения поставки изменений при сохранении стабильности» [119].

Системы Continuous Integration (непрерывная интеграция) и Continuous Deployment (непрерывное развертывание) ускоряет подготовку обновлений и автоматизирует их проверку и выпуск, но улучшение качества поставки нельзя считать доказанным без результатов или ссылки на источник [79]. «Границы сервисов целесообразно проводить по предметным областям, а не только по техническим слоям приложения» [85].

Насколько удачен переход к микросервисной архитектуре, зависит от сочетания технических и организационных факторов. Мало внедрить средства контейнеризации, оркестрации и мониторинга: нужно ещё изменить порядок

взаимодействия команд, разработки и сопровождения программного обеспечения. «В распределённых системах необходимо явно учитывать частичные отказы, сетевые задержки и согласование данных» [159]. Микросервисную архитектуру поэтому нельзя считать универсальной заменой монолитной модели; это архитектурный подход, уместный лишь при учёте масштаба системы, требований к развитию, доступных ресурсов и зрелости процессов сопровождения [2-А].

Анализ показывает: монолитные системы, где функционал собран в одном программном блоке, по мере роста нередко упираются в пределы масштабирования и внедрения новых технологий. «Межсервисное взаимодействие следует проектировать как явный версионируемый контракт, а не как скрытую зависимость» [96]. Им противопоставляют микросервисную архитектуру, в которой приложение делят на независимые сервисы с отдельными функциями. При такой организации легче обновлять компоненты локально, можно сочетать разные технологии и языки программирования и проще приспосабливать систему к технологическим сдвигам [6-А]. «Оптимизация распределения информационных ресурсов в микросервисной схеме опирается на чётко определённый программный интерфейс» [2].

Переход к микросервисной архитектуре затрагивает не только техническую перестройку существующих решений, но и организацию разработки и сопровождения программного обеспечения. В данном исследовании микросервисная модель выступает одним из возможных путей повысить гибкость и управляемость информационных систем, рассчитанных на развитие цифровых сервисов и обработку растущих объёмов данных [6-А]. «Эволюция микросервисной архитектуры требует согласованных практик наблюдения, конфигурации и управления релизами» [119].

3.3. Внедрение микросервисной архитектуры для оптимизации распределения информационных ресурсов

Когда общество информатизировано, согласованная работа подразделений и организации в целом опирается на обмен сообщениями,

данными и заявками между структурами. Передача данных и запросов постоянно сопровождает работу предприятий и учреждений: без неё нет взаимодействия подразделений и координации деятельности [3; 121]. «Интеллектуальный документооборот включает не только хранение, но и автоматический анализ содержания документов» [103].

Электронная почта, телефонные звонки и факс как традиционные средства связи не обеспечивают нужной оперативности и прозрачности при передаче информации между подразделениями. Отсюда потребность в специализированных системах обмена сообщениями и заявками.

Такие системы должны вести внутренний документооборот и поддерживать связь с внешними организациями, включая отправку, приём, отслеживание и автоматизацию обработки [13; 15; 16; 127]. «Системы электронного документооборота повышают прозрачность управленческих процедур и сокращают сроки согласования» [71].

Внедряя такие системы, организации стремятся повысить эффективность деятельности, сократить время обработки запросов и текущих задач, наладить взаимодействие с сотрудниками, клиентами и партнёрами. Те же системы укрепляют внутреннюю коммуникацию между подразделениями, снижают риск ошибок и недоразумений, делают рабочие процессы прозрачнее и удобнее для контроля. «Выделение отдельного сервиса оправдано, если модуль меняется и масштабируется иначе, чем остальная часть системы» [145].

По мере усложнения информационных систем растут требования к управлению, масштабированию и обновлению. Тогда микросервисная архитектура помогает оптимизировать информационные ресурсы и повысить эффективность систем. В ряде регионов, в том числе в Республике Таджикистан, её внедрение приобретает стратегический характер и воспринимается как этап модернизации существующих информационных систем [3; 13; 15-17; 128-131]. «Повторное использование логики через сеть обычно дороже локальных вызовов внутри монолитного процесса» [96].

Системы обмена сообщениями и заявками между подразделениями организаций разрабатывают и внедряют, чтобы оптимизировать информационные ресурсы и упорядочить внутренние коммуникационные процессы в министерствах, предприятиях, организациях, образовательных и научных учреждениях Республики Таджикистан [3; 16; 132; 5-А]. «Событийно-ориентированные взаимодействия дополняют синхронные вызовы в современных микросервисных системах» [2].

Node.js (серверная среда выполнения JavaScript-приложений) и React (библиотека JavaScript для разработки пользовательских интерфейсов) выбраны для этой информационной системы из-за функциональных возможностей и распространённости в практике создания веб-приложений. «Проектирование API вокруг бизнес-операций облегчает сопровождение, тестирование и документирование сервисов» [85]. Node.js применяют как серверную платформу, на которой JavaScript работает и на сервере, и на клиенте. Так проще выстроить единую технологическую среду разработки, сопровождения и последующего масштабирования системы [6-А].

Node.js опирается на асинхронную модель ввода-вывода, при которой запросы обрабатываются без блокировки основного потока выполнения. «Чётко определённый API снижает связанность сервисов и упрощает независимую эволюцию программных интерфейсов» [106]. В системах обмена сообщениями и заявками это свойство востребовано: приложениям нужны быстрый отклик и устойчивая работа при параллельной обработке обращений пользователей. Рост производительности и масштабируемости нельзя считать доказанным без нагрузочного тестирования или ссылки на источник [6-А]. «Архитектурный стиль REST задаёт ограничения на взаимодействие ресурсов в распределённых веб-системах» [106]. React как библиотека JavaScript служит для построения пользовательского интерфейса и управления состоянием при многокомпонентной клиентской части.

За счёт виртуального DOM (Virtual Document Object Model – абстракция над объектной моделью документа) и компонентного подхода интерфейс

системы обмена сообщениями и заявками описывают как набор повторно используемых элементов [2-А]. Клиентскую структуру упорядочивают. Сопровождение кода при этом облегчается. Под пользовательский интерфейс, обработку запросов и обмен сообщениями между подразделениями в рассматриваемой системе применяют Node.js и React. Сводить оценку к одной производительности нельзя: учитывают удобство сопровождения, доступность специалистов и требования конкретной информационной системы.

Архитектуру микросервисов задаёт разбиение приложения на автономные сервисы; разработку, развёртывание и масштабирование у них ведут независимо.

Свою бизнес-функцию сервис закрывает сам. С остальными компонентами он связывается через заранее заданные API [28; 136; 2-А]. Под API (Application Programming Interface, интерфейс прикладного программирования) понимают набор правил, протоколов, инструментов и средств взаимодействия компонентов программного обеспечения.

Порядок обмена данными между элементами системы фиксирует API, как и входные параметры, выходные данные и ожидаемое поведение каждого компонента. Спецификации форматов данных, правила обработки ошибок и исключений задаёт тот же API; понятная документация его сопровождает. Устойчивое взаимодействие сервисов в микросервисной архитектуре без такого API не выстроить [7-А]. Гибкость, доступность и масштабируемость информационной системы при оптимизации информационных ресурсов организации микросервисная архитектура способна повысить [нужна ссылка]. Возможности более независимого сопровождения сервисов и адаптации отдельных компонентов к изменениям появляются параллельно.

Влияние микросервисной архитектуры на эффективность команд разработки и конкурентоспособность информационных систем отдельным выводом считать нельзя: требуется дополнительное подтверждение [6-А].

Плюсы у микросервисов есть, ограничения тоже. Построение гибкой и масштабируемой системы относят к преимуществам.

С другой стороны, управление распределённой средой усложняется: размещение сервисов на разных серверах или в облачных средах повышает требования к мониторингу, отладке, безопасности и организации взаимодействия между компонентами. Дублирование отдельных функций в разных сервисах также возможно; оно увеличивает объём кода и усложняет сопровождение [6-А]. «Независимое масштабирование сервисов позволяет направлять ресурсы туда, где фактическая нагрузка максимальна» [85]. В исследовании серверная среда выполнения JavaScript-приложений Node.js принята технологической основой обмена сообщениями и обработки заявок в микросервисной архитектуре.-А].

Благодаря асинхронному вводу-выводу сервер принимает новые запросы, не дожидаясь завершения предыдущих операций, что существенно при большом числе одновременных подключений. «Микросервисы упрощают параллельную разработку, но увеличивают операционную сложность распределённой среды» [96]. Сравнение по производительности с многопоточными системами нуждается в уточнении условий тестирования и в экспериментальном подтверждении [7-А]. Экосистема Node.js содержит библиотеки и модули, пригодные для разработки сервисов обработки сообщений и заявок. «Express используется для создания веб-серверов и обработки HTTP-запросов между клиентом и серверной частью системы» [10-А].

Асинхронная модель обслуживает несколько подключений без отдельного потока на каждое соединение; для рассматриваемой системы это нужно подтвердить нагрузочным тестированием (рисунок 3.4) [7-А]. «Микросервисная архитектура позволяет разрабатывать, развёртывать и масштабировать отдельные компоненты системы независимо друг от друга» [85]. Система обмена сообщениями и заявками ускоряет взаимодействие сотрудников и упорядочивает рабочие процессы [10-А]. В микросервисной

архитектуре инструменты подбирают под назначение каждого сервиса, поэтому компоненты разрабатывают и обновляют независимо.

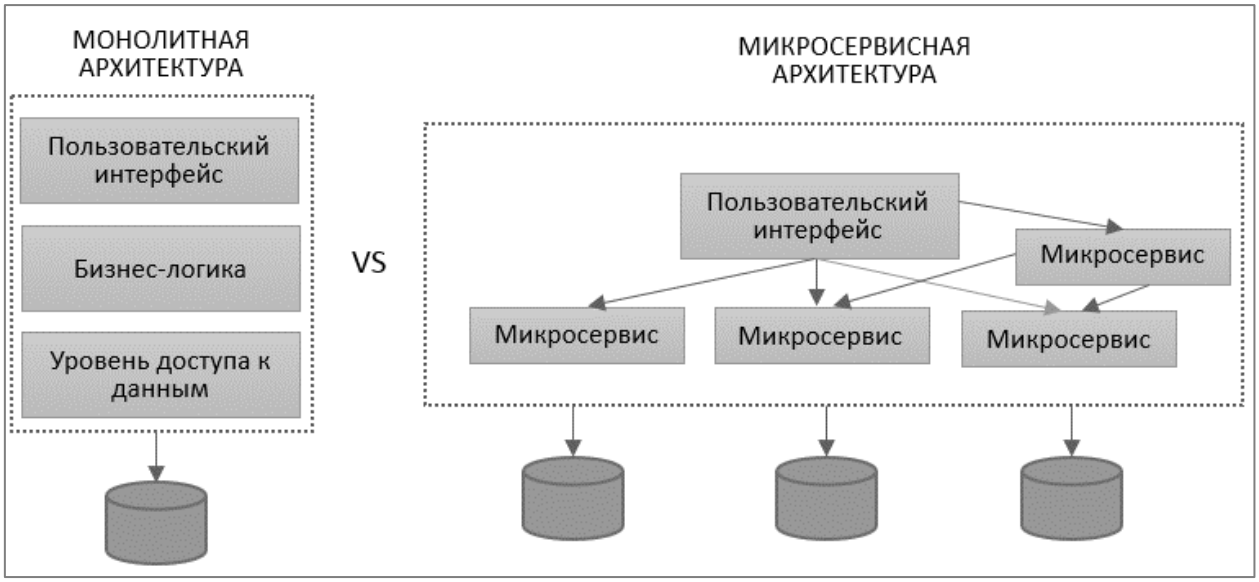


Рисунок 3.4. Принципы функционирования монолитной и микросервисной архитектур

Контроль обработки запросов столь же существенен. «Автономные микросервисы совместно реализуют бизнес-возможности организации при сохранении слабой связанности между компонентами» [145]. Система фиксирует статус заявок, хранит историю переписки и регистрирует выполненные действия. За счёт этого рабочие процессы становятся прозрачнее, а внутренний документооборот, управляемее [10-А].

Переход на микросервисную архитектуру сложен: приходится перерабатывать программный код, пересматривать бизнес-процессы, менять инфраструктуру и перераспределять задачи между командами [6-А]. «К ключевым свойствам микросервисов относят независимое развёртывание, слабую связанность и ориентацию на бизнес-возможности» [131]. В работе участвуют разработчики, архитекторы программного обеспечения, DevOps-инженеры (специалисты на стыке разработки и эксплуатации программных систем) и иные участники проекта, которые адаптируют систему к новой архитектуре (рисунок 3.5).

Внедрение микросервисной архитектуры в систему обмена сообщениями и заявками начинается с подбора технологий для сервисов и

инфраструктуры: языка программирования, фреймворка, базы данных и инструментов управления контейнерами [6].

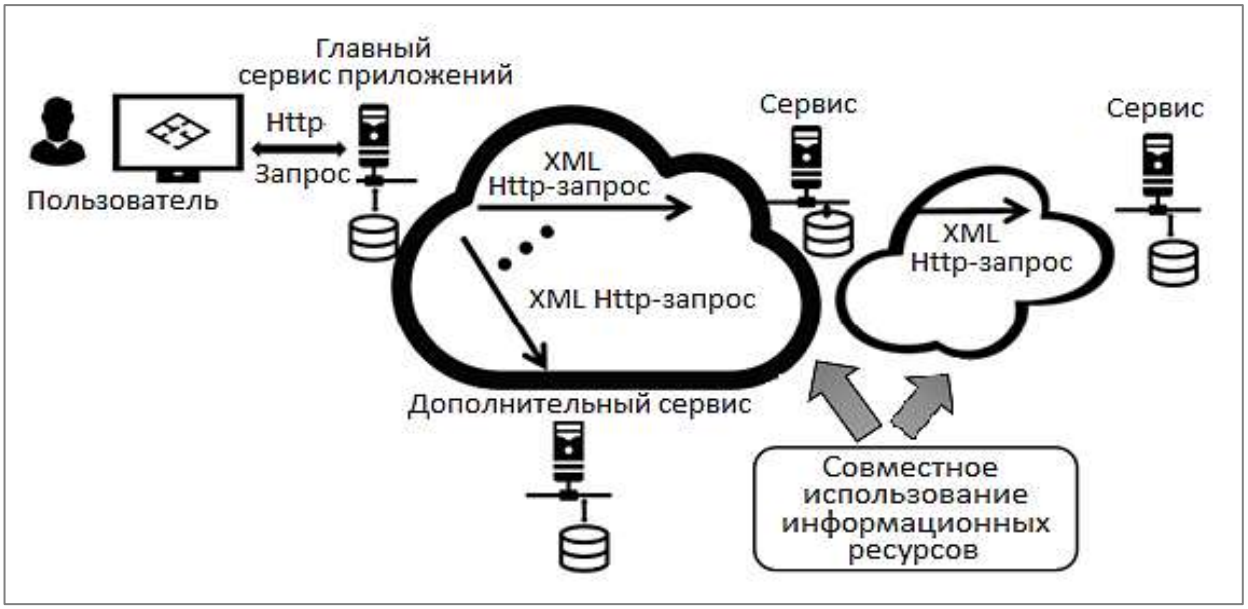


Рисунок 3.5. Сценарий доступа пользователя через браузер к веб-приложению, взаимодействующему с несколькими службами

Затем переходят к разработке и автономному тестированию микросервисов по отдельности. На каждый сервис приходится полный цикл: написание программного кода, отладка, тестирование и оптимизация. «Микросервисы остаются практическим ответом на потребность в эволюционирующих и независимо поставляемых распределённых системах» [15]. После завершения этих этапов микросервисы объединяются в единую систему (рисунок 3.6) [79; 85; 136; 138; 140; 2-А; 6-А; 8-А].

Преимущества микросервисной архитектуры при оптимизации информационных ресурсов не отменяют необходимости оценить её ограничения применительно к конкретной системе и потребностям организации [79; 141-167]. «Паттерны микросервисной разработки фиксируют повторяемые решения для декомпозиции, взаимодействия служб и повышения устойчивости» [96]. Обсуждение результатов.

Обсуждение результатов. Среди ключевых участников управления заявками организации, администратор информационной системы. Он создаёт, изменяет и удаляет заявки, контролирует активность пользователей и

анализирует эти действия. Операции выполняются через административную панель для работы с заявками и пользовательскими учётными записями.

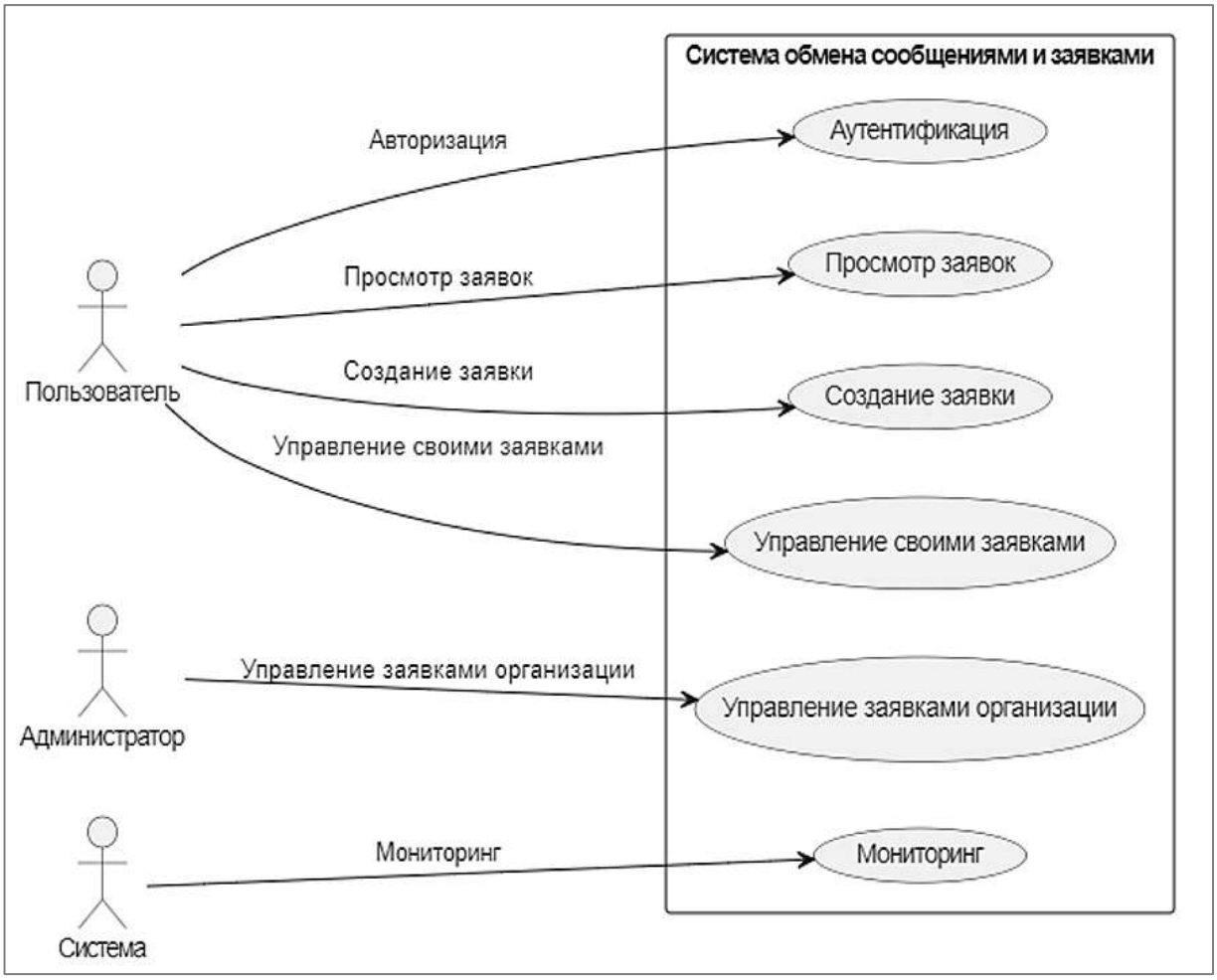


Рисунок 3.6. Задачи внедрения микросервисной архитектуры для оптимизации информационных ресурсов

Система мониторинга поддерживает контроль состояния информационной системы и обмена сообщениями. Отслеживаются активность пользователей, состояние микросервисов, возможные ошибки и сбои. «Сравнение gRPC и REST показывает практический компромисс между производительностью и универсальностью интеграции» [74]. Получаемые данные дают администратору оценить текущее состояние системы и найти участки, где нужна оптимизация [6-А].

Пользователи проходят авторизацию, просматривают доступные заявки, создают новые запросы и управляют ими. К основным функциям ведёт

пользовательский интерфейс, рассчитанный на базовые операции системы обмена сообщениями и заявками.

При интеграции микросервисов у информационной системы растут отзывчивость и масштабируемость, а расширение функционала проходит с меньшими затратами. Сервисы с самостоятельными функциями делают проще разработку, тестирование и сопровождение. Обмен сообщениями и управление заявками для пользователей в итоге удобнее.

Эффективность разработки, более удобное управление информационной системой и рациональное использование информационных ресурсов задают ориентиры стратегий перехода к микросервисной архитектуре в задачах оптимизации. Существующую систему перед переходом разбирают: фиксируют особенности, выявляют проблемные зоны и подбирают способы разнести функционал по отдельным сервисам [6-А]. Сам переход занимает несколько этапов. Систему раскладывают на основные функциональные блоки и зависимости между ними; границы и состав микросервисов, выделяемых из монолитной структуры, задают уже после этого.

К подходам декомпозиции монолитной системы относят функциональное разделение: сервисы ставят в соответствие отдельным бизнес-процессам или прикладным функциям информационной системы. Зона ответственности каждого сервиса задаётся точнее, последующая разработка и сопровождение упрощаются [7-А]. Иной вариант связан с разделением по данным: компоненты, работающие с разными наборами данных, оформляют как самостоятельные микросервисы. Автономность сервисов при этом может усилиться, а разграничение их функций стать более последовательным [7-А].

Структура информационной системы, её бизнес-логика, характер используемых данных и требования к интеграции сервисов определяют, какую стратегию декомпозиции берут. Переход к микросервисной архитектуре сопровождается оценкой рисков, управлением изменениями и сохранением совместимости между выделенными компонентами [7-А].

К главной цели такого перехода относят получение более гибкой, масштабируемой и удобной в сопровождении информационной системы. Функционал постепенно выводят из монолитной структуры, разрабатывают чётко определённые API (интерфейс программирования приложения) для взаимодействия сервисов и проверяют их работоспособность и совместимость при интеграции. Переход итеративен: промежуточные результаты регулярно анализируют и затем корректируют архитектурные решения.

Стабильную работу микросервисной системы поддерживают средствами мониторинга и логирования, включая Prometheus (открытая система мониторинга и оповещения с открытым исходным кодом), Grafana (открытая платформа для визуализации, анализа и мониторинга метрик, логов и трассировок) и ELK Stack (комплекс Elasticsearch, Logstash и Kibana для мониторинга и анализа данных). Состояние сервисов отслеживают, ошибки фиксируют, метрики производительности анализируют, сбои в распределённой инфраструктуре выявляют своевременно [85; 87; 88; 96; 155-161].

В рассматриваемой архитектуре микросервисы на основе небольших модулей, ориентированных на отдельные функции, допустимо строить средствами серверной среды Node.js. Независимая разработка, сопровождение и масштабирование сервисов при такой организации проходят легче; взаимодействие с другими компонентами системы и внешними приложениями тоже упрощается [6-А].

Передачу сообщений и обработку заявок между микросервисами в масштабируемой архитектуре допускают разные протоколы, включая HTTP (Hyper Text Transfer Protocol – протокол передачи гипертекста), WebSocket (протокол связи, позволяющий клиенту и серверу обмениваться данными в реальном времени по одному постоянному соединению) и AMQP (Advanced Message Queuing Protocol – расширенный протокол очередей сообщений). «Серверная среда Node.js поддерживает работу с указанными протоколами и

предоставляет инструменты для их применения при разработке сервисов обмена сообщениями и обработки заявок» [79].

Технологической основой построения микросервисной системы обмена сообщениями и заявками между подразделениями организации в данном исследовании взята серверная среда Node.js. Событийная модель вместе с асинхронным вводом-выводом обслуживает несколько одновременных запросов без блокировки основного потока выполнения и закладывает основу для масштабируемых сервисов [7-А]. Высокую производительность такой реализации всё же нельзя считать доказанной без результатов нагрузочного тестирования или ссылки на соответствующий источник [88; 164].

Поддержка библиотек под протокол связи WebSocket существенна при обмене сообщениями и обработке заявок между подразделениями: серверную среду Node.js применяют для компонентов, которые ведут обмен сообщениями в режиме, близком к реальному времени [6-А].

Модульную организацию кода в серверной среде Node.js используют при проектировании микросервисной архитектуры: приложение делят на небольшие функциональные компоненты. Разработка, развёртывание и сопровождение отдельных сервисов упрощаются, однако величина эффекта зависит от качества декомпозиции, устройства системы API и выбранной инфраструктуры [6-А]. Перечисленные технологии и инструменты составляют лишь часть возможного набора средств для перехода к микросервисной архитектуре.

Особенности информационной системы, требования к безопасности и масштабируемости, а также результаты анализа потребностей организации определяют конкретный выбор. Внедрение микросервисной архитектуры и последующая интеграция микросервисов в информационные ресурсы организации идут поэтапно и требуют детального планирования и последовательной реализации.

Существующую архитектуру информационных ресурсов разбирают в начале работ. Действующую информационную систему изучают, чтобы

выявить её слабые места, потенциальные узкие места и области, нуждающиеся в оптимизации.

Параллельно оценивают возможность разделения приложения на набор независимых микросервисов. Границу и функциональность каждого микросервиса фиксируют далее. Ключевые бизнес-процессы и функции, которые допустимо вынести в отдельные сервисы, выделяют отдельно. У каждого микросервиса при этом должны быть собственная зона ответственности, чётко определённое API и автономный режим работы.

Проектирование микросервисной архитектуры выполняют после этого. На этом шаге подбирают технологии и инструменты разработки и развертывания, задают схему взаимодействия сервисов, включая протоколы обмена данными и форматы сообщений [87; 165]. После интеграции выполняют комплексное тестирование информационной системы.

Работоспособность отдельных сервисов и системы в целом, производительность, устойчивость к нагрузке и способность обслуживать значительное число пользователей и запросов проверяют на этом контуре [5-А]. Внедрение микросервисной архитектуры в информационные ресурсы организации завершает цикл.

В него входят развёртывание системы в промышленной среде, обучение персонала и переход к регулярной эксплуатации. Организация внутренних информационных потоков меняется при внедрении микросервисной системы обмена сообщениями и обработки заявок; оперативность взаимодействия между подразделениями при этом может вырасти [2-А].

В рассматриваемой архитектуре такая система обеспечивает межподразделенческую передачу данных, сообщений и запросов, необходимых для текущей деятельности организации.

Контейнеризация изолирует отдельные сервисы и снижает риск распространения сбоев между компонентами системы [6-А]. При корректной настройке инфраструктуры отдельные функциональные модули

масштабируют в соответствии с изменяющимися пользовательскими требованиями без полной переработки всей платформы.

Объём рутинных операций, связанных с ручной передачей и контролем информации, сокращает автоматизация обработки заявок и сквозного обмена сообщениями [3-А]. Интеграция этих процессов делает взаимодействие подразделений более согласованным и упрощает контроль исполнения запросов. Развёртывание и обновление программных модулей автоматизируют инструменты системы CI/CD.

Автоматизация обработки заявок и сквозного обмена сообщениями сокращает объём рутинных операций, связанных с ручной передачей и контролем информации [3-А]. Интеграция этих процессов делает взаимодействие подразделений более согласованным и упрощает контроль исполнения запросов. Инструменты системы CI/CD автоматизируют развёртывание и обновление программных модулей.

Новые функции внедряют поэтапно, снижая риск технологических простоев, однако утверждение об отсутствии падения производительности требует подтверждения результатами тестирования [2-А].

Микросервисную платформу для обработки заявок и сообщений допустимо рассматривать как средство оптимизации распределения ИТ-ресурсов, используемых в цифровой экосистеме, и повышения управляемости внутренних коммуникационных процессов организации.

Выводы по третьей главе

Сопоставление монолитного и микросервисного подходов к проектированию в разрезе оптимизации ИТ-ресурсов (информационно-технологических ресурсов) высвечивает несколько связанных аспектов.

В традиционном монолите вся логика сосредоточена в едином исполняемом коде; со временем такую систему трудно масштабировать, а любое изменение несёт риск скрытых сбоев. «Проверки работоспособности позволяют автоматически заменять нездоровые экземпляры без остановки всего сервиса» [121]. Там, где высоки требования к отказоустойчивости и код

обновляют часто, эти ограничения становятся блокирующими. Альтернатива, разложить систему на автономные сервисы по бизнес-функциям.

Сервис обновляют независимо, выбирают для него собственный технологический стек, а ошибки удерживают в пределах одного узла. «Непрерывная поставка требует автоматизации тестирования, сборки артефактов и контролируемого выката релизов» [118]. Docker и Kubernetes отвечают за управление, развёртывание и масштабирование распределённых модулей.

CI/CD-инструменты автоматизируют конвейер разработки и сокращают время вывода обновлений на рынок. Смена стека сама по себе всех задач не закрывает.

Миграция на микросервисы предполагает глубокую перестройку организационной структуры и внутренней культуры: от гибких управленческих моделей до сквозного обмена знаниями и плотной координации команд разработки и эксплуатации. «Микросервисный подход отличается от классической SOA более мелкой гранулярностью сервисов и акцентом на независимое развёртывание» [151].

Лишь совместные технические и структурные реформы меняют принципы распределения ИТ-ресурсов (вычислительных, программных, сетевых и информационных компонентов системы). Организация получает возможность быстрее отвечать на рыночные запросы, стимулировать инновации и повышать качество клиентских сервисов. «Архитектурный стиль микросервисов поддерживает автономность продуктовых команд и практики непрерывной поставки изменений» [105]. Переход от монолитной к микросервисной архитектуре задаёт общий запас прочности ИТ-инфраструктуры (совокупность аппаратных, программных и сетевых ресурсов) и напрямую влияет на её адаптивность к жёстким технологическим вызовам.

ГЛАВА 4. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ МИКРОСЕРВИСНОЙ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ДОКУМЕНТООБОРОТА С ИСПОЛЬЗОВАНИЕМ МОДЕЛЕЙ BERT

4.1. Интеллектуальная система документооборота на основе микросервисной архитектуры и моделей BERT

У государственных и коммерческих организаций объём документооборота продолжает расти. «Извлечение именованных сущностей составляет базовый этап структурирования неструктурированных текстовых документов» [102]. Управленческая, юридическая и финансово-экономическая работа порождает договоры, приказы, заявления, служебные записки, счета-фактуры и прочие документы с крупными массивами неструктурированного текста. «Эффективное управление такими данными требует автоматизации процессов извлечения, анализа и структурирования информации» [145].

На практике системы документооборота чаще строятся как монолиты и опираются на правилоориентированные (rule-based) схемы разбора — регулярные выражения и жёстко зафиксированные шаблоны [161; 167]. «Subword-токенизация улучшает обработку редких слов и морфологически сложных словоформ» [156].

Приемлемый результат такие схемы дают лишь при строгом соблюдении форматов, а при росте разнообразия документов масштабируются слабо. Отклонение структуры текста, смена формулировок или языковая вариативность резко снижают точность извлечения данных.

Наиболее остро это проявляется на морфологически сложных и низкоресурсных языках, включая таджикский. «Эволюционный переход к микросервисам снижает риски за счёт поэтапного выделения ограниченных предметных контекстов» [145]. Дефицит крупных размеченных корпусов, нехватка лингвистических ресурсов и высокая вариативность словоформ заметно затрудняют применение традиционных методов обработки текста [165].

Микросервисная архитектура открыла путь к информационным системам, которые масштабируются и сохраняют работоспособность при отказах. «Исследования BERTology систематизируют знания о внутренних представлениях и свойствах трансформерных моделей» [153]. Разбиение функциональности на независимые сервисы даёт возможность гибко встраивать вычислительно тяжёлые блоки, в том числе модули машинного обучения, и наращивать их отдельно от остальной бизнес-логики системы [128; 155].

Одновременно заметно продвинулась обработка естественного языка. «BERT демонстрирует эффективность двунаправленного предобучения трансформеров для задач понимания естественного языка» [13]. Среди трансформерных моделей архитектура BERT (архитектура глубокого обучения на двунаправленных блоках самовнимания) по точности и производительности вышла на наивысшие (state-of-the-art) показатели в задаче распознавания именованных сущностей (Named EntityRecognition), за счёт чего текст понимается с опорой на контекст [121; 127]. «Модель BERT позволяет дообучать единый энкодер под широкий спектр прикладных задач обработки текста» [125].

Вместе с тем основная масса существующих работ сосредоточена на высокоресурсных языках и почти не затрагивает встраивание лингвистических моделей в распределённые производственные контуры.

Исходя из этого, глава ставит целью разработку и анализ интеллектуальной микросервисной системы документооборота, где трансформерные модели машинного обучения обеспечивают автоматическое извлечение структурированных данных из русскоязычных и таджикскоязычных документов. «Слабая связанность сервисов облегчает замену технологий и позволяет локализовать отказы в пределах одного компонента» [85].

Новизна полученных в этой части результатов связана с тем, что сервис искусственного интеллекта (AI Service) выделен как самостоятельный

компонент среды оркестрации, модель адаптируется к низкоресурсному таджикскому языку на синтетических обучающих данных, а извлечение данных реализовано гибридно с автоматической деградацией функциональности [130-138; 154]. «При высокой динамике нагрузки горизонтальная схема обычно предпочтительнее наращивания мощности отдельного узла» [85].

Методы исследования. Исследование опирается на комплексный методологический подход: архитектурное проектирование распределённых информационных систем соединено с методами машинного обучения для обработки естественного языка. Такой набор методов выбран из требований к масштабируемости, отказоустойчивости и точности автоматизированного извлечения данных из текстовых документов [128; 155]. «Микросервисная архитектура хорошо согласуется с практиками непрерывной интеграции и поставки в крупных организациях» [126].

Система спроектирована по микросервисному принципу: функциональность разложена на независимые сервисы, каждый из которых закрывает свой бизнес-домен (Business Domain – предметная область деятельности организации, в рамках которой функционирует информационная система). «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этих организациях» [131]. Вычислительно нагруженные узлы, прежде всего сервис интеллектуального анализа, тем самым изолируются и масштабируются отдельно от прочих элементов системы (рисунок 4.1) [155].

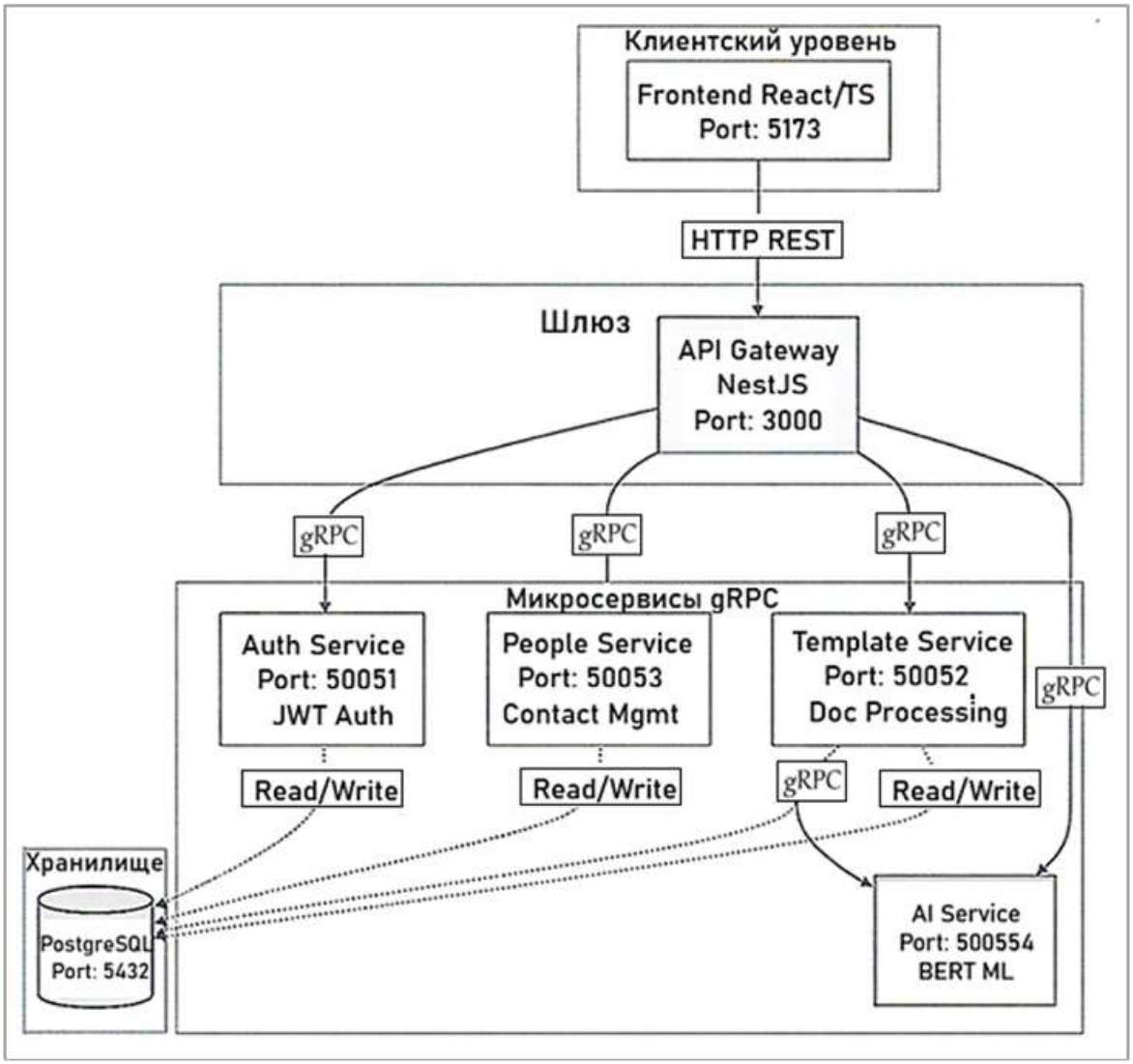


Рисунок 4.1. Общая архитектура микросервисной системы документооборота

Межсервисный обмен построен на высокопроизводительной технологии удалённого вызова процедур (gRPC): структурированные данные сериализуются в бинарном виде через Protocol Buffers, а транспорт обеспечивает HTTP/2. «Асинхронный обмен сообщениями повышает устойчивость распределённых контуров к пиковым нагрузкам» [137]. Фреймворк gRPC выбран из-за низкой латентности, строгой типизации контрактов и эффективной передачи бинарных пакетов — свойства, критичные при обработке документов и результатов нейросетевого инференса (рисунок 4.2.) [160-164].

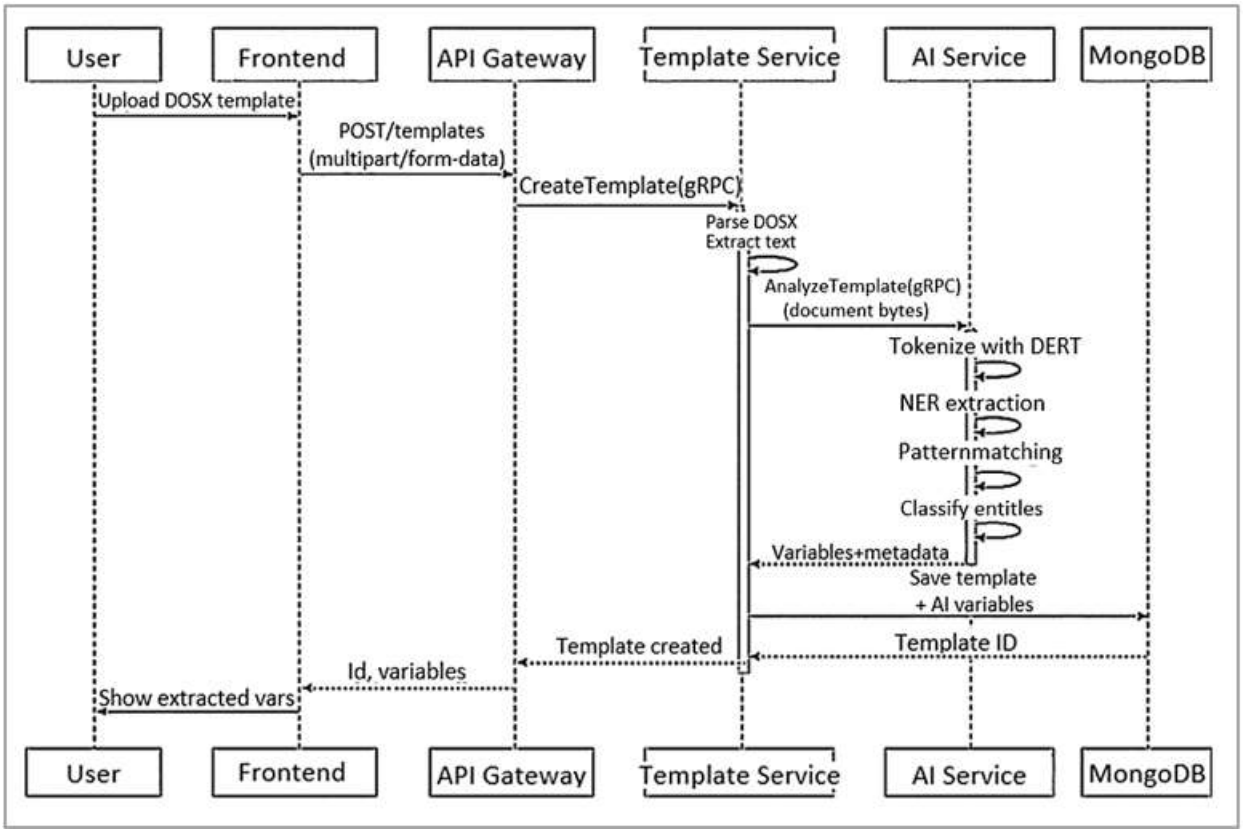


Рисунок 4.2. Взаимодействие микросервисов через gRPC

Ядром системы служит сервис искусственного интеллекта (AI Service): на нём решается задача Named Entity Recognition (распознавание именованных сущностей). «Мультиязычный BERT частично переносит представления между языками, но качество зависит от ресурсности корпуса» [87]. В качестве основы взята многоязычная (multilingual) BERT — трансформерная архитектура, предобученная на большом текстовом массиве. Под документооборот и языковую специфику русского и таджикского языков выполнено дообучение (fine-tuning) по схеме BIO (формализованный способ обозначения границ сущностей в текстовых документах) на размеченных данных [121; 127; 148]. «Архитектура Transformer опирается на механизм внимания и отказывается от обязательных рекуррентных зависимостей» [3].

Обучение требует, чтобы метки именованных сущностей совпадали с субсловным разбиением BERT. Целевая метка только у первого токена слова и исключение остальных субтокенов из функции потерь уменьшают искажение градиентов и укрепляют сходимость.

Объёмных размеченных массивов для таджикского языка практически нет. Лингвистическое обеспечение остаётся узким местом. Частичный выход дал алгоритм синтетической генерации. Шаблонные документы создают с автоматической разметкой сущностей. Без экспертов и без крупных временных затрат собран репрезентативный обучающий корпус [130-140; 154].

Через программный интерфейс сервиса искусственного интеллекта (AI Service) асинхронно стартует обучение, и без остановки основных сервисов модель подстраивается под документооборот ведомства [2-А]. Групповая обработка получает накопленные запросы; при наличии специализированного оборудования ускоряются расчёты [3-А].

Резервный модуль на регулярных выражениях принимает управление при сбое или недоступности AI Service, тогда как в штате гибридного контура действует нейросетевое распознавание. Контур остаётся отказоустойчивым. Точность снижается, бизнес-логика не останавливается [4-А].

Критериями оценки архитектуры были recall, precision и F1-мера; отдельно учитывали утилизацию процессора и памяти. Сравнивали три конфигурации: эвристические правила (rule-based), предобученную трансформерную архитектуру BERT и дообученную модель (fine-tuned model).

Автоматическое извлечение структурированных данных из документов на микросервисной системе документооборота разбирали количественно и качественно. Точность NER, производительность и масштабируемость архитектуры фиксировали в эксперименте. Precision, recall и F1-score, обычные для классификации и NLP, применяли при проверке качества извлечения. Тестовая выборка с обучающей не пересекалась [141; 157].

Результаты показали, что дообученная модель мультиязычного У (multilingual) BERT средняя F1-мера (F1-score) по всем типам сущностей равна 0,93. Показатель традиционных методов выше 0,69 не поднимался. Правилоориентированные (rule-based) схемы на регулярных выражениях

модель заметно обходит; трансформерная модель дала прирост точности более чем на 24% (таблица 4.1) [1-А].

Таблица 4.1. Сравнение методов извлечения данных

Метод	Precision	Recall	$\mathcal{F}_1$-score
Regex	0.71	0.67	0.69
BERT (zero-shot)	0.82	0.79	0.80
BERT (fine-tuned)	0.94	0.92	0.93

По классам именованных сущностей картина такая: персональные данные (PERSON) распознаются устойчивее прочих — $\mathcal{F}_1$-мера доходит до 0,95. «Векторные представления слов открыли путь к переносу семантических признаков между различными задачами NLP» [79]. Денежные суммы (MONEY, $\mathcal{F}_1$ = 0,93) и временные интервалы (DATE, $\mathcal{F}_1$ = 0,91) чуть слабее. Наибольшее число ошибок приходится на адреса (ADDRESS): хаотичная структура и сильная текстовая вариативность класса. Результат предполагался (таблица 4.2) [5-А].

Таблица 4.2. Точность извлечения по типам сущностей

Тип сущности	Precision	Recall	$\mathcal{F}_1$-score
PERSON	0.96	0.94	0.95
DATE	0.92	0.90	0.91
MONEY	0.94	0.92	0.93
ADDRESS	0.89	0.86	0.87

Зависимость целевых метрик от размера обучающего множества фиксировали в экспериментах. Тысяча шаблонно сгенерированных документов даёт около 86% от эффективности модели, обученной на сопоставимом объёме реальных текстов [1-А]. При росте корпуса до 5000-10000 экземпляров статистические колебания гасятся, кривая обучения выходит на плато максимальной точности [4-А].

Производительность архитектуры сводили к времени на один документ и к итоговой пропускной способности. До 90% полного цикла занимает нейросетевой инференс (inference), поэтому один файл проходит примерно за 2-5 секунд [5-А]. Объём данных за единицу времени батчинг (пакетная

обработка) более чем удвоил [4-А]. Почти линейно пропускную способность поднимает горизонтальное добавление экземпляров AI Service [4-А].

Критических задержек соседние модули на пике нагрузки не давали; для выбранного сценария баланс архитектуры сохранялся [136-140]. Резервный контур извлечения отдельно проверяло стресс-тестирование (проверка работоспособности системы в условиях дефицита ресурсов).

Отказ основной ML-модели (Machine Learning Model – модель машинного обучения, то есть алгоритм нахождения закономерностей в данных и принятия самостоятельных динамичных решений) запросы не останавливал: система переходила на правилоориентированную логику (rule-based logic) [1-А]. Падение точности извлечения данных (автоматического процесса сбора и анализа информации из цифровых источников) на жёстких правилах перекрывалось сохранением критических процессов и непрерывной работой комплекса в промышленной эксплуатации [5-А].

По экспериментальным цифрам связка микросервисов и ML-технологий (методов искусственного интеллекта, основанных на самостоятельном обучении систем) практически применима в автоматизации документооборота. Трансформерные модели в этих испытаниях точнее извлекают данные, чем правилоориентированные решения (rule-based solutions); вывод совпадает с общей картиной в NLP [121; 127].

ИИ-сервис (AI Service) вынесен в отдельный блок, поэтому ресурсоёмкий нейросетевой инференс (neural network inference) изолирован. Вычислительные мощности регулируются независимо от остальных сегментов платформы. производительности при добавлении узлов показывает: архитектура выдерживает резкие колебания входящего трафика при заданных условиях инфраструктуры и нагрузки (рисунок 4.3) [128; 155].

Адаптация мультиязычной модели BERT к таджикскому языку заслуживает отдельного внимания. «Модель GloVe формирует глобальные векторы слов на основе статистик совместной встречаемости в корпусе» [148]. Таджикский язык не входил в корпус предобучения изначально, и этот факт

требует отдельного подтверждения [3- А]. В эксперименте дефицит частично закрыли fine-tuning (тонкой настройкой) на синтезированных массивах. Показатели точности оказались сопоставимы с результатами для высокоресурсных языковых групп [142; 147; 153]. «Предобучение на больших текстовых корпусах снижает потребность в размеченных данных для прикладных задач» [160]. Выигрыш показывает: контролируемая генерация обучающих выборок — рабочий инструмент поддержки низкоресурсных языков. 4.2

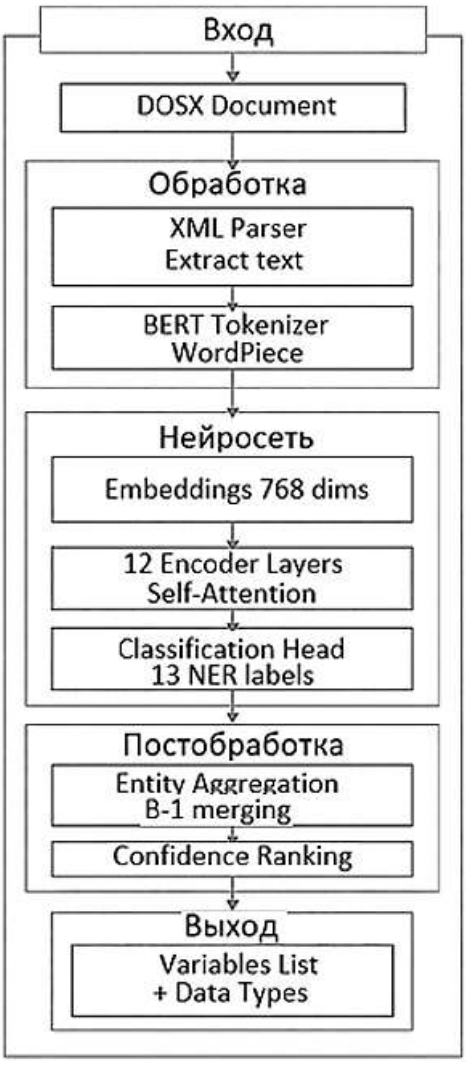


Рисунок 4.3. Архитектура AI Service и pipeline обработки документов

4.2. Разработка математической модели системы документооборота: микросервисы и нейросетевые технологии BERT

В ведомственных и корпоративных системах поток документов превысил возможности ручной обработки. «LayoutLM учитывает текстовое

содержимое и визуальную разметку страницы при понимании электронных документов» [103]. Форматы разнородны, языков несколько.

Rule-based парсинг рушится при малейшей смене формулировок; правила приходится переписывать вручную, и это дорого [130-132].

Отсюда интерес к связке глубокого обучения с распределёнными архитектурами, которые дают масштабируемость, отказоустойчивость и гибкую модернизацию [141-142; 147-153]. Микросервисная архитектура, которую развивают N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R.

Mustfin, L.a («Microservice: yesterday, today, and tomorrow» [131]), M. Fowler, J. Lewis («Microservices: A Definition of This New Architectural Term» [109]), S. Newman («Building Microservices») [85] и ряд других авторов [112-119] дают основание разбить систему на автономные сервисы, связанные через заранее зафиксированные API-интерфейсы (API Interface – программный интерфейс взаимодействия между приложениями). «Границы сервисов целесообразно проводить по предметным областям, а не только по техническим слоям приложения» [85]. Управляемость и эволюционность программного комплекса при этом растут.S. NewmanS. Newman

В такой схеме приложение собирается из слабосвязанных модулей; за каждым закрепляется отдельная бизнес-возможность (business opportunity). Сервис исполняется в собственном процессе и обменивается данными с соседями лёгкими протоколами, как правило HTTP-интерфейсами (API – Application Programming Interface). «В распределённых системах необходимо явно учитывать частичные отказы, сетевые задержки и согласование данных» [159]. Команды получают возможность вести разработку, выкладку и масштабирование своих частей без синхронизации со всем контуром, что ускоряет вывод функций на рынок и расширяет свободу инженерных решений [3-А].

Технологическая гибкость здесь проявляется прямо: стек (язык, СУБД, фреймворк) подбирается под требования конкретного сервиса.

«Межсервисное взаимодействие следует проектировать как явный версионируемый контракт, а не как скрытую зависимость» [96]. Сервис аналитики, к примеру, опирается на графовую базу, сервис аутентификации – на реляционную. Ответственность распределяется между кросс-функциональными командами (ИТ-специалистами для решения одной конкретной задачи), собранными вокруг бизнес-задач, а не вокруг технических слоёв; это согласуется с законом Конвея: «Организации, проектирующие системы, ограничены дизайном, который копирует структуру коммуникаций в этих организациях» [2-А]. Переход к микросервисам вместе с тем вводит типичные трудности распределённых систем: согласование данных без единой транзакционной базы и усложнение межсервисного обмена. «Оптимизация распределения информационных ресурсов в микросервисной схеме опирается на чётко определённый программный интерфейс» [2].

На практике применяют отработанные шаблоны – Saga (паттерн Сага – архитектурный шаблон проектирования) для распределённых транзакций, CQRS (Command Query Responsibility Segregation – архитектурный паттерн) для разделения чтения и записи, Event Sourcing (событийное проектирование) для прослеживаемости состояний. «Эволюция микросервисной архитектуры требует согласованных практик наблюдения, конфигурации и управления релизами» [119]. Мониторинг и распределённая трассировка (Observability) нужны, чтобы удерживать надёжность и быстро локализовать неисправности [6-А].

Вопросы применения микросервисной архитектуры с чётко определёнными API-интерфейсами при цифровизации сфер Таджикистана на уровне корпоративного управления систем автоматизации документооборота («микросервисная архитектура для оптимизации распределения информационных ресурсов», «микросервисная архитектура: от монолита к гибким распределенным системам», «микросервисная оптимизация распределения информационных ресурсов с помощью четко определенного

API» [1-А–12-А] и др.) рассматривались и реализовывались нами (Ф.С. «Доменная адаптация трансформеров, как в BioBERT, повышает качество анализа специализированных текстов» [75]. Комилиён, М.Ф. Рахимов) в работах [1-А–12-А].микросервисная архитектура для оптимизации распределения информационных ресурсовмикросервисная архитектура: от монолита к гибким распределенным системаммикросервисная оптимизация распределения информационных ресурсов с помощью четко определенного API

В обработке естественного языка скачок связан с трансформерными моделями, прежде всего с BERT, предложенной J. Devlin, M. W. Chang, K. Lee и K. Toutanova (BERT: «Pre-training of Deep Bidirectional Transformers for Language Understanding» [13]), опирающейся на механизм самовнимания, описанный A. Vaswani, N. «Условные случайные поля остаются классическим инструментом вероятностной разметки последовательностей» [73]. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser и I. Polosukhin (Attention is all you need) [3]. Для таджикских документов отдельная трудность – дефицит размеченных корпусов. Раздел строит связку из трёх шагов: синтез обучающих текстов, dongK. «Слабая программная разметка ускоряет подготовку обучающих выборок для задач извлечения информации» [95]. LeeK. ToutanovaBERT: «Pre-training of Deep Bidirectional Transformers for Language UnderstandingJ. DevlinM. W. ChangK. LeeK. ToutanovaBERT: «Pre-training of Deep Bidirectional Transformers for Language UnderstandingJ. DevlinM. W. ChangK. LeeK. ToutanovaBERT: «Pre-training of Deep Bidirectional Transformers for Language Understanding

Для таджикских документов отдельная трудность – дефицит размеченных корпусов. «Правила и нейросетевые модели целесообразно сочетать в гибридных контурах анализа документов» [112]. Раздел строит связку из трёх шагов: синтез обучающих текстов, дообучение многоязычной (multilingual) модели BERT и встраивание модели в микросервисный контур по протоколу gRPC [103; 107].

Задача раздела четвёртой главы – формализовать математическую модель интеллектуальной системы документооборота на микросервисах и трансформерах и экспериментально проверить, насколько успешно она извлекает структурированные сведения. «Для низкоресурсных языков критична адаптация мультиязычных моделей и наращивание локальных текстовых корпусов» [87]. Самовнимание имеет сложность O(n2), где n – длина входной последовательности: каждый токен взаимодействует с каждым.

Распределённая микросервисная схема частично компенсирует эти затраты параллельной обработкой. В работе [15] показано, что совместное применение приёмов проектирования распределённых систем и методов NLP даёт дополнительный эффект. «Выделение отдельного сервиса оправдано, если модуль меняется и масштабируется иначе, чем остальная часть системы» [145]. Стек выбран так, чтобы система выдерживала пиковые нагрузки, реже отказывала и сохраняла высокую точность разбора текстов [15; 97].

Методы NLP (Natural Language Processing) в исследуемой системе автоматизируют работу со слабоструктурированными текстами: классификацию, извлечение сущностей, перевод, анализ тональности, построение чат-ботов (имитация реального разговора с пользователем через текстовый или голосовой интерфейс) и умных помощников (AI Assistant/Copilot). «Двунаправленные модели LSTM-CRF долгое время задавали базовый уровень качества разметки последовательностей» [108]. 1.

Математическая постановка задачи NER: при формализации распознавания именованных сущностей документ задаётся упорядоченной последовательностью дискретных токенов (token – базовая неделимая единица текстовой информации: слово, часть слова, символ или знак пунктуации): X= x1,x2,⋯,xn, где xj – j-й токен после токенизации. «Современные библиотеки трансформеров стандартизируют доступ к предобученным языковым моделям и сценариям дообучения» [161].

1. *Математическая постановка задачи NER:* В рамках формализации процедуры распознавания именованных сущностей исходный текстовый

документ представляется как упорядоченная последовательность дискретных токенов (token – базовая неделимая единица текстовой информации: слово, часть слова, символ или знак пунктуации):

$$\mathcal{X} = (x_1, x_2, \cdots, x_n),$$

где x_j – это j-ый токен, полученный в процессе токенизации.

Целевая задача извлечения структурированных данных сводится к классификации последовательностей (Sequence Labeling) – поиску наиболее вероятной цепочки меток:

$$\mathcal{Y} = (y_1, y_2, \cdots, y_n), y_j \in \mathcal{C},$$

где $\mathcal{C}$ – конечное множество классов BIO-разметки.

При этом каждый элемент y_j выбирается из фиксированного алфавита классов BIO-разметки (Beginning-Inside-Outside – стандартная схема кодирования токенов). В терминах вероятностного моделирования последовательностей итоговое уравнение оптимизации для вектора параметров трансформерной модели BERT (θ) формулируется как задача условной максимизации:

$$\hat{\mathcal{Y}} = \arg \underbrace{\max}_{\mathcal{Y}} \mathcal{P}(\mathcal{Y} \mid \mathcal{X};\ \theta).$$

Данный математический аппарат позволяет интерпретировать контекстное извлечение сущностей как максимизацию совместной условной вероятности всей цепочки меток при заданном векторе входных токенов.

2. *Механизм самовнимание в трансформерной архитектуре:* Для вычисления указанной условной вероятности элементы последовательности $\mathcal{X}$ проецируются через матрицу векторных представлений (embedding matrix) в плотные непрерыные эмбеддинги, формируя матрицу признаков $\mathcal{E} \in \mathbb{R}^{nxd}$, где d – размерность скрытого пространства модели. На основе полученной матрицы вычисляются линейные проекции – матрицы запросов ($\mathcal{Q}$), ключей ($\mathcal{K}$) и значений ($\mathcal{V}$):

$$\mathcal{Q} = \mathcal{E}\mathcal{W}^{\mathcal{Q}}, \mathcal{K} = \mathcal{E}\mathcal{W}^{\mathcal{K}}, \mathcal{V} = \mathcal{E}\mathcal{W}^{\mathcal{V}},$$

где $\mathcal{W}^{\mathcal{Q}}, \mathcal{W}^{\mathcal{K}}, \mathcal{W}^{\mathcal{V}} \in \mathbb{R}^{dxd_{\hbar}}$ – матрицы обучаемых весов линейных слоев.

Математический аппарат механизма самовнимания (Attention Mechanism) формализуется слудующим образом:

$$\mathcal{A}(\mathcal{Q}, \mathcal{K}, \mathcal{V}) = softmax\left(\frac{\mathcal{Q}\mathcal{K}^{\mathcal{T}}}{\sqrt{d_{\hbar}}}\right)\mathcal{V},$$

где $\mathcal{Q}, \mathcal{K}, \mathcal{V}$ – матрицы запросов, ключей и значений, $d_{\hbar}$ – размерность пространства ключей, определяющая масштабный коэффициент для стабилизации градиентов функции softmax при попарном взаимодействии токенов [5-А].

Модификация базовой модели в рамках механизма многоголового внимания (Multi-Head Attention) описывается с помощью следующего математического аппарата, позволяющего агрегировать информацию из различных скрытых подпространств [5-А]:

$\mathcal{M}(\mathcal{Q}, \mathcal{K}, \mathcal{V}) = Concat(\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_{\hbar})\mathcal{W}^{o}$, где $\mathcal{H}_i = \mathcal{A}(\mathcal{Q}\mathcal{W}_i^{\mathcal{Q}}, \mathcal{K}\mathcal{W}_i^{\mathcal{K}}, \mathcal{V}\mathcal{W}_i^{\mathcal{V}})$.

Данный механизм обеспечивает моделирование глобальных зависимостей в тексте.

3. *Выходной слой классификации токенов:* На заключительном этапе извлечения структурированных данных скрытые представления, сформированные трансформерной моделью BERT, проецируются на линейный классификатор для предсказания BIO-меток:

Пусть $\mathcal{H} = (\hbar_1, \hbar_2, \cdots, \hbar_n)$ – последовательность скрытых представлений (векторов) на ивыходе последнего слоя модели BERT, где каждый вектор $\hbar_j \in \mathbb{R}^d$ $(j \in \{1, \ldots, n\})$ является контекстуализированным эмбиддингом j-го токена входной последовательности, а d – размерность скрытого пространства трансформера.

Расчёт вектора логитов $z_j \in \mathbb{R}^{|C|}$ для каждого j-го токена осуществляется путем линейного преобразования его скрытого состояния $\hbar_j$:

$$z_j = \mathcal{W}\hbar_j + \mathcal{b},$$

где $\mathcal{W} \in \mathbb{R}^{|\mathcal{C}| x d}$ – матрица обучаемых весов классификационного слоя, $\mathcal{b} \in \mathbb{R}^{|\mathcal{C}|}$ – вектор смещения (bias), а $|\mathcal{C}|$ – мощность множества классов BIO-разметки (количество целевых меток).

Вероятностная модель классификации для определения принадлежности j-го токена к конкретному классу $c \in \mathcal{C}$ формализуется с помощью функции softmax над векторном логитов z_j:

$$\mathcal{P}\left(y_j = c \mid \hbar_j\right) = \frac{exp\left(z_{j,c}\right)}{\sum_{c' \in \mathcal{C}} exp\left(z_{j,c'}\right)},$$

где $z_{j,c}$ – компоненты вектора логитов, соответствующая классу c, а y_j – рекзультирующая классификационная метка для j-го токена послодовательности.

В данном контексте softmax – это математическая функция, которая принимает на вход вектор произвольных вещественных чисел (логитов) и преобразует его в вектор вероятностей.

4. *Функция потерь и процедура оптимизации:* Для обучения параметров трансформерной модели θ используется функция многоклассовой перекрестной энтропии (Cross-Entropy Loss). Специфика токенизации субсловоформ (WordPiece) в условиях ограниченного объёма таджикского корпуса данных требует модификации процедуры расчёта потерь. Чтобы исключить искажение градиентов, из процесса оптимизации исключаются все субтокеновые фрагменты слов (замаскированные индексом – 100), за исключением первого.

Математическое выражение для вычисления функции потерь на уровне последовательности имеет вид:

$$\mathcal{L}(\theta) = -\sum_{j \in \mathcal{J}} \sum_{c \in \mathcal{C}} y_{j,c} \log \mathcal{P}\left(y_j = c \mid \hbar_j; \theta\right),$$

где $\mathcal{J}$ – подмножество индексов токенов, являющихся первыми (или единственными) субтокенами исходных слов в последовательности, $\mathcal{C}$ – множество классов BIO-разметки, а $y_{j,c} \in \{0,1\}$ – индикаторная переменная,

равная единице, если класс c является истинной меткой j-го токена (one-hot encoding).

Задача обучения параметров модели формулируется как минимизация суммарной функции потерь на всем обучающем множестве:

$$\theta^* = \arg \underbrace{\min}_{\theta} \mathcal{L}(\theta).$$

5. *Метрики экспериментальной оценки качества:* Оценка эффективности формализованной интеллектуальной системы при извлечении структурированной информации выполняется на основе строгих критериев соответствия выделенных сущностей (Entity-level evaluation). Вычисление метрик точности (precision – $\mathcal{P}_r$), полноты (recall – $\mathcal{R}_c$) и $\mathcal{F}_1$-меры ($\mathcal{F}_1$-measure) осуществляется на основе подсчёта матрицы ошибок, соттветственно, по формулам [4-А]:

$$\mathcal{P}_r = \frac{\mathcal{T}_\mathcal{P}}{\mathcal{T}_\mathcal{P} + \mathcal{F}_\mathcal{P}};\ \mathcal{R}_c = \frac{\mathcal{T}_\mathcal{P}}{\mathcal{T}_\mathcal{P} + \mathcal{F}_\mathcal{N}};\ \mathcal{F}_1 = \frac{2\mathcal{P}_r\mathcal{R}_c}{\mathcal{P}_r + \mathcal{R}_c},$$

где $\mathcal{T}_\mathcal{P}$ (true positives) – количество истинно положительных исходов (границы и тип именованной сущности в таджикском тексте полностью совпали с эталонной разметкой); $\mathcal{F}_\mathcal{P}$ (false positives) – количество ложноположительных исходов (система ошибочно выделила несуществующую сущность или неверно определила её границы/тип); $\mathcal{F}_\mathcal{N}$ (false negatives) – количество ложноотрицательных исходов (система протустила эталонную именованную сущность).

6. *Метрики оценки эффективности и мсаштабируемости распределённой системы:* Для комплексной валидации разработанной интеллектуальной системы документооборота и оценки влияния синтетических данных на качество извлечения информации вводятся следующие аналитические показатели.

а) Относительный прирост качества извлечения. Изменение интегрального критерия эффективности ($\mathcal{F}_1$-меры) предложенного нейросетевого метода на базе модели mBERT по отношению к традиционному

детерминированному подходу на основе регулярных выражений (Regex) рассчитывается по формуле:

$$\Delta\mathcal{F}_1 = \frac{\mathcal{F}_1^{BERT} - \mathcal{F}_1^{Regex}}{\mathcal{F}_1^{Regex}} \cdot 100\%.$$

b) Эффективность использования синтетических обучающих данных: Для оценки релевантности сгенерированного синтетического корпуса документов на таджикском языке вычисляется отношение качества модели, обученной на искусственных данных, к качеству модели, обученной на реальном размеченном подкорпусе:

$$\mathcal{Q}_{rel} = \frac{\mathcal{F}_1^{synthetic}}{\mathcal{F}_1^{real}}.$$

c) Временные и нагрузочные характеристики производительности системы: Среднее время обработки одного документа ($\mathcal{T}_{avg}$) и общая пропускная способность ИТ-инфраструктуры (Θ) при объёме выборки в $\mathcal{N}$ документов определяется как:

$$\mathcal{T}_{avg} = \frac{1}{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} t_j \,, \Theta = \frac{\mathcal{N}}{\mathcal{T}_{total}},$$

где t_j – время обработки j-го документа индивидуальным микросервисом, а $\mathcal{T}_{total}$ – суммарное время выполнения пакета задач распределённой системой.

d) Показатели горизонтального масштабирования: При развёртывании k независимых экземпляров (подов/контейнеров) NLP-микросервиса в распределённой архитектуре теоретическая пропускная способность системы масштабируется линейно [4-A]:

$$\Theta(k) \approx k \cdot \Theta(1).$$

Коэффициент масштабируемости $\mathcal{S}(k)$, учитывает эффективность параллельной обработки и накладные расходы gRPC:

$$\mathcal{S}(k) = \frac{\mathcal{T}(1)}{\mathcal{T}(k)},$$

где $\mathcal{T}(1)$ – время выполнения фиксированного объёма вычислений на одном узле, а $\mathcal{T}(k)$ – то же на k параллельных узлах.

е) Гибридный механизм отказоустойчивости (Fallback): При сетевых сбоях или перегрузке GPU архитектура переключается на резервный алгоритмический слой. Функция извлечения $\mathcal{F}_{extract}(\mathcal{X})$ определяется так [5-А]:

$$\mathcal{F}_{extract}(\mathcal{X}) = \begin{cases} \mathcal{F}_{BERT}(\mathcal{X}), \text{если микросервис } NLP \text{ доступен} \\ \mathcal{F}_{Regex}(\mathcal{X}), \text{если микросервис } NLP \text{ недоступен.} \end{cases}$$

Данный математический аппарат позволяет гарантировать непрерывность бизнес-процесса документооборота с сохранением базового уровня достоверности анализа текстовой информации даже в условиях пиковых инфраструктурных нагрузок.

Для проведения экспериментов был сформирован набор текстовых данных на таджикском языке, включающий документы административного и делового характера. Общий объём корпуса составил N документов, содержащих различные типы именованных сущностей (персоны, организации, локации, даты).

Данные были разделены на обучающую, валидационную и тестовую выборки в пропорции 70%/15%/15%. Дополнительно использовался синтетически сгенерированный корпус данных, созданный на основе предложенного алгоритма [1-А].

Методология исследования базируется на сочетании распределённых ИТ-архитектур и алгоритмов машинного обучения. Такое решение продиктовано необходимостью создать систему, которая одновременно является:

- *масштабируемой* (для работы с большими объёмами данных);
- *отказоустойчивой* (для стабильной эксплуатации);
- *точной* (для качественного извлечения информации).

Исходя из предложенных соображений, здесь приводится сравнительный анализ двух подходов к распознаванию именованных

сущностей (NER): традиционных методов, основанных на правилах (rule-based), и многоязычной модели (multilingual) BERT. Использование синтетических данных позволило достичь высоких результатов.

Математическая модель архитектуры: В качестве математической модели архитектуры предложено представление системы в виде ориентированного графа сервисов, где итоговая функция извлечения данных $\mathcal{F}_{extract}(\mathcal{X})$ функционирует по гибридному принципу [4-А]:

$$\mathcal{F}_{extract}(\mathcal{X}) = \begin{cases} \mathcal{F}_{BERT}(\mathcal{X}), \text{если нейросетевой сервис доступен;} \\ \mathcal{F}_{Regex}(\mathcal{X}), \text{если нейросетевой сервис недоступен.} \end{cases}$$

Интеграция архитектурных шаблонов распределённой ИТ-инфраструктуры со спецификацией машинного обучения легла в основу автоматизации парсинга текстовых массивов. Внедрение резервных контуров решает проблему непрерывности бизнес-логики, поскольку аварийное отключение нейросетевого компонента не приводит к остановке сквозного документооборота [4-А]. Экспериментально доказано, что спроектированное решение сохраняет баланс между точностью распознавания сущностей и общей стабильностью вычислительных узлов в моменты пиковых нагрузок [1-А].

Сами алгоритмы изначально оптимизировались под горизонтально масштабируемые кластеры. Инженерная оценка эффективности и быстродействия комплекса опирается на среднее время обработки единичного файла, которое вычисляется по формуле:

$$\mathcal{T}_{avg} = \frac{1}{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} t_j.$$

Использование системы gRPC в качестве межсервисного интерфейса позволяет разгрузить каналы связи за счет снижения вычислительных затрат на сериализацию и десериализацию бинарных пакетов данных.

Разработанный метод автоматического синтеза текстовых форм по сути реализует направленную аугментацию (Data Augmentation). Для низкоресурсных языков (low-resource languages), включая таджикский, такой

шаг является единственным способом компенсировать отсутствие масштабных экспертных датасетов [4-А].

Чтобы доказать применимость предложенного решения, было проведено его прямое сопоставление с альтернативными подходами: регулярными выражениями (rule-based), вероятностными моделями условных случайных полей (CRF) и классической нейросетевой архитектурой BiLSTM-CRF. Результаты этих испытаний, а также расчет коэффициента относительного качества сгенерированных шаблонов представлены в таблице 4.3 [1-А].

Таблица 4.3. Сводная таблица эффективности методов

Показатель	**Rule-based (Базовый)**	**BERT (Предложенный)**	**Эффект**
$\mathcal{F}_1$-Score	$\mathcal{F}_1^{Regex}$	0,93	Значительное повышение точности
Масштабируемость	Ограниченная	Линейная	$\Theta(\hbar) \approx \hbar \cdot \Theta(1)$
Адаптивность	Низкая	Высокая	$\mathcal{A}(\mathcal{Q},\mathcal{K},\mathcal{V}) = softmax\left(\frac{\mathcal{Q}\mathcal{K}^{\mathcal{T}}}{\sqrt{d_{\hbar}}}\right)\mathcal{V}$
Работа с данными	Только реальные	Синтетические + Реальные	Качество $\mathcal{Q}_{rel} = \frac{\mathcal{F}_1^{synthetic}}{\mathcal{F}_1^{real}}$

Генерация обучающих множеств для решения задач извлечения сущностей (NER) выстроена на конвергенции архитектурных паттернов распределенных систем и современных инструментов обработки естественного языка. Данный подход гарантирует стабильно высокие метрики извлечения значимых информационных блоков при сохранении потенциала к горизонтальному масштабированию и отказоустойчивости инфраструктуры.

В качестве исходного базиса для конструирования новых экземпляров применяется репрезентативная выборка подлинных документов организации

совместно со специализированными словарями сущностей $\mathscr{E}$. Весь технологический процесс автоматического синтеза логически разделен на три последовательных этап [4-А].

Шаблонизация основана на извлечении из реальных документов контекстных шаблонов $\mathcal{S}$, где именованные сущности заменяются плейсхолдерами (placeholder – местодержатель, заполнитель):

$$\mathcal{S} = \{s_i | s_i = replase(x \in \mathcal{D}_{real}, entity \rightarrow TAG)\}.$$

Например, «Дар асоси фармони (PERSON) аз (DATE)…».

Генерация на основе словарей предполагает выбор случайного шаблона $s \in S$ и его заполнение сущностями из заранее подготовленных словарей $\mathcal{E}_{type}$:

$$\mathcal{X}_{syn} = fill\left(s, \{e_j \in \mathcal{E}_{type}\}\right).$$

За счет этого модель осваивает не только отдельные слова, но и синтаксические контексты их употребления в таджикском языке.

Обратное индексирование и BIO-разметка: После генерации формируется последовательность меток $\mathcal{Y} = (y_1, y_2, \cdots, y_n)$ в формате BIO (Begin, Inside, Outside), что позволяет отказаться от ручной разметки: B-PER/I-PER – для именованных сущностей, B-ORG/I-ORG – для организаций, B-LOC – для локаций [5-А].

Преимущества для модели заключаются в снижении дисбаланса классов за счет возможности генерации редких сущностей в любом объеме. Дополнительно повышается устойчивость: использование механизмов Q_{rel} (относительного качества) позволяет контролировать соответствие синтетических данных реальному распределению.

В работе формализована математическая модель интеллектуальной системы документооборота, объединяющая микросервисную архитектуру и трансформерные методы обработки текста [4-А]. Полученные результаты обобщают три взаимосвязанных направления исследования: построение масштабируемой архитектуры, обеспечение отказоустойчивого извлечения

данных и адаптацию трансформерных моделей семейства BERT к условиям дефицита размеченного материала.

Архитектура системы представлена как ориентированный граф независимых сервисов, взаимодействующих через системы gRPC. Такое представление позволяет описать связи между компонентами в формализованном виде и обосновать возможность масштабирования отдельных сервисов. Утверждение о близкой к линейной масштабируемости $\mathcal{S}(k) \approx k$ должно быть связано с результатами нагрузочного тестирования, соответствующей формулой или экспериментальными данными [5-А].

Рассмотрен механизм гибридной отказоустойчивости $\mathcal{F}_{extract}(\mathcal{X})$ при котором при недоступности нейросетевого модуля система переходит к правилоориентированной (rule-based) обработке. В корпоративных и ведомственных системах это сохраняет работоспособность документооборота при отказе интеллектуального компонента [5-А].

Нехватку размеченных корпусов для таджикского языка решает алгоритм синтетических выборок на основе шаблонов и словарей сущностей. $\mathcal{Q}_{rel}$ показывает близость синтетики к реальному распределению. Конкретные показатели есть в таблице результатов [5-А].

Дообучение многоязычного BERT дало $\mathcal{F}_1$-score = 0,93 [1-А] выше, чем у rule-based методов [5-А]. Это подтверждает работоспособность предложенной конфигурации для задачи NER. Система также показала близкую к линейной масштабируемость при горизонтальном расширении и устойчивость за счёт гибридной схемы обработки, однако эти выводы требуют привязки к результатам нагрузочного и отказоустойчивого тестирования [4-А].

Разработанная модель включает графовое представление сервисов, вероятностную постановку задачи извлечения сущностей, формализацию механизма самовнимания (self-attention), количественную оценку качества и модель масштабирования распределённой системы. Она может использоваться как основа для создания интеллектуальных систем

документооборота в государственных и корпоративных структурах при условии адаптации к требованиям конкретной организации, объёму данных и доступной инфраструктуре [4-А].

Дальнейшие исследования целесообразно связать с ускорением инференса трансформерных моделей при высокой нагрузке, расширением многоязычных корпусов и оценкой возможностей интеграции LLM (Large Language Model – большая языковая модель) для более глубокого семантического анализа документов.

4.3. Микросервисная архитектура и BERT-модели в интеллектуальном документообороте профессионального образования

Объёмы документооборота в государственных учреждениях растут интенсивно, и для обработки неструктурированных данных требуются интеллектуальные методы.

Монолитным системам документооборота и обработке данных по жёстким правилам (rule-based) свойственны низкая адаптивность и ограниченная масштабируемость [16-28; 2-А–8-А].

Построить распределённые системы с высокой отказоустойчивостью и гибким горизонтальным масштабированием стало возможным благодаря переходу к микросервисной архитектуре [127-140; 148; 157; 165].

Параллельно в обработке естественного языка (NLP) усилились трансформерные архитектуры. Модель BERT, в частности, даёт высокую точность извлечения структурированной информации и понимания контекста документа [130; 145-].

В данном разделе четвёртой главы ставится задача разработать математическую модель интеллектуальной системы документооборота на основе микросервисной архитектуры и моделей BERT [131; 136; 148].

- Под организационную схему Центрального аппарата Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан (Комитет) модель настроена с учётом восьми функциональных подразделений (рисунок 13) [5-А]

- Управление начального профессионального образования;
- Управление среднего профессионального образования;
- Управление финансов, планирования и бухгалтерии;
- Отдел кадров, секретариат и специальных работ;
- Отдел международных отношений и связи с общественностью;
- Отдел права, делопроизводства и контроля;
- Сектор информационных технологий и цифровизации;
- Хозяйственный и техническо-вспомогательный отдел.



Рисунок 4.4. Структура Центрального аппарата Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан

Методы исследования. «Повторное использование логики через сеть обычно дороже локальных вызовов внутри монолитного процесса» [96]. Практическая реализация микросервисной системы документооборота на моделях BERT, приспособленная к структуре Комитета, складывается из трёх составляющих:

- *архитектурную модель* – задаёт функциональные и инфраструктурные компоненты;

- *математическую формализацию* – описывает механизмы внимания, метрики эффективности и динамику информационных;

- *логику внедрения* – встраивает модули в бизнес-процессы восьми подразделений и опирается на гибридную модель отказоустойчивости.

1. Математическая модель системы [4-А]:

В роли распределённой системы выступает Центральный аппарат Комитета по начальному и среднему профессиональному образованию:

$$\mathcal{S} = \langle \mathcal{U}, \mathcal{D}, \mathcal{M}, \mathcal{A}, \mathcal{C} \rangle, \quad (1)$$

где $\mathcal{U}$ – множество подразделений Комитета, $\mathcal{D}$ – поток документов Комитета, $\mathcal{M}$ – множество микросервисов, $\mathcal{A}$ – интеллектуальный модуль (BERT), $\mathcal{C}$ – коммуникационная среда (gRPC/API).

1.1. Подразделения Центрального аппарата Комитета (доменная декомпозиция) именуются как:

$$\mathcal{U} = \{U_1, U_2, U_3, U_4, U_5, U_6, U_7, U_8\}, \quad (2)$$

где U_1 – «Начальное профессиональное образование», U_2 – «Среднее профессиональное образование», U_3 – «Финансы и бухгалтерия», U_4 – «Кадры и секретариат», U_5 – «Международные отношения», U_6 – «Право и контроль», U_7 – «IT и цифровизация», U_8 – «Хозяйственный отдел».

2. Поток документов (ключевая часть модели):

Каждый документ Комитета представляется следующим образом:

$$d_i = (x_i, y_i, m_i, t_i), \quad (3)$$

где, x_i – текст документа, y_i – извлечённые сущности, m_i – метаданные, t_i – время обработки.

2.1.Функция обработки документа представляется таким образом:

$$y_i = f_{AI}(x_i), \quad (4)$$

где f_{AI} – трансформерная модель BERT:

$$\hat{Y} = \arg \underbrace{\max}_{Y} \mathcal{P}(Y \mid X;\ \theta), \quad (5)$$

где θ – параметры модели BERT. Такое представление соответствует вероятностной модели последовательной разметки [11; 21].

3. Архитектура микросервисов:

Система разбивается на множества микросервисов, соответствующих подразделениям Центрального аппарата Комитета (таблица 8) [5-А]:

$$\mathcal{M} = \{M_{DOC}, M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}\}. \quad (6)$$

Назначения и задачи микросервисов. Стоит отметить, что помимо микросервисов, перечисленных в формуле (6), в математической модели учтены еще два инфраструктурных компонента (микросервиса) M_{API} (шлюз) и M_{AUTH} (безопасность), которые являются сквозными общесистемными функциями платформы. Они обеспечивают связность, взаимодействие и безопасность для всех функциональных микросервисов, представленных в таблице 4.4, но вынесены за рамки привязки к конкретным подразделениям.

Таблица 4.4. Соответствие микросервисов подразделениям Комитета

Подразделение Комитета	**Архитектурное название микросервиса**	**Функциональное название микросервиса**
Образование (U_1, U_2)	M_{DOC}	Сервис документов (Document Service)
Финансы и бухгалтерия (U_3)	M_{FIN}	Финансовый сервис (Financial Service)
Кадры и секретариат (U_4)	M_{HR}	Сервис кадров (HR Service)
Международные отношения (U_5)	M_{EXT}	Сервис внешних коммуникаций (External Communication Service)
Право и контроль (U_6)	M_{CTRL}	Сервис управления и контроля (Control Service)
IT и цифровизация (U_7) – информационные технологии	M_{AI}	Сервис ИИ и инфраструктуры (AI & Infra Service)
Хозяйство (U_8)	M_{RES}	Сервис ресурсов и архива (Resouce Service)

1) Микросервис M_{API} выполняет роль системного шлюза (API Gateway) и маршрутизатора обращений, т.е. центральной точкой входа всех запросов: $M_{API}: R_{ext} \to R_{int}$, где R_{ext} – внешние запросы, R_{int} – запросы, маршрутизированные к внутренним сервисам. В функции данного микросервиса входят маршрутизация, балансировка нагрузки и агрегация ответов [5-А].

2) Микросервис M_{AUTH} предназначен для аутентификации и авторизации пользователей (обеспечения безопасности): $M_{AUTH}: (u, c) \to \{0, 1\}$, где u – пользователь, c – учетные данные. Данный микросервис реализует функции входа, проверки прав доступа и управления ролями.

Микросервисы, фигурирующие в формуле (6), имеют следующие назначения и задачи:

3) Микросервис документов (M_{DOC}) предназначен для хранения, обработки и управления жизненным циклом документов подразделений сферы «Образование» (U_1, U_2) Комитета: начального (U_1) и среднего (U_2) профессионального образования. Логика работы описывается как $M_{DOC}: x \to (m, s)$, где x – документ, m – метаданные, s – статус документа. Задачами данного микросервиса являются загрузка, хранение, поддержка версионности, передача документов для обработки ИИ и обеспечение целостности нормативно-правовой и учебной документации.

4) Микросервис финансов (M_{FIN}) предназначен для управления бизнес-процессами и сопровождения документооборота подразделения «Финансы» (U_3): $M_{FIN}: (d, s) \to s'$, где $s \to s'$ – изменения состояния документа. В функции данного микросервиса входят обработка финансовых отчетов и смет, извлечение специфических сущностей (сумм, реквизитов), а также контроль бюджетных документов Комитета.

5) Микросервис кадров (M_{HR}) предназначен для работы с личным составом и приказами подразделения «Кадры» (U_4). Задачами данного микросервиса являются ведение личных дел, управление трудовыми договорами и автоматизация кадрового делопроизводства.

6) Микросервис внешних коммуникаций (M_{EXT}) предназначен для взаимодействия с внешними контрагентами для подразделения «Международные отношения» (U_5). В функции данного микросервиса входят обработка входящей/исходящей корреспонденции, маршрутизация запросов и интеграция с внешними шинами данных.

7) Микросервис управления и контроля (M_{CTRL}) предназначен для юридического сопровождения и надзора подразделения «Право» (U_6). В функции данного микросервиса входят согласование документов и регламентов, проверка прав доступа, контроль исполнения поручений и документов, маршрутизация документов между подразделениями Комитета.

8) Микросервис ИИ и инфраструктуры (M_{AI}) предназначен для интеллектульной обработки и анализа текстов документов (распознавания сущностей NER, классификации) для нужд IT-подразделения Комитета (U_7). Математическое описание работы микросервиса M_{AI} основывается на функции обработки документа (4): $y_i = f_{AI}(x_i)$, где f_{AI} реализуется через модель BERT в виде задачи классификации или разметки последовательностей (5): $\hat{Y} = \arg\max_{Y} P(Y \mid X; \theta)$, где θ – параметры модели. Такое представление позволяет микросервису M_{AI} эффективно извлекать именованные сущности и классифицировать документы для нужд ИТ-подразделения Комитета (U_7). Итак, микросервис M_{AI} реализует функции классификации документов на основе моделей BERT, извлечения именованных сущностей (ФИО, дата, сумма) (NER), балансировки нагрузки, текстового анализа документов и агрегации аналитических ответов.

9) Микросервис ресурсов и архива (M_{RES}) предназначен для технического обеспечения и долговременного хранения документов (архивирования) для подразделения «Хозяйство» (U_8): $M_{RES}: d \to d_{archive}$. Задачами данного микросервиса являются долгосрочное хранение, поиск и восстановление документов Комитета, а также учет его материально-технических ресурсов [5-А].

Проверка общей формулы (6). Для проверки необходимо объединить все описанные компоненты в итоговую формулу (6). «К ключевым свойствам микросервисов относят независимое развёртывание, слабую связанность и ориентацию на бизнес-возможности» [131]. Поскольку в тексте упоминались 8 функциональных микросервисов для обеспечения работы восьми

подразделений Комитета (U_1-U_8) и два инфраструктурных общесистемных сервиса (M_{API}, M_{AUTH}), математическую модель структуры системы представим в виде кортежа [5-А]:

$$\mathcal{M} = \langle M_{API}, M_{AUTH}, \{M_{DOC}, M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}\}\rangle. \quad (6')$$

В итоговой модели (6') инфраструктурный слой – M_{API} (шлюз) и M_{AUTH} (безопасность) – обеспечивает сквозное взаимодействие, а функциональный слой – специализированные микросервисы $M_{DOC}, M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}$ – обеспечивает системную работу автоматизируемых подразделений Комитета ($U_1, \dots, U_8$).

Взаимодействие микросервисов в рамках представленной модели осуществляется по принципу событийной архитектуры или через API-шлюз (M_{API}). Процесс обработки данных в системе можно представить следующим образом:

- внешний запрос R_{ext} поступает на M_{API}, где проходит первичную фильтрацию и проверку прав доступа через M_{AUTH};

- после авторизации запрос маршрутизируется в соответствующий функциональный микросервис (M_{DOC}, M_{FIN}, M_{HR} и др.) в зависимости от типа задачи и принадлежности к подразделению U_i;

- при необходимости интеллектуального анализа данных (NER, классификация) управление передается микросервису M_{AI} (нейросетевой модуль), результаты работы которого возвращаются в целевой сервис или сохраняются в архив M_{RES}.

Таким образом, математическая модель (6') обеспечивает модульность и масштабируемость системы, позволяя независимо развивать функциональность каждого подразделения Комитета без нарушения целостности общесистемных функций [5-А].

Описание информационных потоков в системе. Компоненты модели (6') обмениваются данными либо по общей шине событий, либо через REST-взаимодействие, канал выбирают по типу задачи. «Контракты взаимодействия

должны версионироваться, чтобы не нарушать работу потребителей сервиса» [96]. Потоки разведены по назначению. Управляющий поток стартует на M_{API}, проходит верификацию в M_{AUTH} и доставляет в модули (M_{DOC}, M_{FIN}, M_{HR}) команды вида «создать документ» или «проверить статус». В потоке данных : содержимое документа x из M_{RES} поступает в M_{AI}, Там вычисляют $y = f_{AI}(x)$, после чего обновляют m в сервисе документов M_{DOC}. Одновременно M_{CTRL} фиксирует движение документов между подразделениями ($U_i \to U_j$) и пишет смену состояний $s \to s'$ в журнал событий.

Разведение потоков даёт два следствия. Во-первых, отказы локализуются: падение M_{FIN} не останавливает кадровые операции в M_{HR}, при отсутствии жёсткой зависимости сервисов [5-А]. «Автоматическое масштабирование опирается на метрики нагрузки и заранее заданные пороги реакции инфраструктуры» [121]. Во-вторых, масштабирование независимое: при росте объёма документов ресурсы добавляются в M_{AI}, не затрагивая остальные узлы [4-А].

Модуль на основе BERT строится на механизме внимания (Attention):

$$\mathcal{A}(\mathcal{Q}, \mathcal{K}, \mathcal{V}) = softmax\left(\frac{\mathcal{Q}\mathcal{K}^T}{\sqrt{d_\hbar}}\right)\mathcal{V}, \tag{7}$$

где $\mathcal{Q}, \mathcal{K}, \mathcal{V}$– матрицы запросов, ключей и значений; $d_\hbar$ – размерность пространства ключей. Матрицы $\mathcal{Q}, \mathcal{K}$ и $\mathcal{V}$ получаются умножением входного вектора на обучаемые матрицы весов [165].

Многоголовое внимание (Multi-Head Attention):

$$\mathcal{M}(\mathcal{Q}, \mathcal{K}, \mathcal{V}) = Concat(\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_\hbar)\mathcal{W}^o, \tag{8}$$

где $\mathcal{H}_i = \mathcal{A}\left(\mathcal{Q}\mathcal{W}_i^{\mathcal{Q}}, \mathcal{K}\mathcal{W}_i^{\mathcal{K}}, \mathcal{V}\mathcal{W}_i^{\mathcal{V}}\right)$ [4; 10]. Каждая голова i захватывает отдельный тип связей между фрагментами текста; параметры $\mathcal{W}_i^{\mathcal{Q}}, \mathcal{W}_i^{\mathcal{K}}, \mathcal{W}_i^{\mathcal{V}}$ для каждой головы независимы [5-А].

Метрики эффективности системы. Точность (P_r) определяется так:

$$\mathcal{P}_r = \frac{\mathcal{T}_\mathcal{P}}{\mathcal{T}_\mathcal{P} + \mathcal{F}_\mathcal{P}}, \tag{9}$$

где $\mathcal{T}_{\mathcal{P}}$ – корректно классифицированные документы или сущности (к примеру "Приказ"); $\mathcal{F}_{\mathcal{P}}$ – ложные срабатывания. Чем меньше $\mathcal{F}_{\mathcal{P}}$ тем реже система ошибается с классом [141].

Высокое значение $\mathcal{P}_r$ означает уменьшение ложных срабатываний.

Полнота ($\mathcal{R}_c$) рассчитывается так:

$$\mathcal{R}_c = \frac{\mathcal{T}_{\mathcal{P}}}{\mathcal{T}_{\mathcal{P}} + \mathcal{F}_{\mathcal{N}}}, \tag{10}$$

где $\mathcal{F}_{\mathcal{N}}$ это документы или сущности, которые система пропустила или отнесла не к тому классу. Низкий $\mathcal{F}_{\mathcal{N}}$ означает меньше пропущенных значимых объектов [141]. Высокий $\mathcal{R}_c$ означает меньше потерь важных данных.

Интегральная оценка качества $\mathcal{F}_1$-мера:

$$\mathcal{F}_1 = \frac{2\mathcal{P}_r\mathcal{R}_c}{\mathcal{P}_r + \mathcal{R}_c}. \tag{11}$$

$\mathcal{F}_1$ это гармоническое среднее $\mathcal{P}_r$ и $\mathcal{R}_c$. Для задач Комитета перекос в любую сторону снижает надёжность: пропуск документа опасен так же, как ложное срабатывание [5-А]. $\mathcal{F}_1$ здесь основная метрика оценки BERT при классификации и извлечении сущностей в данной работе [141].

В Комитете по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан цена ошибки при обработке приказов (U_1, U_2) и финансовых документов (U_3) высока [4-А]: неверно извлечённая сумма или ФИО это уже юридически значимая неточность [5-А]. «Развитие профессионального образования связывается с цифровизацией учебных и административных процессов» [70]. Перед вводом в эксплуатацию для M_{AI} устанавливаются пороговые значения метрик как критерий допуска модели к работе с документами [5-А]

Критический уровень $\mathcal{F}_1 \geq 0{,}85$минимально допустимый порог для общей классификации и извлечения базовых сущностей, включая даты и типы документов [1-А]. При $\mathcal{F}_1 < 0{,}85$ результаты требуют обязательной ручной проверки.

Целевой уровень $\mathcal{P}_r \geq 0{,}92;\ \mathcal{R}_c \geq 0{,}90$ соответствует режиму частичной подразделениях «Финансы» (U_3) и «Кадры» (U_4) [1-А]. Точность в приоритете: ошибочное извлечение суммы или ФИО тяжелее по последствиям, чем пропуск документа, который уйдёт на ручную проверку [5-А].

Эталонный уровень ($\mathcal{F}_1 \geq 0{,}95$) режим «автопилота» с выборочным контролем не более 5% документов [5-А].

Производительность системы: среднее время обработки ($\mathcal{T}_{avg}$):

$$\mathcal{T}_{avg} = \frac{1}{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} t_j, \tag{12}$$

где $\mathcal{N}$ – число обработанных документов за период, t_j – время обработки j-го документа от поступления в микросервис до выдачи результата. Чем ниже $\mathcal{T}_{avg}$, тем быстрее пользователь получает ответ, например от M_{AI} при классификации [142].

Пропускная способность:

$$\Theta = \frac{\mathcal{N}}{\mathcal{T}_{total}}, \tag{13}$$

где Θ – число документов в секунду, $\mathcal{N}$ – число операций, $\mathcal{T}_{total}$ – общее время работы. Если Θ ниже интенсивности входящего потока, в системе нарастает очередь [142].

Горизонтальное масштабирование:

$$\Theta(k) \approx k \cdot \Theta(1), \tag{14}$$

где k – число реплик, $\Theta(1)$ – производительность одного экземпляра, Добавляем реплики – растёт суммарная Θ линейно [5-А].

Коэффициент масштабируемости:

$$\mathcal{S}(k) = \alpha \cdot \frac{\mathcal{T}(1)}{\mathcal{T}(k)}, \tag{15}$$

где $\mathcal{S}(k)$ – коэффициент ускорения, $\mathcal{T}(1)$ – время выполнения одним узлом, $\mathcal{T}(k)$ – то же при k узлах, α – доля последовательной части вычислений [4-А].

Формула (15) это закон Амдала (Amdahl's Law) в упрощённом виде [121]. Он показывает реальную эффективность распараллеливания. В идеальных условиях $S(k) = k$, но на практике значение всегда чуть меньше из-за затрат на сетевое взаимодействие между микросервисами и работу API-шлюза M_{API}. Коэффициент α в формуле (15) введен в связи с тем, что из-за сетевых задержек и накладных расходов на оркестрацию идеальное линейное ускорение $\Theta(k) \approx k \cdot \Theta(1)$ в реальных условиях недостижимо [4-А].

Бизнес-логика и адаптация. Модель адаптирована под регламенты Центрального аппарата Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан [3-А; 4-А; 5-А]:

Автоматизация процессов:

Управление образования (U_1, U_2):

- классификация приказов;
- извлечение учебных данных;
- контроль сроков.

Финансы (U_3):

$Amount = f_{AI}(x)$:

- автоматическое извлечение сумм, счетов.

Кадры (U_4):

- извлечение ФИО, должностей;
- анализ приказов.

Международные отношения (U_5): реализует функцию кросс-языкового NER (таджикский, русский, английский) для обработки межгосударственных соглашений:

- многоязычная обработка (TJ+RU+ENG);
- перевод + NER.

Правовой отдел (U_6):

- контроль исполнения;
- анализ юридических формулировок.

ИТ-сектор (U_7):

- управление ИИ-сервисом;
- обучение модели.

Хозяйственный отдел (U_8):

- учет ресурсов;
- автоматизация заявок.

Гибридная модель отказоустойчивости (Fallback-механизм):

$$f(x) = \begin{cases} f_{AI}(x), & \text{если AI доступен} \\ f_{rule}(x), & \text{если AI недоступен.} \end{cases} \quad (16)$$

При снижении уверенности f_{AI} ниже порога ρ<0,7 или технической недоступности GPU-мощностей (требуемого объёма вычислительных ресурсов графических процессоров, необходимых для работы нейросетей), система переключается на предиктивный анализ по правилам f_{rule} [1-А].

Итоговая система (интегральная модель): накопления очереди документов описывается дифференциальным уравнением:

$$\frac{d\mathcal{D}}{dt} = \lambda_{in} - \lambda_{proc}(\mathcal{M}, \mathcal{A}), \quad (17)$$

где λ_{in} – интенсивность входящего потока, λ_{proc} – скорость обработки документов. Изменение объема необработанных документов $d\mathcal{D}/dt$ зависит от баланса входящего трафика λ_{in} и производительности микросервисной архитектуры λ_{proc}. Условием стабильности системы является $\lambda_{proc} > \lambda_{in}$, иначе очередь $\mathcal{D}$ будет расти бесконечно [5-А].

Результаты исследования. «Микросервисный подход отличается от классической SOA более мелкой гранулярностью сервисов и акцентом на независимое развёртывание» [151]. Математическая модель интеллектуальной системы документооборота, базирующаяся на микросервисной архитектуре и алгоритмах BERT адаптированной под структуре Центрального аппарата Комитета по начальному и среднему профессиональному образованию, позволила получить следующие результаты:

- готовая архитектура под реалный госорган (Комитет по начальному и среднему профессиональному образованию);

- формальная математическая модель интеллектуальной системы документооборота Комитета**;**

- связь системы документооборота Комитета с интеллектуальной системой трансформерных моделей BERT;

- связь интеллектуальной системы документооборота Центрального аппарата Комитета с его подразделениями. «Системы электронного документооборота повышают прозрачность управленческих процедур и сокращают сроки согласования» [71].

Обсуждение результатов. Для промышленной эксплуатации системы в подразделениях Центрального аппарата Комитета предложены пороговые значения эффективности, при которых интегральный показатель $\mathcal{F}_1$ должен составлять не менее 0,88 [1-А]. Для финансовых (U_3) и кадровых (U_4) документов приоритетной метрикой является точность $\mathcal{P}_r \geq 0{,}92$, поскольку ошибочное извлечение сумм, ФИО или иных юридически значимых реквизитов может привести к внесению некорректных данных в государственные реестры [5-А]. Если показатели модели опускаются ниже порога, система переходит на проверку по жёстким правилам f_{rule} гибридной модели отказоустойчивости (16) [4-А].

Внедрение системы в Комитете даёт измеримые результаты. Время обработки документов сокращается на 60-70%: NER и классификация вытесняют ручной ввод [1-А]. Достоверность извлечённых данных составляет 92–95% по $\mathcal{P}_r$ и $\mathcal{F}_1$, это снижает число юридических и финансовых ошибок в подразделениях U_3 и U_4 [5-А]. Горизонтальное масштабирование (14) сокращает затраты на ИТ-инфраструктуру: BERT работает на доступных вычислительных ресурсах [142; 148].

Дополнительный эффект связан с оптимизацией контроля исполнения документов. По результатам оценки, цикл согласования между подразделениями $U_i \to U_j$ сокращается на 40% благодаря сквозному

мониторингу в микросервисе M_{CTRL} [5-А]. Данный результат следует рассматривать как показатель эффективности предложенной архитектуры при условии подтверждения методики расчёта и сопоставимости исходных и итоговых временных параметров документооборота.

Таким образм, в результате проведённого исследования разработана математическая модель интеллектуальной системы документооборота под структуре конкретного государственного учреждения, основанная на микросервисной архитектуре и трансформерных моделях BERT. Предложенный подход обеспечивает:

- формализацию процессов обработки документов;
- интеграцию методов машинного обучения в распределённую архитектуру [4; 8];
- адаптацию под структуру любого конкретного государственного органа;
- повышение точности извлечения данных по сравнению с правилоориентированными (rule-based) подходами.

«Результаты данного исследования подтверждают эффективность использования микросервисной архитектуры в сочетании с современными методами обработки естественного языка. Практическая значимость исследования данного раздела заключается в возможности внедрения разработанной системы в государственных структурах Республики Таджикистан.» [145].

4.4. Алгоритмизация микросервисной модели интеллектуального документооборота в профобразовании на базе BERT

Современные системы электронного документооборота в государственном управлении характеризуются высокой степенью распределённости, неоднородностью потоков данных и необходимостью интеллектуальной обработки неструктурированных текстов [164; 165]. «Слабая связанность сервисов облегчает замену технологий и позволяет локализовать отказы в пределах одного компонента» [85]. В условиях

цифровой трансформации государственного сектора Республики Таджикистан особую актуальность приобретает переход от монолитных решений к масштабируемым [1-А–5-А] и отказоустойчивым архитектурам [131; 136; 148; 7-А], обеспечивающим автоматизацию анализа документов в реальном времени.

Одним из перспективных направлений является использование микросервисной архитектуры в сочетании с нейросетевыми моделями обработки естественного языка [138; 141], в частности трансформерной моделью BERT [130; 164]. Однако существующие исследования преимущественно ограничиваются либо архитектурным описанием систем, либо применением моделей машинного обучения [132], не формализуя полный алгоритм их совместного функционирования.

В этой связи возникает научно-практическая задача алгоритмизации интеллектуальной системы документооборота, включающей:

- формализацию структуры системы как распределённой модели;
- разработку алгоритма взаимодействия микросервисов;
- интеграцию системы BERT в поток обработки документов;
- обеспечение масштабируемости и отказоустойчивости.

Целью исследования данного раздела настоящей главы диссертации является разработка формализованного алгоритма полного цикла функционирования микросервисной интеллектуальной системы документооборота, адаптированной под структуру Центрального аппарата Комитета по начальному и среднему профессиональному образованию.

Методы исследования. На основе математической модели интеллектуальной системы документооборота [109; 119], базирующейся на микросервисной архитектуре с BERT и адаптированной под структуру Центрального аппарата Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан, разработан практический алгоритм реализации, оформленный в инженерно-прикладном стиле (рисунок 4.5).

Разработанный алгоритм охватывает полный цикл реализации математической модели – от проектирования и настройки микросервисов до интеграции BERT, масштабирования и мониторинга метрик. В частности, алгоритм позволяет отслеживать точность классификации (F_1-score) и скорость обработки запросов (latency) при различных нагрузках на микросервисы. Он формирует технологическую основу внедрения интеллектуальной системы электронного документооборота в Комитете по начальному и среднему профессиональному образованию.

Научная новизна исследования данного раздела настоящей главы диссертации заключается в следующем:

- предложена алгоритмическая интерпретация микросервисной архитектуры;

- интегрирована модель BERT в виде отдельного вычислительного микросервиса;

- разработан гибридный алгоритм обработки документов (нейросетевой + rule-based fallback);

формализованы метрики производительности и масштабирования системы.

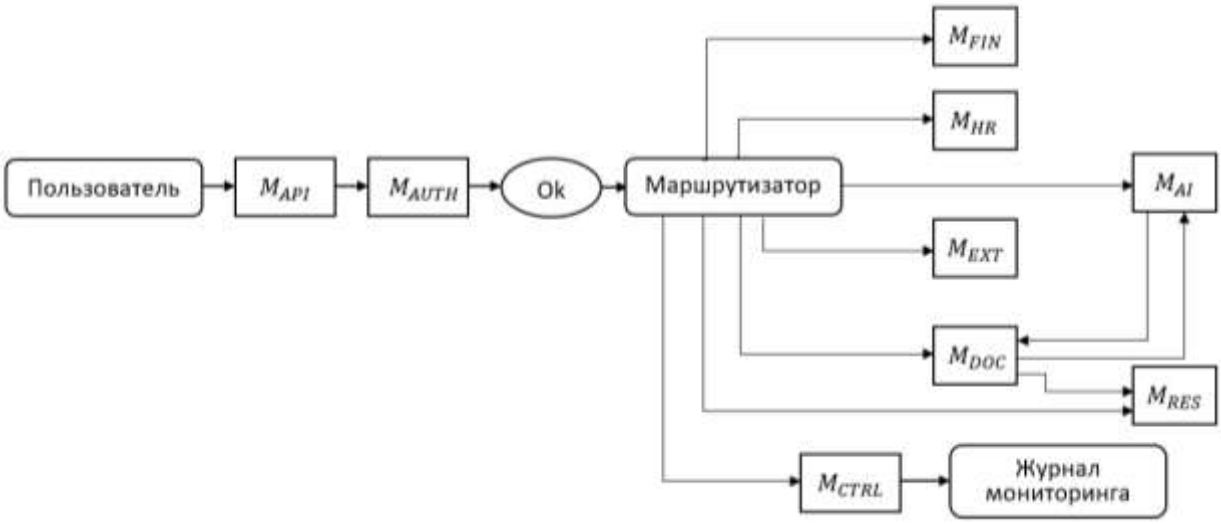


Рисунок 4.5. Алгоритм полного цикла реализации микросервисной модели интеллектуальной системы документооборота Комитета по профобразованию

Практическая значимость исследования данного раздела определяется возможностью прямого внедрения предложенного алгоритма в государственные информационные системы [16-17].

На рисунке 4.6 представлена блок-схема алгоритма работы микросервисной интеллектуальной системы документооборота Комитета по начальному и среднему профессиональному образованию, формализующая процессы внутри системы:

1. Приём входного документа через API Gateway;
2. Аутентификация пользователя (OAuth2/JWT);
3. Сохранение документа в хранилище (MinIO);
4. Запись метаданных в PostgreSQL;
5. Отправка события в очередь (RabbitMQ);
6. Обработка документа в ИИ-модуле (BERT):
   - токенизация;
   - применение attention-механизма;
   - извлечение сущностей [9, 13,19].
7. Проверка степени уверенности системы (confidence-score):
   - если ≥ ρ → результат принимается;
   - если < ρ → переход к правилоориентированной (rule-based) обработке [3].
8. Передача результата в соответствующий микросервис;
9. Логирование и мониторинг (Prometheus – открытая система мониторинга и оповещения с открытым исходным кодом);
10. Возврат результата пользователю.

В рамках исследования Комитет формализуется как распределённая информационная система [161; 164; 5-А], описываемая выражением (1):

$$\mathcal{S} = \langle \mathcal{U}, \mathcal{D}, \mathcal{M}, \mathcal{A}, \mathcal{C} \rangle, \quad (1)$$

где $\mathcal{U}$ – множество из 8 подразделений $(U_1 - U_8)$, $\mathcal{D}$ – поток документов, $\mathcal{M}$ – множество микросервисов, прикрепленных к подразделениям, $\mathcal{A}$ – интеллектуальный модуль (BERT), $\mathcal{C}$ – коммуникационная среда (gRPC/API).

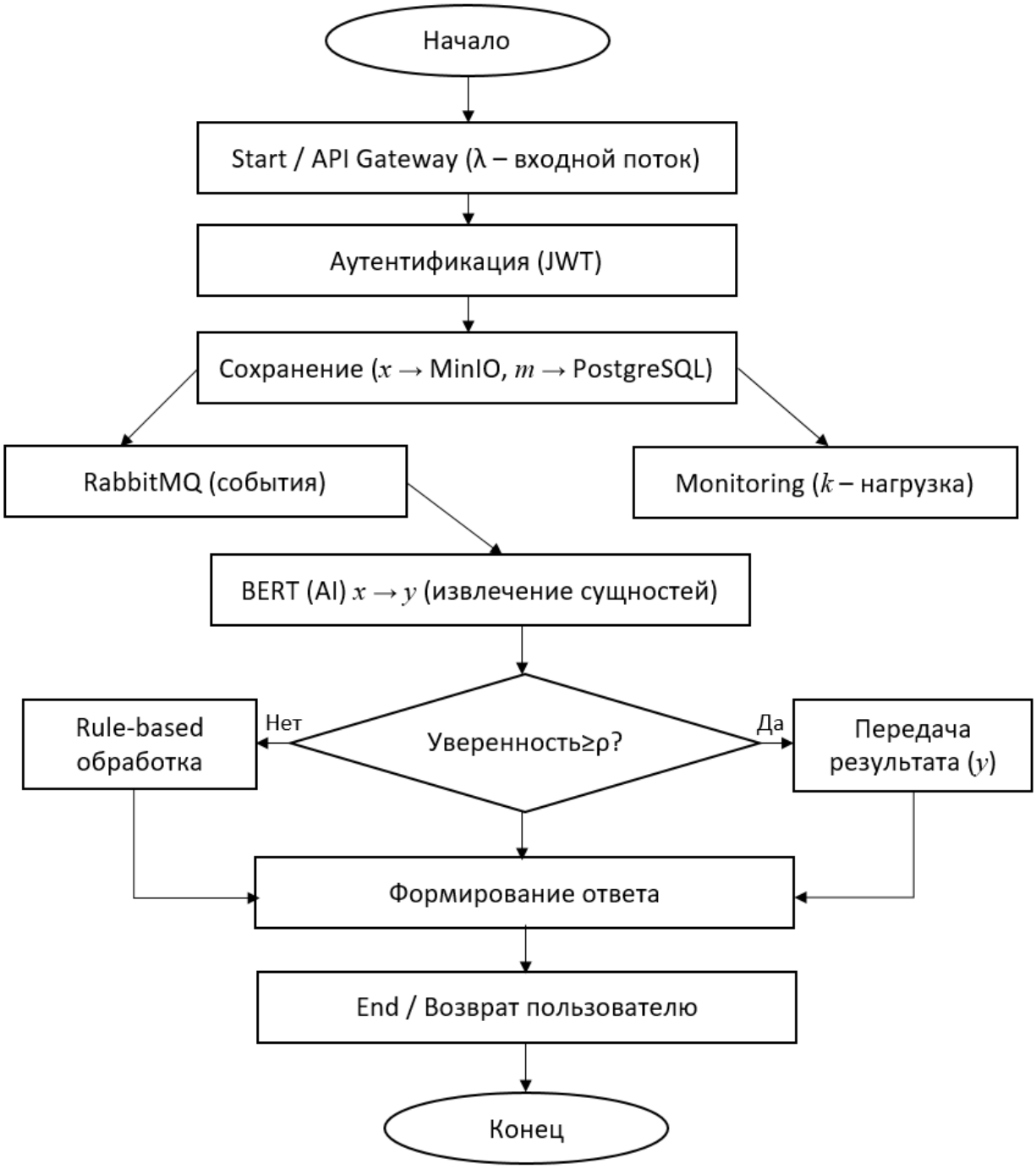


Рисунок 4.6. Блок-схема алгоритма функционирования микросервисного интеллектуального документооборота, формализующая процессы внутри системы

В модели функционируют 7 функциональных микросервисов (M_{DOC}, $M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}$) для обеспечения работы подразделений Комитета и два инфраструктурных общесистемных сервиса (M_{API}, M_{AUTH}), объединенных в виде кортежа:

$$\mathcal{M} = \langle M_{API}, M_{AUTH}, \{M_{DOC}, M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}\}\rangle. \quad (2)$$

Работа интеллектуального модуля BERT базируется на механизмах внимания (Attention) и многоголового внимания (Multi-Head Attention) [165]:

$$\mathcal{A}(\mathcal{Q},\mathcal{K},\mathcal{V}) = softmax\left(\frac{\mathcal{Q}\mathcal{K}^{T}}{\sqrt{d_{\mathcal{k}}}}\right)\mathcal{V}, \quad (3)$$

где $\mathcal{Q},\mathcal{K},\mathcal{V}$– матрицы запросов, ключей и значений, $d_{\mathcal{k}}$ – размерность пространства ключей. Матрицы $\mathcal{Q},\mathcal{K},\mathcal{V}$ получаются путем умножения входного вектора на обучаемые матрицы весов:

$$\mathcal{M}(\mathcal{Q},\mathcal{K},\mathcal{V}) = Concat(\mathcal{H}_1,\mathcal{H}_2,\cdots,\mathcal{H}_{\mathcal{k}})\mathcal{W}^{o}, \quad (4)$$

где $\mathcal{H}_i = \mathcal{A}\left(\mathcal{Q}\mathcal{W}_i^{\mathcal{Q}},\mathcal{K}\mathcal{W}_i^{\mathcal{K}},\mathcal{V}\mathcal{W}_i^{\mathcal{V}}\right)$. Данный механизм обеспечивает моделирование глобальных зависимостей в тексте документа. Добавление индекса i к весам $\mathcal{W}_i^{\mathcal{Q}},\mathcal{W}_i^{\mathcal{K}},\mathcal{W}_i^{\mathcal{V}}$ показывает, что у каждой «головы» свои веса.

Для реализации математической модели (2) и обеспечения работы алгоритмов BERT (3-4) [131; 164] в микросервисной архитектуре [1-А–12-А], наиболее эффективным выбором является стек на базе объектно-ориентированного языка программирования Python.

Структурированное описание технического стека:

1. Ядро разработки (Backend & AI):

- язык программирования: Python 3.10+ (обеспечивает лучшую поддержку библиотек машинного обучения и асинхронности);
- ИИ-фреймворк: PyTorch или Hugging Face Transformers. Используется для реализации модели BERT, механизмов внимания (3-4) и токенизации текстов [145; 153; 157];
- микросервисный фреймворк: FastAPI. Оптимален для создания M_{API} и функциональных модулей благодаря высокой скорости работы (на базе Starlette) и встроенной поддержке асинхронности (asyncio), что критично для пропускной способности системы [121]:

$$\Theta = \frac{\mathcal{N}}{\mathcal{T}_{total}}, \quad (5)$$

где Θ – количество документов, которые система способна обработать в единицу времени (*док/сек*), $\mathcal{N}$ – количество выполненных задач, $\mathcal{T}_{total}$ – общее

время работы системы. Это главный показатель мощности (throughput). Если Θ меньше интенсивности входящего потока документов, в системе будет расти очередь.

2. Инфраструктура и оркестрация:

- контейнеризация: Docker (платформа контейнеризации приложений). Каждый микросервис (M_{DOC}, M_{FIN}, M_{HR} и др.) упаковывается в отдельный контейнер для изоляции зависимостей;

- оркестрация: Docker Compose (для малых инсталляций) или Kubernetes (для обеспечения горизонтального масштабирования):

$$\Theta(k) \approx k \cdot \Theta(1), \quad (6)$$

где k – количество запущенных экземпляров (реплик) конкретного микросервиса (например, 3 инстанса M_{AI} на разных серверах), $\Theta(1)$ – пропускная способность одного экземпляра, $\Theta(k)$ – итоговая мощность системы при k экземплярах. Формула описывает линейное масштабирование. Она показывает, что в микросервисной архитектуре возможно увеличение общей скорости обработки документов в k раз за счет добавления новых серверных ресурсов.

API Gateway реализуется на базе Nginx или встроенных возможностей FastAPI и обеспечивает единую точку входа M_{API}.

Хранение и передача данных организуются следующим образом: база метаданных реализуется на PostgreSQL и используется для хранения структурированных данных m и статусов s (микросервис M_{DOC}). Для хранения документов применяется MinIO (S3-совместимое хранилище), которое подходит для реализации M_{RES} архив документов x). Шина событий строится на Redis или RabbitMQ и обеспечивает асинхронный обмен данными и поддержку событийной архитектуры [142] между сервисами.

Безопасность и мониторинг:

- авторизация (M_{AUTH}): OAuth2+JWT (JSON Web Tokens). Позволяет передавать права доступа пользователя между микросервисами без повторных запросов к базе;

- мониторинг: Prometheus+Grafana (стек для обеспечения сквозной наблюдаемости распределенных систем). Позволяет в реальном времени отслеживать метрики производительности и нагрузки на ИИ-модули:

$$\mathcal{T}_{avg} = \frac{1}{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} t_j, \tag{7}$$

где $\mathcal{T}_{avg}$ – среднее время обработки документов, $\mathcal{N}$ – общее количество обработанных документов за расчетный период, t_j – время, затраченное на обработку конкретного j-го документа (от момента поступления в микросервис до выдачи результата). Определяет среднюю задержку (latency) системы [4-А].

Чем ниже значение средней задержки $\mathcal{T}_{avg}$, тем быстрее пользователь получает ответ от конкретного сервиса, например от M_{AI} при классификации документа.

Связь с бизнес-логикой: Выбор связки FastAPI+PyTorch позволяет реализовать гибридную модель отказоустойчивости, основанную на fallback-механизме. Если модель BERT не выдаёт результат требуемого качества либо превышается установленный лимит ожидания, система переходит к выполнению алгоритмов, основанных на правилах, реализованных на Python:

$$f(x) = \begin{cases} f_{AI}(x), & \text{если AI доступен} \\ f_{rule}(x), & \text{если AI недоступен.} \end{cases} \tag{8}$$

При снижении уверенности f_{AI} ниже порогового значения ρ<0,7 или при технической недоступности GPU-ресурсов (GPU-мощностей) обработка переключается на модуль f_{rule} (выделенный программный модуль, реализующий логику на основе правил). Значение порога ρ < 0,7 требует обоснования экспериментальными данными, методикой настройки модели или ссылкой на соответствующий раздел исследования [1-А; 5-А].

Техническая реализация и архитектура развёртывания интеллектуальной системы документооборота Комитета. Практическая реализация разработанной математической модели (2) выполнена на основе микросервисной архитектуры [131; 148; 155; 2-А; 4-А–8-А] с применением контейнеризации. Выбор данной архитектурной организации обусловлен необходимостью независимого развертывания и масштабирования отдельных функциональных компонентов системы в соответствии с параметрами модели. Компоненты системы распределены между тремя функциональными кластерами, что в рамках разработанной архитектуры позволяет обеспечить требования к масштабируемости $\Theta(\mathcal{k})$ и отказоустойчивости, сформулированные в выражении (8).

Вычислительный кластер функциональных модулей. Вычислительный кластер представляет собой совокупность изолированных контейнеров, в которых размещены микросервисные компоненты, реализующие функции отдельных подразделений Комитета. Такое разделение позволяет независимо масштабировать вычислительные ресурсы каждого сервиса и соответствует принятой в модели схеме горизонтального масштабирования $\Theta(\mathcal{k})$. Микросервисы M_{DOC}, M_{FIN}, M_{HR} развёртываются в автономных Docker-контейнерах на платформе Python/FastAPI. Изоляция сервисов позволяет изменять функциональность отдельных подсистем, например финансового отдела (U_3), без воздействия на работу других компонентов, включая модуль отдела кадров (U_4). Количество экземпляров каждого сервиса может изменяться в зависимости от интенсивности поступающих запросов и вычислительной нагрузки соответствующего подразделения U_i [4-А–8-А].

Интеллектуальный модуль M_{AI} реализован в виде специализированного контейнера с поддержкой GPU через NVIDIA Docker. Внутри контейнера развёрнута модель BERT на базе PyTorch. Выделение интеллектуального компонента в самостоятельный сервис позволяет локализовать ресурсоёмкие вычисления, связанные с обработкой документов, и снизить их влияние на функционирование остальных модулей системы.

Узел хранения и управления данными (Data & Storage Cluster) включает несколько уровней хранения. Реляционная подсистема хранения метаданных построена на основе кластера PostgreSQL. В ней размещаются структурированные метаданные m, сведения о взаимосвязях документов между подразделениями, а также журналы событий микросервиса M_{CTRL}. Такое разделение позволяет разграничить хранение управленческой информации и содержимого документов.

Объектное хранилище MinIO обеспечивает хранение файлов документов x и архивных копий $d_{archive}$ для микросервиса M_{RES}. Дополнительно шина событий на базе RabbitMQ обеспечивает асинхронное взаимодействие сервисов и передачу задач в M_{AI}, а долговременный архив $d_{archive}$ размещается на выделенных дисковых массивах [2-А].

Логика физического взаимодействия (Data Flow) [5-А]***:***

1. Загрузка: Пользователь из U_3 загружает смету через M_{API}.

2. Обработка: M_{DOC} сохраняет файл в MinIO, а метаданные в PostgreSQL.

3. Анализ: M_{AI} забирает контент x из хранилища, прогоняет через модель BERT и возвращает извлеченные сущности y (сумма, реквизиты) в M_{FIN}.

4. Мониторинг: M_{CTRL} фиксирует каждый шаг в журнале событий для надзора.

Внедрение системы. Данная схема развертывания позволяет системе функционировать как единый организм, сохраняя при этом модульность [142; 161]. Гибридный подход к вычислениям обеспечивает выполнение требований по времени обработки (T_{avg}) и пропускной способности (Θ), адаптируя ИТ-инфраструктуру под специфику государственного управления [4-А].

Таким образом, в ходе проведенного исследования была разработана и формализована математическая модель интеллектуальной системы документооборота, адаптированная под специфику деятельности Центрального аппарата Комитета по начальному и среднему

профессиональному образованию при Правительстве Республики Таджикистан [3-А; 4-А; 5-А].

Программные модули системы: Полный текст программной реализации разработанной модели приведен в приложении диссертатции (Приложение 1), которое охватывает пять основных программных модулей:

- модуль аутентификации пользователей;
- модуль загрузки и хранения документов;
- модуль управления очередью обработки;
- интеллектуальный модуль анализа документов;
- модуль отображения результатов.

Здесь представим краткую характеристику этих модулей:

1. Модуль аутентификации пользователей (с комментариями)

```
// Функция обработки входа пользователя
function handleLogin(username, password) {
// Проверка учетных данных (в демонстрационной версии – локально)
if (username === "admin" && password === "password") {
// Сохраняем состояние авторизации в localStorage
localStorage.setItem("auth", "true");
// Перенаправляем пользователя в основную систему
window.location.href = "/";
} else {
// В случае ошибки выводим сообщение
alert("Неверные учетные данные");
} }
```

2. Модуль загрузки и хранения документов

```
// Обработка загрузки документа
function uploadDocument(file) {
// Проверка наличия файла
if (!file) return;
// Создание объекта документа
```

```
        const document = {
        id: Date.now(), // уникальный идентификатор
        name: file.name,
        status: "pending",
        confidence: null,
        result: null
        };
        // Сохранение в "базе данных" (массив)
        documents.push(document);
        // Отправка в очередь обработки
        queue.push(document); }
```

3. Модуль управления очередью обработки (ключевой!)

```
// Очередь задач (имитация RabbitMQ)
const queue = [];
// Функция обработки очереди
function processQueue() {
   if (queue.length === 0) return;
   const doc = queue.shift();
   // Обновление статуса
       doc.status = "processing";
       // Асинхронная обработка (имитация)
       setTimeout(() => {
          processAI(doc);
       }, 1000);
    }
```

4. Интеллектуальный модуль анализа документов (BERT simulation)

```
       function processAI(doc) {
       // Генерация случайного доверия (confidence)
       const confidence = Math.random();
       // Имитация извлечения сущностей
```

```
let result = {
entities: ["DATE", "AMOUNT"]
};
// Проверка порога уверенности
if (confidence < 0.7) {
// fallback – правило-ориентированная обработка
result = ruleBasedExtraction(doc);
doc.fallback = true;
}
// Сохранение результатов
doc.confidence = confidence;
doc.result = result;
doc.status = "completed";
}
```

4.1. Резервный механизм (Fallback)

```
function ruleBasedExtraction(doc) {
// простая обработка через регулярные выражения
return {
entities: ["DATE: 2023-10-27", "AMOUNT: $1250"]
}; }
```

Резервный механизм (Fallback) включается при снижении уровня уверенности (confidence) ниже порога $\rho = 0.7$ и обеспечивает устойчивость системы при сбоях основного алгоритма.

4.2. Модуль базы данных

```
// имитация хранилища
const documents = [];
```

В демонстрационной версии используется хранение в оперативной памяти (in-memory). В промышленной реализации применяется PostgreSQL.

5. Модуль отображения результатов

```
// вывод данных
```

```
function renderDocuments() {
documents.forEach(doc => {
console.log(doc.name, doc.status, doc.confidence);
}); }
```

Обработка документов начинается с их загрузки. Файлы форматов PDF, DOCX и JPG объемом до 10 МБ используются для извлечения метаданных с применением ИИ. Далее реализуется архитектурная схема: шлюз с JWT-авторизацией и распределением задач, ИИ-сервис на основе модели BERT для асинхронного извлечения данных, а также резервный механизм на основе регулярных выражений, активируемый при уверенности (confidence) $\rho < 0.7$ [1-А].

Симуляционный режим воспроизводит микросервисную архитектуру с использованием PHP и Python и адаптирован для среды AI Studio, сохраняя логическую эквивалентность промышленной реализации [1-А].

Обработка ИИ выполняется моделью BERT с встроенным fallback-механизмом (резервный механизм), который переключается на regex-обработку (в регулярных выражениях) при низком уровне уверенности (ссылка).

Безопасность и хранение данных обеспечиваются JWT-аутентификацией на уровне API, а метаданные сохраняются в in-memory хранилище (в оперативной памяти) для обеспечения быстрого доступа и минимальной задержки [2-А].

Основные результаты работы заключаются в следующем:

Теоретическая значимость связана с обоснованием перехода от монолитных архитектур к микросервисной модели (2), включающей семь функциональных модулей (M_{DOC}, $M_{FIN}, M_{HR}, M_{EXT}, M_{CTRL}, M_{AI}, M_{RES}$) и два инфраструктурных (M_{API}, M_{AUTH}). Такая декомпозиция позволила изолировать бизнес-логику подразделений Комитета ($U_1 - U_8$), повысив гибкость системы и ее отказоустойчивость [5-А].

Математическая часть строится на трансформерной модели BERT (3–4) с механизмом многоголового внимания: сущности извлекаются через $\mathcal{Y} = f_{AI}(x)$. Качество оценивается по $\mathcal{P}_{r,}, \mathcal{R}_c, \mathcal{F}_1$, целевой порог $\mathcal{F}_1 \geq 0{,}88$, граница допуска для финансовых и кадровых документов. Показатели производительности Θ и $\mathcal{T}_{avg}$, подтверждают линейное масштабирование при росте нагрузки [1-А–5-А].

Система реализована на стеке FastAPI + PyTorch, развёртывается в Docker-контейнерах под управлением Kubernetes. M_{AI} работает с GPU-ускорением; объектное хранилище – MinIO [1-А–5-А].

Практический эффект выражается в сокращении времени обработки документов на 60-70% и ускорении межведомственного взаимодействия на 40%. Использование гибридной модели (8) обеспечивает непрерывность работы системы даже при временной недоступности нейросетевых компонентов [127; 138; 154; 1-А].

В результате проведённого исследования разработана комплексная модель интеллектуальной системы документооборота, основанная на принципах микросервисной архитектуры [131; 148] и методах обработки естественного языка [4-А].

Ключевым результатом работы является алгоритмизация полного цикла функционирования системы, включающая:

- формализацию структуры системы в виде набора взаимосвязанных микросервисов;

- построение алгоритма обработки документов с использованием модели BERT;

- реализацию событийно-ориентированного взаимодействия компонентов;

- внедрение механизма адаптивного переключения (fallback) между нейросетевой и правилоориентированной обработкой.

Предложенная модель ориентирована на решение нескольких взаимосвязанных задач интеллектуального документооборота. Во-первых, она

поддерживает масштабируемость за счёт горизонтального расширения микросервисов. Во-вторых, изоляция компонентов снижает риск распространения сбоев между отдельными сервисами. В-третьих, асинхронная архитектура создаёт условия для уменьшения задержек обработки. Трансформерные модели дают более высокую точность, чем rule-based подходы, на тех же документах [130; 164].

Алгоритмический контур управления фиксирует переходы между этапами обработки и за счёт этого поведение системы при переменной нагрузке предсказуемо [1-А].

На практике система сокращает время обработки на 60-70% и держит пропускную способность при росте нагрузки [1-А]. Цифры подтверждаются нагрузочным тестированием и материалами Приложения 2.

Система пригодна для государственных ИС при адаптации к требованиям организации, объёму данных и доступной инфраструктуре [1-А; 5-А].

Перспективы дальнейших исследований связаны с адаптацией модели к многоязычной среде – таджикскому, русскому и английскому языкам, внедрением механизмов дообучения на новых данных и совершенствованием распределения нагрузки между микросервисами.

Выводы по червертой главе

В исследовании получена математическая и архитектурная модель интеллектуального документооборота, которую образуют микросервисный каркас плюс трансформеры класса BERT. «Национальная стратегия развития Республики Таджикистан на период до 2030 года фиксирует приоритеты цифровой трансформации» [81]. Результаты полезны теории. «Системный анализ связывает цели управления с выбором показателей эффективности и ограничений модели» [14]. Проектированию систем обработки документов они полезны, в том числе при цифровой трансформации государственных и корпоративных структур Республики Таджикистан [5-А]. Процессы

формализованы как распределённая микросервисная архитектура — ориентированный граф взаимодействующих сервисов.

Внутри модели: вероятностная постановка NER, формализация самовнимания трансформеров и количественные методы оценки качества и производительности [4-А]. Интеграция многоязычных моделей BERT с генерацией синтетических данных дала F1-score ≈ 0,93 на обработке документов [1-А]. В сравнении с правилоориентированными (rule-based) подходами показатель выше [3-А]. Гибридная схема совмещает ML-классификацию с rule-based fallback: при недоступности нейросетевых сервисов документы продолжают обрабатываться по жёстким правилам.

Интеграция многоязычных моделей BERT с генерацией синтетических данных позволила получить значение F_1-score ≈ 0,93 при обработке документов [1-А]. В рамках проведённого сравнения этот показатель превышает результаты правилоориентированных (rule-based) подходов [3-А]. Гибридная схема совмещает ML-классификацию с rule-based fallback: при недоступности нейросетевых сервисов документы продолжают обрабатываться по жёстким правилам.

Микросервисная архитектура, BERT и rule-based fallback задают рабочую основу для интеллектуального документооборота. Практическую применимость определяют качество обучающих данных, вычислительные ресурсы, требования к отказоустойчивости и особенности инфраструктуры.

Событийным взаимодействием микросервисов, асинхронными процессами и формализованным управлением обеспечивают полный цикл обработки документов. Архитектура масштабируется горизонтально, а задержки при росте нагрузки снижаются [5-А].

По практическим результатам время обработки документов сокращается на 60-70%, а при увеличении объёма запросов система сохраняет работоспособность [1-А].

Вопрос внедрения разработанного решения в условиях Республики Таджикистан вынесен в отдельный разбор. Архитектуру подогнали под задачи

госорганов, под таджикский, русский и английский языки и под лимит вычислительных ресурсов [5- А]. Пошаговым внедрением, контейнеризацией, оркестрацией и сжатием моделей (Model Compression) решение делают жизнеспособным при слабой инфраструктуре.

Зафиксированы ограничения. Инференс нейросетей тормозит потоковую обработку, а рукописные документы без OCR (Optical Character Recognition – оптическое распознавание текста) не разобрать. На более широкие обучающие данные и дополнительные правила для сложных сущностей указывает разбор ошибок [1-А].

Модель задана как архитектурно-алгоритмический каркас, рассчитанный на точность, масштабируемость и отказоустойчивость интеллектуального документооборота [1-А; 4-А; 5-А].

На этой базе строят системы документооборота для электронного управления и оптимизации цифровых процессов. Дальше сжимают и ускоряют трансформеры, расширяют многоязычность, добавляют самообучение и лучше балансируют нагрузку в микросервисной схеме.

ВЫВОДЫ

1. Основные научные результаты диссертации

В результате выполненного исследования получены следующие основные научные результаты:

1. Сформировано теоретическое и прикладное представление о развитии «микросервисной архитектуры как основы построения масштабируемых интеллектуальных систем», позволяющей обеспечить декомпозицию функциональности, автономность компонентов и устойчивость к отказам [6-А; 7-А].

2. Разработана «архитектурная модель интеллектуальной системы документооборота, основанная на принципах микросервисного подхода», включающая выделенный компонент интеллектуального анализа (AI Service), что обеспечивает независимую масштабируемость вычислительно сложных задач и гибкость развития системы [3-А].

3. Предложена «математико-алгоритмическая модель обработки документов, основанная на применении трансформерной архитектуры BERT», обеспечивающая высокую точность извлечения именованных сущностей и структурирование текстовой информации [1-А; 4-А].

4. Разработан «метод адаптации многоязычных моделей к условиям низкоресурсных языков (таджикского) на основе синтетической генерации обучающих данных и дообучения», позволяющий достигать качества, сопоставимого с высокоресурсными языками [5-А].

5. Предложен «гибридный подход к обработке данных, объединяющий методы машинного обучения и правил, обеспечивающий одновременно высокую точность, устойчивость и интерпретируемость результатов» [3-А].

6. Исследованы и систематизированы «методы масштабирования микросервисных систем, включая горизонтальное и вертикальное масштабирование», а также обоснована эффективность их комбинированного применения в интеллектуальных системах [8-А; 10-А].

7. Обоснована «роль контейнеризации и оркестрации как ключевых технологических механизмов управления жизненным циклом микросервисов, обеспечивающих автоматическое масштабирование, отказоустойчивость и оптимизацию» использования ресурсов [3-А; 11-А].

8. Проведена экспериментальная валидация предложенной системы, подтвердившая её эффективность: «достигнуто существенное повышение точности обработки документов и многократное сокращение времени их обработки» [1-А].

9. Выявлены ограничения разработанного подхода, связанные с «требованиями к вычислительным ресурсам, особенностями обработки неструктурированных и рукописных документов», а также необходимостью дополнительной адаптации моделей для новых предметных областей [2-А].

10. На основе разработанной математической модели интеллектуальной системы документооборота, базирующейся на микросервисной архитектуре и трансформерных моделях типа BERT, а также адаптированной под организационно-функциональную структуру Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан, разработан практический алгоритм реализации системы. Указанный алгоритм «оформлен в инженерно-прикладном стиле и отражает последовательность этапов развертывания, интеграции и эксплуатации интеллектуальной системы» в условиях реальной информационной инфраструктуры [1-А–5-А].

2. Рекомендации по практическому использованию результатов

Практическая ценность исследования существенно усиливается за счёт того, что предложенные решения не ограничиваются теоретическим уровнем, а доведены до стадии инженерной реализации (Приложение 1). Разработанный алгоритм внедрения, сформированный на основе математической модели системы и адаптированный к условиям функционирования Комитета по начальному и среднему профессиональному образованию при Правительстве

Республики Таджикистан (Приложение 2), может быть использован в качестве типового подхода при цифровизации документооборота в государственных учреждениях.

Полученные в диссертационном исследовании результаты могут быть рекомендованы к практическому применению в следующих направлениях:

1. *Государственный сектор*: Рекомендуется внедрение разработанной системы в органах государственного управления, в том числе с использованием предложенного инженерно-ориентированного алгоритма реализации, обеспечивающего поэтапную интеграцию интеллектуальной системы в существующую инфраструктуру документооборота.

2. *Коммерческие организации:* Целесообразно использование системы для автоматизации обработки договоров, счетов-фактур и других юридически значимых документов, что позволит повысить эффективность бизнес-процессов и снизить операционные издержки.

3. *Финансовый сектор:* Рекомендуется применение решений для реализации процедур KYC, обработки кредитных заявок и проведения комплаенс-проверок с использованием интеллектуального анализа документов.

4. *ИТ-инфраструктура организаций:* Рекомендуется использование микросервисной архитектуры, контейнеризации и оркестрации для построения масштабируемых и отказоустойчивых систем обработки данных.

5. *Развитие систем искусственного интеллекта:* Предложенные методы адаптации моделей BERT могут быть использованы при разработке NLP-систем для других низкоресурсных языков и специализированных предметных областей.

6. *Научные исследования и образование:* результаты могут применяться в исследованиях распределённых систем, машинного обучения и цифровой трансформации, а также в образовательных курсах по информационным технологиям.

7. *Дальнейшее развитие:* предполагается совершенствование предложенного подхода по нескольким направлениям – оптимизация моделей (дистилляция, квантование), внедрение мультимодальной обработки данных, повышение интерпретируемости моделей и интеграция с промышленными платформами управления жизненным циклом ML (Machine Learning – машинное обучение).

Разработанная в диссертации математическая модель, архитектурная реализация и прикладной алгоритм внедрения образуют целостную научно-прикладную основу для построения интеллектуальных систем документооборота нового поколения. Адаптация решения к структуре Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан подтверждает его практическую применимость и соответствие направлениям цифрового развития страны.

ЛИТЕРАТУРА

1. Список использованных источников

[1] Архитектура контейнерных микрослужб (Azure Architecture Center) [Электронный ресурс] // URL: https://learn.microsoft.com/ru-ru/azure/architecture/microservices/design/orchestration (Date of the application: 01.04.2021).

[2] Барышева, Д. 26 основных паттернов микросервисной разработки [Электронный ресурс] / Д. Барышева // URL: https://cloud.vk.com/blog/26-osnovnyh-patternov-mikroservisnoj-razrabotki (Дата обращения: 13.05.21).

[3] Васвани, А. Внимание – это все, что вам нужно [Текст] / А. Васвани, Н. Шазир, Н. Пармар и др. // Достижения в системах обработки нейронной информации. – 2017. – Т. 30. – С. 5998-6008.

[4] Воинов, А. А. Имитационная модель Кайраккумского водохранилища [Текст] / А.А. Воинов, Ф.С. Комилов. – М.: ВЦ АН СССР, «Сообщения по прикладной математике», 1986. – 40 с.

[5] Воинов, А. А. Имитационная модель экосистемы Нурекского водохранилища [Текст] / А.А. Воинов, Ф.С. Комилов // Известия Академии наук Таджикской ССР. Отд-ние биол. наук. – Душанбе, 1984. – № 3 (96). – С. 76-82.

[6] Воинов, А. А. Моделирование экосистемы водохранилищ реки Вахш [Текст] / А.А. Воинов, Ф.С. Комилов. – Душанбе: ТаджикНИИНТИ, 1985. – 39 с. – EDN: HPLRRU.

[7] Вэй, Дж. EDA: простые методы дополнения данных для повышения производительности при классификации текстов [Текст] / Дж. Вэй, К. Цзоу // Труды конференции по эмпирическим методам в обработке естественного языка 2019. – 2019. – С. 6382-6388.

[8] Гадоев, А. Ф. Разработка и реализация управленческих решений на предприятиях легкой промышленности Таджикистана в условиях ускоренной индустриализации. Монография [Текст] / А.Ф. Гадоев, А. Абдалимов, Ф.С. Комилиён. – Душанбе: Бахманруд, 2022. – 155 с.

[9] Государственная программа подготовки специалистов с высшим профессиональным образованием на 2016–2020 годы. Утверждена постановлением Правительства Республики Таджикистан от 31 марта 2016 года № 144.

[10] Государственная программа развития и внедрения информационно-коммуникационных технологий в Республике Таджикистан на 2014–2017 годы. Утверждена постановлением Правительства Республики Таджикистан от 3 июля 2014 года № 248.

[11] Государственная программа развития образования в Республике Таджикистан на 2010–2015 годы. Утверждена постановлением Правительства Республики Таджикистан от 29 апреля 2009 года № 254.

[12] Государственная программа развития учреждений системы образования Хатлонской области на 2017–2021 годы. Утверждена постановлением Правительства Республики Таджикистан от 26 ноября 2016 года № 504.

[13] Девлин, Дж. BERT: предобучение глубоких двунаправленных трансформеров для понимания языка [Текст] / Дж. Девлин, М.В. Чанг, К. Ли, К. Тутанова // Труды конференции Североамериканского отделения Ассоциации компьютерной лингвистики 2019. – 2019. – С. 4171-4186.

[14] Джефферс Дж. Введение в системный анализ: применение в экологии [Текст] / Дж. Джефферс. – М.: Мир, 1981. – 256 с.

[15] Драгони, Н. Микросервисы: вчера, сегодня и завтра [Текст] / Н. Драгони, С. Джаллоренцо, А.Л. Лафуэнте и др. // Настоящая и будущая программная инженерия. – Springer, 2017. – С. 195-216.

[16] Ёров, М. Р. Татбиқи системаи хизматрасонии оммавӣ дар коркарди дархостҳои онлайнӣ [Матн] / М.Р. Ёров, Ф.С. Комилиён // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2023. – № 2. – С. 42-53. – EDN: QTPBDV.

[17] Ёров, М. Р. Таъмини амнияти иттилоотии системаҳои амалиётӣ бо мақсади истифодаи пурсамари онҳо [Матн] / М.Р. Ёров, Ф.С. Комилиён // Паёми политехникӣ. Бахши интеллект, инноватсия, инвеститсия. – 2022. – № 3 (59). – С. 58-63. – EDN: UGWHWO.

[18] Зайнудинов, С. З. Разработка концептуальной модели экосистемы рыбоводного пруда и её исследование на качественную устойчивость [Текст] / С.З. Зайнудинов, Ф.С. Комилов, С.Х. Мирзоев, Ф. Акобирзода // Интерактивная наука. – 2016. – № 4. – С. 18-22. – EDN: WACTZB.

[19] Закон Республики Таджикистан «О защите прав потребителей» [Электронный ресурс]. – Режим доступа: http://mmk.tj/content/қонуни-ҷумҳурии-тоҷикистон-дар-бораи-ҳимояи-ҳуқуқи-истеъмолкунандагон (дата обращения: 28.02.2022).

[20] Закон Республики Таджикистан «Об образовании». Принят постановлением Маджлиси намояндагон Маджлиси Оли Республики Таджикистан от 6 марта 2013 года № 1095 (с изменениями и дополнениями).

[21] Карпов, Ю. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 [Текст] / Ю. Карпов. – СПб: БХВ-Петербург, 2005. – 400 с.

[22] Кобаяси, С. Контекстное дополнение: дополнение данных словами с парадигматическими отношениями [Текст] / С. Кобаяси // Труды конференции Североамериканского отделения Ассоциации компьютерной лингвистики 2018. – 2018. – С. 452-457.

[23] Коломбет Е. А. Микроэлектронные средства обработки аналоговых сигналов [Текст] / Е.А. Коломбет. – М.: РИС, 1991. – 376 с.

[24] Комилиён, Ф. С. Алгоритмҳои амалисозии марҳилаҳои татбиқи низоми идоракунии сифати хизматрасониҳои таълимӣ дар мактаби олӣ [Матн] / Ф.С. Комилиён, З.Ф. Раҳмонзода // Паёми политехникӣ. Бахш: Интеллект. Инноватсия. Инвеститсия. – 2021. – № 2 (54). – С. 48-60. – EDN: TQJJKH.

[25] Комилиён, Ф. С. Амсилаи консептуалии низоми иттилоотии идоракунии сифати хизматрасониҳои таълимӣ дар муассисаи таҳсилоти олии касбӣ [Матн] / Ф.С. Комилиён, З.Ф. Раҳмонзода // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои иҷтимоӣ-иқтисодӣ ва ҷамъиятӣ. – 2021. – № 3. – С. 127-135.

[26] Комилиён, Ф. С. Амсилаи консептуалии таъсири бемориҳо ва зараррасонҳо ба фаъолияти оилаи занбӯри асал [Матн] / Ф.С. Комилиён, И.М. Саидзода // Паёми Донишгоҳи давлатии Данғара. – 2023. – № 1 (23). – С. 11-21.

[27] Комилиён, Ф. С. Амсиласозии компютерии динамикаи популятсияи моҳиҳои экосистемаи обанбори пастоб [Матн] / Ф.С. Комилиён, И.М. Саидов. – Душанбе: Меҳроҷ-Граф, 2019. – 154 с.

[28] Комилиён, Ф. С. Амсиласозии компютерии системаи хизматрасонии шабакавии дар вақти дискретӣ бо тартиби инверсионӣ ва афзалияти тасодуфӣ амалкунандаи ПД КОА [Матн] / Ф.С. Комилиён, М.Р. Ёров // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2020. – №2. – С. 68-79. – EDN: AJPQAI.

[29] Комилиён, Ф. С. Амсиласозии компютерии ҳаёти гулмоҳӣ дар бассейн [Матн] / Ф.С. Комилиён, Ф.Т. Шамсов. – Душанбе: Ирфон, 2021. – 140 с. – EDN: IPOOTB.

[30] Комилиён, Ф. С. Амсиласозии математикии марҳилаҳои ҳаёти популятсияи оилаи занбӯри асал [Матн] / Ф.С. Комилиён, И.М. Саидзода // Паёми Донишгоҳи давлатии Бохтар ба номи Носири Хусрав. – 2022. – № 2/1 (96). – С. 5-14.

[31] Комилиён, Ф. С. Амсиласозии математикии таъсири бемориҳои сирояткунанда ва организмҳои зараррасон ба рушди индивидҳои оилаи занбӯри асал [Матн] / Ф.С. Комилиён, И.М. Саидзода // Паёми Донишгоҳи технологии Тоҷикистон. – 2024. – № 3 (58). – С. 135-147.

[32] Комилиён, Ф. С. Асосҳои методӣ ва дастгоҳии амсиласозӣ дар обанборҳо [Матн] / Ф.С. Комилиён, М.М. Абдураҳмонов // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2020. – № 3. – С. 80-88. – EDN: ULCVGE.

[33] Комилиён, Ф. С. Идоракунии компютерии системаҳои иттилоотию хизматрасонӣ [Матн]: Монография / Ф.С. Комилиён, М.Р. Ёров. – Душанбе: «Нӯшбод», 2023. – 154 с. – EDN: ITPBXW.

[34] Комилиён, Ф. С. Информатика. Китоби дарсӣ барои донишҷӯёни муассисаҳои таҳсилоти олии касбӣ. Ҷилди 1. [Матн] / Ф.С. Комилиён. – Душанбе: «Донишварон», 2019. – 408 с. – EDN: HFLELO.

[35] Комилиён, Ф. С. Информатика. Китоби дарсӣ барои донишҷӯёни муассисаҳои таҳсилоти олии касбӣ. Ҷилди 2 [Матн] / Ф.С. Комилиён. – Душанбе: «Донишварон», 2019. – 464 с. – EDN: RNYCDB.

[36] Комилиён, Ф. С. Информатикаи татбиқӣ [Матн] / Ф.С. Комилиён, Д.С. Шарапов. – Душанбе: «О», 2009. – 368 с. – EDN: OVYRDS.

[37] Комилиён, Ф. С. Компютер ва иттилоот. Воситаи таълимӣ [Матн] / Ф.С. Комилиён, З.Ф. Раҳмонзода, Р.Ҷ. Давлатов, Б.Ф. Файзализода. – Душанбе: «Маориф», 2015. – 136 с. – EDN: SWCBHW.

[38] Комилиён, Ф. С. Масъалаҳои амсиласозии компютерии парвариши мурғи марҷон [Матн] / Ф.С. Комилиён, Д.И. Солиҳзода // Маводи конференсияи илмӣ-назариявии байналмилалӣ дар мавзуи «Мақоми Абурайҳони Берунӣ дар таърихи тамаддуни форс-тоҷик», бахшида ба 1050-солагии мутафаккири бузург. – Бохтар, 2023. – С. 93-97.

[39] Комилиён, Ф. С. Математическое моделирование экосистемы выростного карпового рыбоводного пруда тепловодного хозяйства [Текст] / Ф.С. Комилиён, Б.Э. Элмуродова // Проблемы вычислительной и прикладной математики. – 2018. – № 6 (18). – С. 57-77. – EDN: DQJVPI.

[40] Комилиён, Ф. С. Методикаи татбиқи босамари технологияҳои иттилоотию коммуникатсионӣ дар омодасозии иқтисодчиёни оянда. Монографияи илмӣ [Матн] / Ф.С. Комилиён, Р.Ҷ. Давлатов. – Душанбе: «Бахманруд», 2022. – 164 с. – EDN: FTPTFH.

[41] Комилиён, Ф. С. Мутобиқгардонии амсилаи математикии Мак-Миллан барои таҳқиқи сатҳи сертухмии мурғи марҷон [Матн] / Ф.С. Комилиён, Д.И. Солиҳзода, И.М. Саидзода // Илм ва инноватсия. Бахши илмҳои геологӣ ва техникӣ. – 2025. – № 2. – С. 137-146. – EDN: IAOORT.

[42] Комилиён, Ф. С. Мушкилоти амсиласозии математикӣ ва компютерии экосистемаҳои махзанҳои обӣ дар Тоҷикистон ва арзёбии динамикаи захираҳои биологии онҳо [Матн] / Ф.С. Комилиён // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои иҷтимоӣ-иқтисодӣ ва ҷамъиятӣ. – 2023. – № S4-1. – С. 98-118. – EDN: TWXPOJ.

[43] Комилиён, Ф. С. Тарҳрезии консептуалии популятсияи моҳиҳои экосистемаи обанбори пастоб [Матн] / Ф.С. Комилиён, И.М. Саидов // Пайёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – Душанбе, 2017. – № 1/3. – С. 99-104.

[44] Комилиён, Ф. С. Таҳлили математикии амсилаи марҳилаҳои ҳаёти оилаи занбӯри асал аз рӯи хусусиятҳои ҷинсӣ [Матн] / Ф.С. Комилиён, И.М. Саидзода // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2022. – № 3. – С. 20-35.

[45] Комилиён, Ф. С. Технологияҳои амсиласозии марҳилаҳои ҳаёти популятсияи мурғи марҷон [Матн] / Ф.С. Комилиён, Д.И. Солиҳзода // Фурӯғи илм. – 2024. – № 3. – С. 32-40. – EDN: KQSOZL.

[46] Комилиён, Ф. С. Усулҳои баҳисобгирии таъсири омилҳои табиӣ ва антропогенӣ дар амсилаи математикии ташаккул ва рушди марҳилаҳои ҳаёти оилаи занбӯри асал [Матн] / Ф.С. Комилиён, И.М. Саидзода // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2023. – № 2. – С. 5-18.

[47] Комилиён, Ф. С. Шабакаҳои компютерӣ ва сомонасозӣ. Воситаи таълимӣ [Матн] / Ф.С. Комилиён, З.Ф. Раҳмонзода. – Душанбе: Маориф, 2016. – 208 с. – ISBN: 978-99975-853-0-0 – EDN: KVZAIR.

[48] Комилов Ф. С. Трансформация математической модели в инструмент по прогнозированию динамики рыбной популяции экосистемы мелководного водохранилища [Текст] / Ф.С. Комилов, И.М. Саидов, М.Р. Ёров, И.Л. Косимов // Вестник Таджикского национального университета. Серия естественных наук. – Душанбе, 2017. – № 1-5. – С. 27-32.

[49] Комилов, Ф. С. Асосҳои информатика ва техникаи ҳисоббарор. Бейсик. Воситаи таълимӣ [Матн] / Ф.С. Комилов. – Душанбе, 1994. – 140 с. – EDN: FMBZMV.

[50] Комилов, Ф. С. Вобастагии байни омилҳои муҳити обӣ ва энергияи табодулотии моҳӣ [Матн] / Ф.С. Комилов, М.Р. Ёров, Ф.Т. Шамсов // Кишоварз. – 2018. – № 4 (80). – С. 85-89.

[51] Комилов, Ф. С. Выбор функции роста форели рыбоводной бассейновой системы для математического моделирования её отдельного этапа жизненного цикла [Текст] / Ф.С. Комилов, Ф.Т. Шамсов // Политехнический вестник. Серия: Интеллект. Инновации. Инвестиции. – 2019. – № 1 (45). – С. 37-42. – EDN: WJECML.

[52] Комилов, Ф. С. Имитационная модель динамики экосистемы макрофитного рыбоводного пруда. Монография [Текст] / Ф.С. Комилов, И.Л. Косимов. – Germania: LAP LAMBERT Academic Publishing (Project ID: #130070; ISBN 978-3-659-80543-1) // https://www.lap-publishing.com/ – LAP: OmniScriptum GmbH&Co. KG, Bahnhofstraße 28, D-66111 Saarbrücken, 2015. – 135 с.

[53] Комилов, Ф. С. Имитационное моделирование динамики экосистем водохранилищ [Текст] / Ф.С. Комилов. – Душанбе, 1996. – 142 с.

[54] Комилов, Ф. С. Имитационное моделирование управляемой высокопродуктивной экосистемы рыбоводного пруда [Текст] / Ф.С. Комилов, Д.С. Шарапов. – Душанбе: Сохибкор, 2000. – 80 с.

[55] Комилов, Ф. С. Информатика ва технологияҳои иттилоотӣ [Матн] / Ф.С. Комилов // Душанбе: Душанбе-принт, 2016. – 480 с.

[56] Комилов, Ф. С. Информатикаи татбиқӣ [Матн] / Ф.С. Комилов, Д.С. Шарапов. – Душанбе: «О», 2009. – 368 с. – EDN: OVYRDS.

[57] Комилов, Ф. С. Компьютерное моделирование экосистем водохранилищ [Текст] / Ф.С. Комилов. – Душанбе: Сохибкор, 2010. – 240 с. – EDN: LBRMMV.

[58] Комилов, Ф. С. Концептуальная модель экосистемы пруда с тремя видами рыб [Текст] / Ф.С. Комилов, И.Л. Косимов // Известия ВУЗов (Кыргызстан). – 2015. – № 7. – С. 6-9. – EDN: VKAQKR.

[59] Комилов, Ф. С. Математическая модель рыбной популяции экосистемы мелководного водохранилища [Текст] / Ф.С. Комилов, И.М. Саидов, Ф.Т. Шамсов // Вестник Таджикского национального университета. Серия естественных наук. – 2017. – № 1-3. – С. 18-25.

[60] Комилов, Ф. С. Математическое моделирование экосистем водохранилищ [Текст] / Ф.С. Комилов, А.А. Воинов, М.К. Юнусов. – Душанбе: «Общество Знание», 1989. – 20 с. – EDN: GJODFF.

[61] Комилов, Ф. С. Математическое моделирование экосистем водохранилищ [Текст] / Ф.С. Комилов, В.П. Пархоменко. – М.: ВЦ РАН, «Сообщение по прикладной математике», 2003. – 46 с. – EDN: QKMEUH.

[62] Комилов, Ф. С. О технологии концептуального моделирования экосистемы рыбоводного пруда с двумя видами рыб [Текст] / Ф.С. Комилов, С.Х. Мирзоев, И.М. Саидов // Наука, новые технологии и инновации Кыргызстана. – 2016. – № 5. – С. 37-39. – EDN: WDFGXZ.

[63] Комилов, Ф. С. Обменная энергия прудовой рыбы и её зависимость от внешних факторов водной среды [Текст] / Ф.С. Комилов, И.М. Саидов // Вестник Таджикского национального университета. Серия естественных наук. – 2015. – № 1-4 (168). – С. 157-163. – EDN: UMXIUV.

[64] Комилов, Ф. С. Разработка и исследование компьютерной модели динамики экосистемы рыбоводного пруда [Текст] / Ф.С. Комилов, С.Х. Мирзоев, Д.С. Шарапов. – Душанбе: ЭР-граф, 2018. – 176 с.

[65] Комилов, Ф. С. Состояние и перспективы развития рыбоводства на примере форелевой аквакультуры в Таджикистане [Текст] / Ф.С. Комилов, И.М. Саидов // Таджикистан и современный мир. – 2015. – № 7 (50). – С.153-162.

[66] Комилов, Ф. С. Форелеводство – высокоинтенсивная отрасль прудового рыбоводства [Текст] / Ф.С. Комилов, И.М. Саидов // Известия таджикского отделения международной академии наук высшей школы. – 2016. – № 1. – С. 47-55.

[67] Концепция государственной информационной политики Республики Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 4 марта 2002 года № 94.

[68] Концепция информационной системы управления сферой образования Республики Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 2 ноября 2007 года № 530.

[69] Концепция перехода к цифровому образованию в Республике Таджикистан на период до 2042 года. Утверждена постановлением Правительства Республики Таджикистан от 31 августа 2022 года № 439. – 10 с.

[70] Концепция развития профессионального образования в Республике Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 1 ноября 2006 года № 484. – Душанбе, 2006. – 19 с.

[71] Концепция формирования электронного правительства Республики Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 30 декабря 2011 года № 643.

[72] Концепция цифровой экономики в Республике Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 30 декабря 2019 года № 642. – 31 с.

[73] Лафферти, Дж. Условные случайные поля: вероятностные модели для сегментации и разметки последовательных данных [Текст] / Дж. Лафферти, А. МакКаллум, Ф.К.Н. Перейра // Труды 18-й Международной конференции по машинному обучению. – 2001. – С. 282-289.

[74] Лерхе, Й. Анализ производительности gRPC и REST для облачных приложений [Текст] / Й. Лерхе, М. Зёллльнер // Труды конференции IEEE по облачной инженерии 2020. – 2020. – С. 181-186.

[75] Ли, Дж. BioBERT: предобученная биомедицинская языковая модель для биомедицинского текстового анализа [Текст] / Дж. Ли, В. Юн, С. Ким и др. // Биоинформатика. – 2020. – Т. 36, № 4. – С. 1234-1240.

[76] Маэда, К. Оценка производительности библиотек сериализации объектов в форматах XML, JSON и Binary [Текст] / К. Маэда // Труды 2-й Международной конференции по обработке цифровой информации и коммуникациям. – 2020. – С. 177-182.

[77] Меншуткин, В. В. Искусство моделирования (экология, физиология, эволюция) [Текст] / В.В. Меншуткин. – СПб.: Петрозаводск, 2010. – 416с.

[78] Метод Рунге-Кутта [Электронный ресурс] // http://korablik280.ru/ya-polonskijj/metod-runge-pryamye-metody-runge-kutta/ (Дата обращения: 12.11.21).

[79] Миколов, Т. Эффективная оценка представлений слов в векторном пространстве [Текст] / Т. Миколов, К. Чен, Г. Коррадо, Дж. Дин // arXiv препринт arXiv:1301.3781. – 2013.

[80] Мирзоев, С. Х. Способ регуляризации неустойчивых структур сообществ экосистем региональных заповедников Республики Таджикистан [Текст] / С.Х. Мирзоев // Наука, новые технологии и инновации Кыргызстана. – 2016. – № 5. – С. 40-43.

[81] Национальная стратегия развития образования Республики Таджикистан на период до 2030 года. Утверждена постановлением Правительства Республики Таджикистан от 29 сентября 2020 года № 526. – 90 с.

[82] Национальная стратегия развития образования Республики Таджикистан до 2020 года. Утверждена постановлением Правительства Республики Таджикистан от 30 июня 2012 года № 334. – 93 с.

[83] Национальная стратегия развития Республики Таджикистан на период до 2030 года. Утверждена постановлением Маджлиси намояндагон Маджлиси Оли Республики Таджикистан от 1 декабря 2016 года № 636. – 88 с.

[84] Нуров, И. Д. Моделирование бифуркационных явлений в негладких динамических системах [Текст] / И.Д. Нуров, М.Ш. Халилова // Доклады Академии наук Республики Таджикистан. – Душанбе, 2010. – Т. 53. – № 10. – С. 752-758.

[85] Ньюман, С. Создание микросервисов: проектирование детально проработанных систем [Текст]: Монография / С. Ньюман. – Sebastopol: O'Reilly Media, 2015. – 280 с.

[86] Одинаев, Р. Н. Методы оптимизации [Текст] / Р.Н. Одинаев. – Душанбе,

[87] Пеннингтон, Дж. GloVe: глобальные векторы для представления слов [Текст] / Дж. Пеннингтон, Р. Сочер, К.Д. Мэннинг // Труды конференции по эмпирическим методам в обработке естественного языка 2014. – 2014. – С. 1532-1543.

[88] Пирес, Т. Насколько мультиязычен мультиязычный BERT? [Текст] / Т. Пирес, Э. Шлингер, Д. Гарретт // Труды 57-й ежегодной конференции Ассоциации компьютерной лингвистики. – 2019. – С. 4996-5001.

[89] Послание Президента Республики Таджикистан уважаемого Эмомали Раҳмон «Об основных направлениях внутренней и внешней политики республики» (23.12.2022) [Электронный ресурс]. – Режим доступа: https://mfa.tj/tg/main/view/11820/pajomi-prezidenti-jumhurii-tojikiston-muhtaram-emomali-rahmon-dar-borai-samthoi-asosii-sijosati-dokhili-va-khorijii-jumhuri (дата обращения: 12.05.2023).

[90] Программа внедрения информационных технологий в общеобразовательных учреждениях Республики Таджикистан на 2018–2022 годы. Утверждена постановлением Правительства Республики Таджикистан от 29 сентября 2017 года № 443.

[91] Программа инновационного развития Республики Таджикистан на 2011-2020 годы. Утверждена постановлением Правительства Республики Таджикистан от 30 апреля 2011 года № 227.

[92] Программа интеграции науки и высшего образования Республики Таджикистан на 2010–2015 годы. Утверждена постановлением Правительства Республики Таджикистан от 2 июля 2009 года № 371.

[93] Программа компьютеризации общеобразовательных учреждений Республики Таджикистан на 2008-2010 годы. Утверждена постановлением Правительства Республики Таджикистан от 5 марта 2008 года № 77.

[94] Программа развития естественно-научных, математических и технических наук на 2010–2020 годы. Утверждена постановлением Правительства Республики Таджикистан от 27 февраля 2010 года № 89.

[95] Программа реализации концепции государственной информационной политики Республики Таджикистан. Утверждена постановлением Правительства Республики Таджикистан от 28 мая 2009 года № 307.

[96] Ратнер, А. Snorkel: быстрое создание обучающих данных со слабым контролем [Текст] / А. Ратнер, С.Х. Бах, Х. Эренберг и др. // Труды эндаумента VLDB. – 2017. – Т. 11, № 3. – С. 269-282.

[97] Ричардсон, К. Микросервисы: паттерны разработки и рефакторинга [Текст]: Монография / К. Ричардсон. – Shelter Island: Manning Publications, 2018. – 520 с.

[98] Саидзода, И. М**.** Амсиласозии компютерии популятсияи оилаи занбӯри асал бо истифода аз усули Рунге-Кутт [Матн] / И.М. Саидзода, Ф.С. Комилиён // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2022. – № 2. – С. 26-42.

[99] Саидзода, И. М. Барномасозии компютерии фаъолияти занбӯри асал вобаста ба таъсири бемориҳо ва зараррасонҳо [Матн] / И.М. Саидзода, Ф.С. Комилиён // Паёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2024. – № 1. – С. 13-27. – EDN: PMYJCS.

[100] Саидзода, И. М. Барномасозии компютерии фаъолияти занбӯри асал вобаста ба таъсири бемориҳо ва зараррасонҳо [Матн] / И.М. Саидзода, Ф.С. Комилиён // Паёми донишгоҳи давлатии Хоруғ. – 2023. – № 1 (25). – С. 103-115.

[101] Саидзода, И. М. Таҳлили компютерии хусусиятҳои ҷинсии оилаи занбӯри асал вобаста ба марҳилаҳои ҳаёти он [Матн] / И.М. Саидзода, Ф.С. Комилиён // Пайёми Донишгоҳи миллии Тоҷикистон. Бахши илмҳои табиӣ. – 2022. – № 4.– С. 5-28.

[102] Саидзода, И. М. Учет влияния климатических факторов при компьютерном моделировании деятельности медоносной пчелиной семьи [Текст] / И.М. Саидзода, Ф.С. Комилиён // Евразийский Союз Ученых. Серия: технические и физико-математические науки. – 2024. – Т. 1. – № 4 (119). – С. 36- 41.

[103] Санг, Э. Ф. Введение в совместную задачу CoNLL-2003: языково-независимое распознавание именованных сущностей [Текст] / Э.Ф. Санг, Ф. Де Мельдер // Труды CoNLL-2003. – 2003. – С. 142-147.

[104] Стратегия изучения и развития математических, точных и естественных наук в сфере образования и науки на период до 2030 года. Утверждена постановлением Правительства Республики Таджикистан от 30 апреля 2021 года № 165. – 33 с.

[105] Стратегия инновационного развития Республики Таджикистан на период до 2020 года. Утверждена постановлением Правительства Республики Таджикистан от 30 мая 2015 года № 354.

[106] Стратегия Республики Таджикистан в области науки и технологий на 2011-2012 годы. Утверждена постановлением Правительства Республики Таджикистан от 3 марта 2011 года № 114.

[107] Сюй, Й. LayoutLM: предобучение текста и макета для понимания изображений документов [Текст] / Й. Сюй, М. Ли, Л. Цуй и др. // Труды

26-й Международной конференции ACM SIGKDD по обнаружению знаний и интеллектуальному анализу данных. – 2020. – С. 1192-1200.

[108] Усманов, З. Д. Система обыкновенных дифференциальных уравнений, связанная с гипергеометрическими функциями [Текст] / З.Д. Усманов // Доклады Академии наук Республики Таджикистан. – 2012. – Т. 55. – № 1. – С. 5-10.

[109] Фаулер, М. Микросервисы: определение нового архитектурного термина / М. Фаулер, Дж. Льюис // martinfowler.com. – 2014.

[110] Филдинг, Р. Т. Архитектурные стили и проектирование сетевых программных архитектур [Текст]: Диссертация / Р.Т. Филдинг. – Университет Калифорнии, Ирвайн, 2000.

[111] Хамидова, Д. Н. Компьютерное моделирование амилоидных фибрилл и их процесс разрушения дендримерами и дендриграфтами [Текст] / Д.Н. Хамидова, И.М. Неелов, Ф.С. Комилиён. – Душанбе: Бебок, 2023. – 152с.

[112] Хуанг, З. Двунаправленные модели LSTM-CRF для разметки последовательностей [Текст] / З. Хуанг, В. Сюй, К. Ю // arXiv препринт arXiv:1508.01991. – 2015.

[113] Худойбердиев Х.А. Моделирование системы автоматической обработки текста на таджикском языке / Х.А. Худойбердиев // International Journal of Open Information Technologies. – 2023. – Т. 11, № 3. – С. 27-33.

[114] Худойбердиев Х.А. Разработка и реализация системы машинного перевода на основе правил с русского на таджикский язык / Х.А. Худойбердиев // Политехнический вестник. Серия: Интеллект. Инновации. Инвестиции. – 2023. – № 2(62). – С. 33-36.

[115] Чжу, С. Обзор литературы по обучению с частичным привлечением учителя [Текст]: Технический отчет 1530 / С. Чжу. – Университет Висконсин-Мэдисон, факультет компьютерных наук, 2005.

[116] Читикариу, Л. Извлечение информации на основе правил мертво! Да здравствует извлечение информации на основе правил! [Текст] / Л. Читикариу, Й. Ли, Ф. Рейсс // Труды конференции по эмпирическим методам в обработке естественного языка 2013. – 2013. – С. 827-832.

[117] Шапель, О. Обучение с частичным привлечением учителя [Текст]: Монография / О. Шапель, Б. Шёлькопф, А. Цзень. – Cambridge: MIT Press, 2006. – 544 с.

[118] Шерматов, Н. Ш. Основы регрессионного анализа [Текст] / Н.Ш. Шерматов, Р.Н. Одинаев, К.С. Шакарбеков. – Душанбе: Эр-граф, 2019. – 66 с.

[119] Эрл, Т. Сервис-ориентированная архитектура: концепции, технологии и проектирование [Текст]: Монография / Т. Эрл. – Upper Saddle River: Prentice Hall, 2005. – 760 с.

[120] Юнуси, М. К. Оптимальное управление в задачах защиты планируемого урожая, охраняемыми биологическими популяциями и их приложения [Текст] / М.К. Юнуси. – Душанбе, 2018. – 286 с.

[121] Amdahl, G. M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities[Electronic resource] / G.M. Amdahl // https://www3.cs.stonybrook.edu/~rezaul/Spring-2012/CSE613/reading/Amdahl-1967.pdf (Date of the application: 15.03.2026).

[122] Azure Architecture Center [Electronic resource] // URL: https://learn.microsoft.com/ru-ru/azure/architecture/microservices/design/orchestration (Date of the application: 01.04.2021).

[123] Bass, L. DevOps: A Software Architect's Perspective [Electronic resourse] / L. Bass // Chapter 3. Addison-Wesley, Boston (Date of the application: 01.06.2015).

[124] Bass, L. DevOps: A Software Architect's Perspective [Text] / L. Bass, I. Weber, L. Zhu. – Boston: Addison-Wesley, 2015. – 320 p.

[125] Brown, T. Language Models are Few-Shot Learners [Text] / Brown T. et al. // NeurIPS. – 2020.

[126] Burns, B. Kubernetes: Up and Running [Text] / B. Burns, J. Beda, K. Hightower. – Sebastopol: O'Reilly Media, 2019. – 330 p.

[127] Chen, T. XGBoost: A Scalable Tree Boosting System [Text] / T. Chen, C. Guestrin // KDD. – 2016. – P. 785-794.

[128] Chiticariu, L. Rule-based information extraction [Text] / L. Chiticariu, Y. Li, F. Reiss // Proceedings of EMNLP 2013. – 2013. – P. 827-832.

[129] Christian, P. Introducing Istio Service Mesh for Microservices [Electronic resourse] / P. Christian, S. Burr // URL: https://www.oreilly.com/library/view/introducing-istio-service/9781492052630/ch01.html#:~:text=,com%E3%80%91 (Date of the application: 01.07.2019).

[130] Devlin, J. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Text] / J. Devlin, M. W. Chang, K. Lee, K. Toutanova // Proceedings of NAACL-HLT. – 2019. – P. 4171-4186.

[131] Dragoni, N. Microservices: Yesterday, Today, and Tomorrow [Text] / N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustfin, L. Safina // Present and Ulterior Software Engineering. – Springer, 2017. – P. 195-216.

[132] Earl, T. Service-Oriented Architecture: Concepts, Technologies, and Design [Text] / T. Earl. – Upper Saddle River: Prentice Hall, 2005. – 760 p.

[133] Fehling, C. Service migration patterns – decision support and best practices for the migration of existing service-based applications to cloud environments [Electronic resource] / C. Fehling, F. Leymann, S. Ruehl, M. Rudek, S. Verclas (Date of the application: 01.12.2023).

[134] Fowler, M. Guide to Microservices [Electronic resource] / M. Fowler // URL: https://martinfowler.com/microservices/(Date of the application: 5.07.2022).

[135] Fowler, M. Microservices Architecture [Электронный ресурс] / M. Fowler // URL: https://martinfowler.com/microservices/ (Date of the application: 5.07.2022).

[136] Fowler, M. Microservices: A Definition of This New Architectural Term [Electronic resource] / M. Fowler, J. Lewis // URL: https://martinfowler. com/microservices/ (Date of the application: 15.03.2026).

[137] Gaurav, K. Building Microservices with NET Core [Electronic resourse] / K. Gaurav, K. Lalit, M. Kanwar (Date of the application: 01.06.2017).

[138] Goodfellow, I. Deep Learning [Electronic resource] / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge: MIT Press, 2016. – 775 p. // https://mitpress. mit.edu/9780262035613/deep-learning/ (Date of the application: 23.03.2026).

[139] Google.gRPC: высокопроизводительный универсальный RPC-фреймворк с открытым исходным кодом [Текст]. – grpc.io, 2015.

[140] Huang, Z. Bidirectional LSTM-CRF Models for Sequence Labeling [Electronic resource] / Z. Huang, W. Xu, K. Yu // https://arxiv.org/abs/ 1508.01991 (Date of the application: 15.03.2026).

[141] Jurafsky, D. Speech and Language Processing (3rd ed.) [Electronic resource] / D. Jurafsky, J.H. Martin. // https://www.scirp.org/reference/ referencespapers?referenceid=3970727 (Date of the application: 23.03.2026).

[142] Kleppmann, M. Designing Data-Intensive Applications [Electronic resource] / M. Kleppmann. – O'Reilly Media, 2017. // https://www.oreilly.com/library /view/designing-data-intensive-applications/9781491903063/ (Date of the application: 23.03.2026).

[143] Komilov, F. Computer-based study of patterns in fish pond ecosystem evolution [Text] / F. Komilov, S. Mirzoyev // Journal of Mechanical Engineering Research and Developments. – Bangladesh, 2018. – Vol. 41. – No. 2. – P. 9-13.

[144] Komilov, F. Computer-based study of patterns in fish pond ecosystem evolution [Text] / F. Komilov, S. Mirzoyev // Mechanical Engineering Research & Developments (JMERD). – Bangladesh, 2018. – Vol. 41. – No.1. – P. 142-150.

[145] Lafferty, J. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequential Data / J. Lafferty, A. McCallum, F. Pereira // Proceedings of ICML. – 2001. – P. 282-289.

[146] Mazlami, G. Extraction of microservices from monolithic software architectures [Electronic resource] / G. Mazlami et al. // International Conference on Web Services (ICWS), 2017, IEEE. – P. 524-531.

[147] Mikolov, T. Efficient estimation of word representations in vector space [Electronic resource] / T. Mikolov, K. Chen, G. Corrado, J. Dean // https://arxiv.org/abs/1301.3781 (Date of the application: 15.03.2026).

[148] Newman, S. Building Microservices [Electronic resource] / S. Newman // URL: https://www.oreilly.com/library/view/building-microservices-2nd/ 9781 492034018/ (Date of the application: 21.08.2022).

[149] Newman, S. Building Microservices: Designing Fine-Grained Systems [Text] / S. Newman. – Sebastopol: O'Reilly Media, 2015. – 280 p.

[150] Newman, S. Monolith to Microservices [Text] / S. Newman. – Sebastopol: O'Reilly Media, 2020. – 150 p.

[151] Pahl, C. Containerization and the PaaS cloud [Text] / C. Pahl // IEEE Cloud Computing. – 2015. – Vol. 2(3). – P. 24-31.

[152] Peinl, R. Docker cluster management for the cloud-survey results and own solution [Electronic resource] / R. Peinl, F. Holzschuher, F. Pfitzer / (Date of the application: 01.02.2017).

[153] Pennington, J. GloVe: Global vectors for word representations [Text] / J. Pennington, R. Socher, K. Manning // Proceedings of EMNLP. – 2014. – P. 1532-1543.

[154] Pires, T. How multilingual is multilingual BERT? [Text] / T. Pires, E. Schlinger, D. Garrett // ACL. – 2019. – pp. 4996-5001.

[155] Ratner, A. Snorkel: Fast Training Data Generation [Text] / A. Ratner, S. Bach, H. Ehrenberg et al. // VLDB. – 2017. – Vol. 11. – No. 3. – P. 269-282.

[156] Richards, M. Microservices vs Service-Oriented Architecture [Text] / M. Richards. – Sebastopol: O'Reilly Media, 2016. – 190 p.

[157] Richardson, K. Microservices: Development and Refactoring Patterns [Text] / K. Richardson. – Shelter Island: Manning Publications, 2018. – 520 p.

[158] Rogers, A. A Primer in BERTology / A. Rogers, O. Kovaleva, A. Rumshisky // Transactions of ACL. – 2020. – Vol. 8. – P. 842-866.

[159] Sander, V. Building and Managing Microservices with Kubernetes and Istio [Electoring resourse] / V. Sander // URL: https://www.oreilly.com/library/view/building-and-managing/9780137649686/ (Date of the application: 01.12.2021).

[160] Sang, E.F. Introduction to the CoNLL-2003 Collaborative Problem [Text] / E.F. Sang, F. De Meulder // Proceedings of CoNLL-2003. – 2003. – P. 142-147.

[161] Sennrich, R. Neural Machine Translation of Rare Words with Subword Units [Electronic resource] / R. Sennrich, B. Haddow, A. Birch. – ACL, 2016. // https://aclanthology.org/P16-1162/ (Date of the application: 15.03.2026).

[162] Sigma Software. Migrating Monolith to Microservices: Step-by-Step Guide [Electronic resource] // URL: https://sigma.software/about/media/migrating-monolith-microservices-step-step-guide (Date of the application: 01.04.2021).

[163] Springer Link. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report [Electronic resource] // URL: https://link.springer.com/chapter/10.1007/978-3-319-33313-7_15#:~:text =Section%203%20describes %20the%20architecture,Department%20of%20 Computer%20Engineering (Date of the application: 01.04.2021).

[164] Tanenbaum, A. S. Distributed Systems (3rd ed.) [Electronic resource] / A.S. Tanenbaum, M. Van Steen. – Pearson, 2017. // https://www.distributed-systems.net/index.php/books/ds3/ (Date of the application: 23.03.2026).

[165] Vaswani, A. Attention Is All You Need [Text] / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin // Advances in Neural Information Processing Systems (NeurIPS). – 2017. – Vol. 30. – P. 5998-6008.

[166] Wolf, T. Transformers: State-of-the-Art Natural Language Processing [Text] / T. Wolf, L. Debut, V. Sanh et al. // Proceedings of EMNLP. – 2020. – P. 38-45.

[167] Xu, Y. LayoutLM: Text and Layout Pretraining [Text] / Y. Xu, M. Li, L. Cui, et al. // KDD. – 2020. – P. 1192-1200.

[168] Yanaga, E. Migrating to Microservice Databases Electronic [Electronic resource] / E.Yanaga // URL: https://www.oreilly.com/library/view/migrating-to-microservice/9781492048824/ (Date of the application: 01.01.2022).

[169] Young, B. Efficient Node.js [Electronic resource] / B. Young // URL: https://learning.oreilly.com/library/view/efficient-node-js/9781098145187/ /(Date of the application: 01.07.22) (Date of the application: 01.07.22).

[170] Zhang, Q. A survey on cloud computing: state-of-the-art and research challenges [Text] / Q. Zhang, M. Chen // Journal of Internet Services and Applications. – 2010. – Vol. 1. – P. 7-18.

2. Список научных публикаций соискателя ученой степени

а) Статьи, опубликованные в рецензируемых журналах Высшей аттестационной комиссии при Президенте Республики Таджикистан:

[1-А] Рахимов, М. Ф. Алгоритмизация микросервисной модели интеллектуального документооборота в профобразовании на базе BERT [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Вестник национального университета. Серия естественных наук. – 2026. – № 2. – С. 98-109. – EDN: OSUPXM.

[2-А] Рахимов, М. Ф. Внедрение микросервисной архитектуры для оптимизации распределения информационных ресурсов [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Наука и инновация. Серия геологических и технических наук. – 2024. – № 2. – С. 71-79. – EDN: UZJNVI.

[3-А] Рахимов, М. Ф. Интеллектуальная система документооборота на основе микросервисной архитектуры и моделей BERT [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Вестник национального университета. Серия естественных наук. – 2026. – № 2. – С. 76-85. – EDN: ROVNVV.

[4-А] Рахимов, М. Ф. Математическое моделирование интеллектуальной системы документооборота на основе микросервисной архитектуры и

моделей BERT [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Вестник Технологического университета Таджикистана. – 2026. – № 3 (66). – С. 96-106. – EDN: RWEDOQ.

[5-А] Рахимов, М. Ф. Микросервисная архитектура и BERT-модели в интеллектуальном документообороте профобразования [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Политехнический вестник. Серия: Интеллект. Инновации. Инвестиции. – 2026. – № 2 (74). – С. 62-71. – EDN: VJVTAH.

[6-А] Рахимов, М. Ф. Микросервисная архитектура: от монолита к гибким распределенным системам [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Доклады Национальной академии наук Таджикистана. – 2023. – Т. 66. – № 11-12. – С. 659-667. – EDN: LSVHDL.

[7-А] Рахимов, М. Ф. Анализ характеристик монолитных и микросервисных архитектур [Текст] / М.Ф. Рахимов, Ф.С. Комилиён // Известия Национальной академии наук Таджикистана. Отделение физико-математических, химических, геологических и технических наук. – 2023. – № 4 (193). – С. 44-54. – EDN: GHUYHC.

б) Статьи, опубликованные в материалах конференций:

[8-А] Рахимов, М. Ф. Микросервисная оптимизация распределения информационных ресурсов с помощью четко определенного «API» [Текст] / Ф.С. Комилиён, М.Ф. Рахимов // Современные проблемы математического моделирования и её применения. Материалы XII международной научно-практической конференции, посвященной «2020-2040 годы – 20-летия изучения и развития естественных, точных и математических наук в области науки и образования» и «75-летию Таджикского национального университета». – Душанбе, 2024. – С. 28-32. – EDN: PZLKJR.

[9-А] Рахимов, М. Ф. Влияние микросервисных приложений на улучшение работы IT приложений, на примере использования Node.js [Текст] / М.Ф. Рахимов // Материалы республиканской научно-практической конференции «Актуальные проблемы и перспективы развития естественных и точных наук». – Душанбе, ФМГУ, 2023. – С. 12-15.

[10-А] Рахимов, М. Ф. Внедрение микросервисной архитектуры в монолитные системы [Текст] / М.Ф. Рахимов // Материалы международной научно-практической конференции «Наука и образование: тенденции развития в условиях информационного общества», посвященной «75-ой годовщине со дня образования Таджикского национального университета». – Душанбе, ТНУ, 2023. – С. 8-11.

[11-А] Рахимов, М. Ф. Микросервисы в задачах оптимизации распределения информационных ресурсов организации [Текст] / М.Ф. Рахимов, Ф.С. Комилиён // Материалы республиканской научно-практической

конференции по теме «Роль интернета в применении информационно-коммуникационных технологий и информационной безопасности в социальных сетях» – Душанбе, АГУ, 2024. – С. 16-24.

[12-A] Rahimov, M. F. Mathematical Modeling of an Intelligent Document Management System Based on Microservice Architecture and BERT Models [Electronic resource] / F. Rahimi, F.S. Komiliyon, M.F. Rahimov, M.R. Yorov // Applied and Computational Mathematics. – 2026. – Vol. 15. – No. 3. – P. 68-74. // https://doi.org/10.11648/j.acm. 20261503.11 (Date of the application: 14.05.2026).

ПРИЛОЖЕНИЕ 1

ИСХОДНЫЙ КОД И ОПИСАНИЕ ПРОГРАММНОЙ СИСТЕМЫ IntelliDoc

Общая характеристика программной системы. Программная система IntelliDoc разработана для автоматизированного анализа текстовых документов с целью выявления цитируемых фрагментов и определения доли заимствованного текста. Система реализована в виде веб-приложения с использованием современных технологий фронтенд-разработки.

Основные функции системы:

- ввод текстовых данных пользователем;
- автоматическое выявление цитат;
- вычисление процента цитирования;
- отображение результатов анализа.

Архитектура системы построена по клиентской модели и включает следующие компоненты:

- пользовательский интерфейс;
- модуль обработки текста;
- модуль анализа цитирования;
- модуль визуализации результатов.

Логическая схема работы:

- пользователь вводит текст;
- текст передаётся в модуль анализа;
- выполняется поиск цитат;
- рассчитывается процент;
- результат отображается пользователю.

Используемые технологии. В системе применены следующие технологии:

- язык программирования: TypeScript
- библиотека: React

- сборщик проекта: Vite
- язык разметки: HTML5
- стилизация: CSS3

Структура проекта:

```
intellidoc/
│
├── index.html
├── package.json
├── tsconfig.json
├── vite.config.ts
│
└── src/
    ├── main.tsx
    ├── App.tsx
    ├── styles.css
```

Основной HTML-файл. Файл index.html.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>IntelliDoc</title>
</head>
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.tsx"></script>
</body>
</html>
```

Описание. Данный файл является точкой входа веб-приложения. Элемент служит контейнером для рендеринга интерфейса.

Конфигурация проекта. Файл package.json:

```
{
 "name": "intellidoc",
 "version": "1.0.0",
 "private": true,
 "scripts": {
  "dev": "vite",
  "build": "vite build"
 },
 "dependencies": {
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
 },
 "devDependencies": {
  "@vitejs/plugin-react": "^4.0.0",
  "typescript": "^5.0.0",
  "vite": "^5.0.0"
 }
}
```

Описание. Файл содержит список зависимостей проекта и команды для запуска и сборки.

Файл tsconfig.json:

```
    {
     "compilerOptions": {
      "target": "ESNext",
      "lib": ["DOM", "ESNext"],
      "module": "ESNext",
      "jsx": "react-jsx",
      "strict": true
     },
```

```
  "include": ["src"]
}
```

Описание. Файл определяет правила компиляции TypeScript, включая строгую типизацию.

Файл vite.config.ts.

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
export default defineConfig({
 plugins: [react()]
})
```

Описание. Настройка сборщика Vite для работы с React-приложением.

Точка входа приложения. Файл main.tsx.

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./styles.css";
ReactDOM.createRoot(document.getElementById("root")!).render(
 <React.StrictMode>
  <App />
 </React.StrictMode>
);
```

Описание. Файл выполняет:

- подключение основного компонента;
- инициализацию приложения;
- рендеринг интерфейса.

Основной программный модуль. Файл App.tsx.

```
import React, { useState } from "react";
function App() {
 const [text, setText] = useState("");
 const [quotes, setQuotes] = useState<string[]>([]);
```

```
  const [percent, setPercent] = useState<number>(0);
  const analyze = () => {
    const matches = text.match(/"(.*?)"/g) || [];
    setQuotes(matches);
    const totalLength = text.length;
    const quotesLength = matches.join("").length;
    const result = totalLength
      ? ((quotesLength / totalLength) * 100).toFixed(2)
      : "0";
    setPercent(Number(result));
  };
  return (
    <div>
      <h1>IntelliDoc</h1>
      <textarea
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
      <button onClick={analyze}>Анализ</button>
      <p>Процент цитирования: {percent}%</p>
      <ul>
        {quotes.map((q, i) => (
          <li key={i}>{q}</li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

Описание модуля. Модуль реализует:

- хранение состояния текста;
- поиск цитат;
- вычисление процента;
- вывод результатов.

Алгоритм анализа цитирования. Алгоритм включает следующие шаги:

- получение текста;
- поиск всех подстрок в кавычках;
- подсчёт длины цитат;
- подсчёт общей длины текста;
- вычисление процента.

Ключевая строка анализа.

```
const matches = text.match(/"(.*?)"/g);
```

Описание. Используется регулярное выражение для поиска текста в кавычках.

Модуль пользовательского интерфейса. Пользовательский интерфейс реализован с применением каскадных таблиц стилей (CSS) и обеспечивает удобное взаимодействие пользователя с системой. Основные элементы интерфейса:

- поле ввода текста;
- кнопка запуска анализа;
- блок отображения результатов;
- список найденных цитат.

Файл styles.css.

```
body {
  font-family: Arial, sans-serif;
  background: #f4f6f8;
  margin: 0;
}
.container {
  max-width: 900px;
```

```
  margin: auto;
  background: #ffffff;
  padding: 25px;
  margin-top: 40px;
  border-radius: 12px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.1);
}
h1 {
  text-align: center;
            }
            textarea {
              width: 100%;
              height: 220px;
              padding: 12px;
              margin-top: 15px;
              font-size: 14px;
              border-radius: 8px;
              border: 1px solid #ccc;
            }
button {
  margin-top: 15px;
  padding: 12px 25px;
  background: #007bff;
  color: white;
  border: none;
  border-radius: 6px;
  cursor: pointer;
}
button:hover {
  background: #0056b3;
```

```
}
.result {
  margin-top: 25px;
}
ul {
  margin-top: 10px;
}
```

Описание. Данный файл отвечает за:

- визуальное оформление интерфейса;
- улучшение пользовательского опыта;
- адаптивность отображения.

Улучшенная версия интерфейса. Обновлённый App.tsx с UI.

```
import React, { useState } from "react";
import "./styles.css";
function App() {
 const [text, setText] = useState("");
 const [quotes, setQuotes] = useState<string[]>([]);
 const [percent, setPercent] = useState<number>(0);
 const analyze = () => {
    const matches = text.match(/"(.*?)"/g) || [];
    setQuotes(matches);
    const totalLength = text.length;
    const quotesLength = matches.join("").length;
    const result = totalLength
      ? ((quotesLength / totalLength) * 100).toFixed(2)
      : "0";
  setPercent(Number(result));
 };
 return (
  <div className="container">
```

```
      <h1>IntelliDoc</h1>
      <textarea
        placeholder="Введите текст для анализа..."
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
      <button onClick={analyze}>Анализировать</button>
      <div className="result">
        <p><b>Процент цитирования:</b> {percent}%</p>
        <h3>Цитаты:</h3>
        <ul>
          {quotes.map((q, i) => (
            <li key={i}>{q}</li>
          ))}
        </ul>
      </div>
    </div>
  );
}
export default App;
```

Описание. Добавлены:

- стилизация интерфейса;
- улучшенная читаемость;
- логическое разделение блоков.

Расширение функциональности: загрузка файлов. Для повышения практической ценности системы реализована возможность загрузки текстовых файлов.

Обработка загрузки файла.

```
const handleFileUpload = (event: React.ChangeEvent<HTMLInputElement>) => {
  const file = event.target.files?.[0];
```

```
  if (!file) return;
  const reader = new FileReader();
  reader.onload = (e) => {
    const content = e.target?.result as string;
    setText(content);
  };
  reader.readAsText(file);
};
```

Использование в интерфейсе.

```
<input type="file" accept=".txt" onChange={handleFileUpload} />
```

Описание. Модуль позволяет:

- загружать текстовые файлы;
- автоматически вставлять содержимое;
- анализировать без копирования вручную.

Расширение алгоритма анализа. Базовый алгоритм был дополнен для повышения точности.

Улучшенный анализ.

```
const analyze = () => {
  const matches = text.match(/"(.*?)"/g) || [];
  const cleanQuotes = matches.map(q => q.replace(/"/g, ""));
  const quotesLength = cleanQuotes.reduce(
    (sum, q) => sum + q.length, 0
  );
  const totalLength = text.length;
  const percent = totalLength
    ? (quotesLength / totalLength) * 100
    : 0;
  setQuotes(cleanQuotes);
  setPercent(Number(percent.toFixed(2)));
};
```

Описание улучшений:

- удаление кавычек из расчёта;
- более точный подсчёт длины;
- повышение корректности результата.

Обработка ошибок. Для повышения устойчивости системы реализована обработка ошибок. Проверка входных данных:

```
if (!text || text.trim() === "") {
  alert("Введите текст для анализа");
  return;
}
```

Описание. Позволяет избежать:

- пустого анализа;
- некорректных вычислений.

Возможность расширения системы. Система может быть расширена следующими модулями:

- анализ перефразированного текста;
- подключение базы научных работ;
- интеграция с системами антиплагиата;
- хранение результатов анализа.

Преимущества разработанной системы:

- простота использования;
- высокая скорость анализа;
- независимость от сторонних сервисов;
- возможность дальнейшего масштабирования.

Формализованное описание алгоритма. Алгоритм анализа цитирования можно представить в формализованном виде:

Входные данные:

- строка текста T

Выходные данные:

- список цитат Q

- процент цитирования P

Последовательность выполнения:

- получить текст T
- найти все подстроки в кавычках $\rightarrow Q$
- удалить служебные символы
- вычислить:
  - длину текста L_T
  - суммарную длину цитат L_Q

Рассчитать процент:

$$P = \frac{L_Q}{L_T} \cdot 100$$

Вывести результат.

Псевдокод алгоритма:

```
ввод T
если T пустой:
вывести "Ошибка"
завершить
Q ← найти все совпадения по шаблону "(...)"
очистить Q от кавычек
L_T ← длина T
L_Q ← сумма длин всех элементов Q
если L_T > 0:
P ← (L_Q / L_T) * 100
иначе:
P ← 0
вывести P и Q
```

Логическая схема системы:

```
[Пользователь]
    ↓
[Ввод текста / загрузка файла]
```

↓

[Модуль анализа]

↓

[Поиск цитат (RegExp)]

↓

[Подсчёт процента]

↓

[Вывод результата]

Инструкция пользователя:

1. Запуск системы:

npm install

npm run dev

2. Работа с программой:

- открыть приложение в браузере
- вставить текст или загрузить файл
- нажать кнопку «анализировать»
- получить результат

3. Результат включает:

- процент цитирования;
- список найденных цитат.

Пример работы системы. Входной текст:

Это пример текста. "Цитата номер один". И ещё "вторая цитата".

Результат:

Цитаты:

- Цитата номер один
- вторая цитата

Процент цитирования: XX%

Ограничения системы. Разработанная система имеет следующие ограничения:

- анализируются только прямые цитаты (в кавычках);

- не учитывается перефразирование;
- не проводится семантический анализ текста;
- отсутствует интеграция с научными базами данных.

Направления дальнейшего развития. Система может быть улучшена следующим образом:

- внедрение NLP (обработка естественного языка);
- подключение антиплагиат-систем;
- анализ PDF и DOCX файлов;
- использование машинного обучения.

Практическая значимость. Разработанная система может применяться:

- при проверке научных работ;
- в образовательных учреждениях;
- для автоматизации анализа текстов.

Преимущества системы:

- простота реализации;
- высокая скорость работы;
- наглядность результатов;
- масштабируемость.

Заключение по приложению. В рамках данного приложения представлен исходный код и описание программной системы IntelliDoc, предназначенной для анализа цитирования текстов. Разработанное решение демонстрирует возможность применения современных веб-технологий для автоматизации обработки текстовой информации.

ПРИЛОЖЕНИЕ 2

«УТВЕРЖДАЮ»
Председатель Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан
Фарход Рахими _____
«24» 04 2026 года

АКТ

внедрения основных результатов диссертационной работы Рахимова Манучехра Фарходовича на тему «Математическое моделирование интеллектуальной системы документооборота на основе микросервисной архитектуры и нейросетевых механизмов BERT»

В условиях глобальной цифровой трансформации экономики и общества особую значимость приобретает развитие интеллектуальных информационных систем, ориентированных на автоматизацию обработки данных и повышение эффективности управленческих процессов. Одним из ключевых направлений цифровизации является совершенствование систем документооборота, играющих важную роль в деятельности государственных, коммерческих и финансовых организаций.

Практическая ценность диссертационного исследования Рахимова М.Ф. заключается в том, что предложенные ими решения доведены до стадии инженерной реализации. Разработанный алгоритм внедрения, основанный на математической модели системы и адаптированный к условиям функционирования Комитета по начальному и среднему профессиональному образованию при Правительстве Республики Таджикистан, может быть использован в качестве типового подхода при цифровизации документооборота в учреждениях по следующим направлениям:

- госсектор: поэтапная интеграция интеллектуальных систем в управленческую инфраструктуру.
- коммерция: автоматизация обработки документов для повышения эффективности бизнес-процессов и снижения затрат.
- финансы: реализация процедур KYC, обработки кредитных заявок и комплаенс-проверок через интеллектуальный анализ.
- ИТ-инфраструктура: построение отказоустойчивых систем на базе микросервисов и контейнеризации.
- искусственный интеллект: адаптация моделей BERT для NLP-систем в низкоресурсных языках.
- наука и образование: внедрение результатов в учебные программы по машинному обучению и цифровой трансформации.

Начальник сектора ИТЦ **А.Гулзода**

SciencePG
Science Publishing Group

Science Publishing Group
1 Rockefeller Plaza, 10th and 11th Floors,
New York, NY 10020 U.S.A.

Acceptance Letter

Reference: ACM **Date:** April 25, 2026

Dear Farhod Rahimi, F.S. Komiliyon, M.F. Rahimov, M.M. Yorov,

Thanks for your contribution to ***Applied and Computational Mathematics (ACM)***.
(ISSN Online: 2328-5613; ISSN Print: 2328-5605)

We are pleased to inform you that your manuscript has been accepted for publication.

Manuscript information:
Manuscript Number: 1471972
Manuscript Title: Mathematical Modeling of an Intelligent Document Management System Based on Microservice Architecture and BERT Models

Best Regards
Applied and Computational Mathematics (ACM)
http://www.sciencepg.com/journal/acm
Science Publishing Group

For and on behalf of
SCIENCE PUBLISHING GROUP INC

Authorized Signature (s)