API-технологии в разработке ИИ-систем

Учебно-методическое пособие

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Российский государственный аграрный университет –
МСХА имени К. А. Тимирязева»

API-ТЕХНОЛОГИИ В РАЗРАБОТКЕ ИИ-СИСТЕМ

Учебно-методическое пособие для студентов
направления подготовки 09.03.03 Прикладная информатика
направленностей «Программные решения для бизнеса»
и «Системы искусственного интеллекта»

Электронное текстовое издание

Санкт-Петербург
Наукоемкие технологии
2026

© Греченева А. В., Потапов Д. С.,
Лапшин С. М., Бакнин М. Д., 2026
ISBN 978-5-00271-131-4

УДК 004.4:004.8
ББК 32.813:32.973
А76

Авторы:
А. В. Греченева, Д. С. Потапов, С. М. Лапшин, М. Д. Бакнин

Рецензент:
Дмитрий Игоревич Суржик, д.т.н., доцент,
Московский университет МВД России имени В. Я. Кикотя

Греченева, А. В.
А76 API-технологии в разработке ИИ-систем [Электронный ресурс]: учебно-методическое пособие / А. В. Греченева, Д. С. Потапов, С. М. Лапшин, М. Д. Бакнин. – СПб.: Наукоемкие технологии, 2026. – 37 с. – URL: https://publishing.intelgr.com/archive/API-tekhnologii-v-razrabotke-II-sistem.pdf.

ISBN 978-5-00271-131-4

Пособие предназначено для студентов направления подготовки 09.03.03 «Прикладная информатика», обучающихся по направленностям «Программные решения для бизнеса» и «Системы искусственного интеллекта» в рамках образовательных программ топ-уровня в сфере искусственного интеллекта. В пособии рассматриваются принципы проектирования, разработки, тестирования, профилирования и эксплуатации API-сервисов, обеспечивающих интеграцию моделей машинного обучения и компонентов искусственного интеллекта в прикладные информационные системы.

Пособие построено как сквозной инженерный проект: от постановки задачи и разработки API-контракта до реализации сервиса, подключения модели, проведения нагрузочного тестирования, оптимизации и защиты итогового решения. Для направленности «Программные решения для бизнеса» предусмотрена траектория на Python/FastAPI и инструментах обработки данных. Для направленности «Системы искусственного интеллекта» предусмотрена траектория на C++ с акцентом на многопоточность, профиль производительности, рантаймы инференса и аппаратно-программные ограничения.

Издание подготовлено в рамках договора о предоставлении гранта № 70-2025-000753 от 29 мая 2025 г., источником финансового обеспечения которого является субсидия, предоставленная из федерального бюджета, на обеспечение обучения студентов по образовательным программам высшего образования для топ-специалистов в сфере искусственного интеллекта (уровня «ДС») по федеральному проекту «Искусственный интеллект» национального проекта «Экономика данных и цифровая трансформация государства».

УДК 004.4:004.8
ББК 32.813:32.973

© Греченева А. В., Потапов Д. С.,
Лапшин С. М., Бакнин М. Д., 2026

ISBN 978-5-00271-131-4

Учебное издание

Греченева Анастасия Владимировна
Потапов Даниил Сергеевич
Лапшин Сергей Михайлович
Бакнин Максим Дмитриевич

API-технологии в разработке ИИ-систем

Учебно-методическое пособие

Электронное текстовое издание

Издается в авторской редакции
Верстка *Ю. Н. Сафонкина*
Главный редактор *В. М. Коровин*

Подписано к использованию 06.05.2026.
Объем издания – 0,7 Мб.

Издательство «Наукоемкие технологии»
ООО «Корпорация «Интел Групп»
https://publishing.intelgr.com
E-mail: publishing@intelgr.com
Тел.: +7 (812) 945-50-63
Интернет-магазин издательства
https://shop.intelgr.com/

ISBN 978-5-00271-131-4



ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 6

1. Теоретическая часть 8

1.1. API как инженерный контур ИИ-системы 8
1.2. Архитектуры API-сервисов для инференса 8
1.3. REST, OpenAPI, gRPC и Proto-контракты 9
1.4. Данные, валидация, ошибки и версионирование 10
1.5. Безопасность API ИИ-сервисов 10
1.6. Тестирование, мониторинг и трассировка 11
1.7. Производительность: latency, throughput, p95/p99 11
1.8. Профильные траектории ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА и СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА 12

2. Практическая часть 14

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 1. Проектирование архитектуры API-сервиса инференса и потоков данных 14
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 2. Метрики и SLO для API инференса 15
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 3. Проектирование REST API и подготовка спецификации OpenAPI 16
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 4. Проектирование gRPC интерфейса и proto-описания 18
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 5. Реализация базового REST API 19
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 6. Реализация API по контракту 21
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 7. Интеграция модели в API и измерение базовой latency 22
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 8. Очередь задач и батчинг инференса 23
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 9. Конкурентная обработка запросов 24
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 10. Потокобезопасность общих ресурсов 25
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 11. Нагрузочное тестирование API 26
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 12. Профилирование и оптимизация API 27
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 13. Развертывание API-сервиса в целевом окружении 28
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 14. Оптимизация модели и вычислений 29
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 15. Настройка ускоренного инференса 30

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 16. Профилирование ускоренного контура 31
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 17. Контрактные и интеграционные тесты API 32
ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 18. Итоговый мини-проект: API-сервис для ИИ-задачи 33
Рекомендуемая литература и интернет-ресурсы 34
Приложения 35
Приложение А. Шаблон отчета по практической работе 35
Приложение Б. Чек-лист качества API ИИ-сервиса 35
Приложение В. Критерии оценивания итогового мини-проекта 36
Приложение Г. Рекомендуемая структура репозитория 36
Приложение Д. Минимальный шаблон README 37

ВВЕДЕНИЕ

Разработка систем искусственного интеллекта в прикладной среде редко ограничивается обучением модели. Для использования модели в цифровом продукте требуется инженерный контур: прием запроса, проверка данных, запуск инференса, постобработка результата, контроль ошибок, защита доступа, логирование, тестирование, мониторинг и измерение производительности. API-технологии становятся границей между моделью, данными, бизнес-процессами и пользовательскими интерфейсами.

В агропромышленном комплексе и финтехе API-сервисы применяются для онлайн-скоринга хозяйств, геоаналитики, прогнозирования урожайности, обработки обращений клиентов, поддержки операторов, анализа документов, мониторинга IoT-данных, построения цифровых двойников и интеграции ИИ-компонентов в CRM, BI, BPM, ERP и мобильные приложения. Поэтому практическая подготовка должна включать не только синтаксис фреймворка, но и полный цикл инженерной работы с сервисом.

Настоящее учебно-методическое пособие делает упор на практические задания по дисциплине «API-технологии» для двух направленностей подготовки 09.03.03 «Прикладная информатика»: «Программные решения для бизнеса» и «Системы искусственного интеллекта». Материал учитывает различие технологических акцентов: для ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА центральным является Python-стек прикладной разработки API и интеграции ИИ в бизнес-системы; для СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА – C++-стек, многопоточность, производительность, аппаратные платформы и ускорители.

Цель пособия – сформировать у обучающихся способность проектировать, реализовывать, документировать, тестировать, профилировать и сопровождать API-сервисы ИИ-систем, а также принимать аргументированные инженерные решения по выбору архитектуры, контракта, инструментов реализации и способов оптимизации.

После выполнения практических заданий студент должен уметь:

- формулировать требования к API ИИ-сервиса и переводить их в контракт взаимодействия;
- проектировать REST и gRPC интерфейсы для задач инференса и обработки данных;
- реализовывать API-сервис по выбранной профильной траектории;
- интегрировать модель машинного обучения или демонстрационный inference-компонент;

- обеспечивать валидацию данных, обработку ошибок, логирование и базовую защиту API;
- проводить нагрузочное тестирование, профилирование и оптимизацию;
- оформлять инженерный отчет и защищать выбранные архитектурные решения.

Пособие может использоваться как для аудиторных практических занятий, так и для самостоятельной работы. Практические задания образуют сквозной проект; при необходимости преподаватель может выбрать отдельные работы или объединить несколько заданий в командный мини-проект.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. API как инженерный контур ИИ-системы

API (Application Programming Interface) – это формализованная граница взаимодействия между программными компонентами. В ИИ-системах API выполняет роль управляемого входа к модели, данным и вычислительному контуру. Через API внешняя система передает признаки, изображения, документы, текстовые запросы или идентификаторы объектов, а в ответ получает прогноз, классификацию, извлеченные сущности, рекомендации или служебный статус.

ИИ-сервис должен быть не только точным с точки зрения модели, но и инженерно надежным. Ошибка в формате входных данных, рост задержки, недоступность модели, неконтролируемый batch, отсутствие версионирования или слабая защита доступа могут сделать решение непригодным для промышленного применения. Поэтому API следует рассматривать как часть жизненного цикла ИИ-продукта, а не как вспомогательную оболочку вокруг модели.

Типовой запрос к API инференса проходит следующие этапы: прием запроса; аутентификация и авторизация; валидация входных данных; преобразование признаков; запуск модели; постобработка результата; запись событий и метрик; формирование ответа. Для каждой стадии необходимо определить ответственность, допустимые ошибки, ограничения времени и способы наблюдаемости.

1.2. Архитектуры API-сервисов для инференса

API-сервис может быть реализован как монолитное приложение, отдельный микросервис инференса, часть распределенной системы с API gateway или элемент событийной архитектуры. Выбор зависит от характера нагрузки, требований к масштабированию, ограничений платформы, критичности данных и роли модели в бизнес-процессе.

Синхронный API подходит для интерактивных сценариев: оператор в CRM запрашивает оценку риска, пользователь портала получает рекомендацию, мобильное приложение отображает прогноз. Асинхронная обработка нужна, когда вычисления длительные или массовые: пакетный скоринг, обработка документов, анализ изображений, пересчет витрин данных. В таком случае API может возвращать идентификатор задачи, а результат становится доступен позднее.

В ИИ-сервисах часто выделяют несколько уровней: внешний API для потребителей, внутренний inference engine, сервис подготовки данных,

модельное хранилище, сервис мониторинга качества и инфраструктура логирования. Даже в учебном прототипе полезно отделять контракт API от логики модели, поскольку это повышает сопровождаемость и облегчает тестирование.

1.3. REST, OpenAPI, gRPC и Proto-контракты

REST API использует ресурсы, методы HTTP и коды статусов. Он удобен для веб-приложений, интеграции с BI/CRM, ручной отладки через curl или Postman и автоматического документирования через OpenAPI. Для задач инференса обычно проектируются endpoint-ы /health, /models, /predict, /batch/predict, /predictions/{id}.

OpenAPI-спецификация описывает доступные операции, схемы запроса и ответа, параметры, ошибки, требования безопасности и примеры использования. Она является контрактом между разработчиками сервиса, фронтендом, тестировщиками и внешними потребителями. Спецификация должна обновляться вместе с реализацией.

gRPC использует Protobuf-контракты и хорошо подходит для внутренних высокопроизводительных взаимодействий между сервисами. Он обеспечивает строгие типы, компактную сериализацию и поддержку streaming-сценариев. Нативный gRPC не может быть вызван напрямую из браузера, поскольку браузерные API не предоставляют необходимого контроля над HTTP/2-стримами. Для browser-facing сценариев применяют gRPC-Web – адаптер поверх HTTP/1.1 с проксированием через Envoy или аналог – либо gRPC JSON Transcoding, который автоматически генерирует REST/JSON-интерфейс из .proto-файла. В учебном пособии REST и gRPC рассматриваются как взаимодополняющие подходы: REST – для внешней интеграции и наглядности, gRPC – для внутренних сервисов и производительного инференса.

```
# Пример REST-запроса к сервису инференса
POST /v1/predict HTTP/1.1
Content-Type: application/json
X-Request-Id: demo-001

{
  "region": "Московская область",
  "area_ha": 128.5,
  "features": [12.4, 0.31, 45.0, 6.8]
}
```

1.4. Данные, валидация, ошибки и версионирование

Главная особенность API ИИ-сервиса состоит в зависимости качества ответа от качества входных данных. Недопустимые значения, измененный порядок признаков, отсутствующие поля или несовпадение единиц измерения могут привести к некорректному прогнозу без явной программной ошибки. Поэтому схема данных должна быть жестко описана, а входы – проверены до запуска модели.

Типовой ответ API должен включать не только прогноз, но и служебные поля: request_id, model_version, schema_version, timestamp, confidence или иной показатель уверенности, а также диагностические сообщения при ошибке. Это позволяет расследовать инциденты, воспроизводить результаты и проводить регрессионные проверки.

Версионирование необходимо вести как минимум на трех уровнях: версия API, версия схемы признаков и версия модели. Изменение модели без изменения API может быть допустимо, если формат входов и выходов сохраняется. Изменение схемы признаков требует более строгого контроля совместимости.

1.5. Безопасность API ИИ-сервисов

API ИИ-системы обрабатывает данные, которые могут иметь коммерческую, персональную или технологическую ценность. Минимальный уровень защиты включает аутентификацию, авторизацию, ограничение частоты запросов, контроль размера входных данных, журналирование действий, запрет вывода чувствительных данных в логи и разделение публичных и внутренних endpoint-ов.

Для ИИ-сервисов важны и специфические риски: попытки извлечения модели, подбор входов для перегрузки вычислительного контура, отправка вредоносных данных, prompt injection в LLM-сценариях, утечки обучающих данных, неконтролируемое использование дорогого ускорителя. В учебных проектах эти риски рассматриваются на базовом уровне через лимитирование, проверку входов и разграничение доступа.

Особого внимания требует формат pickle/joblib, широко используемый для сохранения моделей scikit-learn. Модуль pickle не является безопасным: при десериализации вредоносного файла возможно выполнение произвольного кода на сервере. Это означает, что загрузка pickle-файла модели из ненадёжного источника – например, из пользовательского запроса, внешнего хранилища без контроля целостности или скомпрометированного репозитория – может привести к полной компрометации сервиса. Рекомендуемые меры: хранить

файлы моделей только во внутреннем доверенном хранилище, проверять целостность через HMAC или хэш-сумму перед загрузкой, использовать ONNX как безопасную альтернативу pickle для production-окружения.

1.6. Тестирование, мониторинг и трассировка

Тестирование API ИИ-сервиса включает несколько уровней. Unit-тесты проверяют функции валидации, препроцессинга, постпроцессинга и отдельные бизнес-правила. Интеграционные тесты проверяют endpoint-ы и взаимодействие с моделью или хранилищем. Conformance-тесты (тесты соответствия спецификации) проверяют, что реализация endpoint-ов соответствует описанию OpenAPI или proto-контракту – корректные коды статусов, схемы полей, типы данных. Contract testing в классическом понимании (например, инструмент Pact) – это другой подход: он фиксирует ожидания конкретного потребителя API и проверяет, что поставщик этим ожиданиям удовлетворяет, независимо от остальных деталей реализации. В рамках данного пособия используется более простой вариант – проверка соответствия спецификации; полноценный consumer-driven contract testing рассматривается как тема для углублённого изучения. Нагрузочные тесты оценивают поведение сервиса под параллельными запросами.

Мониторинг должен показывать как технические метрики API, так и модельные характеристики. В рамках данного пособия основной акцент делается на latency, throughput, error rate, размер очереди, долю таймаутов, загрузку CPU/RAM/GPU и количество запросов по endpoint-ам. Для промышленного контура дополнительно нужны метрики качества модели, дрейфа данных и стабильности распределений признаков.

Трассировка позволяет связать события одного запроса в разных компонентах. Для этого используется request_id или trace_id, который проходит от входного API до модели, логов, хранилища и ответа клиенту. Без идентификатора запроса анализ производственных инцидентов становится существенно сложнее.

1.7. Производительность: latency, throughput, p95/p99

Производительность API ИИ-сервиса нельзя оценивать только средним временем ответа. Среднее значение скрывает длинный хвост задержек, который часто определяет качество пользовательского опыта. Поэтому используются percentile-метрики: p95 означает, что 95% запросов выполняются быстрее указанного времени, p99 – что быстрее выполняются 99% запросов.

Throughput показывает количество запросов, обработанных за единицу времени. Он должен анализироваться совместно с error rate и latency: высокий throughput не является преимуществом, если значительная доля запросов завершается ошибкой или таймаутом. Для ИИ-сервисов также важно учитывать стоимость запроса: использование GPU, объем памяти, размер batch, число копирований данных и накладные расходы сериализации.

Оптимизация должна начинаться с измерений. Типичная последовательность инженерной работы: сформулировать гипотезу, измерить baseline, провести профилирование, внести одно изменение, повторить измерения в тех же условиях, зафиксировать эффект и побочные последствия. Этот подход используется во всех практических заданиях, связанных с производительностью.

1.8. Профильные траектории ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА и СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Практические задания имеют общую методическую структуру, но разные технологические акценты. Для направленности «Программные решения для бизнеса» основное внимание уделяется разработке API-сервисов на Python, интеграции с библиотеками обработки данных и машинного обучения, документированию OpenAPI, тестированию pytest, контейнеризации и внедрению в прикладные бизнес-системы.

Для направленности «Системы искусственного интеллекта» акцент переносится на C++-реализацию API, многопоточность, потокобезопасность, рантаймы инференса, оптимизацию под платформы, использование GPU/FPGA или их программных аналогов, профилирование и анализ узких мест в высокопроизводительном контуре.

Компонент	Программные решения для бизнеса	Системы искусственного интеллекта
Основной язык	Python	C++17/20
API-стек	FastAPI, Flask, DRF, gRPC Python	Boost.Beast, Drogon/Crow, gRPC C++
ИИ-компонент	scikit-learn, PyTorch, ONNX Runtime	ONNX Runtime, OpenVINO, TensorRT
Конкурентность	Async/await, workers, очереди задач; многопоточность	thread pool, atomics, locks, lock-free подходы

Компонент	Программные решения для бизнеса	Системы искусственного интеллекта
	ограничена GIL (CPython ≤ 3.12); начиная с Python 3.13 доступна экспериментальная free-threaded сборка без GIL (PEP 703)	
Контроль качества	pytest, httpx, Locust/k6	GoogleTest, wrk/hey/k6, perf/VTune/Nsight
Итоговый результат	API-сервис для бизнес-сценария ИИ	высокопроизводительный API-сервис инференса

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

Практическая часть построена как последовательность из 18 заданий. Каждое задание может выполняться индивидуально или в малой команде. Рекомендуемая продолжительность одного задания – 2 академических часа аудиторной работы и самостоятельная доработка. Итогом является репозиторий учебного проекта, отчет и защита инженерных решений.

Единый сквозной сценарий пособия – API-сервис для ИИ-задачи в АПК или финтехе. Преподаватель может предложить единый кейс для группы либо разрешить студентам выбрать один из вариантов: агроскоринг, прогноз урожайности, классификация обращений, OCR документов, антифрод, геоаналитика, рекомендательная система, LLM-ассистент оператора.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 1.
Проектирование архитектуры API-сервиса инференса и потоков данных

Тема: Роль API в ИИ-системах, контуры инференса и обмен данными.

Цель работы: Сформировать архитектурное описание API-сервиса, выделить внешних потребителей, модель, данные, хранилища, компоненты наблюдаемости и ограничения по качеству.

Исходная ситуация: Сервис агроскоринга: по данным хозяйства, региону, площади посевов, погодным признакам и истории платежей возвращается оценка риска и рекомендации для оператора.

Порядок выполнения:

1. Определите потребителей сервиса: веб-портал, CRM, BI-панель, мобильное приложение или внутренний сервис банка.
2. Опишите контур данных: входной запрос, препроцессинг, обращение к модели, постпроцессинг, запись результата и возврат ответа.
3. Выделите синхронные и асинхронные операции: быстрый инференс, пакетная обработка, переобучение, обновление справочников.
4. Нарисуйте компонентную схему и укажите протоколы взаимодействия.
5. Зафиксируйте риски: некорректные входы, перегрузка, утечка данных, деградация модели, недоступность хранилища.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Описать Python-сервис на FastAPI: endpoint /predict, библиотека модели scikit-learn/PyTorch, хранилище результатов PostgreSQL или SQLite, логирование request_id.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Описать C++-сервис: REST/gRPC шлюз, модуль инференса ONNX Runtime/OpenVINO, пул потоков, очередь задач, метрики p95/p99 latency.

Результаты, представляемые к защите:

- архитектурная схема;
- таблица компонентов и интерфейсов;
- описание данных запроса и ответа;
- перечень рисков и ограничений.

Контрольные вопросы:

1. Почему API является границей ответственности ИИ-сервиса?
2. Какие операции лучше выполнять асинхронно?
3. Какие компоненты нужно добавить для промышленной эксплуатации?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 2.
Метрики и SLO для API инференса

Тема: Latency, throughput, error rate, SLO, лимитирование и backpressure.

Цель работы: Задать измеримые показатели качества API и подготовить план измерения производительности.

Исходная ситуация: Сервис должен обслуживать запросы от CRM и веб-портала. Для одиночного запроса требуется быстрый ответ, а для пакетной оценки хозяйств допускается очередь.

Порядок выполнения:

1. Сформулируйте пользовательские сценарии: одиночный прогноз, пакетная оценка, служебная проверка /health.
2. Определите SLI: p50/p95/p99 latency, throughput, error rate, availability, timeout rate, размер очереди.
3. Задайте SLO для каждого сценария и обоснуйте значения.
4. Определите способ лимитирования: rate limit по API-ключу, максимальный размер batch, максимальная длина очереди.
5. Подготовьте таблицу экспериментов: тип нагрузки, число клиентов, длительность теста, ожидаемый результат.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Подготовить конфигурацию Locust/k6 для FastAPI-сервиса и определить метрики, которые будут собираться на уровне приложения.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Подготовить сценарий wrk/hey/k6 для C++ сервиса и перечень метрик CPU, памяти, очередей и блокировок.

Результаты, представляемые к защите:

- таблица SLO;
- план нагрузочного тестирования;
- описание лимитов сервиса;
- перечень метрик мониторинга.

Контрольные вопросы:

1. Чем p95 отличается от среднего времени ответа?
2. Почему throughput нельзя оценивать без error rate?
3. Как backpressure защищает сервис от перегрузки?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 3.
Проектирование REST API и подготовка спецификации OpenAPI

Тема: REST, endpoint-ы, схемы данных, статусы HTTP, OpenAPI.

Цель работы: Разработать REST-контракт API инференса, пригодный для реализации, тестирования и документирования.

Исходная ситуация: API принимает признаки хозяйства и возвращает прогноз риска, доверительный интервал, версию модели и объясняющие факторы.

Порядок выполнения:

1. Определите ресурсы и операции: /health, /models, /predict, /batch/predict, /predictions/{id}.
2. Опишите схемы request/response, обязательные и необязательные поля.
3. Определите коды ошибок: 400, 401, 403, 404, 409, 422, 429, 500, 503.
4. Опишите версионирование API и модели: /v1, model_version, schema_version.
5. Составьте OpenAPI-фрагмент и проверьте его в валидаторе.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Сформировать OpenAPI через FastAPI и Pydantic-модели, добавить примеры запросов и ответов.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Сформировать OpenAPI вручную или через генератор/аннотации; определить соответствие C++ DTO структурам.

Пример фрагмента:

```
openapi: 3.0.3
info:
  title: AgroScore AI API
  version: 1.0.0
paths:
  /v1/predict:
    post:
      summary: Оценка риска хозяйства
      requestBody:
        required: true
      responses:
        '200':
          description: Прогноз выполнен
        '422':
          description: Ошибка валидации входных данных
```

Результаты, представляемые к защите:

- OpenAPI YAML/JSON;
- таблица endpoint-ов;
- примеры успешных и ошибочных ответов.

Контрольные вопросы:

1. Почему контракт должен фиксироваться до реализации?
2. Когда лучше использовать 422, а когда 400?
3. Зачем указывать версию модели в ответе?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 4.
Проектирование gRPC интерфейса и proto-описания

Тема: gRPC, Protobuf, streaming, batch, совместимость контрактов.

Цель работы: Разработать gRPC-контракт для высокопроизводительного инференса и сопоставить его с REST-контрактом.

Исходная ситуация: Пакетная оценка сотен хозяйств перед формированием кредитного отчета выполняется внутренним сервисом, которому важны строгие типы и компактная сериализация.

Порядок выполнения:

1. Выделите сервис и методы: Predict, BatchPredict, GetModelInfo, Health.
2. Опишите сообщения запроса и ответа в формате .proto.
3. Определите статусы ошибок и соответствие HTTP-кодам.
4. Продумайте совместимость: запрет удаления полей, резервирование номеров, optional-поля.
5. Сравните REST и gRPC по удобству интеграции, скорости, поддержке браузеров и внутреннему использованию.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Сгенерировать Python gRPC stubs или описать контракт как альтернативу REST для внутреннего обмена между сервисами.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Сгенерировать C++ stubs, описать серверную реализацию Predict и BatchPredict.

Пример фрагмента:

```
syntax = "proto3";
package agro.ai.v1;

service AgroScoreService {
  rpc Predict(PredictRequest) returns (PredictResponse);
  rpc BatchPredict(BatchPredictRequest) returns
(BatchPredictResponse);
}

message PredictRequest {
  string request_id = 1;
  string region = 2;
  double area_ha = 3;
  repeated double features = 4;
}

message PredictResponse {
```

```
  string request_id = 1;
  double risk_score = 2;
  string model_version = 3;
}
```

Результаты, представляемые к защите:

- файл predict.proto;
- таблица соответствия REST/gRPC;
- описание стратегии версионирования.

Контрольные вопросы:

1. Почему Protobuf требует аккуратного обращения с номерами полей?
2. Когда streaming лучше batch-запроса?
3. Почему gRPC часто используют во внутренних контурах?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 5.
Реализация базового REST API

Тема: HTTP endpoint-ы, валидация, единый формат ошибок, health-check.

Цель работы: Создать минимально работоспособный API-сервис с проверкой готовности и демонстрационным endpoint-ом инференса.

Исходная ситуация: Перед интеграцией реальной модели команда должна предоставить стабильный mock-сервис, чтобы фронтенд и тестировщики могли начать работу.

Порядок выполнения:

1. Создайте структуру проекта: app, models/schemas, service, tests, docs.
2. Реализуйте /health и /v1/predict.
3. Добавьте валидацию входных данных: диапазоны, обязательные поля, типы.
4. Верните единый формат ошибок: code, message, details, request_id.
5. Добавьте логирование входящего запроса без вывода персональных данных.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Реализовать сервис на FastAPI. Использовать Pydantic-схемы, uvicorn, автогенерацию OpenAPI.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Реализовать HTTP endpoint-ы на C++ с Boost.Beast/Drogon/Crow; подготовить CMake и DTO-структуры.

Пример фрагмента:

```
     from fastapi import FastAPI
from pydantic import BaseModel, Field

app = FastAPI(title="AgroScore AI API", version="1.0.0")

class PredictRequest(BaseModel):
    region: str = Field(min_length=2)
    area_ha: float = Field(gt=0)
    features: list[float]

@app.get("/health")
def health():
    return {"status": "ok"}

@app.post("/v1/predict")
def predict(req: PredictRequest):
    score = min(1.0, max(0.0, sum(req.features) / 100.0))
    return {"risk_score": score, "model_version": "demo-1"}
```

Вариант 1 – async def (для чисто вычислительного кода без блокировок)

```
     @app.post("/v1/predict") async def predict(req:
PredictRequest): score = min(1.0, max(0.0, sum(req.features) /
100.0)) return {"risk_score": score, "model_version": "demo-1"}
```

Вариант 2 – def с явным выносом тяжёлой работы (для реального инференса)

```
import asyncio from concurrent.futures import ThreadPoolExecutor
executor = ThreadPoolExecutor() def run_inference(features:
list[float]) -> float: # Здесь может быть блокирующий вызов модели
return min(1.0, max(0.0, sum(features) / 100.0))
@app.post("/v1/predict") async def predict(req: PredictRequest):
loop = asyncio.get_running_loop() score = await
loop.run_in_executor(executor, run_inference, req.features) return
{"risk_score": score, "model_version": "demo-1"}
```

Результаты, представляемые к защите:

- репозиторий с базовым API;
- инструкция запуска;
- скриншот/вывод Swagger UI или curl-запросов.

Контрольные вопросы:

1. Чем readiness отличается от liveness?
2. Почему ошибки должны иметь единый формат?
3. Какие поля нельзя писать в лог?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 6.
Реализация API по контракту

Тема: Контрактная реализация, таймауты, ограничения входных данных, request_id.

Цель работы: Привести реализацию API в соответствие с утвержденным контрактом и добавить эксплуатационные ограничения.

Исходная ситуация: Внутренний потребитель требует стабильного набора полей и предсказуемых ошибок; сервис должен отклонять слишком большие запросы до запуска модели.

Порядок выполнения:

1. Сверьте реализацию endpoint-ов со спецификацией OpenAPI или proto.
2. Добавьте request_id: прием из заголовка или генерация на стороне сервиса.
3. Добавьте таймаут инференса и ограничение размера batch.
4. Реализуйте обработку ошибок модели и недоступности зависимостей.
5. Подготовьте набор curl/grpcurl примеров для демонстрации.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Добавить middleware FastAPI для request_id, обработчики исключений и настройки timeout на уровне сервиса/клиента.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Реализовать таймауты в HTTP/gRPC сервере, прокинуть request_id через логирование и ответы.

Результаты, представляемые к защите:

- обновленная реализация API;
- набор примеров запросов;
- перечень отклоняемых сценариев.

Контрольные вопросы:

1. Зачем нужен request_id?
2. Какая разница между таймаутом клиента и таймаутом инференса?
3. Почему большие batch-запросы следует ограничивать?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 7.
Интеграция модели в API и измерение базовой latency

Тема: Инференс, препроцессинг, постпроцессинг, прогрев модели.

Цель работы: Подключить модель машинного обучения к API-сервису и измерить базовое время ответа.

Исходная ситуация: Демонстрационный скоринговый алгоритм заменяется обученной моделью. Необходимо обеспечить воспроизводимость входных признаков и корректность результата.

Порядок выполнения:

1. Подготовьте модель или заглушку модели, возвращающую устойчивый результат.
2. Опишите порядок преобразования входных данных в feature vector.
3. Добавьте прогрев модели при старте сервиса.
4. Сохраните версию модели и схему входных признаков.
5. Измерьте latency для 20–50 одиночных запросов и зафиксируйте baseline.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Подключить pickle/joblib-модель scikit-learn либо PyTorch/ONNX модель; вынести инференс в отдельный сервисный слой.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Подключить ONNX Runtime/OpenVINO; описать конфигурацию сессии. Системы искусственного интеллекта, входные/выходные тензоры и режим CPU.

Результаты, представляемые к защите:

- работающий endpoint с моделью;
- описание feature pipeline;
- таблица baseline latency.

Контрольные вопросы:

1. Почему feature pipeline должен быть версионирован?
2. Что такое прогрев модели?
3. Какие ошибки возникают при несоответствии схемы входов?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 8.
Очередь задач и батчинг инференса

Тема: Batching, queue, backpressure, сравнение latency/throughput.

Цель работы: Реализовать пакетную обработку запросов и сравнить влияние батчинга на задержку и пропускную способность.

Исходная ситуация: Во время сезонной кампании банк пакетно оценивает тысячи хозяйств. Одиночный инференс удобен для интерфейса, но пакетная обработка эффективнее для массового расчета.

Порядок выполнения:

1. Опишите два режима: single predict и batch predict.
2. Задайте максимальный размер batch и максимальное время ожидания накопления batch.
3. Реализуйте очередь задач или имитацию очереди.
4. Сравните одиночные и пакетные запросы по latency и throughput.
5. Сформулируйте вывод: когда батчинг полезен, а когда ухудшает пользовательский опыт.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать asyncio.Queue, Celery/RQ или простую очередь внутри приложения; оценить влияние workers.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Реализовать thread-safe очередь и batch executor; измерить потребление памяти.

Результаты, представляемые к защите:

- реализация batch endpoint;
- таблица экспериментов;
- выводы по настройке batch_size.

Контрольные вопросы:

1. Почему батчинг может увеличить p95 latency?
2. Что означает backpressure для клиента?
3. Какие данные нельзя смешивать в одном batch?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 9.
Конкурентная обработка запросов

Тема: Workers, async, thread pool, выделение вычислительных задач.

Цель работы: Настроить конкурентную обработку запросов и оценить влияние числа исполнителей на качество API.

Исходная ситуация: Сервис одновременно обслуживает интерактивных пользователей и внутренние интеграции. Нужно подобрать настройки, при которых рост нагрузки не приводит к лавинообразному росту ошибок.

Порядок выполнения:

1. Выберите модель конкурентности: async I/O, процессы, потоки, thread pool.
2. Определите, какие операции являются CPU-bound, а какие I/O-bound.
3. Настройте число workers/threads и повторите нагрузочный сценарий.
4. Сравните p95/p99, throughput, error rate и загрузку CPU.
5. Сделайте вывод о разумных настройках для учебного стенда.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Настроить uvicorn/gunicorn workers, async endpoint-ы и вынести CPU-bound операции в executor или очередь задач.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Реализовать или настроить thread pool, определить число потоков, сравнить режимы 1, 2, 4, 8 потоков.

Результаты, представляемые к защите:

- конфигурация конкурентности;
- график или таблица сравнения;
- вывод о bottleneck.

Контрольные вопросы:

1. Почему GIL важен для Python CPU-bound задач?

(Примечание (Python 3.13+). С Python 3.13 (PEP 703) доступна экспериментальная free-threaded сборка CPython, где GIL отключается через PYTHON_GIL=0; в Python 3.14 поддержка расширена. В перспективе это позволит эффективно параллелить CPU-bound задачи в потоках. Однако функция остаётся экспериментальной: большинство C-расширений (NumPy, PyTorch) требуют пересборки, для production не

рекомендуется. На учебном стенде по-прежнему актуально: CPU-bound задачи – в ProcessPoolExecutor или C-расширения, освобождающие GIL.)

2. Почему больше потоков не всегда быстрее?
3. Как отличить I/O-bound от CPU-bound операции?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 10.
Потокобезопасность общих ресурсов

Тема: Кэш, счетчики метрик, блокировки, атомарные операции, корректность под нагрузкой.

Цель работы: Обеспечить корректную работу общих структур данных при параллельных запросах.

Исходная ситуация: Сервис кэширует результаты для повторяющихся запросов и ведет счетчики ошибок. Под нагрузкой обнаруживаются некорректные значения метрик и редкие сбои.

Порядок выполнения:

1. Выделите общие ресурсы: cache, model session, counters, connection pool.
2. Опишите риски гонок и deadlock.
3. Добавьте синхронизацию или замените структуру на потокобезопасную.
4. Смоделируйте параллельную нагрузку и проверьте корректность результатов.
5. Сравните накладные расходы разных вариантов защиты.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Проверить общие словари/кэш, добавить lock при необходимости, оценить влияние многопроцессного режима на общий кэш.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать std::mutex, std::shared_mutex, std::atomic; проверить ThreadSanitizer при наличии инструмента.

Результаты, представляемые к защите:

- описание общих ресурсов;
- код синхронизации;
- результаты проверки под нагрузкой.

Контрольные вопросы:

1. Что такое data race?
2. Почему кэш может стать bottleneck?
3. Когда atomic достаточно, а когда нужна блокировка?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 11.
Нагрузочное тестирование API

Тема: Сценарии нагрузки, p95/p99, throughput, error rate, отчет по измерениям.

Цель работы: Провести управляемое нагрузочное тестирование и оформить инженерный отчет по результатам.

Исходная ситуация: Перед демонстрацией индустриальному партнеру нужно доказать, что сервис выдерживает учебную нагрузку и корректно деградирует при превышении лимитов.

Порядок выполнения:

1. Подготовьте сценарии: одиночный predict, batch predict, ошибочные запросы.
2. Запустите тест с нарастающим числом пользователей/соединений.
3. Соберите p50/p95/p99, throughput, error rate, CPU/RAM.
4. Определите точку насыщения сервиса.
5. Оформите результаты в таблицу и сформулируйте инженерные выводы.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать Locust, k6 или pytest-benchmark; сохранить конфигурацию теста в репозитории.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать wrk, hey, ab или k6; дополнить измерения системными метриками.

Результаты, представляемые к защите:

- файл сценария нагрузки;
- таблица результатов;
- вывод о пропускной способности.

Контрольные вопросы:

1. Почему тест должен включать негативные сценарии?
2. Что означает точка насыщения?
3. Почему тесты нужно повторять несколько раз?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 12.
Профилирование и оптимизация API

Тема: Bottleneck, сериализация, копирования, память, измерение эффекта «до/после».

Цель работы: Найти узкое место в API-сервисе и подтвердить эффект оптимизации измерениями.

Исходная ситуация: Нагрузочный тест показал, что сервис тратит много времени не на модель, а на сериализацию, преобразование данных или блокировки.

Порядок выполнения:

1. Выберите инструмент профилирования и измеряемый сценарий.
2. Постройте профиль: время в препроцессинге, инференсе, постпроцессинге, сериализации.
3. Выберите одну оптимизацию: кэширование, отказ от лишних копирований, изменение формата, предвыделение памяти.
4. Повторите нагрузочный тест в тех же условиях.
5. Сравните показатели “до/после” и оцените побочные эффекты.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать cProfile, py-spy, line_profiler; оптимизировать pandas/NumPy преобразования или JSON-сериализацию.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать perf/VTune/valgrind massif по возможности; оптимизировать копирования, аллокации, блокировки, сериализацию.

Результаты, представляемые к защите:

- фрагмент профиля;
- описание оптимизации;
- таблица сравнения до/после.

Контрольные вопросы:

1. Почему оптимизацию нельзя считать успешной без повторных измерений?
2. Какие операции часто оказываются дороже инференса?
3. Как оптимизация может ухудшить сопровождаемость?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 13.
Развертывание API-сервиса в целевом окружении

Тема: Управление зависимостями, контейнеризация, конфигурация, среда выполнения.

Цель работы: Подготовить сервис к воспроизводимому запуску на учебном или целевом стенде.

Исходная ситуация: Команда должна передать сервис преподавателю или ментору так, чтобы он запускался без ручного восстановления окружения.

Порядок выполнения:

1. Опишите переменные окружения и конфигурационные параметры.
2. Подготовьте requirements.txt/pyproject.toml или CMake/vcpkg/conan manifest.
3. Создайте Dockerfile или инструкцию запуска без контейнера.
4. Добавьте /health и проверку зависимостей при старте.
5. Проверьте запуск на чистой машине или в новом контейнере.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Собрать Docker-образ FastAPI-сервиса, настроить порт, переменные окружения и health-check.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Собрать C++ сервис через CMake, описать зависимости рантайма модели, подготовить Dockerfile или пакет запуска.

Результаты, представляемые к защите:

- Dockerfile или инструкция сборки;
- файл конфигурации;
- протокол проверки запуска.

Контрольные вопросы:

1. Почему важно отделять конфигурацию от кода?
2. Какие зависимости модели нужно фиксировать?
3. Как health-check помогает оркестратору?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 14.
Оптимизация модели и вычислений

Тема: Квантование, оптимизация графа, сравнение latency, памяти и качества.

Цель работы: Оценить влияние оптимизации модели на производительность API и качество результата.

Исходная ситуация: Для запуска на ограниченной платформе требуется снизить задержку и потребление памяти без существенной потери качества прогноза.

Порядок выполнения:

1. Зафиксируйте baseline: latency, RAM, размер модели, качество на контрольном наборе.
2. Выберите метод оптимизации: квантование, упрощение графа, уменьшение числа признаков, кэширование препроцессинга.
3. Примените оптимизацию и повторите измерения.
4. Оцените допустимость потери качества.
5. Сформулируйте рекомендацию для целевого сценария.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать ONNX/сконвертированную модель либо упрощенную scikit-learn модель; сравнить joblib/pickle/ONNX по времени ответа.

Важно: pickle и joblib не предназначены для загрузки файлов из ненадёжных источников – десериализация вредоносного файла позволяет выполнить произвольный код. В API-сервисах файл модели должен загружаться только из доверенного внутреннего хранилища с проверкой целостности. Для production предпочтителен формат ONNX.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать ONNX Runtime/OpenVINO/TensorRT по доступности; проверить режимы FP32/FP16/INT8 или имитировать ограниченную платформу.

Результаты, представляемые к защите:

- сравнительная таблица моделей;
- вывод о компромиссе скорость/качество;
- обновленная конфигурация инференса.

Контрольные вопросы:

1. Почему ускорение модели может снизить качество?
2. Как выбирать допустимый порог потери качества?
3. Что важнее для edge-сценария: RAM или latency?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 15.
Настройка ускоренного инференса

Тема: GPU/ускорители, провайдеры выполнения, FP16/INT8, сравнение с CPU.

Цель работы: Подключить ускоренный режим инференса или спроектировать план его подключения при отсутствии оборудования.

Исходная ситуация: Для анализа спутниковых снимков или потокового видео CPU-инференс недостаточен; требуется оценить выгоду от ускорителя.

Порядок выполнения:

1. Опишите целевой ускоритель: GPU, NPU, FPGA или облачный accelerator.
2. Определите, какие части пайплайна будут ускоряться: препроцессинг, модель, постпроцессинг.
3. Подключите провайдер выполнения или подготовьте проектную конфигурацию.
4. Сравните CPU и ускоренный режим на одинаковом наборе запросов.
5. Укажите накладные расходы передачи данных и ограничения оборудования.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Подключить GPU-провайдер PyTorch/ONNX Runtime при наличии оборудования; иначе подготовить конфигурационный план и CPU baseline.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Подготовить TensorRT/ONNX Runtime CUDA/OpenVINO провайдер; описать калибровку и проверки FP16/INT8.

Результаты, представляемые к защите:

- конфигурация ускоренного режима;
- сравнение CPU/GPU или проектный план;
- анализ ограничителей производительности.

Контрольные вопросы:

1. Почему GPU не всегда ускоряет одиночные запросы?
2. Какие накладные расходы возникают при передаче данных?
3. Почему важно проверять качество после FP16/INT8?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 16.
Профилирование ускоренного контура

Тема: Nsight/CPU-профилировщик, анализ bottleneck, настройка батчинга и передачи данных.

Цель работы: Проанализировать производительность ускоренного API-контура и определить, где теряется время.

Исходная ситуация: После подключения ускорителя latency снизилась меньше ожидаемого. Необходимо понять, что ограничивает сервис: CPU-preprocessing, сериализация, копирование или сама модель.

Порядок выполнения:

1. Разбейте latency на этапы: прием запроса, валидация, препроцессинг, копирование, инференс, постпроцессинг, ответ.
2. Соберите профиль доступным инструментом.
3. Проверьте влияние batch_size и числа параллельных запросов.
4. Выберите одну оптимизацию ускоренного контура.
5. Оформите вывод с указанием главного ограничителя.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать torch profiler, ONNX Runtime profiling, py-spy или логирование таймингов этапов.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать NVIDIA Nsight Systems/Compute при наличии GPU либо CPU-профилировщик для API-части.

Результаты, представляемые к защите:

- разложение latency по этапам;
- профиль или таблица таймингов;
- рекомендации по настройке.

Контрольные вопросы:

1. Почему ускоренная модель может простаивать?
2. Как batch_size связан с загрузкой GPU?
3. Что такое end-to-end latency?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 17.
Контрактные и интеграционные тесты API

Тема: Unit, integration, contract testing, негативные сценарии, актуализация спецификации.

Цель работы: Подготовить набор тестов, подтверждающий соответствие API контракту и устойчивость к ошибочным входам.

Исходная ситуация: После нескольких оптимизаций изменились схемы ответов. Нужно убедиться, что внешние потребители не сломаются.

Порядок выполнения:

1. Подготовьте unit-тесты для валидации, препроцессинга и форматирования результата.
2. Подготовьте integration-тесты для endpoint-ов.
3. Проверьте соответствие OpenAPI/proto контракту.
4. Добавьте негативные сценарии: пустые поля, неверные типы, слишком большой batch, timeout.
5. Обновите спецификацию по результатам тестов.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Использовать pytest и TestClient/httpx, добавить проверку JSON schema и snapshot ключевых ответов.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Использовать GoogleTest для модулей и внешний клиент k6/postman/pytest для интеграционных API-тестов.

Результаты, представляемые к защите:

- набор тестов;
- отчет о прохождении;
- актуализированная спецификация.

Контрольные вопросы:

1. Чем unit-тест отличается от conformance-теста? В чём дополнительно отличие conformance-теста от consumer-driven contract-теста (Pact)?
2. Почему негативные сценарии обязательны?
3. Как тесты помогают сохранять совместимость API?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ № 18.
Итоговый мини-проект: API-сервис для ИИ-задачи

Тема: Сборка сервиса, измерения, профилирование, отчет и защита решения.

Цель работы: Собрать завершенный прототип API-сервиса с моделью, документацией, тестами, измерениями и инженерным отчетом.

Исходная ситуация: Команда выбирает один отраслевой сценарий: агроскоринг, прогноз урожайности, классификация обращений, антифрод, OCR документов, геоаналитика или ассистент оператора.

Порядок выполнения:

1. Выберите кейс и сформулируйте пользовательский сценарий.
2. Опишите архитектуру, контракт и модель данных.
3. Реализуйте API и подключите модель или демонстрационный inference-компонент.
4. Добавьте тесты, документацию и сценарий нагрузки.
5. Проведите измерения, выполните одну оптимизацию и оформите отчет.

Траектория ПРОГРАММНЫЕ РЕШЕНИЯ ДЛЯ БИЗНЕСА: Итоговый проект на Python: FastAPI-сервис, ML-компонент, OpenAPI, pytest, нагрузочный сценарий и Docker.

Траектория СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: Итоговый проект на C++: REST/gRPC сервис, рантайм инференса, потокобезопасность, профилирование и отчет по производительности.

Результаты, представляемые к защите:

- Репозиторий;
- спецификация API;
- тесты;
- нагрузочный отчет;
- презентация защиты.

Контрольные вопросы:

1. Какая инженерная проблема была решена?
2. Как доказано качество API?
3. Какие ограничения остались у прототипа?

Критерии зачета: работа считается выполненной, если представлен воспроизводимый артефакт, результат соответствует заявленному контракту, студент объясняет принятые инженерные решения и демонстрирует проверку корректности.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА И ИНТЕРНЕТ-РЕСУРСЫ

1. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. University of California, Irvine.
2. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. O'Reilly Media.
3. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media.
4. Newman S. Building Microservices. O'Reilly Media.
5. Nygard M. Release It! Design and Deploy Production-Ready Software. Pragmatic Bookshelf.
6. Документация OpenAPI Specification: https://spec.openapis.org/
7. Документация FastAPI: https://fastapi.tiangolo.com/
8. Документация gRPC: https://grpc.io/docs/
9. Документация Protocol Buffers: https://protobuf.dev/
10. Документация ONNX Runtime: https://onnxruntime.ai/docs/
11. Документация OpenVINO: https://docs.openvino.ai/
12. Документация Docker: https://docs.docker.com/
13. Документация pytest: https://docs.pytest.org/
14. Документация k6: https://k6.io/docs/
15. Документация CMake: https://cmake.org/documentation/
16. Документация GoogleTest: https://google.github.io/googletest/

ПРИЛОЖЕНИЯ

Приложение А. Шаблон отчета по практической работе

1. Титульная часть: ФИО, группа, номер практической работы, название кейса.
2. Цель работы и выбранная профильная траектория.
3. Краткое описание архитектуры и используемых инструментов.
4. Описание API-контракта: endpoint-ы или методы gRPC, схемы данных, ошибки.
5. Описание реализации: структура проекта, ключевые модули, особенности интеграции модели.
6. Результаты тестирования: позитивные и негативные сценарии.
7. Результаты измерений: latency, throughput, error rate, условия эксперимента.
8. Выводы: достигнутые результаты, ограничения, предложения по улучшению.

Приложение Б. Чек-лист качества API ИИ-сервиса

Проверка	Да/нет	Комментарий
Есть описанный контракт OpenAPI или proto		
Все входные данные валидируются до инференса		
Ошибки возвращаются в едином формате		
Ответ содержит request_id и model_version		
Есть /health или аналогичная проверка состояния		
Есть unit и integration tests		
Проведено нагрузочное тестирование		
Проведено профилирование или разложение latency		
Есть README и инструкция запуска		
Чувствительные данные не выводятся в лог		

Приложение В. Критерии оценивания итогового мини-проекта

Критерий	Баллы	Описание
Архитектура и контракт	0–20	Корректная декомпозиция, понятный контракт, версионирование, схема ошибок.
Реализация API	0–25	Рабочие endpoint-ы, валидация, обработка ошибок, единый формат ответов.
Интеграция ИИ-компонента	0–15	Корректный препроцессинг/инференс/постпроцессинг, версия модели, контроль входов.
Тестирование и надежность	0–15	Unit/integration/contract tests, негативные сценарии, воспроизводимость запуска.
Производительность	0–15	Измерения latency/throughput/error rate, профилирование, подтвержденная оптимизация.
Документация и защита	0–10	README, спецификация, отчет, аргументированная защита инженерных решений.

Итоговая оценка может быть выставлена по сумме баллов: 86–100 – «отлично», 71–85 – «хорошо», 56–70 – «удовлетворительно», менее 56 – требуется доработка. Для зачета рекомендуется порог не ниже 56 баллов при обязательном наличии работающего API и отчета.

Приложение Г. Рекомендуемая структура репозитория

```
   agro-ai-api/
README.md
docs/
  openapi.yaml
  report.md
app/                  # Python trajectory
  main.py
  schemas.py
  service.py
cpp_service/           # C++ trajectory
  CMakeLists.txt
  src/
  include/
model/
  model.onnx
```

```
    model_card.md
  tests/
  load_tests/
    k6_predict.js
  docker/
    Dockerfile
```

Приложение Д. Минимальный шаблон README

```
     # API-сервис для ИИ-задачи

## Назначение
Кратко опишите кейс, потребителя API и модель.

## Запуск
Команды установки зависимостей и запуска сервиса.

## API
Ссылка на OpenAPI/proto, примеры запросов и ответов.

## Тестирование
Команды запуска unit, integration и load tests.

## Измерения
Таблица baseline и результата оптимизации.

## Ограничения
Что не реализовано в учебном прототипе и что нужно для
промышленного контура.
```