АО «АЛМАТИНСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет инженерных и информационных технологий

Кафедра компьютерной инженерии

ПРОЕКТИРОВАНИЕ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

ЛЕКЦИЯ № 3

Архитектура AI-систем: компоненты, слои и потоки данных

Теоретико-практический конспект

Составитель: Алтыбай А.

Кафедра компьютерной инженерии

Алматы, 2026

Аннотация и назначение лекции

Лекция 3 раскрывает тему «Архитектура AI-систем: компоненты, слои и потоки данных» в контексте разработки промышленной AI-системы. Материал опирается на тематический план дисциплины и соединяет определения, принципы проектирования, математические оценки, проектные артефакты, практический код и аргументированный анализ рисков. Вопросы построены вокруг решений, которые предстоит принимать инженеру, а не только вокруг названий алгоритмов. [1, 2]

План занятия

1. Границы AI-подсистемы
2. Контейнеры и компоненты
3. Монолит и микросервисы
4. Синхронные и асинхронные запросы
5. Пакетный и потоковый режим
6. Хранилище признаков и артефактов
7. Контракт данных
8. Отказоустойчивость и деградация
9. Практические сценарии, контроль качества и проектные задания.

Ожидаемые результаты обучения

По завершении студент способен:

1. объяснить и аргументированно применить принцип «Границы AI-подсистемы» к инженерному решению;
2. объяснить и аргументированно применить принцип «Контейнеры и компоненты» к инженерному решению;

3. объяснить и аргументированно применить принцип «Монолит и микросервисы» к инженерному решению;
4. объяснить и аргументированно применить принцип «Синхронные и асинхронные запросы» к инженерному решению;
5. объяснить и аргументированно применить принцип «Пакетный и потоковый режим» к инженерному решению;
6. объяснить и аргументированно применить принцип «Хранилище признаков и артефактов» к инженерному решению;
7. реализовать упрощённый проверяемый прототип и сформулировать критерии его приемки.
8. указать угрозы достоверности результатов и предложить проверки до эксплуатации.

Сквозной кейс. В течение лекции рассматриваем платформу обработки обращений. Существенные условия: рост пиковой нагрузки, вызовы внешней модели. Основная проверяемая характеристика: доступность и p95 latency. Числа в расчётных примерах учебные и не являются результатами реальных измерений.

1. Базовые теоретические положения из конспекта курса

Ниже представлены исходные положения учебно-методического конспекта, связанные с данной темой. Следующие разделы раскрывают их с инженерными деталями и проверяемыми примерами. [1, 2] Цель лекции. познакомить с компонентами и архитектурными шаблонами AI-систем, научить проектировать потоки данных и взаимодействие training/inference контуров.

Ключевые понятия. архитектура, data flow, training pipeline, inference pipeline, model serving, batch inference, online inference, monolith, microservices, event-driven architecture.

1.1 Архитектурное мышление

Архитектура AI-системы определяет границы компонентов и потоки информации между ними. Хорошая архитектура делает систему изменяемой: модель можно переобучить без переписывания приложения, источник данных можно заменить без изменения бизнес-логики, а мониторинг подключается независимо от конкретного алгоритма.

Архитектуру удобно описывать на нескольких уровнях: контекст системы, контейнеры/сервисы, компоненты и критические последовательности взаимодействий. На раннем этапе важнее логические границы, чем конкретные названия облачных сервисов.

1.2 Training и inference как два разных контура

Training pipeline получает исторические данные, формирует обучающую выборку, вычисляет признаки, обучает модель и публикует артефакт. Inference pipeline получает текущий объект, применяет совместимые преобразования и вызывает выбранную версию модели.

Критическая проблема — training-serving skew: когда логика подготовки данных при обучении отличается от логики в production. Один из способов снизить риск — использовать общий код preprocessing или feature store с едиными определениями признаков.

1.3 Batch, online и streaming inference

Batch inference подходит, когда прогноз можно вычислить заранее для большого набора объектов: например, ночной пересчет риска клиентов. Online inference применяется, когда результат нужен непосредственно при запросе. Streaming inference используется для непрерывного потока событий и часто связан с event-driven архитектурой.

Выбор режима влияет на требования к latency, устойчивости и инфраструктуре. Batch обычно проще и дешевле; online требует масштабирования сервиса, кэширования и отказоустойчивости; streaming требует управ-

ления порядком событий и состоянием.

1.4 Монолит, микросервисы и событийная архитектура

Монолит удобен для небольшого проекта: меньше сетевых вызовов и операционной сложности. Микросервисы дают независимое масштабирование компонентов, но увеличивают требования к наблюдаемости, версиям API и распределенным отказам. Событийная архитектура полезна, когда действия должны запускаться реактивно по факту появления событий.

Нет универсально «лучшей» архитектуры. Основной принцип — выбирать минимальную сложность, которая удовлетворяет требованиям по масштабу, скорости изменений и надежности.

1.5 Архитектурные документы

Для AI-проекта полезны: контекстная диаграмма, схема потоков данных, описание API, модель данных, decision log, список зависимостей, требования к ресурсам и deployment diagram. Документация должна помогать отвечать на вопрос «что изменится, если мы заменим модель, источник данных или способ развертывания?».

1.6 Выводы

- Training и inference — разные контуры, но должны использовать согласованную логику данных.
- Режим inference выбирается по требованиям к свежести и latency.
- Архитектурный стиль следует выбирать по контексту, а не по моде.
- Документирование границ и потоков данных снижает стоимость будущих изменений.

1.7 Контрольные вопросы

- ▷ Что такое training-serving skew?
- ▷ Когда batch inference предпочтительнее online inference?
- ▷ Какие преимущества и издержки у микросервисной архитектуры?
- ▷ Для чего используется event-driven подход?
- ▷ Какие диаграммы полезно подготовить при проектировании AI-системы?

2. Систематическое изложение темы и инженерные решения

2.1 Границы AI-подсистемы

Начинать проектирование нужно с контекстной диаграммы: кто поставляет данные, кто потребляет решения, кто поддерживает контракт. Границы защиты совпадают не всегда с границами процессов. [2]

Проверяемый пример. Разделите систему обращений на UI, API, очередь событий, inference worker, хранилище и журнал.

Вопрос для обсуждения. Как проверить выполнение такого требования: Разделите систему обращений на UI, API, очередь событий, inference worker, хранилище и журнал.

2.2 Контейнеры и компоненты

Логические компоненты описывают обязанности; контейнеры обозначают единицы развертывания. Чрезмерное дробление увеличивает число сетевых вызовов и отказов. [3]

Проверяемый пример. Сравните одно FastAPI-приложение с отдельными сервисами preprocessing и inference.

Вопрос для обсуждения. Как проверить выполнение такого требования: Сравните одно FastAPI-приложение с отдельными сервисами preprocessing и inference.

2.3 Монолит и микросервисы

Монолит проще согласованно тестировать; микросервисы позволяют независимо масштабировать сложные этапы. Выбор оправдывается нагрузкой, командой и частотой изменений. [4]

Проверяемый пример. Опишите при каких условиях классификатор изображений нужно выделить из основного приложения.

Вопрос для обсуждения. Как проверить выполнение такого требования: Опишите при каких условиях классификатор изображений нужно выделить из основного приложения.

2.4 Синхронные и асинхронные запросы

Синхронный API отвечает сразу, а асинхронный принимает задание и возвращает идентификатор для последующего статуса. Время работы модели и UX определяют интерфейс. [5]

Проверяемый пример. Для анализа видео длиной 10 минут выберите очередь задач и вебхук, а не блокирующий HTTP-запрос.

Вопрос для обсуждения. Как проверить выполнение такого требования: Для анализа видео длиной 10 минут выберите очередь задач и вебхук, а не блокирующий HTTP-запрос.

2.5 Пакетный и потоковый режим

Batch эффективен для регулярной обработки большого объема; online нужен при принятии решения в реальном времени. Гибрид использует предвычисленные признаки и быстрый скоринг. [6]

Проверяемый пример. Составьте архитектуру ночного скоринга клиентов и интерактивного просмотра результатов утром.

Вопрос для обсуждения. Как проверить выполнение такого требования: Составьте архитектуру ночного скоринга клиентов и интерактивного просмотра результатов утром.

2.6 Хранилище признаков и артефактов

Различают данные-источники, feature store, model registry и хранилище логов. Их назначение разное, объединение без контрактов приводит к путанице версий. [1]

Проверяемый пример. Покажите, где сохраняются immutable model artifact и mutable пользовательская запись.

Вопрос для обсуждения. Как проверить выполнение такого требования: Покажите, где сохраняются immutable model artifact и mutable пользовательская запись.

2.7 Контракт данных

Схема содержит типы, допустимые диапазоны, обязательные поля и семантику времени. Нарушение схемы должно фиксироваться до предсказания. [2]

Проверяемый пример. Разработайте JSON Schema или Pydantic-модель входного события.

Вопрос для обсуждения. Как проверить выполнение такого требования: Разработайте JSON Schema или Pydantic-модель входного события.

2.8 Отказоустойчивость и деградация

Timeout, повторная попытка, circuit breaker, кеш и fallback улучшают стабильность, но неверные повторные вызовы способны дублировать действия. [3]

> **Проверяемый пример.** Для API с ошибками 503 определите политику retry только для идемпотентных запросов.

Вопрос для обсуждения. Как проверить выполнение такого требования: Для API с ошибками 503 определите политику retry только для идемпотентных запросов.

2.9 Безопасность по границам

Аутентификация, авторизация, сетевые политики и секреты применяются на границах доверия. Модель не должна читать любые файловые пути из запроса пользователя. [4]

> **Проверяемый пример.** Постройте threat model для загрузки пользовательского PDF в RAG-систему.

Вопрос для обсуждения. Как проверить выполнение такого требования: Постройте threat model для загрузки пользовательского PDF в RAG-систему.

2.10 Архитектурные компромиссы

Изменение архитектуры нужно оценивать по метрикам latency, cost, availability и maintainability, а не по модности технологии. [5]

Проверяемый пример. Составьте ADR для выбора монолита на первом релизе и критерии будущего выделения inference worker.

Вопрос для обсуждения. Как проверить выполнение такого требования: Составьте ADR для выбора монолита на первом релизе и критерии будущего выделения inference worker.

2.11 Observability by design

Идентификатор запроса связывает события между API, очередью, моделью и пользовательской реакцией. Метрики должны отражать ошибки каждого компонента. [6]

Проверяемый пример. Проследите запрос через correlation ID, построив временную диаграмму событий.

Вопрос для обсуждения. Как проверить выполнение такого требования: Проследите запрос через correlation ID, построив временную диаграмму событий.

2.12 Версионирование API и модели

Версия интерфейса не совпадает с версией весов. Совместимость определяется схемой и семантикой выходов, а не только URL сервиса. [1]

Проверяемый пример. Обоснуйте, в каких случаях достаточно изменить model_version, а когда потребуется /v2/predict.

Вопрос для обсуждения. Как проверить выполнение такого требования: Обоснуйте, в каких случаях достаточно изменить model_version, а когда потребуется /v2/predict.

3. Математические инструменты и количественные критерии

Для перехода от неформального решения к проверяемой постановке введём следующие величины.

$$T_{\text{end}} = T_{\text{network}} + T_{\text{validation}} + T_{\text{features}} + T_{\text{model}} + T_{\text{post}} \tag{1}$$

$$L = \lambda W \tag{2}$$

Последнее соотношение является законом Литтла при стационарности: среднее число запросов в системе L равно интенсивности λ на среднее время пребывания W.

При выборе числового критерия нужно отличать обучающую функцию потерь, продуктовую ценность и эксплуатационные ограничения. Первую можно оптимизировать дифференцированием, вторая определяется процессом принятия решения, а третьи проверяются нагрузочными испытаниями, политиками безопасности и мониторингом.

3.1 Пошаговый расчёт на учебных данных

Рассмотрим две конфигурации одной системы. Вариант A имеет условную оценку качества $q_A = 0.86$ и задержку $t_A = 90$ мс; вариант B — $q_B = 0.89$ и $t_B = 280$ мс. Предположим, что требование заказчика задаёт $q \geq 0.85$ и $p95 \leq 150$ мс. Тогда A удовлетворяет обоим условиям, тогда как B нарушает второе. Следовательно, выбор B только потому, что его offline-метрика выше, не является обоснованным. Числа иллюстративны; реальный вывод требует измерений при одинаковой нагрузке и одинаковом наборе данных.

$$\begin{aligned} \text{accept}(A) &= \mathbf{1}\{0.86 \geq 0.85 \wedge 90 \leq 150\} = 1, \\ \text{accept}(B) &= \mathbf{1}\{0.89 \geq 0.85 \wedge 280 \leq 150\} = 0. \end{aligned}$$

3.2 Как интерпретировать число, а не только вычислить его

Сначала следует проверить единицы измерения, состав выборки и смысл агрегирования. Например, p95 на спокойном стенде нельзя переносить на рабочую нагрузку с иным числом одновременных запросов. Затем исследуют чувствительность к изменениям входных данных и неопределённость оценок. Наконец, фиксируют последствия порога: какую ошибку он предотвращает, а какую может увеличить. Этот подход делает количественный критерий частью инженерной аргументации, а не украшением отчёта. [1, 2]

4. Практическое программирование на Python

Пример предназначен для демонстрации ключевого инженерного принципа. Он не претендует на обработку реального production-трафика. Запускать код следует в изолированном окружении и фиксировать версии библиотек, если они используются. Вывод зависит от указанных входных данных; для стохастических алгоритмов он может меняться.

4.1 Основной воспроизводимый пример

```python
from pydantic import BaseModel, Field
class Request(BaseModel):
    text: str = Field(min_length=1, max_length=5000)
    language: str = Field(pattern=r"^(ru|kk|en)$")

class Response(BaseModel):
    label: str
    score: float = Field(ge=0, le=1)
    model_version: str

# Contract tests should check invalid lengths and unknown languages.
example = Request(text="Please help", language="en")
print(example.model_dump())
```

Проверка результатов. Сначала убедитесь, что код завершился без исключений; проверьте формы массивов, схему результата и отсутствие недопустимых значений. Если используются внешние данные, включите в отчёт способ их получения и идентификатор версии. Измените один входной параметр и объясните, почему наблюдаемый результат изменился или остался прежним.

4.2 Компактная проверка контракта или метрики

```python
latencies_ms = {"network": 23, "validation": 5,
                "features": 16, "model": 57, "post": 8}
print("total latency ms:", sum(latencies_ms.values()))
```

При самостоятельном повторении предложите минимум один негативный тест: некорректное значение, пустой вход, несовместимую версию схемы либо нарушение SLO. Переход от успешного демонстрационного запуска к надежной системе требует именно таких проверок.

5. Разбор трёх инженерных ситуаций

5.1 Ситуация 1: Границы AI-подсистемы

Заказчик использует платформу обработки обращений. Новое требование заставляет рассмотреть принцип «Границы AI-подсистемы». Основная неопределённость состоит в том, как одновременно обеспечить доступность и p95 latency, не допустить вызовы внешней модели и сохранить возможность изменения модели.

Постановка. Начинать проектирование нужно с контекстной диаграммы: кто поставляет данные, кто потребляет решения, кто поддерживает контракт. Границы защиты совпадают не всегда с границами процессов.

Действие инженера. Разделите систему обращений на UI, API, очередь событий, inference worker, хранилище и журнал.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

5.2 Ситуация 2: Хранилище признаков и артефактов

Заказчик использует платформу обработки обращений. Новое требование заставляет рассмотреть принцип «Хранилище признаков и артефактов». Основная неопределённость состоит в том, как одновременно обеспечить доступность и p95 latency, не допустить вызовы внешней модели и сохранить возможность изменения модели.

Постановка. Различают данные-источники, feature store, model registry и хранилище логов. Их назначение разное, объединение без контрактов приводит к путанице версий.

Действие инженера. Покажите, где сохраняются immutable model artifact и mutable пользовательская запись.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

5.3 Ситуация 3: Observability by design

Заказчик использует платформу обработки обращений. Новое требование заставляет рассмотреть принцип «Observability by design». Основная неопределённость состоит в том, как одновременно обеспечить доступность и p95 latency, не допустить вызовы внешней модели и сохранить возможность изменения модели.

Постановка. Идентификатор запроса связывает события между API, очередью, моделью и пользовательской реакцией. Метрики должны отражать ошибки каждого компонента.

Действие инженера. Проследите запрос через correlation ID, построив временную диаграмму событий.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

6. Самостоятельная работа: индивидуальные проектные задания

Для каждого задания студент создаёт отдельный небольшой артефакт: схему, набор тестов, код либо таблицу сравнения, прикладывает наблюдаемый результат и объясняет ограничения.

1. **Границы AI-подсистемы.** Разделите систему обращений на UI, API, очередь событий, inference worker, хранилище и журнал. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

2. **Контейнеры и компоненты.** Сравните одно FastAPI-приложение с отдельными сервисами preprocessing и inference. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

3. **Монолит и микросервисы.** Опишите при каких условиях классификатор изображений нужно выделить из основного приложения. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

4. **Синхронные и асинхронные запросы.** Для анализа видео длиной 10 минут выберите очередь задач и вебхук, а не блокирующий HTTP-запрос. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

5. **Пакетный и потоковый режим.** Составьте архитектуру ночного скоринга клиентов и интерактивного просмотра результатов утром. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

6. **Хранилище признаков и артефактов.** Покажите, где сохраняются immutable model artifact и mutable пользовательская запись. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

7. **Контракт данных.** Разработайте JSON Schema или Pydantic-модель входного события. Сформулируйте проверяемое требование, продемон-

стрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

8. **Отказоустойчивость и деградация.** Для API с ошибками 503 определите политику retry только для идемпотентных запросов. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

9. **Безопасность по границам.** Постройте threat model для загрузки пользовательского PDF в RAG-систему. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

10. **Архитектурные компромиссы.** Составьте ADR для выбора монолита на первом релизе и критерии будущего выделения inference worker. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

11. **Observability by design.** Проследите запрос через correlation ID, построив временную диаграмму событий. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

12. **Версионирование API и модели.** Обоснуйте, в каких случаях достаточно изменить model_version, а когда потребуется /v2/predict. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

7. Вопросы для текущего и итогового контроля

1. В чём назначение и ограничения принципа «Границы AI-подсистемы»?

2. Как проверить на практике утверждение: «Разделите систему обращений на UI, API, очередь событий, inference worker, хранилище и журнал.»?

3. В чём назначение и ограничения принципа «Контейнеры и компоненты»?

4. Как проверить на практике утверждение: «Сравните одно FastAPI-приложение с отдельными сервисами preprocessing и inference.»?
5. В чём назначение и ограничения принципа «Монолит и микросервисы»?
6. Как проверить на практике утверждение: «Опишите при каких условиях классификатор изображений нужно выделить из основного приложения.»?
7. В чём назначение и ограничения принципа «Синхронные и асинхронные запросы»?
8. Как проверить на практике утверждение: «Для анализа видео длиной 10 минут выберите очередь задач и вебхук, а не блокирующий HTTP-запрос.»?
9. В чём назначение и ограничения принципа «Пакетный и потоковый режим»?
10. Как проверить на практике утверждение: «Составьте архитектуру ночного скоринга клиентов и интерактивного просмотра результатов утром.»?
11. В чём назначение и ограничения принципа «Хранилище признаков и артефактов»?
12. Как проверить на практике утверждение: «Покажите, где сохраняются immutable model artifact и mutable пользовательская запись.»?
13. В чём назначение и ограничения принципа «Контракт данных»?
14. Как проверить на практике утверждение: «Разработайте JSON Schema или Pydantic-модель входного события.»?
15. В чём назначение и ограничения принципа «Отказоустойчивость и деградация»?
16. Как проверить на практике утверждение: «Для API с ошибками 503 определите политику retry только для идемпотентных запросов.»?
17. В чём назначение и ограничения принципа «Безопасность по границам»?

18. Как проверить на практике утверждение: «Постройте threat model для загрузки пользовательского PDF в RAG-систему.»?
19. В чём назначение и ограничения принципа «Архитектурные компромиссы»?
20. Как проверить на практике утверждение: «Составьте ADR для выбора монолита на первом релизе и критерии будущего выделения inference worker.»?
21. В чём назначение и ограничения принципа «Observability by design»?
22. Как проверить на практике утверждение: «Проследите запрос через correlation ID, построив временную диаграмму событий.»?
23. В чём назначение и ограничения принципа «Версионирование API и модели»?
24. Как проверить на практике утверждение: «Обоснуйте, в каких случаях достаточно изменить model_version, а когда потребуется /v2/predict.»?
25. Почему высокое offline-качество не гарантирует надежность промышленной AI-системы?
26. Какие артефакты нужны для воспроизводимости и анализа инцидента?

8. Глоссарий ключевых понятий

Границы AI-подсистемы
Начинать проектирование нужно с контекстной диаграммы: кто поставляет данные, кто потребляет решения, кто поддерживает контракт.

Контейнеры и компоненты
Логические компоненты описывают обязанности; контейнеры обозначают единицы развертывания.

Монолит и микросервисы
Монолит проще согласованно тестировать; микросервисы позволяют независимо масштабировать сложные этапы.

Синхронные и асинхронные запросы
Синхронный API отвечает сразу, а асинхронный принимает задание и возвращает идентификатор для последующего статуса.

Пакетный и потоковый режим
Batch эффективен для регулярной обработки большого объема; online нужен при принятии решения в реальном времени.

Хранилище признаков и артефактов
Различают данные-источники, feature store, model registry и хранилище логов.

Контракт данных
Схема содержит типы, допустимые диапазоны, обязательные поля и семантику времени.

Отказоустойчивость и деградация
Timeout, повторная попытка, circuit breaker, кеш и fallback улучшают стабильность, но неверные повторные вызовы способны дублировать действия.

Безопасность по границам
Аутентификация, авторизация, сетевые политики и секреты применяются на границах доверия.

Архитектурные компромиссы
Изменение архитектуры нужно оценивать по метрикам latency, cost,

availability и maintainability, а не по модности технологии.

Observability by design

Идентификатор запроса связывает события между API, очередью, моделью и пользовательской реакцией.

Версионирование API и модели

Версия интерфейса не совпадает с версией весов.

9. Итоги и связь со следующей темой

Представленные методы имеют смысл как части целостного управляемого жизненного цикла. Ключевой результат инженерного проектирования состоит не в максимальном числе компонентов или сложности модели, а в убедительной связи требований, проектных решений, измерений и механизмов сопровождения. Дальнейшее изучение курса развивает эти связи на следующем этапе жизненного цикла.

Список литературы

[1] Huyen C. Designing Machine Learning Systems. O'Reilly Media, 2022. https://www.oreilly.com/library/view/designing-machine-learning/9781098107956/.

[2] Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. https://dataintensive.net/.

[3] Sculley D. et al. Hidden Technical Debt in Machine Learning Systems. NeurIPS, 2015. https://research.google/pubs/hidden-technical-debt-in-machine-learning-systems/.

[4] Google Developers. Rules of Machine Learning: Best Practices for ML Engineering. https://developers.google.com/machine-learning/guides/rules-of-ml.

[5] FastAPI. Official documentation. https://fastapi.tiangolo.com/.

[6] Docker Inc. Docker documentation. https://docs.docker.com/.