Отчёт

Пункт 5. Анализ тенденций (arXiv): AI + Kubernetes + контейнеризация (2024–2025)

Введение

Kubernetes за последние годы стал "базовой платформой" для развёртывания и эксплуатации распределённых приложений, включая сервисы машинного обучения. На практике это означает, что многие вопросы, которые раньше решались на уровне инфраструктуры или вручную , всё чаще формулируются как задачи управления динамической системой. В 2024–2025 в исследованиях заметно усиливается фокус на том, чтобы делать оркестрацию более "умной": использовать прогнозирование, обучение с подкреплением и многофакторные модели принятия решений вместо фиксированных правил.

Ниже приведён обзор 5 работ с arXiv. Затем — выделение основных тенденций и выводы, как это соотносится с учебным проектом.

Подбор статей (5+)

№	Статья	Год	Почему релевантно теме "AI + Kubernetes/оркестрация"
1	**Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques…** — arXiv:2403.07905	2024	DL+RL для автоматизации task scheduling в больших облачных системах/кластерной оркестрации. [1]
2	**A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling in Cloud Computing Environments** — arXiv:2408.02926	2024	RL-управление расписанием workflow в облаке; в тексте упоминается Argo на Kubernetes как контекст выполнения. [2]
3	**KIS-S: A GPU-Aware Kubernetes Inference Simulator with RL-Based Auto-Scaling** — arXiv:2507.07932	2025	GPU-aware симулятор + RL-автоскейлер (PPO), интеграция с Prometheus и Kubernetes API. [1]

№	Статья	Год	Почему релевантно теме "AI + Kubernetes/оркестрация"
4	**An SLO Driven and Cost-Aware Autoscaling Framework for …** — arXiv:2512.23415	2025	Переосмысление autoscaling в Kubernetes как SLO+cost control problem, "AI-Ops-driven" подход с совместимостью с Kubernetes-примитивами. [3][4]
5	**Key Considerations for Auto-Scaling** — arXiv:2510.02585	2025	Обзор/сравнение подходов к autoscaling, акцент на том, что нужны и низкоуровневые метрики (CPU/mem), и высокоуровневые (latency/SLO). [5][6]

Обзор и ключевые идеи работ

1) arXiv:2403.07905 — DL + RL как следующий шаг планировщика

В arXiv:2403.07905 авторы описывают подход, который можно понимать как "планировщик нового поколения": deep learning используется для мониторинга и предсказания параметров системы, а reinforcement learning — для выбора решения по размещению/расписанию задач в реальном времени.[1]

Важно, что мотивация в работе типичная для больших распределённых систем: сложность и динамичность нагрузки не позволяют полагаться на один раз настроенные правила. Поэтому появляется идея контурного управления: наблюдаем состояние → принимаем решение → измеряем результат → обновляем стратегию.[1]

Что здесь можно вынести как тенденцию: Kubernetes (и системы вокруг него) всё чаще рассматриваются не только как набор API и контроллеров, а как среда, где можно внедрять более обучаемые политики планирования и управления ресурсами.[1]

2) arXiv:2408.02926 — RL для scheduling workflow и оптимизации стоимости

В arXiv:2408.02926 внимание смещено с отдельных pod’ов/контейнеров на выполнение workflow: наборов задач с зависимостями, типичных для data-pipeline и ML-pipeline.[2]

Авторы используют DRL-подход для оптимизации стоимости выполнения workflow при наличии разных типов ресурсов (включая spot/on-demand). Такая постановка важна, потому что в проде инженер обычно выбирает компромисс между (а) стоимостью, (б) временем выполнения, (в) риском прерываний/нестабильности, и этот компромисс плохо описывается простыми правилами.[2]

Для темы "AI + Kubernetes" это показывает общий тренд: оркестрация всё чаще происходит на более высоком уровне абстракции (workflow/пайплайны), а не только на уровне "поднять pod и отдать трафик".[2]

3) arXiv:2507.07932 — почему HPA недостаточно для AI-инференса (особенно GPU)

Статья arXiv:2507.07932 актуальна для Kubernetes-проектов с ИИ, потому что говорит о практической применении: стандартный HPA по CPU/memory для инференса на GPU часто плохо работает, особенно при всплесках трафика и при требованиях по latency.[1]

Авторы предлагают KIS-S: связку симулятора Kubernetes-инференса (KISim) и RL-автоскейлера (KIScaler), который обучается в симуляции, а затем управляет репликами через Kubernetes API, используя метрики из Prometheus.[1]

Ключевой практический смысл: чтобы реально улучшать autoscaling под AI-нагрузки, нужны (1) богатые метрики (вплоть до GPU-уровня и latency-уровня), (2) возможность безопасно учить политику без риска для прода, (3) интеграция с существующими инструментами наблюдаемости (Prometheus) и механизмами оркестрации (Kubernetes API).[1]

4) arXiv:2512.23415 — autoscaling как SLO+cost control problem

В arXiv:2512.23415 autoscaling явно формулируется как задача управления с ограничениями: держать SLO и одновременно не уходить в неконтролируемые расходы, причём авторы подчёркивают совместимость с "native Kubernetes primitives".[3][4]

Даже если не углубляться в детали реализации, важна сама рамка: вместо "CPU>50% → прибавь реплику" предлагается multi-signal подход с guardrails, и это укладывается в современную линию "AI-Ops" (объединение телеметрии, прогнозирования и управляемой автоматизации).[3]

5) arXiv:2510.02585 — метрики и сигналы: почему CPU недостаточно

arXiv:2510.02585 в явном виде подчёркивает, что для эффективного autoscaling нужны как низкоуровневые системные метрики (CPU/mem/I/O), так и высокоуровневые прикладные (latency, error rate и т.п.).[6][5]

Основные тенденции (синтез по статьям)

Тренд A: "Обучаемое" планирование и автоскейлинг вместо фиксированных правил

В 2024–2025 отчётливо видно смещение от статики и порогов к адаптивным стратегиям, которые могут подстраиваться под нагрузку и неопределённость.[2][1]
При этом RL появляется сразу в двух местах: (1) scheduling/placement, (2) autoscaling, то есть управление ведётся и на уровне куда поставить, и на уровне сколько реплик держать.[1]

Тренд B: Multi-objective оптимизация (стоимость, latency, эффективность)

Оптимизация одной метрики становится недостаточной: работы явно обсуждают баланс между производительностью/задержкой и стоимостью ресурсов.[3][2]
Особенно это видно в контексте облака (spot/on-demand) и в контексте инференса, где хвостовые задержки (например P95) часто важнее среднего значения.[1]

Тренд C: Переход к SLO-ориентированному управлению

Вместо "CPU utilization" как главного сигнала всё чаще обсуждаются SLO и прикладные сигналы качества: latency, error rate, tail latency, throughput.[5][3]
Это логично: для пользователя важен не CPU, а ответ сервиса. CPU — только прокси-сигнал, и в AI-инференсе он особенно плох (GPU-узкие места, очереди, batching).[1]

Тренд D: GPU-aware и inference-специфичная оркестрация

Отдельная линия — GPU-ориентированные сценарии: интеграция с Prometheus, сбор GPU-метрик и обучение/оценка политик автоскейлинга под реальный профиль нагрузки инференса.[1]
Это показывает, что контейнеризация ИИ — уже не про "запаковать модель в Docker", а про полноценную эксплуатацию: наблюдаемость, корректное масштабирование, устойчивость к всплескам.[1]

Тренд E: Симуляция как мост между исследованием и production

Для RL-подходов повторяется мотив: учить агента на реальной системе дорого и рискованно, поэтому симуляторы/тестбеды становятся отдельным важным направлением.[1]

Как это связано с учебным проектом

Даже в минимальной реализации (REST API + Kubernetes) видно, что Kubernetes-примитивы (Deployment, Service, Ingress, HPA) — это базовый уровень автоматизации эксплуатации. Однако исследования показывают, что этот уровень часто недостаточен для реальных AI-нагрузок: нужны более богатые сигналы и более гибкие политики принятия решений.[5][1]

В проекте уже реализован HPA по CPU. Это хороший учебный аналог "реактивного, порогового управления". Дальнейшее развитие в логике рассмотренных статей — подключение полноценной наблюдаемости (Prometheus), переход к SLO-ориентированным метрикам (например latency), а затем — к предиктивным/обучаемым автоскейлерам.[3][1]

Если бы проект расширялся до настоящего AI-инференса, то именно направления из статей (GPU-aware autoscaling, RL/forecasting, cost-aware политики) стали бы логичными кандидатами на улучшение архитектуры.

Заключение

1. В 2024–2025 в исследованиях усиливается тренд на применение DL/RL и AI-Ops принципов для scheduling и autoscaling в Kubernetes-подобных средах, особенно при динамической и стохастической нагрузке.[3][1]
2. Autoscaling переосмысливается как задача управления с несколькими целями: SLO/latency, стоимость, устойчивость и объяснимость решений.[5][3]
3. Для AI-инференса, особенно GPU-сценариев, стандартных CPU-порогов недостаточно, поэтому растёт интерес к специализированным метрикам и стратегиям управления, а также к симуляторам и тестбедам для безопасного обучения/оценки политик.[1]

Ссылки

- Xu et al., "Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques…" — arXiv:2403.07905[1]
- Jayanetti et al., "A Deep Reinforcement Learning Approach for Cost Optimized Workflow Scheduling…" — arXiv:2408.02926[2]
- Zhang et al., "KIS-S: A GPU-Aware Kubernetes Inference Simulator with RL-Based Auto-Scaling" — arXiv:2507.07932[1]
- "An SLO Driven and Cost-Aware Autoscaling Framework for …" — arXiv:2512.23415[4][3]
- "Key Considerations for Auto-Scaling" — arXiv:2510.02585[6][5]

1
2
3
4
5
6
7

8
9
10
11
12
13
14
15
16
17