Проблематика інтеграції ML-компонентів у розподілену мікросервісну архітектуру веб-сервісу для контент-аналізу

Андрій Коляда, магістрант[1], (ORCID: 0009-0001-9763-4071)
Тамара Лященко, ст. викладач [1] (ORCID: 0000-0001-9092-0297)

[1] *Київський національний університет будівництва та архітектури, Україна*

АНОТАЦІЯ

Робота присвячена розгорнутому аналізу інтеграції компонентів машинного навчання (ML) у мікросервісну архітектуру веб-сервісу для контент-аналізу. Показано, що поєднання вимог до низької латентності, високої доступності та відтворюваності з притаманними ML обмеженнями (ресурсоємний інференс, потреба у прискорювачах, дрейф даних і моделей) формує комплекс взаємопов'язаних викликів на перетині архітектури, даних, експлуатації та безпеки. Узагальнено підходи до декомпозиції на сервіси з чіткими контрактами (REST/gRPC, подієві шини), до організації єдиного контуру даних/ознак із паритетом офлайн/онлайн (feature store), до керування життєвим циклом моделей (MLOps: пайплайни, модельний реєстр, безпечні стратегії розгортання, а також до спостережуваності якості та продуктивності (SLI/SLO, трейси, «ML Test Score»). Обґрунтовано використання розподілених обчислень (Spark) і автоскейлінгу у Kubernetes для стабільного виконання SLO без деградації якості та з керованою вартістю.

Ключові слова: мікросервіси; машинне навчання; контент-аналіз; MLOps; Kubernetes; model registry; feature store; latency SLO.

1. ВСТУП

Зростання обсягів і різноманіття контенту (текст, зображення, відео, потоки подій) зумовлює попит на інтелектуальні веб-сервіси, здатні в реальному часі класифікувати, фільтрувати, маркувати та модератувати матеріали. Мікросервісна архітектура природно підтримує незалежне масштабування ingestion, нормалізації, інференсу та аналітики, але інтеграція ML у такий ландшафт стикається з низкою характерних ризиків: неоднорідні профілі навантаження, потреба в GPU/TPU, «холодні старти» контейнерів інференсу, дрейф даних/ознак/передбачень, а також необхідність наскрізної спостережуваності й чітких контрактів. Крім того, промислова готовність ML-рішень вимагає формалізації перевірок і процедур якості (рубрика «ML Test Score») та дисципліни керування змінами (canary/blue-green), що ускладнює проєктування й експлуатацію.

2. МЕТА

Сформулювати та обґрунтувати практичні рішення інтеграції ML-компонентів у мікросервісну архітектуру веб-сервісу для контент-аналізу, які забезпечують: масштабованість і відмовостійкість сервісів, стабільність метрик якості моделей при змінних даних і навантаженнях, виконання SLO з латентності/доступності, керовану вартість обчислень у хмарному середовищі.

3. ДОСЛІДЖЕННЯ ПРОБЛЕМАТИКИ І ПІДХОДИ ДО ЇЇ ВИРІШЕННЯ

Інтеграція ML у мікросервісну архітектуру починається з продуманої декомпозиції та чітких інтеграційних контрактів. Функції збирання й очищення даних, формування та видачі ознак, тренування моделей, реєстру версій і онлайн-інференсу відокремлюються на спеціалізовані сервіси, що взаємодіють через стандартизовані API. Для зовнішніх клієнтів доцільно використовувати REST (сумісність, проста еволюція контрактів), натомість низьколатентну міжсервісну взаємодію реалізовувати через gRPC із бінарною серіалізацією та підтримкою стримінгу. Подієві сценарії з високою пропускною здатністю (модерація стрімів, масовий коментарний трафік) ефективніше обробляти шинами повідомлень на кшталт Kafka/NATS з патернами outbox, ідемпотентністю, ретраями з backoff та circuit breaker для локалізації збоїв. Вбудовані correlation/trace id у кожному запиті забезпечують наскрізний трейсинг і керованість SLO на «гарячому» шляху інференсу [1, 2].

Ключем до стабільної якості є узгодженість даних та ознак між офлайновим навчанням і онлайн-сервінгом. Єдиний feature store фіксує визначення ознак, їхні джерела, частоти оновлення, SLA свіжості та правила валідації (діапазони, sparsity, захист від витоків), а також забезпечує відтворюваність обчислень у дослідах і продакшні. Щоб уникнути training/serving skew, профілі розподілів ознак, зафіксовані під час тренування, регулярно звіряються з продакшн-розподілами; значущі відхилення інтерпретуються як дрейф і слугують тригером для розслідувань або перенавчання. Еволюцію схем (Avro/Proto) варто контролювати через реєстр із вимогою зворотної сумісності, а походження даних — відслідковувати каталогами та data lineage, що спрощує аудит і ретроспективи інцидентів. Масове переобчислення ознак і побудову датасетів доцільно виконувати розподіленими рушіями (Apache Spark), що скорочує час backfill і підвищує передбачуваність експлуатації [2, 5].

Життєвий цикл моделей формалізується практиками MLOps. Оркестровані пайплайни (Airflow/Prefect/Kubeflow) фіксують версії коду, середовищ і датасетів; експерименти та артефакти відслідковуються, а model registry зберігає метрики, список ознак, хеші та політики промоції між стадіями staging/production. Перед продакшном моделі проходять інваріантні тести стабільності, перевірки на витоки ознак та відповідність контрактам; далі застосовуються безпечні стратегії випуску: shadow (0% трафіку, спостереження), canary (градуальне розширення частки запитів із порогами на метрики) або blue-green із миттєвим rollback. Така дисципліна зменшує «прихований

технічний борг» ML-систем і робить релізи передбачуваними [3, 4, 6, 7].

Виконання SLO з латентності й доступності досягається поєднанням інженерних і модельних оптимізацій. На шляху запиту це мікробатчинг/динамічний батчинг на сервері інференсу, прогрівання моделей, кешування ембеддингів/результатів, а також квантизація, прюнінг або дистиляція моделей. Для семантичного пошуку/зіставлення корисний спеціалізований сервіс ANN-індексів із кешами top-K. Автоскейлінг HPA/KEDA з окремими GPU-пулами та pod disruption budgets забезпечує стабільну пропускну здатність під час піків. Спостережуваність вибудовується на SLI/SLO (P50/P95/P99, throughput, error-rate), «золотих сигналах» (latency/traffic/errors/saturation), OpenTelemetry-трейсингу та профілі моделі з індикаторами дрейфу даних, ознак і передбачень, що з'єднані із тригерами перенавчання [1, 2, 4].

Питання безпеки й економіки рішень опрацьовуються паралельно з технічною архітектурою. Ідентифікація/авторизація реалізується OIDC/OAuth2 з RBAC/ABAC і mTLS для міжсервісної взаємодії; секрети централізуються (KMS/Vault) і ротуються, дані шифруються у транзиті та на зберіганні, а PII мінімізуються та псевдонімізуються відповідно до політик ретеншну. Вартість контролюється через cost allocation (теги/лейбли), rightsizing контейнерів і вузлів, масштабування GPU-пулів і scale-to-zero для неактивних тренувальних задач; архітектурно зменшують синхронні «ланцюжки» REST на користь подієвих інтеграцій, розводять «гарячі» та «холодні» шляхи й завчасно обчислюють ознаки — усе це допомагає тримати cost per 1k requests у межах бюджету без порушення SLO [1, 2, 5].

4. МЕТОДИКА ЕКСПЕРИМЕНТАЛЬНОЇ ОЦІНКИ ТА КРИТЕРІЇ ПРИЙНЯТТЯ

Щоб підтвердити, що запропонована інтеграція ML-компонентів у мікросервісну архітектуру забезпечує стабільну якість контент-аналізу та дотримання SLO під змінним навантаженням, оцінювання проводиться на трьох рівнях: офлайнова якість моделей, онлайнова якість/латентність, а також експлуатаційна надійність і вартість у постановці з чіткими контрактами та незалежним масштабуванням сервісів [1, 2].

На офлайновому рівні формується репрезентативний корпус даних (баланс класів і тематик, часові зрізи, різні джерела), усуваються дублікати й запобігається feature leakage; версії даних/коду/середовищ фіксуються оркестрованими пайплайнами навчання (Airflow/Prefect/Kubeflow), а експерименти й артефакти відслідковуються у реєстрі моделей (MLflow/Kubeflow Registry) [6, 7]. Базові метрики — precision/recall, macro-F1, AUROC для релевантних задач; кандидат у staging приймається за умови покращення відносно базової моделі та проходження інваріантних перевірок, рекомендованих підходами «Hidden Technical Debt…» і «ML Test Score» (стабільність метрик, відтворюваність, відсутність leakage) [3, 4].

Онлайнове оцінювання стартує у shadow-режимі (нульовий вплив на користувача) з порівнянням передбачень і латентності нової та поточної моделей, після чого виконується canary із контрольованим нарощуванням частки трафіку та гейтингом за P95/P99 latency, error-rate, throughput і стабільністю бізнес-метрик. У разі деградації активується rollback у схемі blue-green. Такі стратегії відповідають практикам розгортання у мікросервісах та рекомендованим підходам до життєвого циклу моделей у MLOps-платформах [1, 6, 7].

Експлуатаційну надійність перевіряють навантажувальними тестами (ступінчасті та пікові профілі RPS), випробуваннями на backpressure (штучні лаги брокера подій) і «хаос»-сценаріями часткових відмов (kill pod, мережеві затримки, деградація залежних сервісів) із верифікацією circuit breaker, ідемпотентності та коректного повторного споживання подій [1, 2]. Для дрейфу даних/ознак/передбачень застосовують порівняння розподілів (наприклад, PSI/KL-дивергенція) і встановлюють пороги сповіщень, що ініціюють перенавчання; відповідність цим практикам підтримується рекомендаціями щодо інженерної дисципліни ML та тестування [3, 4, 6].

Компонент економіки вимірюється як cost per 1k requests і GPU-години для тренування/інференсу. Проводять абляційні досліди ефекту квантизації, прюнінгу, дистиляції, мікробатчингу та кешування ембеддингів на якість/латентність/вартість; пакетні конвеєри підготовки даних і backfill виконують на розподілених рушіях (Apache Spark), що скорочує цикл оновлення ознак і стабілізує експлуатацію [2, 5]. Фінальні пороги SLO/якості/вартості формалізуються як автоматичні гейткіпери у CI/CD та MLOps-пайплайнах [1–4, 6, 7].

5. ВИСНОВКИ

Інтеграція ML-компонентів у мікросервісну архітектуру вимагає узгодженості рішень щодо даних, моделей, безпеки та експлуатації. Поєднання практик мікросервісної інженерії, інженерії даних і MLOps забезпечує стабільність якості та керовану вартість масштабування сервісів. Практична імплементація має ґрунтуватися на стандартизованих контрактах даних і API, єдиному feature store, формалізованому життєвому циклі моделей та наскрізній спостережуваності. Це дозволяє створювати надійні, керовані та масштабовані ML-сервіси у сучасних розподілених системах.

Список літератури.

[1] Newman, S. Building Microservices. O'Reilly, 2015.

[2] Kleppmann, M. Designing Data-Intensive Applications. O'Reilly, 2017.

[3] Sculley, D., et al. Hidden Technical Debt in Machine Learning Systems. NeurIPS, 2015.

[4] Breck, E., et al. The ML Test Score: A Rubric for ML Production Readiness. Google Research, 2017–2019 (whitepaper).

[5] Zaharia, M., et al. "Apache Spark: A Unified Engine for Big Data Processing." Communications of the ACM, 2016.

[6] Kubeflow Project. Kubeflow: Machine Learning Toolkit for Kubernetes. Project documentation.

[7] MLflow. MLflow Model Registry – Concepts & Workflows. Documentation.