

ПРОЄКТУВАННЯ ТА АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ КОРПОРАТИВНОГО ІІІ-СЕРВІСУ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Баландюк Антон Вікторович

студент групи МІПМ-1-26-1.4 д.,

Київський столичний університет імені Бориса Грінченка, м. Київ,

науковий керівник — к.т.н., доц.Машкіна І.В.

ВСТУП

Актуальність і постановка проблеми

Великі мовні моделі (Large Language Models, LLM) поступово переходять від експериментального використання до інтеграції у внутрішні інформаційні системи підприємств. Корпоративні ІІІ-сервіси можуть застосовуватися для інтелектуального пошуку в документах, аналізу текстових даних, підтримки фахівців, генерації програмного коду та автоматизації окремих бізнес-процесів. Перехід від використання зовнішнього чат-сервісу до керованого корпоративного рішення потребує створення власної або гібридної обчислювальної інфраструктури, у якій контрольованими є моделі, дані, доступ користувачів і життєвий цикл програмних компонентів.

На відміну від традиційних вебзастосунків, LLM-сервіси характеризуються значними вимогами до GPU-пам'яті та обчислювальних ресурсів, нерівномірністю навантаження і залежністю часу відповіді від довжини контексту та кількості одночасних запитів. Ефективність обслуговування залежить також від керування KV-cache і пакетування запитів. У роботі W. Kwon та співавторів запропоновано PagedAttention та систему vLLM, орієнтовані на ефективніше використання пам'яті під час LLM-serving[1].

Корпоративне використання генеративного ІІІ висуває додаткові вимоги до безпеки та керування ризиками, оскільки система може опрацьовувати внутрішні документи, персональні дані, програмний код і службову інформацію. Профіль NIST AI 600-1 визначає специфічні ризики генеративного ІІІ та рекомендує управляти ними протягом життєвого циклу системи [2]. OWASP серед актуальних ризиків LLM-застосунків розглядає prompt injection, розкриття чутливої інформації, ризики ланцюга постачання, отруєння даних і моделей та неконтрольоване споживання ресурсів [3].

Отже, створення корпоративного ІІІ-сервісу не зводиться до запуску мовної моделі на сервері. Потрібна цілісна інфраструктура, що охоплює обчислювальні ресурси, контейнеризацію, оркестрацію, LLM inference, керування моделями, автентифікацію й авторизацію, спостережуваність, автоматизоване розгортання та оновлення.

Окремою проблемою є відтворюваність інфраструктури. Ручне налаштування серверів, мережі, контейнерів і прикладних компонентів ускладнює повторне розгортання, масштабування та відновлення. Підхід Infrastructure as Code (IaC) дає змогу описувати інфраструктуру декларативними конфігураціями, версіонувати, повторно використовувати та

автоматично застосовувати їх. Terraform підтримує керування життєвим циклом хмарних і локальних ресурсів за таким принципом [4]. Для оркестрації контейнеризованих компонентів доцільним є Kubernetes, який підтримує планування GPU через device plugins і надання спеціалізованих ресурсів контейнерам [5, 6].

Мета дослідження

Метою дослідження є розроблення та експериментальне обґрунтування архітектурних і технологічних рішень для автоматизованого розгортання інфраструктури корпоративного ІІІ-сервісу на основі великих мовних моделей із забезпеченням відтворюваності, масштабованості, спостережуваності та контрольованого використання обчислювальних ресурсів.

Об'єкт дослідження — процес розгортання та функціонування інфраструктури корпоративних ІІІ-сервісів на основі великих мовних моделей.

Предмет дослідження — архітектурні рішення, методи Infrastructure as Code, контейнеризації, оркестрації, автоматизованого розгортання, LLM inference та моніторингу, що забезпечують відтворюване й кероване функціонування корпоративного ІІІ-сервісу.

Для досягнення мети передбачається: проаналізувати сучасні підходи до LLM-serving та MLOps; визначити функціональні й нефункціональні вимоги; розробити концептуальну архітектуру; обґрунтувати технологічний стек; автоматизувати створення інфраструктури й розгортання сервісу; сформувати систему показників; реалізувати прототип і провести порівняльні експерименти.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ І ПУБЛІКАЦІЙ

Проблематика розгортання великих мовних моделей знаходиться на перетині високопродуктивного inference, MLOps, контейнерної оркестрації, Infrastructure as Code, DevOps/GitOps, кібербезпеки та observability. Одним із ключових завдань LLM-serving є ефективне використання GPU-пам'яті. W. Kwon та співавтори показали можливості PagedAttention для керування KV-cache; реалізована на його основі система vLLM у наведених авторами експериментах забезпечувала вищу пропускну здатність порівняно з досліджуваними системами за співставного рівня затримки [1]. Це обґрунтовує розгляд inference engine як окремого об'єкта експериментального порівняння.

Інший підхід до промислового inference представлений NVIDIA Triton Inference Server, що підтримує HTTP/REST і gRPC, репозиторії моделей, пакетування запитів та метрики продуктивності [7]. Для Kubernetes-native serving розвиваються спеціалізовані рішення, зокрема KServe, що надає засоби розгортання inference-сервісів у Kubernetes [8].

Контейнеризація забезпечує переносимість програмного середовища та фіксацію його залежностей [9]. Для LLM це особливо важливо через залежність inference-середовища від версій CUDA, фреймворків машинного навчання та серверів моделей. Kubernetes створює рівень оркестрації, а device plugin framework дозволяє рекламувати GPU як ресурси кластера й надавати їх Pod [5, 6].

Для відтворюваного формування базової інфраструктури доцільно застосовувати IaC. Terraform дозволяє описувати compute-, storage- і network-ресурси декларативними конфігураціями, зберігати їх у системі контролю версій та повторно застосовувати [4]. Поєднання IaC з GitOps поширює декларативний підхід на конфігурацію застосунків. Argo CD підтримує декларативне визначення та синхронізацію Kubernetes-застосунків із Git-репозиторієм [10].

Важливим компонентом MLOps є керування версіями моделей. MLflow Model Registry забезпечує централізоване ведення моделей, їх версій, метаданих та зв'язків із відповідними експериментами [11]. Для спостережуваності OpenTelemetry використовує traces, metrics і logs як основні телеметричні сигнали [12], а Prometheus орієнтований на збирання та зберігання часових рядів метрик [13]. Для LLM-сервісу ці показники мають доповнюватися latency, time to first token (TTFT), throughput, кількістю токенів і використанням GPU-пам'яті.

Проведений аналіз показує, що наявні засоби вирішують окремі складові задачі. Тому предметом подальшого дослідження є їх узгодження в єдиному керованому процесі: «опис інфраструктури → створення ресурсів → оркестрація → розгортання LLM → контроль версій → моніторинг → оновлення або відновлення».

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Оскільки дослідження перебуває на початковому етапі, його поточними результатами є постановка задачі, визначення вимог, формування концептуальної архітектури, попередній вибір технологічного стеку та формулювання гіпотез для подальшої експериментальної перевірки. Кількісні висновки щодо продуктивності або ефективності запропонованих рішень на цьому етапі не формулюються.

Корпоративний ІІІ-сервіс пропонується розглядати як багаторівневу систему. На інфраструктурному рівні розміщуються серверні або хмарні обчислювальні ресурси, GPU, мережа та сховища. Наступний рівень утворюють контейнеризація та Kubernetes. Рівень LLM-serving забезпечує завантаження моделі, виконання inference та API-доступ. Над ним розташовуються корпоративний API, автентифікація та прикладні сервіси. Наскрізними контурами є автоматизація розгортання, керування моделями, безпека та observability (рис. 1).

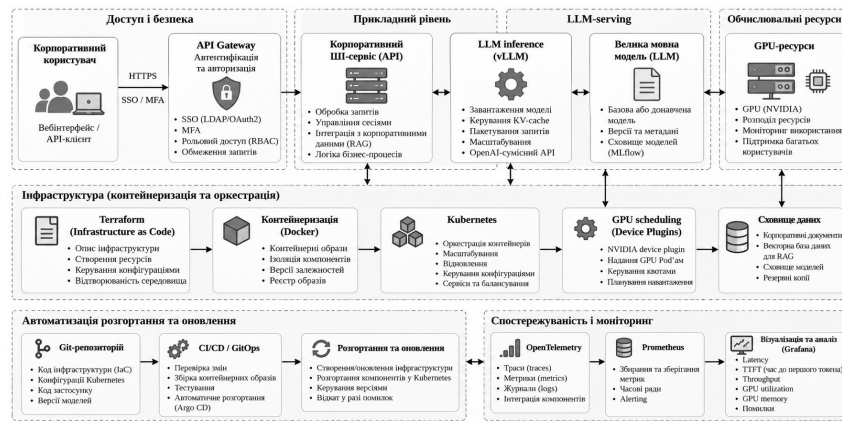


Рис. 1. Концептуальна архітектура та автоматизований процес розгортання корпоративного LLM-сервісу

Попередньо визначено такий технологічний стек: Docker — контейнеризація; Kubernetes — оркестрація та керування GPU-ресурсами; Terraform — опис і створення інфраструктури як коду; Helm/Kubernetes manifests — декларативне конфігурування; Git та CI/CD/GitOps — контроль і автоматичне застосування змін; vLLM — основний кандидат для LLM inference; NVIDIA Triton — один із можливих засобів порівняння; MLflow — керування моделями; Prometheus та OpenTelemetry — метрики й телеметрія.

Гіпотеза 1. Використання Infrastructure as Code та декларативного опису конфігурації дасть змогу скоротити тривалість повторного розгортання корпоративного LLM-сервісу та зменшити конфігураційні розбіжності порівняно з ручним розгортанням. Передбачається порівняти частково ручний сценарій із Terraform та декларативними конфігураціями Kubernetes за часом, кількістю ручних операцій, помилками, відтворюваністю та часом відновлення.

Гіпотеза 2. Контейнеризація та Kubernetes-оркестрація забезпечать відтворюваніше оновлення й відновлення LLM-сервісу та спростять керування GPU. Перевірка охоплюватиме первинне розгортання, перезапуск компонента, відмову, оновлення версії та повторне розгортання на іншому вузлі. GPU планується надавати контейнерам через device plugin [6].

Гіпотеза 3. Спеціалізований LLM inference engine забезпечить ефективніше використання GPU та вищу пропускну здатність порівняно з базовим запуском моделі без оптимізованого serving-рівня. Основним кандидатом є vLLM [1]. За однакової моделі, апаратної платформи та набору запитів планується вимірювати throughput, end-to-end latency, TTFT, швидкість генерації токенів, GPU utilization та GPU memory.

Гіпотеза 4. Автоматизований CI/CD/GitOps-процес скоротить час між зміною конфігурації та отриманням працездатної версії сервісу й забезпечить контрольоване повернення до попередньої конфігурації. Репозиторій міститиме IaC і конфігурації застосунку; зміни проходять перевірку, побудову образу, тестування та автоматизоване розгортання, зокрема з використанням Argo CD [10].

Гіпотеза 5. Поеднання інфраструктурної та LLM-специфічної телеметрії дозволить швидше локалізувати причини деградації продуктивності, ніж

контроль лише базових показників сервера. Планується збирати CPU/RAM/GPU utilization, GPU memory, кількість і тривалість запитів, throughput, latency, TTFT та помилки inference засобами Prometheus і OpenTelemetry [12, 13, 14].

Окремим напрямом є безпека: автентифікація користувачів, рольове розмежування доступу, захист секретів, мережеві політики та аудит. Рекомендації NIST і OWASP використовуватимуться для формування моделі ризиків і вимог до захисту [2, 3].

Для експериментальної перевірки передбачається створити тестовий стенд із GPU та розгорнути відкриту LLM, параметри якої відповідатимуть доступним ресурсам. Вибір конкретної моделі доцільно виконувати на етапі реалізації, щоб архітектура не була прив'язана до одного сімейства моделей. Кожен експеримент планується повторювати за однакових початкових умов. Основні залежні показники: час розгортання і відновлення, throughput, медіана та 95-й перцентиль latency, TTFT, GPU utilization, GPU memory, частка успішних запитів. Незалежні змінні: спосіб розгортання, inference engine, кількість одночасних запитів, довжина контексту та конфігурація ресурсів.

ВИСНОВКИ

Сформульовано задачу проєктування та автоматизації розгортання інфраструктури корпоративного ІІ-сервісу на основі великих мовних моделей. Встановлено, що така інфраструктура повинна забезпечувати не лише LLM inference, але й керування GPU-ресурсами, відтворюваність середовища, автоматизоване розгортання й оновлення, контроль версій моделей, захищений доступ і комплексну спостережуваність.

За результатами початкового аналізу сформовано концептуальну архітектуру та технологічний контур на основі Docker, Kubernetes, Terraform, vLLM, засобів GitOps, MLflow, Prometheus та OpenTelemetry. Сформульовано п'ять гіпотез щодо впливу ІаС на відтворюваність, Kubernetes — на керованість GPU-навантаження, спеціалізованого inference engine — на продуктивність, GitOps — на керованість оновлень, а комплексної телеметрії — на діагностування деградації.

Подальша робота передбачає реалізацію прототипу, автоматизацію його розгортання, проведення контрольованих експериментів і кількісне оцінювання продуктивності, відтворюваності, надійності та використання обчислювальних ресурсів. Таким чином, на поточному етапі результатом є обґрунтована постановка задачі та план експериментальної перевірки, а не передчасне декларування ефективності ще не реалізованої системи.

ДЖЕРЕЛА

1. Kwon W., Li Z., Zhuang S. et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23). 2023. DOI: 10.1145/3600006.3613165.
2. Autio C., Schwartz R., Dunietz J. et al. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1. 2024. DOI: 10.6028/NIST.AI.600-1.

3. OWASP Foundation. OWASP Top 10 for LLM Applications 2025. OWASP GenAI Security Project.
4. HashiCorp. Terraform Documentation: Infrastructure as Code. HashiCorp Developer Documentation.
5. The Kubernetes Authors. Production Environment. Kubernetes Documentation.
6. The Kubernetes Authors. Schedule GPUs; Device Plugins. Kubernetes Documentation.
7. NVIDIA. NVIDIA Triton Inference Server Documentation.
8. KServe Project. KServe Documentation: LLM / Generative Inference.
9. Docker Inc. Docker Documentation: What is a Container?
10. Argo Project. Argo CD: Declarative GitOps Continuous Delivery for Kubernetes.
11. MLflow Project. MLflow Model Registry Documentation.
12. OpenTelemetry Authors. OpenTelemetry Documentation: Signals and Observability.
13. Prometheus Authors. Prometheus Documentation: Overview.
14. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science. URL: <https://doi.org/10.28925/9720213284km>