Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
Магістр

(назва освітнього ступеня)

на тему: **Розробка малої мовної моделі SLM для системи з обмеженими обчислювальними ресурсами на базі AZURE ML та AI Foundry**

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Містерман П. М. (прізвище та ініціали)

Керівник (підпис) Петрик М. Р. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)
Рецензент (підпис) Гром'як Р. С. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістра
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Містерману Петру Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка малої мовної моделі SLM для системи з обмеженими обчислювальними ресурсами на базі AZURE ML та AI Foundry.

Керівник роботи Петрик М. Р.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз вимог до програмної системи.
Проєктування та розробка програмної системи
Тестування, оцінювання та навантажувальний аналіз системи
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації, схема архітектури системи, порівняльні таблиці SLM та LLM

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Стручок В. С. ст. викл.		
основи охорони праці	Осухівська Г. М. зав. каф.		
Нормоконтроль	Стоянов Ю.М. к.т.н., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз завдання		
3	Виконання розділу 1		
4	Виконання розділу 2		
5	Виконання розділу 3		
6	Оформлення пояснювальної записки		
7	Оформлення графічного та презентаційного матеріалу		
8	Попередній захист		
9	Захист		

Студент _____ Містерман П. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Містерман Петро Михайлович. Розробка малої мовної моделі для системи з обмеженими обчислювальними ресурсами на базі Azure ML та AI Foundry. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 73, таблиць 2, рисунків 19, додатків 2, бібліографічних посилань 35.

Метою роботи є розробка програмної системи класифікації голосової пошти на основі малої мовної моделі для використання в умовах обмежених обчислювальних ресурсів із застосуванням платформи Azure Machine Learning.

Об’єктом дослідження є процес автоматизованої класифікації голосових повідомлень у програмних системах з використанням технологій штучного інтелекту. Предметом дослідження є методи та засоби розробки програмної системи класифікації голосової пошти на основі малих мовних моделей.

У роботі проаналізовано підходи до використання великих і малих мовних моделей та реалізовано життєвий цикл малої мовної моделі, включаючи підготовку синтетичних даних, тонке налаштування, розгортання та тестування. Проведено експериментальне оцінювання якості класифікації та навантажувальне тестування системи.

Результати дослідження показали, що тонко налаштована мала мовна модель забезпечує точність класифікації понад 96 % та ефективне масштабування при нижчих експлуатаційних витратах у порівнянні з великою мовною моделлю. Отримані результати підтверджують доцільність використання малих мовних моделей у системах з обмеженими обчислювальними ресурсами.

Практичне значення роботи полягає у можливості використання розробленої системи для автоматизованої класифікації голосової пошти в реальних інформаційних системах.

Ключові слова: інженерія програмного забезпечення, штучний інтелект, малі мовні моделі, класифікація голосової пошти, Azure Machine Learning.

ABSTRACT

Misterman Petro Mykhailovych. Development of a Small Language Model for Systems with Limited Computational Resources Based on Azure ML and AI Foundry. Master's qualification thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, Specialty 121 "Software Engineering". Ternopil, 2025. Pages 73, tables 2, figures 19, appendices 2, references 35.

The aim of this thesis is to develop a voicemail classification system based on a small language model suitable for operation in environments with limited computational resources using the Azure Machine Learning platform.

The object of the research is the process of automated classification of voice messages in software systems using artificial intelligence technologies. The subject of the research is the methods and tools for developing a voicemail classification system based on small language models.

The thesis analyzes existing approaches to the use of large and small language models for natural language processing tasks and implements the full lifecycle of a small language model, including synthetic data preparation, fine-tuning, deployment, and testing. Experimental evaluation of classification quality and load testing of the system were conducted.

The results of the study demonstrate that the fine-tuned small language model achieves classification accuracy above 96% and supports effective horizontal scaling while maintaining lower operational costs compared to a large language model. These results confirm the feasibility and efficiency of using small language models in systems with limited computational resources.

The practical significance of the work lies in the possibility of applying the developed system in real-world information systems for automated voicemail classification.

Keywords: software engineering, artificial intelligence, small language models, voicemail classification, Azure Machine Learning.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AI - Artificial Intelligence (галузь інформатики, що займається створенням інтелектуальних систем).

LLM – Large Language Model (велика мовна модель, що містить мільярди параметрів).

LoRA – Low-Rank Adaptation (метод тонкого налаштування мовних моделей із використанням низькорозмірних матриць).

MLOps – Machine Learning Operations (практики розгортання, моніторингу та підтримки моделей машинного навчання).

RPS – Requests per second (кількість запитів, що обробляються системою за одну секунду).

SLM – Small Language Model (мала мовна модель, оптимізована для систем з обмеженими обчислювальними ресурсами).

VM – Virtual Machine (віртуальна машина).

ВСТУП

Стрімкий розвиток технологій штучного інтелекту та обробки природної мови зумовив широке впровадження мовних моделей у різні сфери інформаційних систем, зокрема у сервіси автоматизованої обробки голосових повідомлень. Однією з поширених практичних задач є класифікація голосової пошти з метою визначення, чи було повідомлення залишене живою людиною, чи автоматизованою системою. Така задача є актуальною для контакт-центрів, телекомунікаційних платформ та систем автоматичного обслуговування клієнтів, де своєчасна та коректна обробка повідомлень безпосередньо впливає на якість сервісу.

Сучасні підходи до розв'язання подібних задач часто ґрунтуються на використанні великих мовних моделей, які забезпечують високий рівень точності, однак потребують значних обчислювальних ресурсів або залежать від зовнішніх хмарних сервісів. Це створює обмеження щодо масштабованості, передбачуваності витрат та контролю над процесами навчання і розгортання моделей. У зв'язку з цим актуальним є пошук альтернативних підходів, які дозволяють поєднати достатній рівень якості результатів із економічною доцільністю та можливістю використання в умовах обмежених обчислювальних ресурсів. Одним із таких підходів є застосування малих мовних моделей із подальшим тонким налаштуванням під конкретну предметну область.

Метою даної магістерської роботи є розробка та дослідження програмної системи класифікації голосової пошти на основі малої мовної моделі, адаптованої для роботи в умовах обмежених обчислювальних ресурсів із використанням платформи Azure Machine Learning та Azure AI Foundry.

Для досягнення поставленої мети у роботі необхідно розв'язати такі задачі:

- проаналізувати предметну область класифікації голосової пошти та існуючі підходи до використання мовних моделей;
- сформулювати вимоги до програмної системи з урахуванням обмежень обчислювальних ресурсів;

- підготувати та розширити тренувальний датасет для задачі класифікації;
- реалізувати процес тонкого налаштування малої мовної моделі з використанням керованих обчислювальних ресурсів;
- розгорнути модель у вигляді endpoint та забезпечити програмну взаємодію з нею;
- виконати тестування програмної системи, включаючи оцінювання якості класифікації та навантажувальне тестування;
- проаналізувати отримані результати та обґрунтувати доцільність використання малої мовної моделі замість великої мовної моделі.

Об’єктом дослідження є процес автоматизованої обробки та класифікації голосових повідомлень у програмних системах із використанням технологій штучного інтелекту.

Предметом дослідження є методи та засоби розробки програмної системи класифікації голосової пошти на основі малої мовної моделі з урахуванням обмежень обчислювальних ресурсів і вимог до продуктивності.

Наукова новизна роботи полягає у дослідженні можливості ефективного застосування малої мовної моделі як альтернативи великим мовним моделям для задачі класифікації голосової пошти, а також у порівняльному аналізі якості, продуктивності та експлуатаційних характеристик моделей у різних сценаріях розгортання.

Практичне значення одержаних результатів полягає у створенні працездатної програмної системи класифікації голосової пошти, яка може бути використана у реальних інформаційних системах. Запропоноване рішення демонструє можливість зниження вимог до обчислювальних ресурсів та вартості експлуатації без суттєвої втрати якості, що робить його доцільним для впровадження в умовах обмеженої інфраструктури.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Аналіз вимог є початковим етапом проєктування програмної системи та визначає її функціональні можливості, обмеження й критерії якості. Для систем, що базуються на машинному навчанні, особливу роль відіграє формалізація вимог не лише до програмних компонентів, але й до даних, моделей та інфраструктури. У даному розділі виконано аналіз вимог до програмної системи класифікації голосової пошти на основі малої мовної моделі з урахуванням результатів її реалізації та тестування.

1.1 Аналіз предметної області

Предметна область даної роботи охоплює автоматизовану класифікацію голосової пошти з метою визначення типу джерела повідомлення — живої людини або автоматизованої системи. У сучасних інформаційних та телекомунікаційних системах обсяги вхідних голосових повідомлень постійно зростають, що робить ручну обробку таких даних малоефективною та економічно невиправданою. Це зумовлює необхідність створення автоматизованих програмних рішень, здатних працювати стабільно в умовах обмежених обчислювальних ресурсів.

Особливістю задачі класифікації голосової пошти є її залежність від якості текстової транскрипції та варіативності мовлення. Повідомлення можуть бути короткими, фрагментованими або містити стандартні фрази, характерні для автовідповідачів. Водночас мовлення реальних людей часто є непередбачуваним і неструктурованим. Така різноманітність вхідних даних ускладнює використання простих правил або класичних алгоритмів машинного навчання та зумовлює доцільність застосування мовних моделей.

Використання великих мовних моделей у даній предметній області дозволяє досягати високої якості класифікації, однак супроводжується суттєвими обмеженнями. До них належать високі вимоги до обчислювальних ресурсів, залежність від зовнішніх serverless API, обмежений контроль над процесом

навчання та складність прогнозування експлуатаційних витрат при масштабуванні системи. Для рішень, орієнтованих на довготривалу роботу та обробку великої кількості повідомлень, такі фактори можуть стати критичними.

У системах з обмеженими обчислювальними ресурсами ключовими вимогами є стабільність роботи, передбачувана продуктивність та економічна ефективність. У цьому контексті використання малих мовних моделей є більш доцільним підходом, оскільки вони дозволяють суттєво зменшити вимоги до апаратного забезпечення без істотної втрати якості класифікації. Тонке налаштування SLM під конкретну задачу дає змогу адаптувати модель до специфіки предметної області та досягти результатів, співставних з великими мовними моделями.

Крім того, малі мовні моделі забезпечують більшу гнучкість у виборі архітектури розгортання. Вони можуть використовуватися як у serverless-середовищах, так і на керованих обчислювальних ресурсах, що дозволяє оптимізувати баланс між продуктивністю та витратами. Це є особливо важливим для систем класифікації голосової пошти, де обробка може виконуватися як у потоковому, так і в пакетному режимі, залежно від навантаження.

Таким чином, предметна область класифікації голосової пошти в умовах обмежених обчислювальних ресурсів вимагає застосування ефективних та масштабованих рішень. Використання малих мовних моделей у поєднанні з сучасними MLOps-підходами дозволяє створити програмну систему, яка забезпечує необхідний рівень якості, продуктивності та економічної доцільності.

1.2 Аналіз існуючих рішень та підходів до використання великих і малих мовних моделей

У сучасних системах обробки природної мови все ширше застосовуються мовні моделі, які здатні виконувати складні завдання класифікації, узагальнення та інтерпретації текстових даних. Для задач автоматизованої обробки голосової пошти такі моделі дозволяють працювати з неструктурованими текстовими

транскрипціями без необхідності ручного проєктування правил або використання вузькоспеціалізованих ознак. Водночас вибір типу мовної моделі суттєво впливає на архітектуру системи, вимоги до інфраструктури та економічну доцільність рішення.

Одним із поширених підходів є використання великих мовних моделей, які надаються у вигляді готових сервісів через serverless API. Такі моделі демонструють високий рівень узагальнення та стабільні результати для широкого спектра задач обробки тексту. Вони не потребують локального розгортання та дозволяють швидко інтегрувати інтелектуальні функції в існуючі програмні системи. Для задач класифікації голосової пошти великі мовні моделі можуть забезпечувати високу точність без додаткового навчання, що робить їх привабливими на етапі прототипування або швидкої перевірки гіпотез [1].

Водночас використання великих мовних моделей супроводжується низкою обмежень. Основними з них є залежність від зовнішніх сервісів, обмежений контроль над процесом навчання та конфігурацією моделі, а також складність прогнозування витрат при масштабуванні системи. Оскільки обробка кожного запиту тарифікується окремо, загальна вартість використання LLM може суттєво зростати зі збільшенням обсягів даних. Для систем, орієнтованих на тривалу експлуатацію та обробку великої кількості повідомлень, такі фактори можуть стати критичними [2].

Альтернативним підходом є використання малих мовних моделей, які мають значно меншу кількість параметрів і можуть бути розгорнуті на керованих або локальних обчислювальних ресурсах. Хоча базові можливості таких моделей є обмеженішими порівняно з LLM, їх тонке налаштування під конкретну задачу дозволяє суттєво підвищити якість результатів. Для задач класифікації тексту, зокрема голосової пошти, SLM здатні досягати високої точності за умови наявності репрезентативного тренувального датасету та правильно підібраних параметрів навчання.

Використання малих мовних моделей надає розробникам більший контроль над усім життєвим циклом моделі — від підготовки даних до розгортання та

масштабування. Це дозволяє оптимізувати використання обчислювальних ресурсів, адаптувати архітектуру системи до реальних навантажень та забезпечити передбачувану вартість експлуатації. Крім того, SLM можуть використовуватися як у serverless-середовищах, так і на керованих обчислювальних ресурсах, що забезпечує гнучкість у виборі сценарію розгортання залежно від вимог системи.

Таким чином, аналіз існуючих підходів свідчить, що великі мовні моделі є ефективним інструментом для швидкої інтеграції інтелектуальних функцій, однак їх використання не завжди є доцільним у системах з обмеженими обчислювальними ресурсами. У таких умовах застосування малих мовних моделей із тонким налаштуванням під конкретну задачу дозволяє досягти балансу між якістю, продуктивністю та економічною ефективністю, що робить цей підхід більш придатним для практичної реалізації програмних систем класифікації голосової пошти.

1.3 Постановка завдання та цілей програмної системи

Метою розробки програмної системи є створення ефективного рішення для автоматизованої класифікації голосової пошти в умовах обмежених обчислювальних ресурсів. Система повинна забезпечувати коректне визначення типу джерела повідомлення – жива людина або автоматизована система – з прийнятним рівнем точності, стабільною продуктивністю та прогнозованими витратами на експлуатацію.

Для досягнення поставленої мети необхідно розв’язати комплекс взаємопов’язаних задач, що охоплюють як програмну інженерію, так і компоненти машинного навчання. Зокрема, система має забезпечувати обробку текстових транскрипцій голосових повідомлень, формування запитів до мовної моделі, отримання та інтерпретацію результатів класифікації, а також інтеграцію з хмарною інфраструктурою для навчання та розгортання моделі.

Однією з ключових задач є вибір та адаптація мовної моделі, яка здатна ефективно працювати в умовах обмежених ресурсів. Це передбачає використання

малої мовної моделі з можливістю її тонкого налаштування під конкретну предметну область, що дозволяє досягти високої якості класифікації без використання великих мовних моделей. Важливою складовою завдання є забезпечення повного життєвого циклу моделі, включаючи підготовку даних, навчання, реєстрацію, розгортання та подальше тестування.

З огляду на практичне використання системи, до неї висуваються вимоги щодо продуктивності та масштабованості. Система повинна стабільно працювати при зростанні кількості запитів та підтримувати можливість горизонтального масштабування шляхом збільшення кількості інстансів мовної моделі. Водночас необхідно забезпечити контроль над затримкою відповіді та рівнем помилок при високому навантаженні.

Додатковим завданням є забезпечення відтворюваності результатів та керованості процесів навчання і тестування. Це досягається шляхом використання MLOps-підходів, які дозволяють стандартизувати процеси підготовки даних, конфігурації навчання, оцінювання якості моделей та управління версіями. Такий підхід є необхідним для подальшого розвитку та підтримки програмної системи.

Таким чином, постановка завдання для даної програмної системи полягає у створенні масштабованого, економічно доцільного та ефективного рішення для класифікації голосової пошти на основі малої мовної моделі, яке відповідає вимогам систем з обмеженими обчислювальними ресурсами та може бути використане в реальних умовах експлуатації.

1.4 Вимоги до програмної системи

Вимоги до програмної системи визначають перелік її функціональних можливостей, а також обмеження та характеристики якості, яким вона повинна відповідати під час експлуатації. Для систем на основі машинного навчання такі вимоги охоплюють не лише програмні компоненти, але й вимоги до моделей, даних, продуктивності та інфраструктури. У даному підрозділі сформульовано

вимоги до програмної системи класифікації голосової пошти з урахуванням особливостей її архітектури та результатів експериментального оцінювання.

1.4.1 Функціональні вимоги

Програмна система повинна забезпечувати автоматизовану класифікацію текстових транскрипцій голосової пошти з метою визначення типу джерела повідомлення. Система має приймати на вхід текстове повідомлення у визначеному форматі та повертати результат класифікації у вигляді однієї з наперед заданих міток без додаткових пояснень. Такий формат вихідних даних забезпечує можливість подальшої автоматизованої обробки результатів іншими компонентами інформаційної системи.

Система повинна забезпечувати взаємодію з мовною моделлю через програмний інтерфейс на основі кінцевої точки інференсу. Формування запитів до моделі має виконуватися з використанням чітко визначеної інструкції, що гарантує детерміновану поведінку моделі та стабільність результатів класифікації. Усі виклики моделі повинні виконуватися без необхідності участі користувача.

Програмна система повинна підтримувати повний життєвий цикл мовної моделі, включаючи підготовку тренувальних даних, тонке налаштування, реєстрацію моделі та її розгортання у вигляді endpoint. Процеси навчання та повторного навчання моделі мають бути автоматизованими та відтворюваними, що досягається шляхом використання конфігураційних файлів і тренувальних скриптів.

Система повинна забезпечувати можливість масштабування шляхом збільшення кількості інстансів endpoint без зміни прикладної логіки. Це дозволяє адаптувати систему до зростання кількості запитів та забезпечити стабільну роботу в умовах підвищеного навантаження.

1.4.2 Нефункціональні вимоги

Однією з ключових нефункціональних вимог є обмеження обчислювальних ресурсів, у межах яких повинна працювати система. Програмне рішення має бути орієнтоване на використання малих мовних моделей, що дозволяє зменшити вимоги до апаратного забезпечення та забезпечити економічну доцільність експлуатації. Використання великих мовних моделей допускається лише на етапах порівняльного аналізу або прототипування.

Система повинна забезпечувати прийнятний рівень продуктивності, зокрема обмеження на середню затримку відповіді та стабільну обробку паралельних запитів. Продуктивність системи має зберігатися на передбачуваному рівні при масштабуванні, а збільшення навантаження не повинно призводити до неконтрольованого зростання кількості помилок.

Важливою нефункціональною вимогою є надійність та стабільність роботи системи. Програмна система повинна коректно обробляти помилки виклику endpoint, тимчасову недоступність інфраструктурних компонентів та нестандартні вхідні дані без порушення загальної логіки роботи.

До нефункціональних вимог також належить відтворюваність результатів та керованість процесів машинного навчання. Усі експерименти з навчання моделей повинні мати можливість повторного запуску з ідентичними параметрами, а результати — бути зафіксованими у відповідних артефактах. Це є необхідною умовою для подальшої підтримки та розвитку системи.

Таким чином, сформульовані функціональні та нефункціональні вимоги визначають рамки, в межах яких розроблятиметься програмна система забезпечує коректну, стабільну та економічно доцільну роботу в умовах обмежених обчислювальних ресурсів.

1.5 Аналіз обмежень обчислювальних ресурсів та економічної доцільності

Аналіз обмежень обчислювальних ресурсів є важливим етапом формування вимог до програмної системи, оскільки безпосередньо впливає на вибір архітектурних рішень, тип мовної моделі та спосіб її розгортання. Для систем автоматизованої класифікації голосової пошти, які орієнтовані на обробку значної кількості повідомлень, критичними чинниками є стабільність роботи, прогнозована продуктивність та контроль експлуатаційних витрат.

Використання великих мовних моделей, як правило, пов’язане з високими вимогами до обчислювальних ресурсів або залежністю від зовнішніх serverless API. У таких сценаріях розробник має обмежений контроль над інфраструктурою, а вартість обробки запитів зростає пропорційно до обсягу даних. Для систем, що функціонують у режимі постійного навантаження або потребують масштабування, це може призводити до суттєвого збільшення операційних витрат та ускладнювати довготривалу експлуатацію рішення.

У межах даної роботи програмна система розглядається як компонент більшої інформаційної інфраструктури, що накладає додаткові обмеження на доступні обчислювальні ресурси. Такі обмеження можуть бути зумовлені як апаратними можливостями середовища виконання, так і організаційними факторами, зокрема лімітами на використання хмарних сервісів або вимогами до оптимізації витрат. У цьому контексті застосування малих мовних моделей є інженерно обґрунтованим підходом, оскільки вони дозволяють ефективно використовувати доступні ресурси без істотної втрати якості результатів.

Економічна доцільність розроблюваної системи визначається співвідношенням між якістю класифікації, продуктивністю та витратами на експлуатацію. Малі мовні моделі, особливо після тонкого налаштування під конкретну задачу, забезпечують можливість досягнення необхідного рівня точності при значно нижчих витратах на обчислювальні ресурси. Крім того,

можливість розгортання таких моделей на керованих обчислювальних ресурсах дозволяє оптимізувати витрати шляхом масштабування лише за потреби.

Важливим аспектом економічної доцільності є також передбачуваність витрат. На відміну від повністю serverless-підходів, де вартість залежить від кількості запитів і може суттєво змінюватися з часом, використання керованих обчислювальних ресурсів забезпечує більш стабільну модель витрат. Це спрощує планування бюджету та робить систему більш придатною для інтеграції в корпоративні або промислові середовища.

Таким чином, аналіз обмежень обчислювальних ресурсів та економічної доцільності підтверджує доцільність вибору малої мовної моделі як основи програмної системи класифікації голосової пошти. Обраний підхід дозволяє забезпечити баланс між якістю, продуктивністю та витратами, що є ключовим чинником для створення ефективного та масштабованого програмного рішення в умовах обмежених ресурсів.

1.6 Висновки до першого розділу

У даному розділі було виконано комплексний аналіз вимог до програмної системи класифікації голосової пошти в умовах обмежених обчислювальних ресурсів. Розглянуто предметну область задачі, особливості обробки неструктурованих текстових даних та актуальність автоматизації процесів класифікації голосових повідомлень у сучасних інформаційних системах.

У межах розділу проаналізовано існуючі підходи до використання великих і малих мовних моделей для задач обробки природної мови. Показано, що великі мовні моделі забезпечують високий рівень якості, однак їх застосування супроводжується обмеженнями щодо обчислювальних ресурсів, контролю над процесами навчання та економічної доцільності. Натомість малі мовні моделі після адаптації під конкретну предметну область є більш придатними для використання в системах, орієнтованих на довготривалу експлуатацію та масштабування.

Також було сформульовано постановку задачі та цілей розробки програмної системи, що визначило основні напрями подальшої реалізації. Окрему увагу приділено формуванню функціональних та нефункціональних вимог, які охоплюють вимоги до якості класифікації, продуктивності, надійності та керованості процесів машинного навчання.

Аналіз обмежень обчислювальних ресурсів та економічної доцільності підтвердив доцільність використання малих мовних моделей як основи розроблюваної системи. Обраний підхід дозволяє досягти балансу між якістю результатів, продуктивністю та витратами на експлуатацію, що є ключовим фактором для практичного впровадження програмного рішення.

Таким чином, у першому розділі створено теоретичне та методичне підґрунтя для подальшої розробки програмної системи. Отримані результати визначили вимоги та обмеження, які були враховані під час проєктування, реалізації та експериментального дослідження, представлених у наступних розділах магістерської роботи.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі описано проєктні рішення та підхід до розробки програмної системи на основі малої мовної моделі для класифікації голосової пошти в умовах обмежених обчислювальних ресурсів. Основну увагу приділено вибору процесу розробки, формуванню архітектури системи, а також організації життєвого циклу машинного навчання з використанням платформ Azure Machine Learning та Azure AI Foundry. Окремо розглядаються питання забезпечення ефективності, відтворюваності та масштабованості розроблюваного рішення.

2.1 Вибір процесу розробки програмної системи

Розробка програмних систем на основі машинного навчання та штучного інтелекту має низку особливостей, які відрізняють її від класичних підходів програмної інженерії. Зокрема, результат роботи такої системи значною мірою залежить від якості даних, параметрів навчання та обраної архітектури моделі, а процес створення рішення часто носить експериментальний характер. У зв’язку з цим вибір процесу розробки є критично важливим для забезпечення гнучкості, відтворюваності та керованості життєвого циклу програмної системи.

У даній магістерській роботі для розробки системи класифікації голосової пошти було обрано гібридний підхід, що поєднує принципи Agile-розробки та життєвий цикл MLOps. Agile-методологія передбачає ітеративну розробку, поступове вдосконалення функціональності та постійний аналіз отриманих результатів, що є доцільним у проєктах з високим рівнем невизначеності та необхідністю швидкої адаптації рішень [3]. Застосування Agile дозволяє гнучко змінювати підходи до підготовки даних, налаштування моделей та вибору метрик без порушення загальної логіки проєкту.

Разом із тим, класичні Agile-підходи не враховують специфіку машинного навчання, зокрема необхідність управління версіями датасетів, відстеження експериментів, контролю якості моделей та їх деградації з часом. Для розв’язання

цих задач у роботі використовується концепція MLOps, яка є розвитком ідей DevOps у контексті машинного навчання [4]. MLOps охоплює повний життєвий цикл моделі — від збору та підготовки даних до навчання, валідації, розгортання та моніторингу в робочому середовищі.

Поєднання Agile та MLOps забезпечує можливість коротких ітерацій розробки, під час яких здійснюється уточнення синтетичних тренувальних даних, тонке налаштування малої мовної моделі, оцінювання її якості та прийняття рішень щодо подальших змін. Такий підхід дозволяє системно порівнювати різні мовні моделі, оптимізувати використання обчислювальних ресурсів та поступово наближати систему до заданих функціональних і нефункціональних вимог [5].

Платформи Azure Machine Learning та Azure AI Foundry надають інструментальні засоби, які безпосередньо підтримують реалізацію MLOps-підходу, зокрема керування експериментами, запуск тренувальних задач, реєстрацію моделей, розгортання кінцевих точок та масштабування обчислювальних ресурсів [6][7]. Це дозволяє інтегрувати процес навчання та експлуатації моделей у єдиний керований пайплайн, що є важливим для створення промислово придатних AI-рішень.

Таким чином, вибір гібридного процесу розробки Agile + MLOps є обґрунтованим для даної магістерської роботи, оскільки він забезпечує баланс між гнучкістю експериментального підходу та структурованістю інженерного процесу. Це дозволяє ефективно розробити програмну систему на основі малої мовної моделі, придатну для використання в умовах обмежених обчислювальних ресурсів.

2.2 Проєктування архітектури системи класифікації голосової пошти

Архітектура програмної системи класифікації голосової пошти проєктувалася з урахуванням вимог до обмежених обчислювальних ресурсів, масштабованості та можливості інтеграції з зовнішніми сервісами. Основною метою архітектурного проєктування було створення такого рішення, яке дозволяє

використовувати малу мовну модель як сервіс, забезпечуючи стабільну якість класифікації та мінімальні витрати на інференс.

Розроблювана система побудована за сервісно-орієнтованим підходом, де мовна модель розгортається у вигляді окремого endpoint, доступного через програмний інтерфейс. Такий підхід є типовим для сучасних систем машинного навчання та дозволяє відокремити логіку моделі від клієнтських застосунків, спрощуючи масштабування та підтримку рішення [8].

Загальна архітектура системи включає етапи підготовки даних, навчання моделі, її реєстрації та подальшого використання для інференсу. Тренувальні та тестові дані зберігаються у сховищах, керованих платформою Azure AI Foundry, що забезпечує централізоване управління даними без необхідності ручного проєктування баз даних. Після завершення процесу навчання модель реєструється в реєстрі моделей Azure Machine Learning, що дозволяє відстежувати версії та використовувати модель у різних середовищах [9].

Інференс виконується через кінцеву точку, розгорнуту в середовищі Azure ML, яка приймає текстові повідомлення голосової пошти та повертає результат класифікації. Взаємодія із системою здійснюється виключно через програмний інтерфейс, що відповідає типовим сценаріям використання AI-сервісів у корпоративних системах та дозволяє легко інтегрувати рішення з іншими компонентами інформаційної інфраструктури [10].

Архітектурне рішення також враховує можливість масштабування обчислювальних ресурсів. Кількість віртуальних машин, що обслуговують endpoint, може змінюватися залежно від навантаження, що є важливим для забезпечення стабільної роботи системи під час пікових запитів. Такий підхід відповідає принципам хмарних обчислень і дозволяє оптимізувати співвідношення між продуктивністю та вартістю використання ресурсів [11].

На рисунку 2.1 представлено загальну архітектуру системи класифікації голосової пошти з використанням малої мовної моделі згенерованої за допомогою gpt-5.2, що демонструє взаємозв’язок між основними компонентами та етапами життєвого циклу моделі.

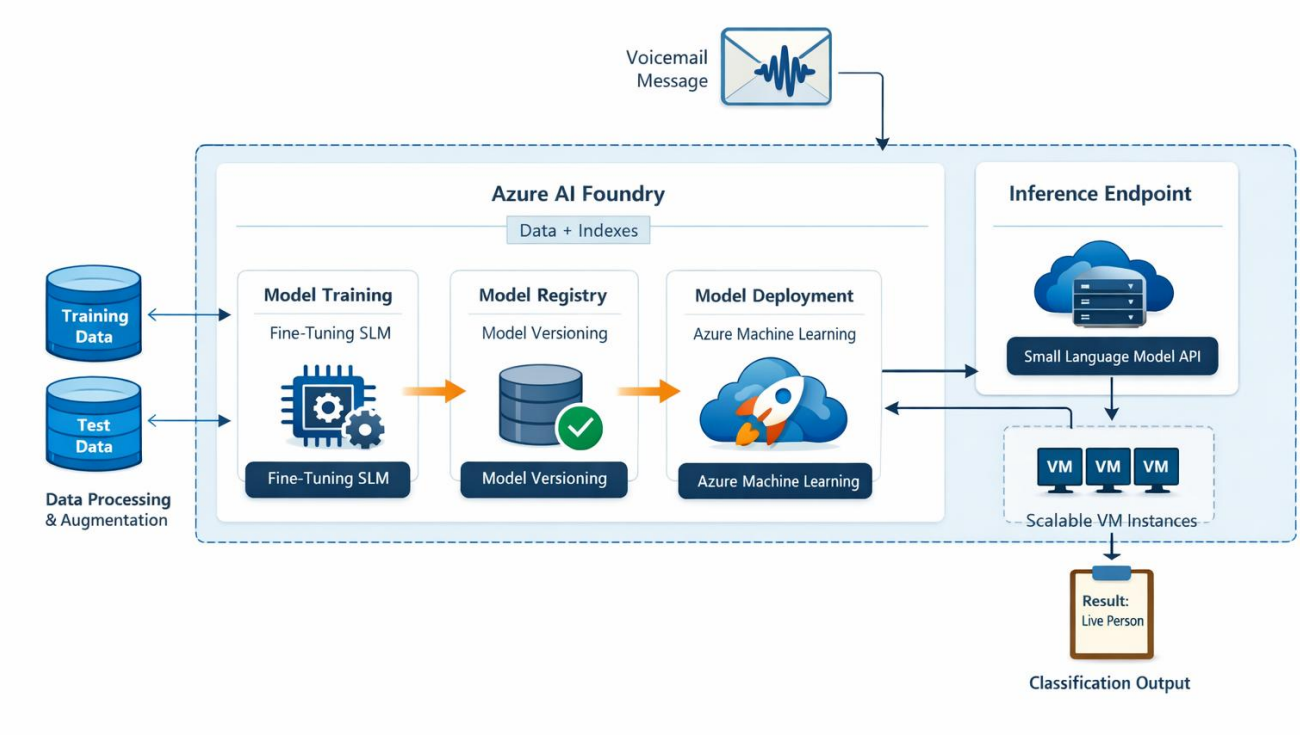

Рисунок 2.1 – Архітектура системи класифікації голосової пошти на основі малої мовної моделі

Запропонована архітектура забезпечує гнучкість розгортання, можливість подальшого розширення функціональності та відповідність вимогам до систем з обмеженими обчислювальними ресурсами. Вона створює основу для реалізації наступних етапів розробки, зокрема організації зберігання даних, підготовки синтетичних датасетів і тонкого налаштування мовної моделі.

2.3 Організація зберігання даних та ML-артефактів у Azure AI Foundry

Ефективна організація зберігання даних та артефактів машинного навчання є одним із ключових аспектів побудови надійних та масштабованих AI-систем. У випадку хмарних платформ сучасні підходи до управління даними передбачають максимальну автоматизацію інфраструктурних процесів та мінімальну участь розробника у фізичному проєктуванні сховищ. Саме такий підхід реалізовано в середовищі Azure AI Foundry та Azure Machine Learning.

У межах розроблюваної системи фізичне створення та адміністрування баз даних не виконується користувачем безпосередньо. Зберігання тренувальних, валідаційних і тестових даних, а також проміжних результатів експериментів повністю керується платформою Azure AI Foundry.

Тренувальні дані, додані до проекту в AI Foundry, автоматично зберігаються в Azure Blob Storage і можуть бути повторно використані в різних експериментах. Також Azure AI Foundry забезпечує централізоване управління цими даними, включно з контролем доступу, версій та інтеграцією з тренувальними пайплайнами. Це відповідає рекомендаціям MLOps щодо відтворюваності результатів та прозорості експериментів [12].

Окрім даних, важливим елементом системи є зберігання ML-артефактів, до яких належать навчені моделі, їх конфігурації та метадані. Усі ці артефакти, так само як тренувальні дані, містяться в зберігаються в Blob Storage проекту, що дозволяє зберігати декілька версій однієї моделі, порівнювати їх між собою та використовувати для подальшого розгортання. Реєстр моделей є логічним сховищем і не потребує ручного налаштування з боку розробника, оскільки базується на керованій хмарній інфраструктурі.

Такий підхід до організації зберігання даних і ML-артефактів має низку переваг для систем з обмеженими обчислювальними ресурсами. По-перше, він зменшує складність інфраструктури та знижує ймовірність помилок, пов’язаних із ручним управлінням сховищами. По-друге, централізоване зберігання даних і моделей спрощує масштабування системи та інтеграцію з іншими компонентами хмарного середовища. По-третє, автоматизоване управління артефактами відповідає вимогам сучасних MLOps-практик і забезпечує основу для подальшого моніторингу та підтримки моделі в експлуатації.

Таким чином, використання вбудованих механізмів Azure AI Foundry для зберігання даних і ML-артефактів дозволяє реалізувати ефективну та керовану інфраструктуру без необхідності ручного створення баз даних. Це є важливою складовою розроблюваної програмної системи та забезпечує її відповідність вимогам до сучасних хмарних AI-рішень.

2.4 Формування та підготовка синтетичних тренувальних даних

Підготовка якісного тренувального датасету є критичним етапом розробки систем машинного навчання, особливо у випадку спеціалізованих задач, для яких відсутні відкриті або придатні до використання реальні дані. У задачі класифікації голосової пошти використання реальних записів є обмеженим через питання конфіденційності та доступності, тому в межах даної роботи було застосовано підхід генерації синтетичних даних за допомогою великої мовної моделі.

Синтетичні дані генерувалися з використанням моделі gpt-4o у вигляді текстових діалогів, що імітують реальні сценарії взаємодії між абонентом і автоматичною системою голосової пошти або живою людиною. Кожен згенерований приклад містив послідовність повідомлень із зазначенням ролей та відповідну цільову мітку класифікації. Дані зберігалися у форматі CSV, що спрощує подальшу обробку та інтеграцію з пайплайнами Azure Machine Learning.

Приклад структури одного запису в датасеті наведено на рисунку 2.2.

```
messages,outcome
"[{'role': 'user', 'content': 'Hi'},
  {'role': 'assistant', 'content': 'Hello! My name is Nio, and I am a virtual assistant calling
from ...'},
  {'role': 'user', 'content': 'Hello?'},
  {'role': 'user', 'content': 'Yes.'}]",
"{'vm_disposition': True, 'disposition': 'Live Person'}"
```

Рисунок 2.2 – Приклад структури одного запису в датасеті

Початковий обсяг синтетичного датасету становив 1000 прикладів. Для підвищення різноманітності даних та зменшення ризику перенавчання було реалізовано окремий пайплайн обробки та розширення даних, який автоматизує очищення, нормалізацію та генерацію додаткових варіацій прикладів.

Очищення даних включало перевірку коректності структури діалогів, видалення порожніх або некоректних повідомлень, а також уніфікацію форматів ролей і тексту. Код функції для очищення даних наведено в лістингу 2.1, додаток Б.

Для розширення датасету використовувався підхід автоматичної генерації варіацій діалогів із застосуванням мовної моделі gpt-4o. Основною ідеєю було збереження семантики початкового прикладу з одночасною зміною формулювань, довжини відповідей та стилю мовлення. Для цього використовувався промпт, який задавав чіткі правила генерації нових варіантів, який зображений на рисунку 2.3.

```
paraphrase_prompt = """
You are a text augmentation assistant. Your task is to generate all possible paraphrased variations of a given sentence while keeping the original meaning intact.

- Return the paraphrased sentences as a Python-style list.
- Use different word choices, sentence structures, and phrasings.
- You can shorten the sentence if it still conveys the original meaning.
- Do not add any explanations, return only the list.

Example:

Input: "Please leave a message after the tone."
Output:
[
    "Leave your message after the beep.",
    "Please record your message once you hear the tone.",
    "You can leave a message when the tone sounds.",
    "Kindly leave a message after the signal.",
    "Feel free to leave a voicemail after the tone.",
    "Message me after the signal.",
    "Drop a voicemail after the beep.",
    "Speak after the tone."
]

Now, paraphrase this sentence:
"{sentence}"
"""
```

Рисунок 2.3 – Промпт для розширення датасету

Відповідний код для розширення даних у межах пайплайну зображений в лістингу 2.2.

Лістинг 2.2 – Код функції для розширення даних

```
def paraphrase_replies(replies, llm):
    paraphrased_replies = []
    misstep = 5

    for sample in replies:
        res =
llm.invoke([SystemMessage(content=paraphrase_prompt.format(sentence=
sample))])
        response = ast.literal_eval(res.content)

        for item in response:
            if item not in paraphrased_replies:
                if len(item) > len(sample)-misstep and len(item) <
len(sample)+misstep:
                    paraphrased_replies.append(item)
```

Після етапів очищення та розширення даних пайплайн за допомогою вбудованої функції (див лістинг 2.3) автоматично формує тренувальні семпли в очікуваному форматі, який зображений на рисунку 2.4.

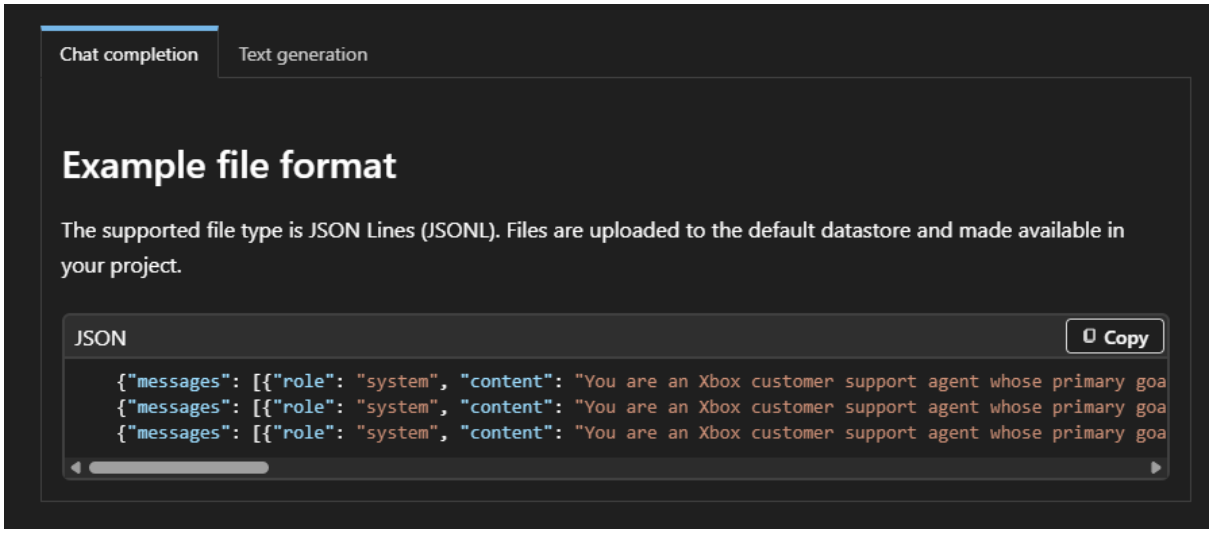

Рисунок 2.4 – Очікуваний формат даних для тренування [13]

Лістинг 2.3 – Код функції для формування тренувальних семплів в правильному форматі

```
def data_to_jsonl(df):
    system_prompt = "You are a text classification model. Determine whether the following response is from a real person or an automated voicemail system.\n\nReturn:\nLive Person - if it's a real person.\nAnswering Machine - if it's a voicemail system.\n\nProvide only the classification result without additional explanations."

    data = []
    for _, row in df.iterrows():
        sample = row["replies"]
        label = "Answering Machine" if row["labels"] == "1" else "Live Person"

        json_obj = {
            "messages": [
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": sample},
                {"role": "assistant", "content": label}
            ]
        }
        data.append(json_obj)

    return data
```

Наступним етапом пайплайн формує фінальний датасет і виконує його розбиття на тренувальну, валідаційну та тестову вибірки. Розподіл даних здійснювався у співвідношенні, рекомендованому для задач машинного навчання, з фіксацією випадкових зерен для забезпечення відтворюваності результатів [14]. У підсумку було сформовано близько 3000 тренувальних, 300 валідаційних та 300 тестових прикладів.

Завершальним етапом пайплайну було автоматичне збереження сформованих датасетів у кероване сховище платформи Azure AI Foundry (див. лістинг 2.4, додаток Б). Дані зберігалися в blob storage проєкту, що дозволяє безпосередньо використовувати їх у тренувальних задачах Azure Machine Learning без додаткових ручних операцій. Завантажені дані в проекті AI Foundry зображено на рисунку 2.5.

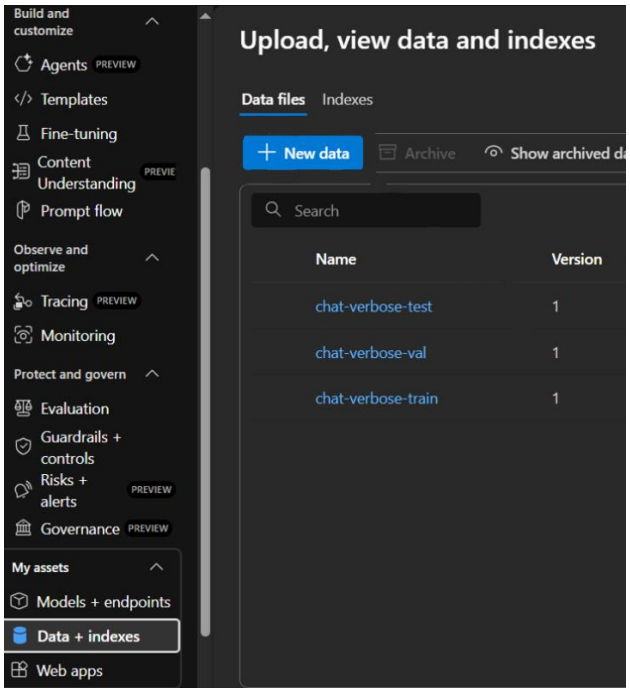

Рисунок 2.5 – Завантажені датасети в проекті для тренування SLM

Таким чином, реалізований пайплайн формування та підготовки синтетичних даних дозволив створити масштабований, відтворюваний та якісно збалансований датасет, придатний для тонкого налаштування малої мовної моделі в межах розроблюваної програмної системи.

2.5 Тонке налаштування SLM у середовищі Azure AI Foundry

У межах даного підрозділу розглядається процес тонкого налаштування малої мовної моделі з використанням інструментів платформи Azure AI Foundry. Особливу увагу приділено різним підходам до навчання моделі, які відрізняються рівнем контролю над параметрами, вимогами до обчислювальних ресурсів та економічною доцільністю.

2.5.1 Serverless fine-tuning як початковий етап навчання

Перед початком повноцінного навчання моделі на керованих обчислювальних ресурсах доцільно виконати початкове тонке налаштування з мінімальними інфраструктурними витратами. Для цього в межах даної роботи було використано serverless-підхід до fine-tuning, який надається платформою Azure AI Foundry. Такий підхід дозволяє швидко оцінити придатність вибраної мовної моделі до цільової задачі без необхідності резервування власних обчислювальних ресурсів.

Serverless fine-tuning у Azure AI Foundry реалізується як керований сервіс, у якому вся інфраструктура для навчання моделі надається платформою. Користувачеві не потрібно створювати або налаштовувати віртуальні машини, GPU-кластери чи мережеві ресурси, оскільки всі ці компоненти автоматично управляються хмарним середовищем. Це робить serverless-підхід особливо привабливим для початкових експериментів, швидкого прототипування та перевірки гіпотез [15].

На етапі налаштування навчання користувачеві пропонується вибір між serverless API та керованими обчислювальними ресурсами. Це зображено на рисунку 2.6. В межах даної роботи на початковому етапі було обрано serverless-підхід, що дозволило уникнути вимог до GPU-квот та значних фінансових витрат.

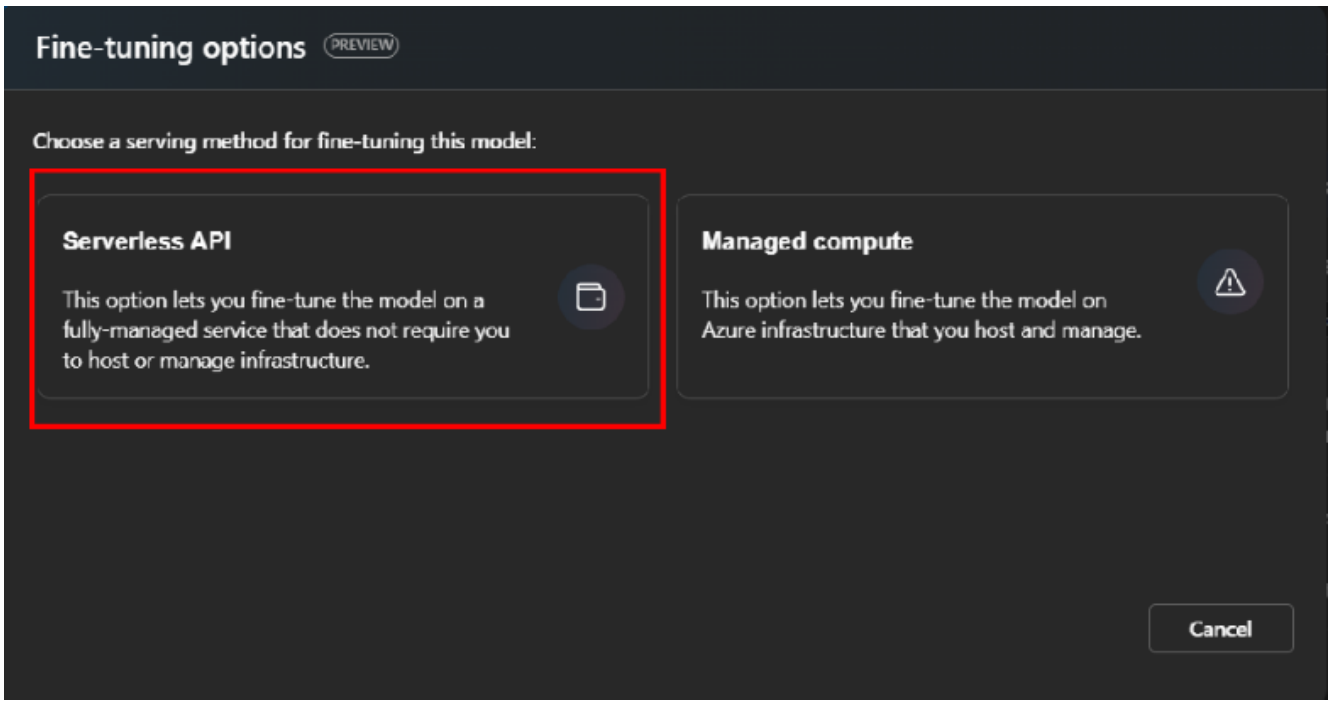

Рисунок 2.6 –Варіанти налаштування

Однією з особливостей serverless fine-tuning є обмежений набір параметрів, доступних для конфігурації навчального процесу. Зокрема, користувач має можливість налаштовувати лише базові параметри, такі як кількість епох, розмір пакету та коефіцієнт навчання [16]. Процес налаштування параметрів тренування зображено на рисунку 2.7.

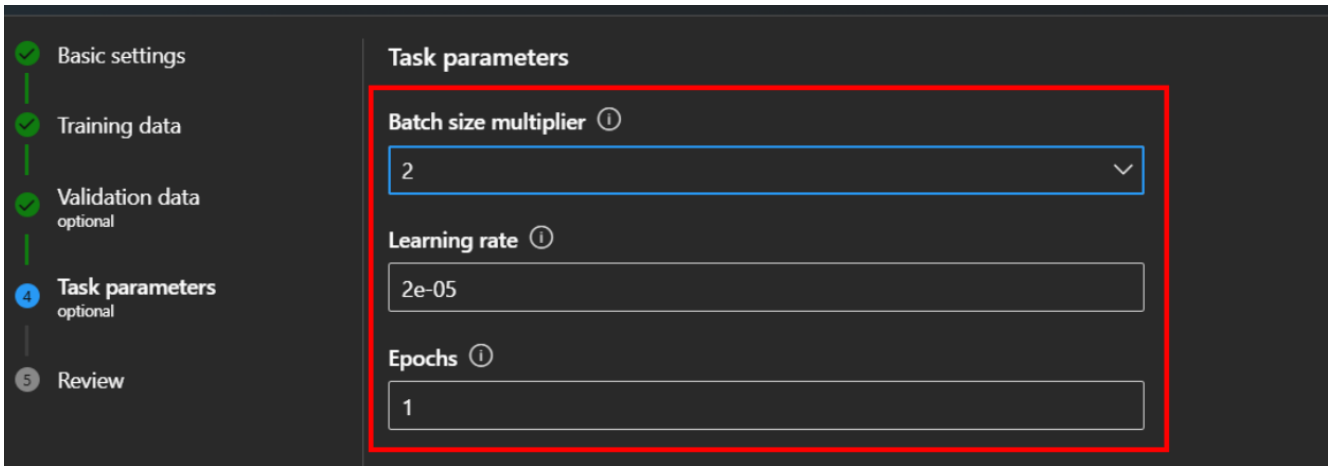

Рисунок 2.7 – Параметри тренування

З одного боку, це зменшує гнучкість експериментів, однак з іншого — значно спрощує процес налаштування та знижує ризик некоректної конфігурації моделі, що є важливим на початковому етапі дослідження.

Використання serverless fine-tuning дозволило швидко отримати перші результати навчання та оцінити вплив тонкого налаштування на якість класифікації голосової пошти. Отримані моделі реєструвалися в реєстрі моделей Azure Machine Learning, що забезпечувало можливість їх подальшого порівняння та повторного використання. Водночас обмеження serverless-підходу щодо кількості доступних параметрів та контролю над обчислювальними ресурсами стали підставою для переходу до наступного етапу — навчання моделі на керованих обчислювальних ресурсах.

Таким чином, serverless fine-tuning був використаний як початковий етап навчання, що дозволив з мінімальними витратами перевірити ефективність обраної мовної моделі та підготувати основу для більш глибокого та керованого процесу тонкого налаштування в середовищі Azure Machine Learning.

2.5.2 Навчання моделі на керованих обчислювальних ресурсах Azure ML

Після початкового етапу тонкого налаштування за допомогою serverless-підходу виникає необхідність у більш гнучкому та контрольованому процесі навчання моделі. Це зумовлено потребою детальнішого налаштування параметрів, використання власних обчислювальних ресурсів та інтеграції навчання у повноцінний MLOps-пайплайн. Для цього в межах даної роботи було використано керовані обчислювальні ресурси платформи Azure Machine Learning.

Навчання мовної моделі на керованих обчислювальних ресурсах Azure ML може виконуватися двома способами: через графічний інтерфейс Azure AI Foundry, що зображено на рисунку 2.8, а також за допомогою власних програмних скриптів. Використання UI дозволяє швидко налаштувати тренування, вибрати модель, датасети та основні параметри, що є зручним для демонстраційних або початкових експериментів.

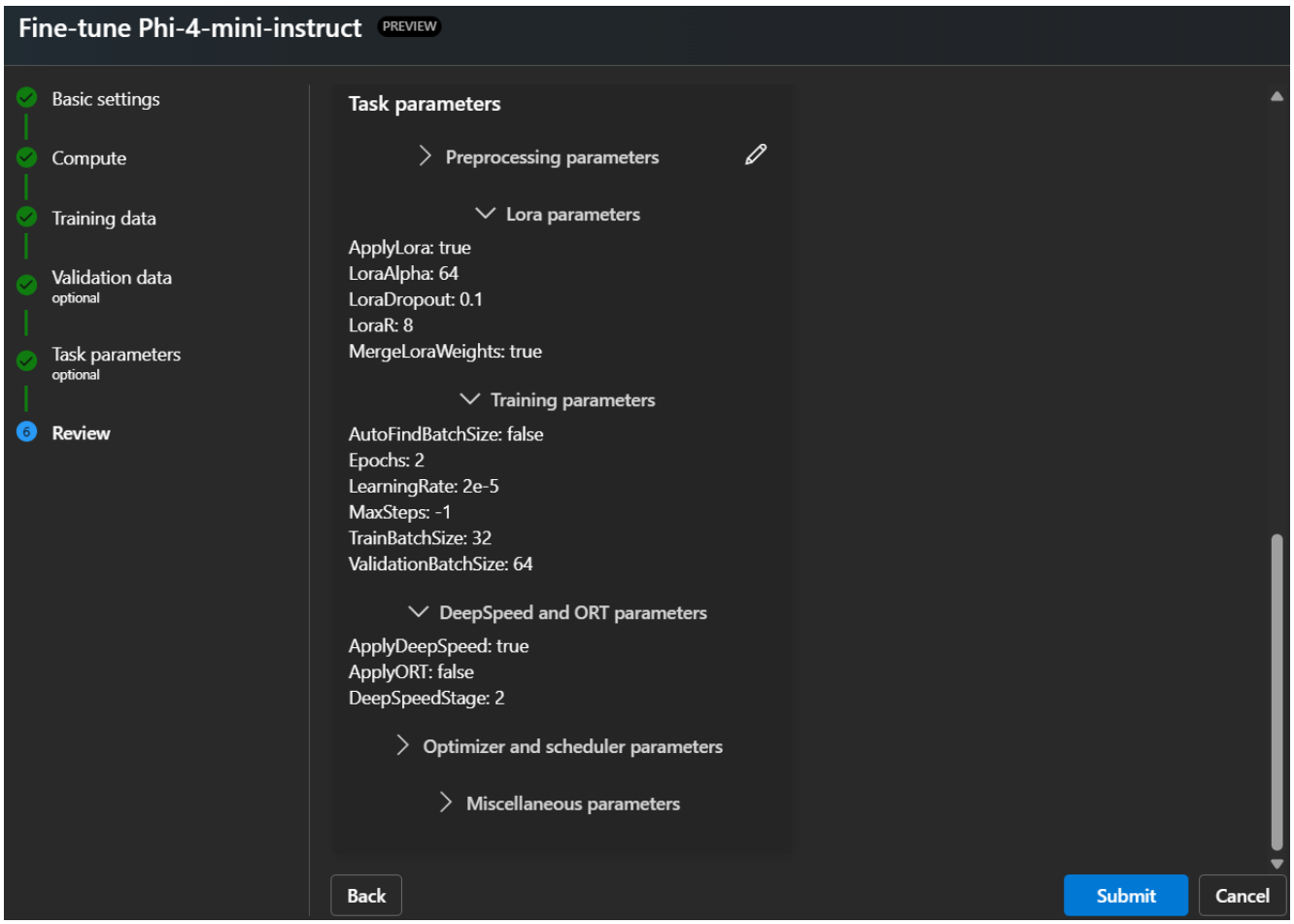

Рисунок 2.8 – Запуск точного налаштування Phi-4-mini-instruct через UI

Однак у межах розроблюваної програмної системи основний акцент було зроблено на використанні власного тренувального скрипта, що забезпечує повний контроль над процесом навчання, вибором обчислювальних ресурсів і логікою пайплайну.

Даний скрипт запускався не на локальному середовищі, а у VS Code контейнері val-chat-agent розгорнутому на Azure. Це зображено на рисунку 2.9.

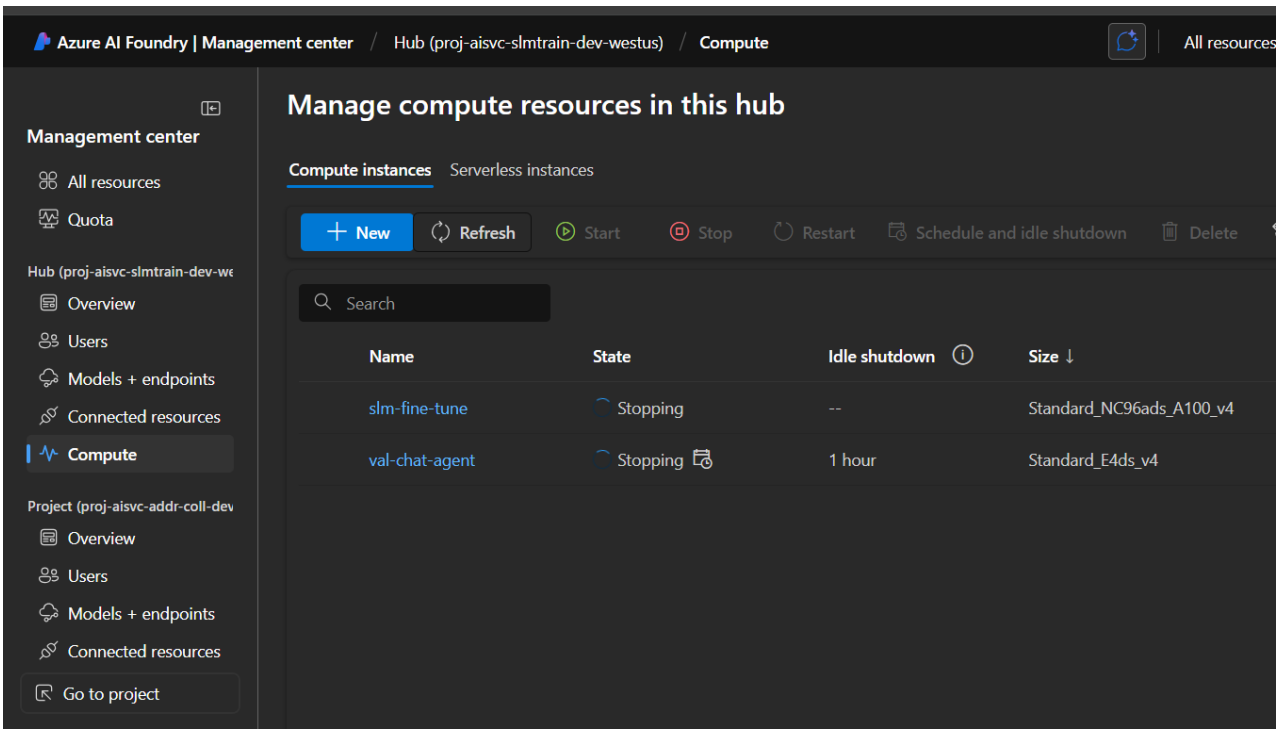

Рисунок 2.9 – Розгорнуті VS Code контейнери на Azure

Першим етапом скрипта є автентифікація та підключення до робочого простору Azure Machine Learning. Для цього використовується стандартний механізм DefaultAzureCredential з автоматичним fallback на інтерактивну автентифікацію, що дозволяє запускати скрипт як у локальному середовищі, так і в CI/CD-сценаріях. Далі відбувається ініціалізація клієнтів для доступу до workspace та системних реєстрів моделей AzureML.

Вибір базової мовної моделі здійснюється шляхом отримання останньої версії моделі Phi-4-mini-instruct із системного реєстру. Відповідний код для вибору моделі зображено на лістингу 2.5.

Лістинг 2.5 – Код для вибору базової мовної моделі

```
model_name = "Phi-4-mini-instruct"

try:
    foundation_model =
registry_ml_client_meta.models.get(model_name, label="latest")
except:
    foundation_model = registry_ml_client.models.get(model_name,
label="latest")
```

```
print(
    "\n\nUsing model name: {0}, version: {1}, id: {2} for fine
tuning".format(
        foundation_model.name, foundation_model.version,
foundation_model.id
    )
)
```

Після вибору моделі виконується аналіз допустимих типів обчислювальних ресурсів для її тонкого налаштування. Деякі мовні моделі мають обмеження щодо типів віртуальних машин, на яких дозволено виконувати fine-tuning, що визначається у їх метаданих.

Далі здійснюється створення або повторне використання GPU-кластера для навчання. У межах роботи було використано кластер з типом віртуальної машини Standard_NC96ads_A100_v4 (див. рис. 2.9), що забезпечує достатню кількість GPU для ефективного навчання з використанням підходу LoRA [17]. Код створення та перевірки compute-кластера наведено на лістингу 2.6.

Лістинг 2.6 – Створення та перевірка GPU-кластера для навчання

```
compute_cluster_size = "Standard_NC96ads_A100_v4"
compute_cluster = "slm-fine-tune"

try:
    compute = workspace_ml_client.compute.get(compute_cluster)
except:
    compute = AmlCompute(
        name=compute_cluster,
        size=compute_cluster_size,
        tier="Dedicated",
        max_instances=2,
    )

workspace_ml_client.compute.begin_create_or_update(compute).wait()
```

Основною особливістю навчання моделі в даній роботі є використання підходу Low-Rank Adaptation (LoRA), який дозволяє суттєво зменшити обчислювальні витрати на тонке налаштування великих і малих мовних моделей. Замість оновлення всіх ваг моделі, LoRA навчає лише додаткові низькорозмірні матриці, що значно знижує вимоги до пам’яті та часу навчання [18].

Параметри тонкого налаштування, включно з конфігурацією LoRA, кількістю епох, розміром батчу та оптимізатором, задаються у вигляді словника. Це дозволяє централізовано керувати конфігурацією експериментів та легко відтворювати результати. Відповідний фрагмент коду наведено на лістингу 2.7 додаток Б.

Навчання моделі реалізується у вигляді ML-пайплайну з використанням готового компонента chat_completion_pipeline, доступного в системному реєстрі AzureML. Пайплайн об’єднує етапи імпорту моделі, попередньої обробки даних, тонкого налаштування та оцінювання, а також автоматично приймає шляхи до тренувального, валідаційного та тестового датасетів. Визначення пайплайну та його запуск наведено на лістингу 2.8.

Лістинг 2.8 – Створення та запуск пайплайну навчання моделі

```
@pipeline(name=pipeline_display_name)
def create_pipeline():
    chat_completion_pipeline = pipeline_component_func(
        mlflow_model_path=foundation_model.id,
        compute_finetune=compute_cluster,
        train_file_path=Input(type="uri_file",
path=os.getenv("TRAIN_FILE_PATH")),
        validation_file_path=Input(type="uri_file",
path=os.getenv("VALIDATION_FILE_PATH")),
        test_file_path=Input(type="uri_file",
path=os.getenv("TEST_FILE_PATH")),
        number_of_gpu_to_use_finetuning=gpus_per_node,
        **finetune_parameters
    )
    return {"trained_model":
chat_completion_pipeline.outputs.mlflow_model_folder}
```

Після запуску тренувальної джоби в Azure ML Studio автоматично відображаються граф навчання, який дозволяє моніторити стан джоби і використовуваних ресурсів під час процесу fine-tuning. Це зображено на рисунках 2.10 і 2.11 відповідно.

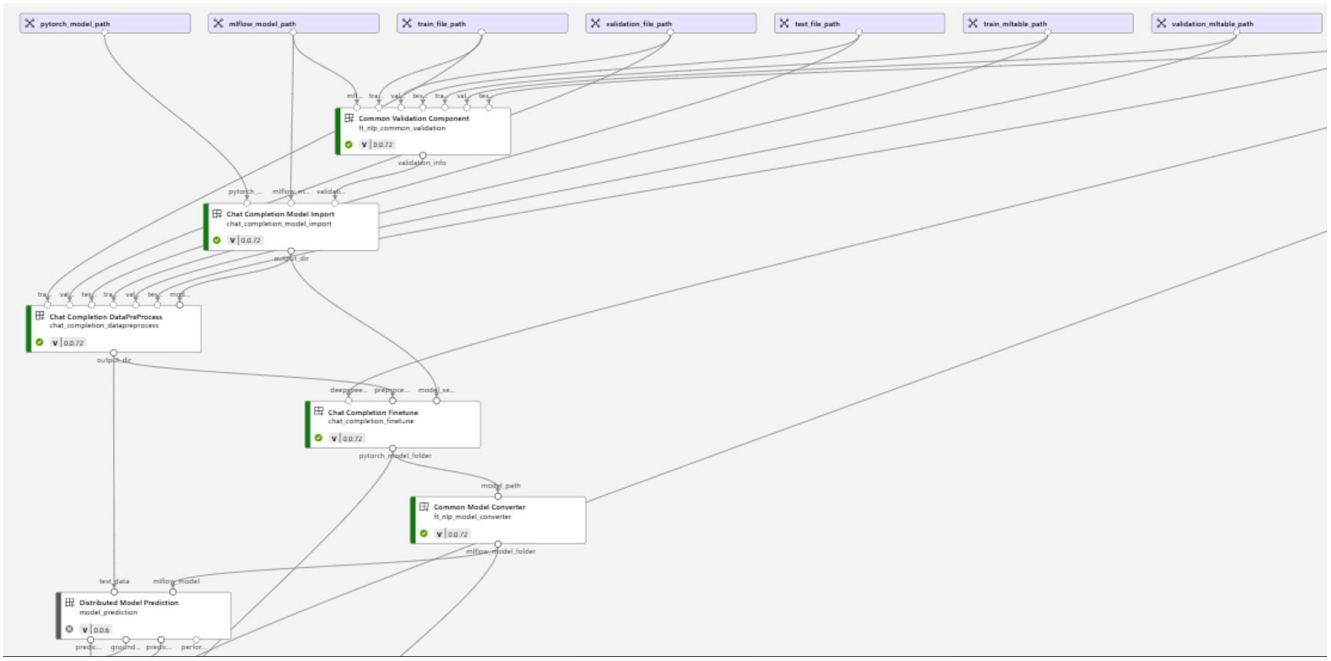

Рисунок 2.10 – Тренувальна джоба

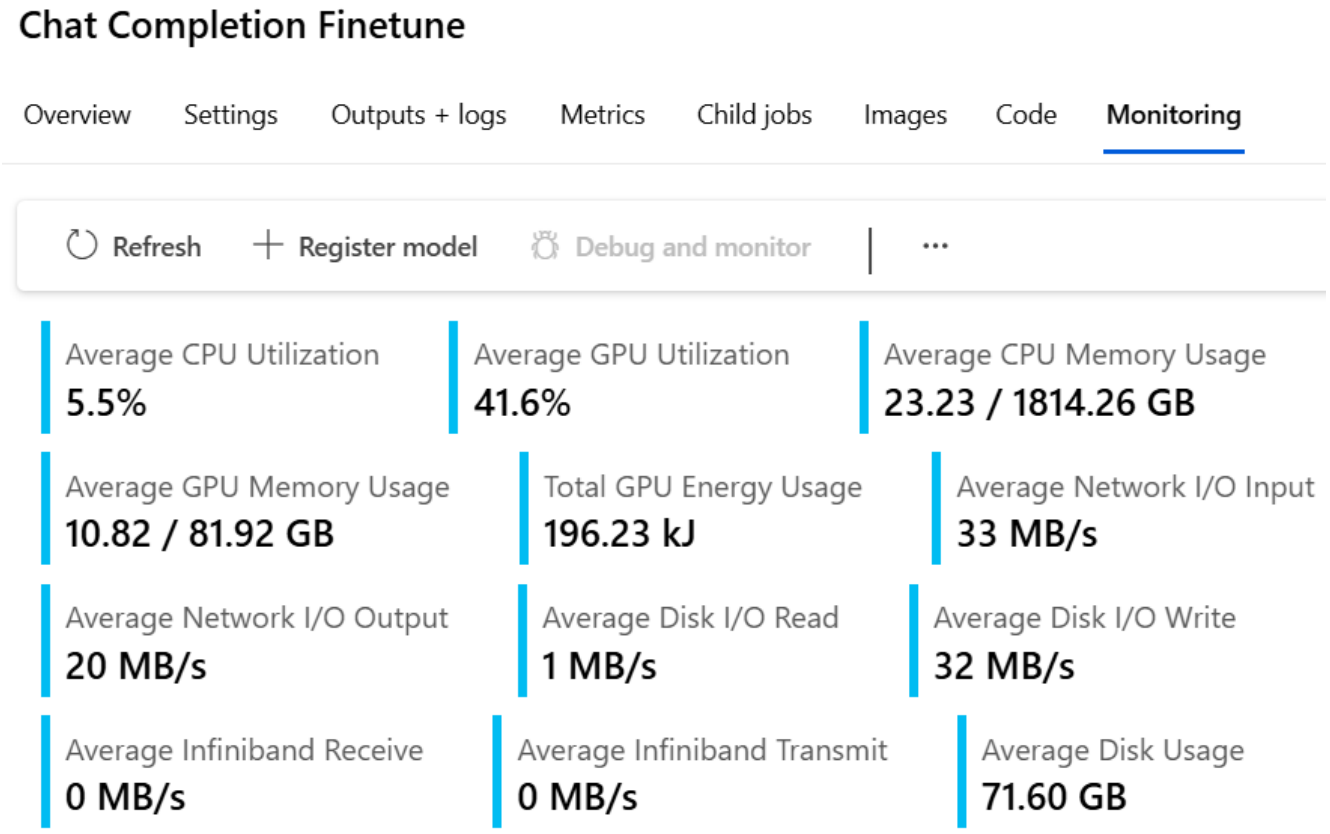

Рисунок 2.11 – Моніторинг використання ресурсів під час fine-tuning SLM

Важливою особливістю реалізованого підходу є можливість не чекати завершення всіх етапів пайплайну, зокрема автоматичної валідації та реєстрації

моделі. За необхідності тренувальну джобу можна зупинити після завершення етапу навчання та вручну зареєструвати модель з використанням MLflow-артефактів. Це забезпечує додаткову гнучкість і дозволяє використовувати власні скрипти для оцінювання якості моделі перед її подальшим розгортанням.

Таким чином, використання керованих обчислювальних ресурсів Azure Machine Learning у поєднанні з власним тренувальним скриптом забезпечило повний контроль над процесом тонкого налаштування малої мовної моделі. Реалізований підхід дозволив поєднати ефективність LoRA, масштабованість хмарної інфраструктури та вимоги до відтворюваності експериментів, що є критично важливим для розробки програмної системи з обмеженими обчислювальними ресурсами.

2.6 Реєстрація моделі та розгортання endpoint для інференсу

Після завершення процесу тонкого налаштування мовної моделі необхідним етапом є її реєстрація та підготовка до використання в робочому середовищі. Реєстрація моделі забезпечує централізоване управління версіями, можливість повторного використання результатів навчання та подальше розгортання моделі для інференсу. У межах даної роботи ці етапи реалізовано з використанням стандартних механізмів платформи Azure Machine Learning.

Реєстрація моделі виконується шляхом збереження артефактів навчання у форматі MLflow та їх додавання до реєстру моделей Azure ML. Такий підхід дозволяє зберігати не лише саму модель, але й супровідні метадані, зокрема опис моделі, параметри навчання та версію, що є важливим для відстеження експериментів і забезпечення відтворюваності результатів.

У процесі розробки було передбачено два сценарії реєстрації моделі. Перший сценарій передбачає автоматичну реєстрацію після завершення всіх етапів тренувального пайплайну. Другий сценарій, який активно використовувався в межах даної роботи, дозволяє зупинити тренувальну джобу після завершення етапу навчання та виконати реєстрацію моделі вручну. Такий підхід є доцільним у

випадках, коли оцінювання якості моделі виконується за допомогою власних скриптів або коли необхідно уникнути додаткових витрат часу та ресурсів на автоматичні етапи валідації.

Після реєстрації модель стає доступною для розгортання у вигляді кінцевої точки інференсу. Розгортання endpoint у середовищі Azure Machine Learning дозволяє використовувати модель як сервіс, що приймає текстові запити та повертає результат класифікації голосової пошти. Взаємодія з endpoint здійснюється через програмний інтерфейс, що відповідає REST-підходу та дозволяє інтегрувати модель з іншими компонентами програмної системи або зовнішніми сервісами.

Під час розгортання endpoint може бути налаштовано тип обчислювальних ресурсів, кількість екземплярів та параметри масштабування. Це дозволяє адаптувати систему до різного рівня навантаження та забезпечити стабільну роботу інференсу в умовах змінної кількості запитів. Такий підхід є особливо важливим для систем класифікації голосової пошти, які можуть обробляти значну кількість повідомлень у пікові періоди.

Відсутність графічного інтерфейсу користувача в межах розроблюваної системи є усвідомленим архітектурним рішенням. Оскільки модель використовується як сервіс для автоматизованої обробки даних, її інтеграція через програмний інтерфейс є більш доцільною, ніж створення окремого користувацького інтерфейсу. Це спрощує архітектуру системи, знижує витрати на розробку та відповідає типовим сценаріям використання AI-моделей у корпоративних інформаційних системах.

Таким чином, етапи реєстрації моделі та розгортання endpoint для інференсу забезпечують завершення циклу розробки програмної системи та її готовність до практичного використання. Реалізований підхід дозволяє ефективно управляти версіями моделей, контролювати використання обчислювальних ресурсів і забезпечувати інтеграцію з іншими компонентами хмарної інфраструктури.

2.7 Опис програмного інтерфейсу взаємодії з моделлю

Для інтеграції тонко налаштованої мовної моделі в програмну систему було реалізовано програмний інтерфейс взаємодії на основі кінцевої точки інференсу Azure Machine Learning. Такий підхід дозволяє використовувати модель як сервіс, забезпечуючи уніфікований доступ до неї з різних компонентів системи без необхідності створення окремого користувацького інтерфейсу.

Використання API для взаємодії з моделлю є доцільним рішенням для задач автоматизованої обробки голосової пошти, оскільки класифікація повідомлень виконується без участі користувача. Це зменшує складність архітектури, спрощує масштабування та відповідає типовим сценаріям використання AI-моделей у корпоративних інформаційних системах, де моделі інтегруються як бекенд-сервіси.

Підключення до розгорнутого endpoint виконується за допомогою спеціального клієнта, який інкапсулює параметри доступу, форматування запитів і налаштування генерації відповіді. Відповідний код ініціалізації клієнта для виклику моделі наведено на лістингу 2.8.

Лістинг 2.8 – Ініціалізація клієнта для виклику endpoint Azure ML

```
model = AzureMLChatOnlineEndpoint(
    endpoint_url=os.getenv("SLM_ENDPOINT_URL"),
    endpoint_api_type=AzureMLEndpointApiType.dedicated,
    endpoint_api_key=os.getenv("SLM_ENDPOINT_API_KEY"),
    content_formatter=CustomOpenAIChatContentFormatter(),
    model_kwargs={
        "do_sample": False,
        "temperature": 0.0,
        "top_p": 0.7,
        "max_tokens": 14,
        "num_beams": 1,
    }
)
```

У наведеному прикладі використовуються параметри генерації, орієнтовані на детермінований результат, що є важливим для задачі класифікації. Зокрема, нульове значення температури та вимкнене семплювання забезпечують

стабільність відповідей моделі, тоді як обмеження кількості токенів запобігає генерації надлишкового тексту.

Для отримання прогнозу від моделі реалізовано окрему функцію, яка приймає підготовлений список повідомлень та повертає результат класифікації. Такий підхід дозволяє ізолювати логіку виклику моделі та спростити її повторне використання в різних частинах системи. Код функції наведено на лістингу 2.9.

Лістинг 2.9 – Функція отримання відповіді від мовної моделі

```
def get_predictions(model, input_messages, model_name=None):
    response = model.predict_messages(input_messages)
    return response.content
```

Формат запиту до endpoint відповідає стандартному chat-підходу, де повідомлення передаються у вигляді списку ролей із текстовим контентом. Системне повідомлення задає чітку інструкцію для моделі щодо задачі класифікації та формату відповіді, що є важливим для коректної роботи інструкційних мовних моделей. Приклад формату запиту наведено на рисунку 2.12.

```
[
    {
        "role": "system",
        "content": (
            "You are a text classification model. Determine whether the following "
            "response is from a real person or an automated voicemail system.\n\n"
            "Return:\n"
            "Live Person - if it's a real person.\n"
            "Answering Machine - if it's a voicemail system.\n\n"
            "Provide only the classification result without additional explanations."
        )
    },
    {
        "role": "user",
        "content": (
            "forwarded to an automatic voice message system [PHONE] "
            "is not available at the tone please record your message"
        )
    }
]
```

Рисунок 2.12 – Формат запиту до endpoint-у моделі

У відповідь модель повертає «Live Person» або «Answering Machine» без додаткових пояснень, що спрощує подальшу автоматизовану обробку результатів.

Запропонований формат взаємодії забезпечує чітке розділення між логікою формування запиту, обробкою відповіді та бізнес-логікою системи. Це дозволяє легко модифікувати інструкції, змінювати параметри генерації або замінювати модель без необхідності суттєвих змін у коді клієнтської частини.

Таким чином, реалізований програмний інтерфейс взаємодії з моделлю забезпечує просту, масштабовану та ефективну інтеграцію мовної моделі в програмну систему класифікації голосової пошти. Обраний API-підхід відповідає сучасним практикам розробки AI-систем і створює основу для подальшого тестування, навантажувального аналізу та впровадження рішення в реальне середовище експлуатації.

2.8 Висновки до другого розділу

У другому розділі було розглянуто проєктування та реалізацію програмної системи класифікації голосової пошти на основі малої мовної моделі з використанням хмарних сервісів Azure Machine Learning та Azure AI Foundry. Основну увагу приділено вибору процесу розробки, архітектурних рішень та організації повного життєвого циклу машинного навчання з урахуванням обмежених обчислювальних ресурсів.

У межах розділу обґрунтовано вибір гібридного підходу Agile + MLOps, який дозволив поєднати ітеративність розробки з керованістю процесів навчання, розгортання та експлуатації мовної моделі. Було описано архітектуру системи, що передбачає використання мовної моделі як окремого сервісу з доступом через програмний інтерфейс, що забезпечує масштабованість та спрощує інтеграцію з іншими компонентами інформаційної системи.

Окрему увагу приділено організації зберігання даних і ML-артефактів, де фізичне управління базами даних передано хмарній платформі. Такий підхід дозволив зменшити складність інфраструктури та забезпечити централізоване

управління датасетами, експериментами і версіями моделей. Було детально описано процес формування синтетичного тренувального датасету, його автоматизовану обробку, розширення та розбиття на тренувальну, валідаційну і тестову вибірки.

У розділі також описано реалізацію процесу fine-tuning, який був виконаний у два етапи:

1) використання serverless-підходу для швидкої перевірки гіпотез;
2) використання керованих обчислювальних ресурсів Azure ML для повного контролю над навчанням.

Особливістю реалізованого рішення стало застосування підходу LoRA, який дозволив суттєво зменшити обчислювальні витрати без значної втрати якості моделі.

Завершальним етапом розділу стало описання процесів реєстрації моделі, її розгортання у вигляді endpoint для інференсу та реалізації програмного інтерфейсу взаємодії з моделлю. Обраний API-підхід забезпечив детерміновану поведінку моделі, простоту інтеграції та готовність системи до подальшого тестування і масштабування.

Таким чином, у другому розділі було сформовано цілісну основу програмної системи, що поєднує ефективні інженерні рішення, сучасні підходи до машинного навчання та практичні вимоги до використання малих мовних моделей у ресурсно обмежених середовищах. Отримані результати створюють підґрунтя для подальшого оцінювання якості, продуктивності та стійкості системи, що буде розглянуто в наступному розділі.

3 ТЕСТУВАННЯ, ОЦІНЮВАННЯ ТА НАВАНТАЖУВАЛЬНИЙ АНАЛІЗ ПРОГРАМНОЇ СИСТЕМИ

Тестування програмної системи є завершальним і водночас критично важливим етапом її розробки, оскільки саме на цьому етапі перевіряється відповідність отриманого рішення поставленим вимогам. Для систем на основі машинного навчання тестування виходить за межі класичної перевірки коректності роботи програмного коду та охоплює оцінювання якості моделей, стабільності інференсу та ефективності використання обчислювальних ресурсів. У цьому розділі здійснюється комплексний аналіз розробленої системи класифікації голосової пошти з використанням кількісних та експериментальних методів.

3.1 Загальна методологія тестування програмної системи

Методологія тестування розроблюваної програмної системи ґрунтується на поєднанні підходів програмної інженерії та практик оцінювання моделей машинного навчання. Оскільки система побудована навколо мовної моделі, основною метою тестування є перевірка не лише коректності взаємодії компонентів, але й якості, надійності та продуктивності процесу класифікації голосової пошти.

Тестування системи проводилося у декілька взаємопов’язаних етапів. На першому етапі виконувалося функціональне тестування, спрямоване на перевірку коректності роботи програмного інтерфейсу взаємодії з моделлю. Зокрема, перевірялася правильність формування запитів до endpoint, обробка відповідей та відповідність формату результатів очікуваним значенням. Це дозволило впевнитися, що система стабільно виконує базову функцію класифікації повідомлень без помилок на рівні API-взаємодії [19].

Другий етап тестування був зосереджений на оцінюванні якості роботи мовних моделей. Для цього використовувалися заздалегідь підготовлені тестові та валідаційні вибірки, сформовані в процесі обробки синтетичного датасету.

Оцінювання виконувалося за стандартними метриками класифікації, що дозволяє об’єктивно порівнювати результати різних моделей та конфігурацій. Такий підхід відповідає загальноприйнятим практикам оцінювання моделей машинного навчання та забезпечує відтворюваність експериментів [14].

Окрему увагу в межах тестування було приділено аналізу продуктивності системи. Оскільки модель використовується у вигляді сервісу, важливими показниками є затримка відповіді та стабільність роботи при зростанні кількості запитів. Для цього застосовувалося навантажувальне тестування з імітацією паралельних викликів endpoint та аналізом масштабування системи при зміні кількості обчислювальних ресурсів. Такий підхід дозволяє оцінити придатність системи до використання в реальних умовах експлуатації.

Загальна методологія тестування була побудована таким чином, щоб забезпечити всебічну оцінку програмної системи: від правильності її функціонування до ефективності використання обчислювальних ресурсів. Отримані результати тестування стали основою для подальшого порівняльного аналізу моделей, який буде представлено в наступних підрозділах цього розділу.

3.2 Оцінювання якості класифікації голосової пошти

Оцінювання якості роботи програмної системи класифікації голосової пошти є ключовим етапом тестування, оскільки саме воно дозволяє визначити, наскільки ефективно мовна модель виконує поставлену задачу. Для систем на основі машинного навчання якість класифікації визначається не лише загальною точністю, але й здатністю моделі коректно розпізнавати різні класи в умовах реалістичних даних. У межах даної роботи оцінювання здійснювалося з використанням стандартних метрик бінарної класифікації.

Оцінювання якості виконувалося на окремій тестовій вибірці, яка не використовувалася під час навчання або валідації моделі. Такий підхід дозволяє отримати об’єктивну оцінку узагальнювальної здатності моделі та зменшує ризик

перенавчання. Для всіх моделей застосовувався однаковий формат вхідних даних і ідентичні інструкції, що забезпечувало коректність порівняння результатів [14].

Основною метрикою оцінювання була Accuracy, яка відображає частку правильно класифікованих прикладів від загальної кількості тестових зразків. Дана метрика є інтуїтивно зрозумілою та дозволяє отримати загальне уявлення про ефективність моделі. Однак у задачах класифікації голосової пошти використання лише Accuracy може бути недостатнім, оскільки помилки різних типів можуть мати різний вплив на практичне використання системи.

Для більш детального аналізу якості класифікації використовувалися метрики Precision та Recall. Precision характеризує частку правильних позитивних передбачень серед усіх позитивних відповідей моделі та є важливою у сценаріях, де критичними є хибнопозитивні помилки. У контексті класифікації голосової пошти це дозволяє зменшити кількість випадків, коли автоматичне повідомлення помилково класифікується як жива людина. Recall, у свою чергу, відображає здатність моделі знаходити всі релевантні позитивні приклади та є важливим для мінімізації хибнонегативних результатів [20].

Для узагальненої оцінки балансу між Precision та Recall використовувалася метрика F1-score, яка є гармонійним середнім цих двох показників. Використання F1-score є доцільним у випадках, коли необхідно враховувати як повноту, так і точність класифікації, особливо за наявності потенційного дисбалансу між класами [38].

Окрім показників якості класифікації, додатково аналізувалася затримка відповіді (latency) моделі під час інференсу. Даний показник не впливає безпосередньо на коректність класифікації, проте є критично важливим для оцінювання придатності системи до практичного використання, особливо в умовах обробки великої кількості повідомлень у режимі близькому до реального часу. Аналіз затримки дозволяє пов’язати якість моделі з вимогами до продуктивності та обчислювальних ресурсів.

Таким чином, у межах даного підрозділу було визначено набір метрик та підхід до оцінювання якості класифікації голосової пошти, який забезпечує

всебічний аналіз ефективності мовних моделей. Отримані значення метрик стануть основою для порівняльного аналізу великих і малих мовних моделей, який буде представлено в наступному підрозділі.

3.3 Порівняльний аналіз використання малих та великих мовних моделей

Порівняльний аналіз малих та великих мовних моделей проводився з метою оцінювання доцільності використання SLM як альтернативи LLM у системах з обмеженими обчислювальними ресурсами. Аналіз базується на двох ключових аспектах: якості класифікації голосової пошти та економічній ефективності хостингу моделей у хмарному середовищі. Такий підхід дозволяє комплексно оцінити практичну придатність моделей для реального використання.

Для порівняння було використано наступні LLMs без тонкого налаштування:

- gpt-4o-mini;
- gpt-35-turbo-0125.

Для тонкого налаштування на Serverless API було обрано наступні SLMs:

- Llama-3.1-8B-Instruct;
- Phi-3.5-mini-instruct;
- Phi-4-mini-instruct;
- Ministral-3B.

Оцінювання якості класифікації виконувалося за такими метриками:

- Accuracy
- Precision
- Recall
- F1-score
- та середньої затримки відповіді.

В таблиці 3.1. зображено результати тестування LLM та тонко налаштованих SLM на Serverless API.

Таблиця 3.1 – Порівняння якості класифікації та затримки інференсу моделей

Model	Accuracy, %	Precision, %	F1 score, %	Recall, %	Latency, s	Availability
gpt-4o-mini	95.38	97.20	95.21	93.29	0.4822	Serverless
gpt-35-turbo-0125	98.68	99.32	98.65	97.99	0.2405	Serverless
Llama-3.1-8B-Instruct	98.68	98.66	98.66	98.66	0.8130	Serverless / VM
Phi-3.5-mini-instruct	98.68	99.32	98.65	97.99	0.8025	Serverless / VM
Phi-4-mini-instruct	94.06	99.25	93.62	88.59	0.7272	Serverless / VM
Ministral-3B	79.21	74.16	80.73	88.59	0.6018	Serverless

З результатів тестування видно, що моделі gpt-35-turbo-0125, Llama-3.1-8B-Instruct та Phi-3.5-mini-instruct демонструють найвищі показники якості класифікації, практично не поступаючись одна одній. Це підтверджує можливість досягнення рівня якості, співставного з LLM, за допомогою тонко налаштованих SLM.

Модель Phi-4-mini-instruct демонструє нижчі значення Accuracy та Recall, проте характеризується високою Precision, що вказує на меншу кількість хибнопозитивних результатів. Для задачі класифікації голосової пошти це є важливим фактором, оскільки помилкове визначення автовідповідача як живої людини може негативно впливати на подальші бізнес-процеси. Водночас модель Ministral-3B показала найгірші результати за більшістю метрик, що обмежує її практичне застосування в межах даної системи.

Окрім якості класифікації, важливим критерієм вибору моделі є вартість її використання. Порівняння витрат на хостинг моделей у serverless-середовищі та на керованих обчислювальних ресурсах наведено в таблиці 3.2 [21].

Таблиця 3.2 – Порівняння вартості хостингу мовних моделей

Model	**Serverless hosting**		**Compute hosting**		**Input cost, $/1000 tokens**	**Output cost, $/1000 tokens**
	$/hour	**$/month**	**$/hour**	**$/month**		
gpt-4o-mini	-	-	-	-	0.00015	0.0006
gpt-35-turbo-0125	1.7	1241	-	-	0.00055	0.00165
Llama-3.1-8B-Instruct	0.74	540.2	7.3460	5,362.58	0.0003	0.00061
Phi-3.5-mini-instruct	0.80	584	3.67	2,681.29	0.00013	0.00052
Phi-4-mini-instruct	0.80	584	3.67	2,681.29	0.000075	0.0003
Ministral-3B	0.65	474.5	3.67	2,681.29	0.00004	0.00004

Не зважаючи на свої хороші результати, Llama-3.1-8B-Instruct потребує значно потужніших обчислювальних ресурсів для навчання та інференсу, що призводить до збільшення затримки відповіді та вартості хостингу, особливо у сценаріях використання керованих віртуальних машин. А Phi-3.5-mini-instruct на момент проведення дослідження втратила частину функціональних можливостей щодо тонкого налаштування в середовищі Azure AI Foundry, що створювало ризики для подальшої інтеграції та підтримки в межах розроблюваної системи.

З урахуванням сукупності факторів — прийнятного рівня якості, високої точності прогнозів, стабільної підтримки в Azure Machine Learning, можливості використання як у serverless-середовищі, так і на керованих обчислювальних ресурсах — як базову модель для подальшого тонкого налаштування було обрано Phi-4-mini-instruct.

Після вибору базової моделі було виконано її тонке налаштування на керованих обчислювальних ресурсах Azure Machine Learning з використанням власного тренувального скрипта. Результати тренування зображені на рисунку 3.1.

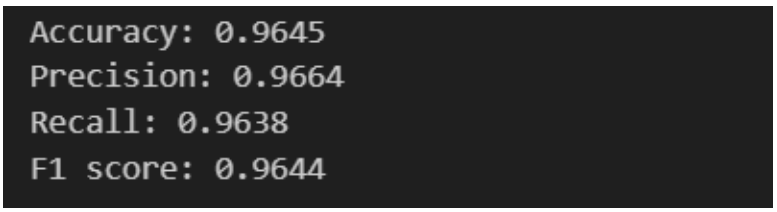

Рисунок 3.1 – Результати тренування Phi-4-mini-instruct на керованих обчислювальних ресурсах

Отримані результати свідчать про покращення якості класифікації порівняно з serverless fine-tuning, що підтверджує ефективність використання керованих обчислювальних ресурсів для задачі класифікації голосової пошти, оскільки розширені можливості конфігурації навчального процесу дозволяють точніше адаптувати модель до специфіки предметної області.

Також було досягнуто нижчої затримки моделі. Це зображено на рисунку 3.2.

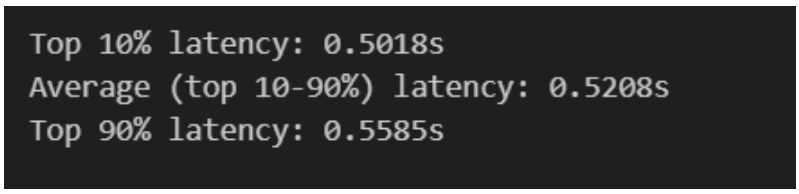

Рисунок 3.2 – Результати затримки тонко налаштованої Phi-4-mini-instruct на керованих обчислювальних ресурсах

Таким чином, тонке налаштування моделі Phi-4-mini-instruct на керованих обчислювальних ресурсах стало завершальним етапом вибору та підготовки мовної моделі для подальшої експлуатації. Отримані показники якості підтверджують доцільність обраного підходу та створюють основу для наступного етапу дослідження – аналізу продуктивності системи під навантаженням та оцінювання її масштабованості.

3.4 Навантажувальне тестування програмної системи та аналіз масштабування

Навантажувальне тестування програмної системи класифікації голосової пошти виконувалося з метою оцінювання її продуктивності, стабільності та здатності до масштабування в умовах зростання кількості запитів. Оскільки розроблена система використовує мовну модель, розгорнуту у вигляді endpoint в Azure Machine Learning, важливими показниками є кількість оброблюваних запитів за секунду (RPS), рівень помилок та поведінка системи при зміні кількості інстансів.

Для проведення навантажувального тестування було використано інструмент k6, який дозволяє моделювати паралельні HTTP-запити та аналізувати продуктивність сервісів у контрольованих умовах [22]. Тестування виконувалося шляхом надсилання запитів до endpoint моделі Phi-4-mini-instruct із використанням однакового формату вхідних даних та фіксованих параметрів інференсу.

Перед початком навантажувального тестування було проаналізовано стандартну конфігурацію endpoint, яка за замовчуванням передбачає менші значення параметрів, що обмежують кількість одночасно оброблюваних запитів. Зокрема, початкові значення параметрів WORKER_COUNT та max_concurrent_requests_per_instance є заниженими з точки зору високонавантажених сценаріїв і можуть суттєво обмежувати пропускну здатність системи.

З метою коректного оцінювання максимальної продуктивності розробленої програмної системи перед проведенням навантажувального тестування ці параметри були змінені вручну. Для тестових запусків встановлено наступні значення:

- WORKER_COUNT: 32
- max_concurrent_requests_per_instance: 32

Збільшення зазначених параметрів дозволило розширити можливості паралельної обробки запитів кожним інстансом endpoint та забезпечило більш

репрезентативну оцінку пропускної здатності системи в умовах підвищеного навантаження.

Перший етап навантажувального тестування проводився з використанням одного інстансу endpoint. У цьому режимі система продемонструвала стабільну роботу з пропускною здатністю близько 78 запитів за секунду (RPS) без втрати відповідей або зростання кількості помилок. Це зображено на рисунку 3.3.

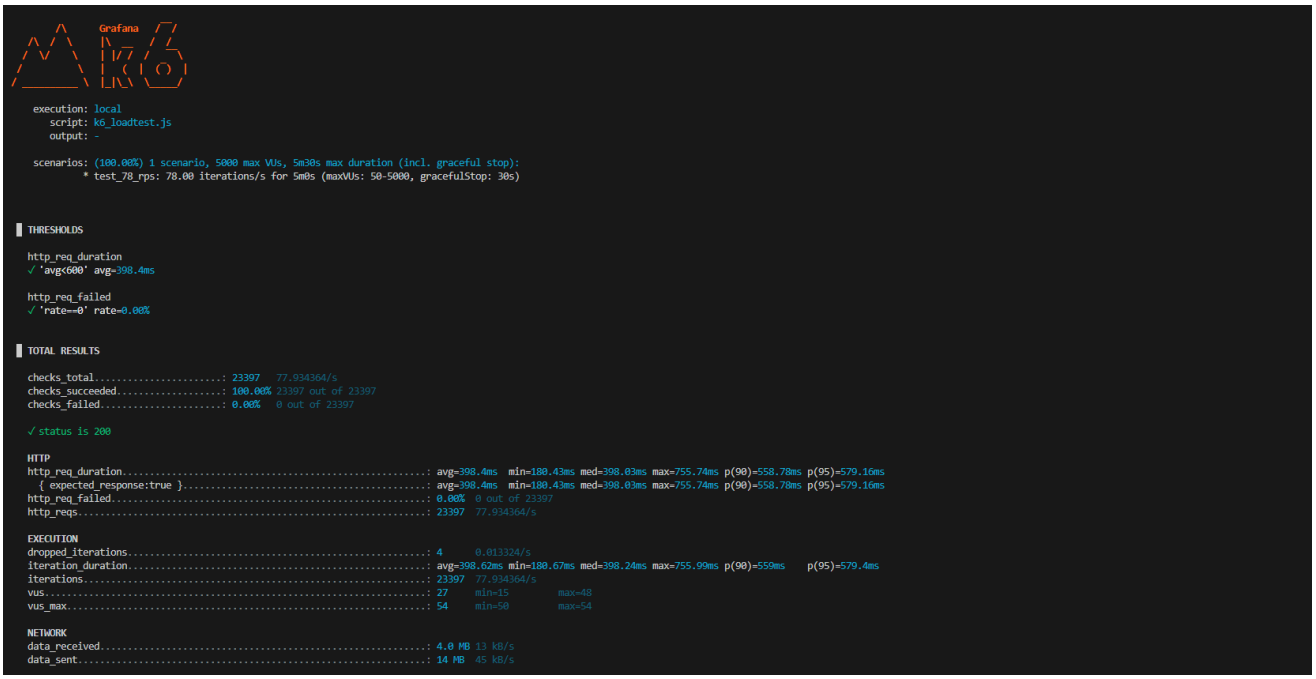

Рисунок 3.3 – Результатів k6-тестування для 1 інстансу

При подальшому збільшенні навантаження, що перевищувало можливості одного інстансу за заданих обмежень конфігурації, було зафіксовано появу помилок обробки запитів. Частина HTTP-запитів завершувалася невдало, що свідчить про досягнення граничної пропускної здатності одного інстансу endpoint. Невдале завершення тесту зображено на рисунку 3.4.

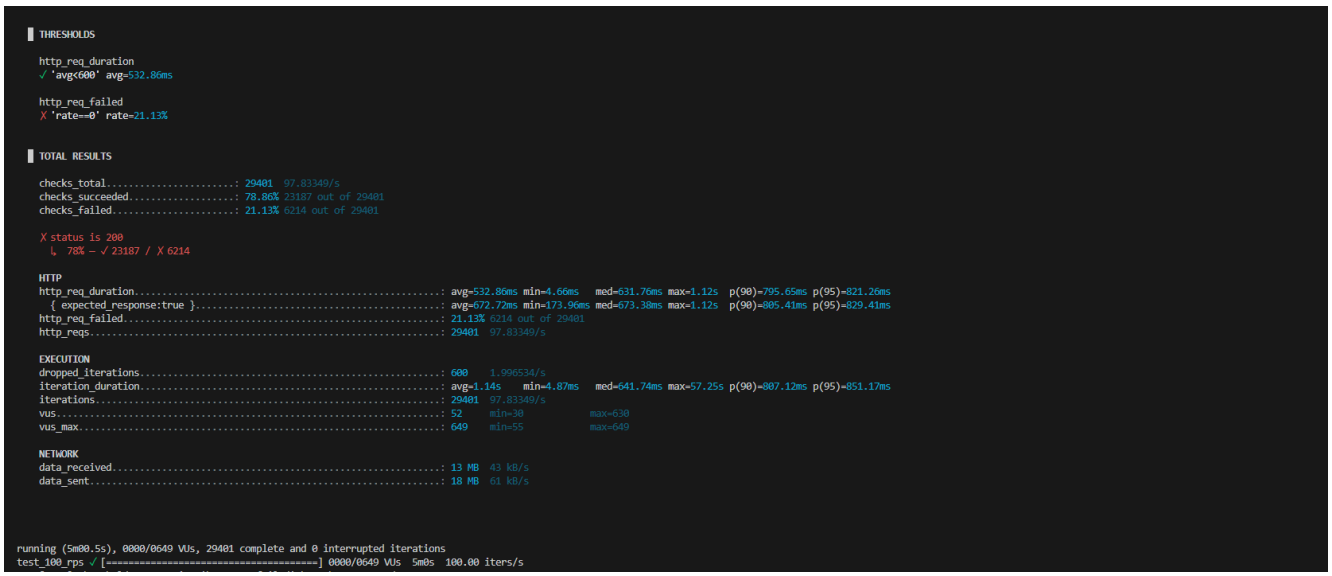

Рисунок 3.4 – Отримання помилок під час навантажувального тесту

Для підвищення продуктивності системи було виконано масштабування endpoint до двох інстансів через користувацький інтерфейс Azure, який зображений на рисунку 3.5.

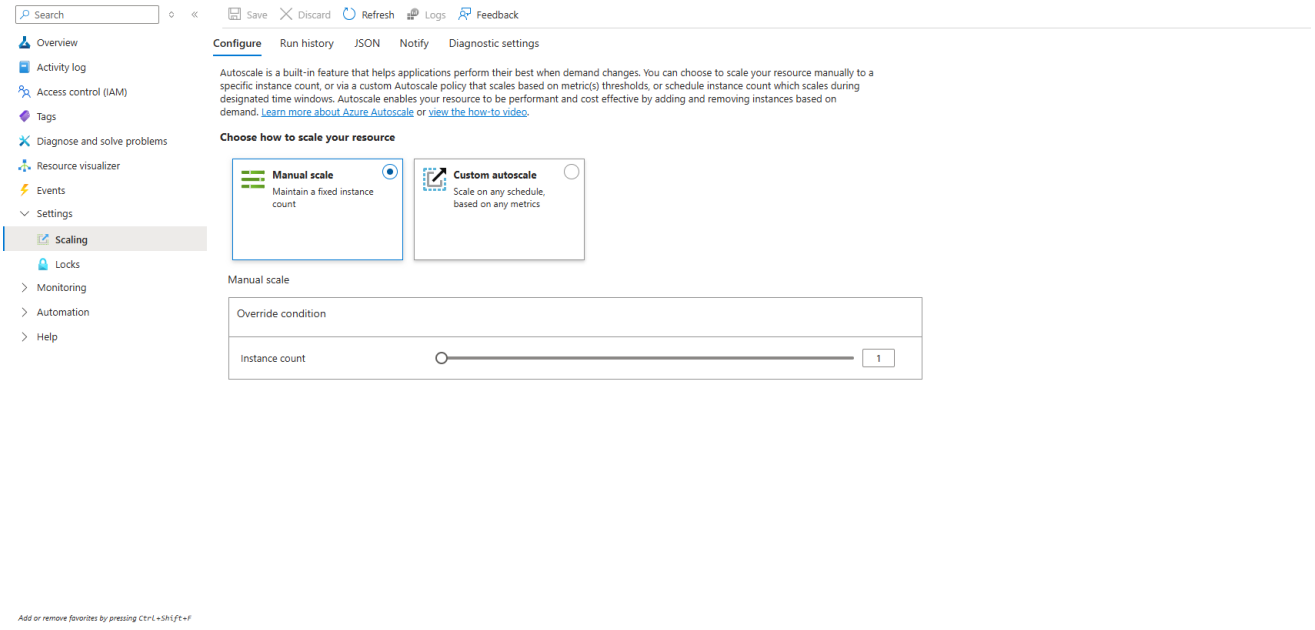

Рисунок 3.5 – Інтерфейс масштабування endpoint

Масштабування здійснювалося засобами Azure Machine Learning шляхом збільшення кількості активних інстансів без зміни логіки роботи системи. У результаті пропускна здатність зросла до приблизно 125 RPS, при цьому кількість

помилок була відсутня, про що свідчить результат тесту, який представлений на рисунку 3.6.

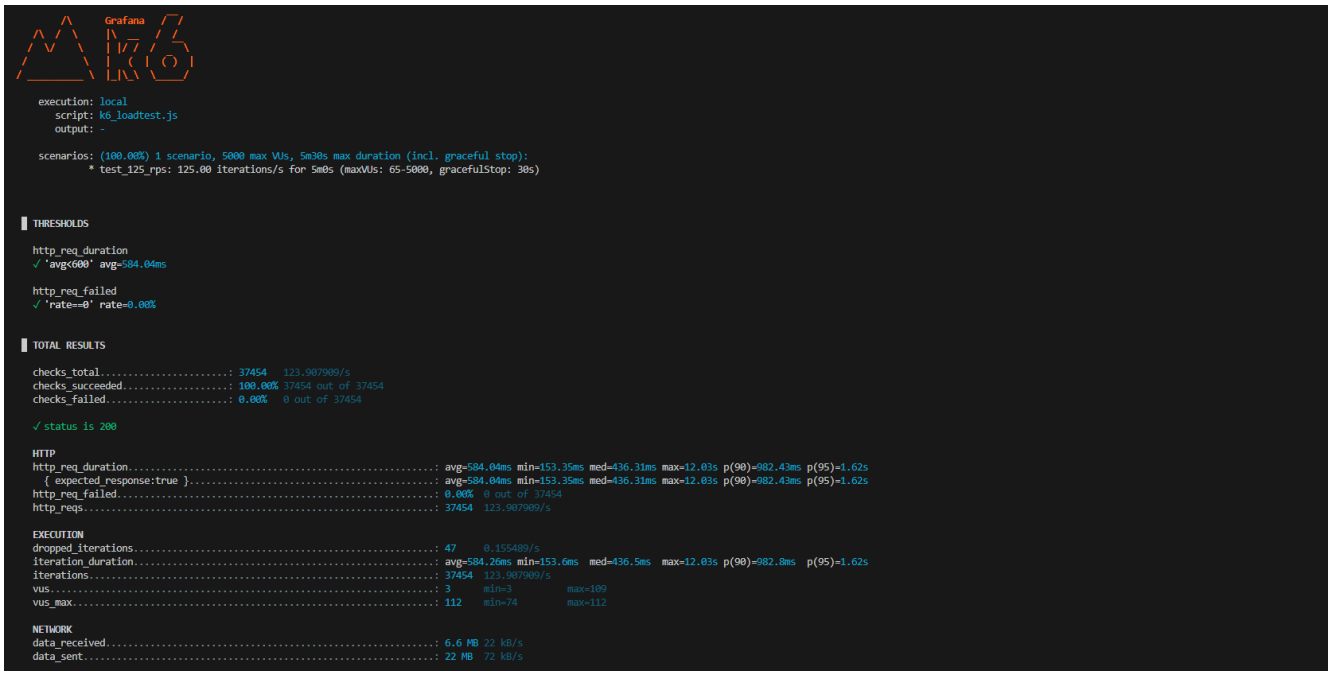

Рисунок 3.6 – Результат навантажувального тесту ендпоінту на двох інстансах

На наступному етапі тестування було виконано масштабування endpoint до трьох інстансів. За таких умов система продемонструвала пропускну здатність понад 150 RPS, зберігаючи стабільну обробку запитів без суттєвого зростання затримки або кількості помилок. Це зображено на рисунку 3.7.

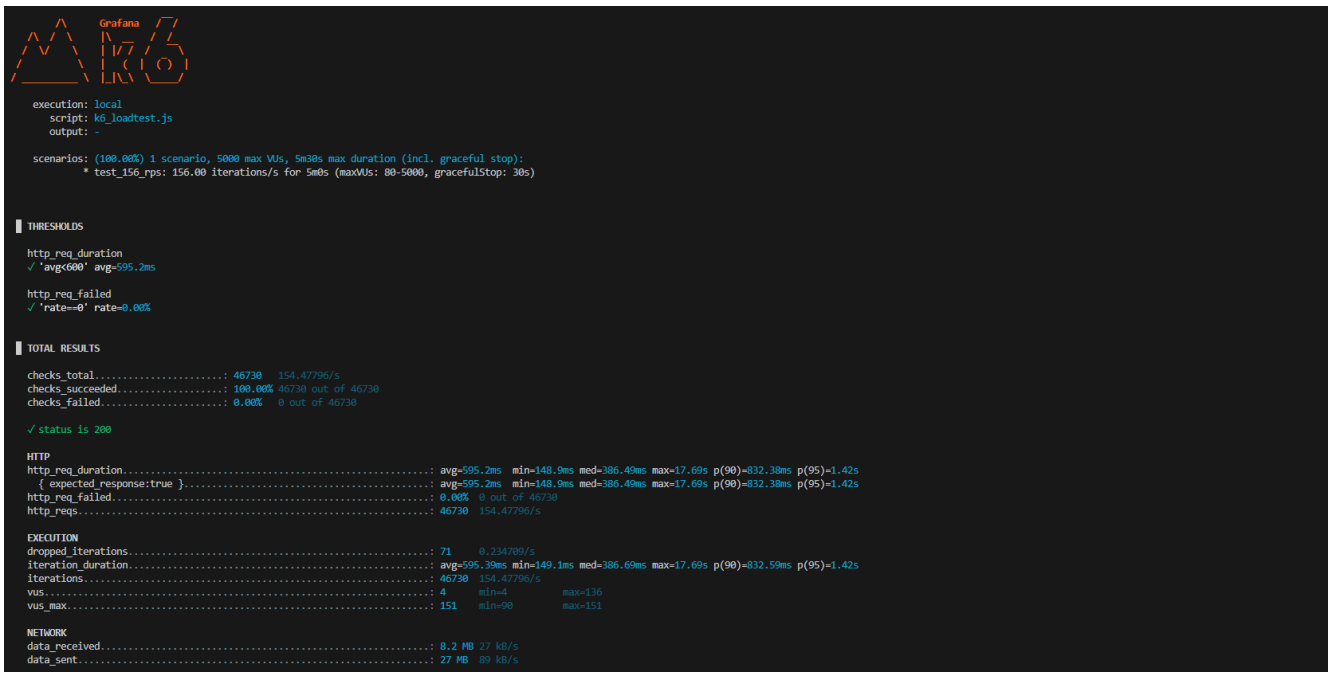

Рисунок 3.7 – Тестування ендпоінту моделі на трьох інстансах

Отримані результати свідчать про лінійне зростання пропускної здатності системи при збільшенні кількості інстансів, що підтверджує коректність обраної архітектури та ефективність масштабування endpoint. Важливою перевагою платформи Azure Machine Learning є можливість налаштування автоматичного масштабування, яке дозволяє динамічно змінювати кількість інстансів залежно від поточного навантаження. Це забезпечує баланс між продуктивністю системи та витратами на обчислювальні ресурси.

Таким чином, навантажувальне тестування підтвердило, що розроблена програмна система на основі малої мовної моделі Phi-4-mini-instruct здатна стабільно працювати під значним навантаженням та ефективно масштабуватися відповідно до зростання кількості запитів. Отримані результати демонструють практичну придатність системи для використання в реальних умовах експлуатації та завершують експериментальну частину дослідження.

3.5 Висновки до третього розділу

У третьому розділі було виконано комплексне тестування та експериментальне оцінювання розробленої програмної системи класифікації голосової пошти на основі малої мовної моделі. Основну увагу приділено перевірці якості класифікації, продуктивності інференсу та здатності системи до масштабування в умовах зростання навантаження, що є критично важливим для практичного використання AI-рішень у промислових середовищах.

У межах розділу було визначено та обґрунтовано набір метрик для оцінювання якості класифікації, зокрема Accuracy, Precision, Recall та F1-score. Проведене порівняння великих та малих мовних моделей показало, що тонко налаштовані SLM здатні досягати рівня якості, співставного з великими мовними моделями, при значно нижчих вимогах до обчислювальних ресурсів та вартості використання. Це підтверджує доцільність застосування малих мовних моделей у задачах класифікації голосової пошти.

Результати порівняльного аналізу дозволили обґрунтовано обрати модель Phi-4-mini-instruct як базову для подальшої експлуатації. Подальше тонке налаштування цієї моделі на керованих обчислювальних ресурсах Azure Machine Learning із використанням власного тренувального скрипта забезпечило покращення показників якості класифікації порівняно з serverless-підходом. Отримані значення метрик підтвердили ефективність використання керованих ресурсів для фінального навчання моделі.

Навантажувальне тестування із застосуванням інструмента k6 продемонструвало, що розроблена система здатна стабільно обробляти значну кількість запитів за секунду та ефективно масштабуватися шляхом збільшення кількості інстансів endpoint. Аналіз результатів показав лінійне зростання пропускної здатності системи при масштабуванні, а також дозволив виявити обмеження одного інстансу за умов високого навантаження. Можливість налаштування автоматичного масштабування в Azure Machine Learning створює додаткові передумови для адаптації системи до змінних робочих навантажень.

Таким чином, результати тестування та експериментального аналізу підтверджують, що розроблена програмна система відповідає поставленим вимогам щодо якості, продуктивності та масштабованості. Отримані результати створюють практичне підґрунтя для впровадження системи в реальне середовище експлуатації та завершують експериментальну частину магістерської роботи.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Розділ присвячено питанням охорони праці та забезпечення безпеки в надзвичайних ситуаціях під час розробки та експлуатації програмних систем на основі технологій штучного інтелекту. У ньому розглядаються потенційні небезпеки, що виникають під час роботи з комп'ютерною технікою та хмарною інфраструктурою, а також заходи щодо їх мінімізації відповідно до чинних нормативно-правових актів і державних стандартів України.

4.1 Охорона праці

Під час розробки програмної системи малої мовної моделі для систем з обмеженими обчислювальними ресурсами особлива увага приділялася питанням охорони праці та пожежної безпеки. Роботи з проєктування, навчання та тестування мовних моделей виконуються з використанням комп'ютерної техніки та хмарної інфраструктури, що потребує дотримання встановлених норм безпеки праці, санітарно-гігієнічних вимог та правил пожежної безпеки.

Відповідно до Закону України «Про охорону праці», охорона праці визначається як система правових, соціально-економічних, організаційно-технічних та лікувально-профілактичних заходів і засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності [23]. Під час виконання робіт, пов'язаних із розробкою програмного забезпечення, основними шкідливими та небезпечними чинниками є тривала робота за комп'ютером, підвищене зорове навантаження, статичне навантаження на опорно-руховий апарат, а також вплив електромагнітного випромінювання від засобів обчислювальної техніки.

Вимоги до організації робочого місця програміста регламентуються державними санітарними нормами та правилами, зокрема НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [24]. Згідно з цими нормами, робоче місце повинно бути обладнане

ергономічним столом і кріслом з регулюванням висоти, а монітор розташований на відстані 50–70 см від очей користувача. Освітлення робочої зони має бути рівномірним, без відблисків на екрані, з рівнем освітленості не менше 300–500 лк.

Тривалість безперервної роботи за комп'ютером повинна відповідати встановленим нормам, з обов'язковими перервами для відпочинку та зменшення зорового навантаження. Для працівників, діяльність яких пов'язана з інтенсивною розумовою працею та використанням комп'ютерної техніки, рекомендовано робити регламентовані перерви тривалістю 10–15 хвилин кожні 1–2 години роботи [24].

Питання пожежної безпеки під час експлуатації комп'ютерної техніки регламентуються Кодексом цивільного захисту України та Правилами пожежної безпеки в Україні [25, 26]. Приміщення, у яких розміщується комп'ютерне обладнання, повинні бути оснащені справною електромережею, автоматичними вимикачами та первинними засобами пожежогасіння. Забороняється використання пошкоджених кабелів живлення, перевантаження електромережі та залишення увімкненого обладнання без нагляду.

Згідно з вимогами ДСТУ EN 62305 «Захист від блискавки. Загальні положення.» та загальних правил електробезпеки, усі електротехнічні пристрої повинні бути заземлені, а доступ до них — безпечним для користувачів [27]. Дотримання цих вимог зменшує ризик ураження електричним струмом та виникнення аварійних ситуацій під час роботи з обчислювальною технікою.

Отже, під час розробки та експлуатації програмної системи класифікації голосової пошти було враховано основні вимоги охорони праці, санітарно-гігієнічні норми та правила пожежної безпеки. Дотримання чинних законодавчих і нормативних документів забезпечує безпечні умови праці, збереження здоров'я розробника та зменшує ризик виникнення надзвичайних ситуацій під час виконання робіт.

4.2 Стійкість роботи обчислювальної системи під час НС воєнного часу

У сучасних умовах воєнного стану питання безпеки в надзвичайних ситуаціях набуває особливої актуальності, зокрема для забезпечення стійкості роботи обчислювальних систем. Надзвичайні ситуації воєнного характеру створюють підвищені ризики для життя і здоров’я персоналу, а також для цілісності технічної та інформаційної інфраструктури. Втрата електропостачання, пошкодження будівель, порушення каналів зв’язку та загрози інформаційній безпеці можуть призвести до повної або часткової зупинки функціонування обчислювальної системи. Тому організація заходів безпеки в надзвичайних ситуаціях повинна бути спрямована як на захист персоналу, так і на збереження працездатності критично важливих інформаційних ресурсів [28].

Законодавчою основою забезпечення безпеки в надзвичайних ситуаціях є Кодекс цивільного захисту України, у якому зазначено, що *«цивільний захист спрямований на захист населення, територій, навколишнього природного середовища та майна від надзвичайних ситуацій шляхом запобігання таким ситуаціям, ліквідації їх наслідків і надання допомоги постраждалим»* [30]. Дотримання вимог цивільного захисту є обов’язковим для всіх підприємств, установ і організацій незалежно від форми власності, у тому числі для тих, що експлуатують обчислювальні системи та інформаційні технології.

У методичному посібнику В.С. Стручка наголошується, що безпека в надзвичайних ситуаціях передбачає завчасну підготовку персоналу, створення систем оповіщення, розроблення планів реагування та чітке визначення дій працівників у разі виникнення небезпеки [28]. Зазначається, що ефективність заходів цивільного захисту значною мірою залежить від рівня поінформованості працівників і регулярного проведення навчань та тренувань.

Для обчислювальних систем у період воєнного часу характерними є такі основні загрози: фізичне пошкодження обладнання внаслідок бойових дій, тривалі відключення електроенергії, збої в роботі телекомунікаційних мереж, а також підвищена активність кібератак. У навчальному посібнику «Техноекологія та

цивільна безпека» зазначено, що при оцінюванні ризиків надзвичайних ситуацій необхідно враховувати як ймовірність настання події, так і масштаби можливих наслідків для персоналу та матеріальних цінностей [29]. Такий підхід дозволяє визначити пріоритетні напрями захисту та розробити адекватні заходи реагування.

Важливим організаційним заходом є розроблення плану цивільного захисту для підрозділу, що обслуговує обчислювальну систему. Згідно з методичними рекомендаціями ДСНС України, у плані повинні бути визначені порядок оповіщення персоналу, маршрути евакуації, місця укриття, відповідальні особи та алгоритми дій у разі виникнення надзвичайної ситуації воєнного характеру [31]. Наявність такого плану забезпечує злагоджені дії персоналу та мінімізує втрати часу у критичних ситуаціях.

Особлива увага в умовах надзвичайних ситуацій приділяється забезпеченню електробезпеки та пожежної безпеки. Відповідно до Правил пожежної безпеки в Україні, *«електроустановки та електропроводка повинні експлуатуватися відповідно до вимог нормативних документів, а у разі виникнення пожежі персонал зобов'язаний діяти згідно з установленими інструкціями»* [32]. Для обчислювальних систем це означає необхідність використання джерел безперебійного живлення, резервних генераторів, систем заземлення та автоматичного захисту електромереж.

З метою забезпечення стійкості функціонування обчислювальної системи в умовах воєнного часу важливу роль відіграє захист і збереження даних. Згідно з міжнародним стандартом ISO 22301, організація повинна *«планувати та впроваджувати заходи для забезпечення безперервності діяльності та відновлення критичних процесів у разі надзвичайних ситуацій»* [33]. Практична реалізація цих вимог передбачає регулярне резервне копіювання даних, географічне рознесення копій, а також тестування процедур відновлення.

Не менш важливим аспектом безпеки в надзвичайних ситуаціях є психофізіологічна безпека персоналу. У посібниках з цивільної безпеки підкреслюється, що під час воєнних дій працівники можуть зазнавати значного психологічного навантаження, що негативно впливає на їхню працездатність і

здатність приймати правильні рішення [28], [29]. Тому роботодавець повинен передбачати заходи щодо раціональної організації праці, чергувань, відпочинку та надання психологічної підтримки.

Крім того, в умовах воєнного стану зростає значення інформаційної безпеки. Державні рекомендації у сфері захисту інформації наголошують, що *«заходи інформаційної безпеки повинні бути посилені з урахуванням підвищених ризиків несанкціонованого доступу та кібератак»* [34]. Це передбачає застосування багатофакторної автентифікації, розмежування прав доступу, шифрування даних і постійний моніторинг подій безпеки.

Отже, безпека в надзвичайних ситуаціях є невід'ємною складовою забезпечення стійкості роботи обчислювальної системи під час воєнного часу. Реалізація вимог цивільного захисту, організація захисту персоналу, впровадження технічних і організаційних заходів безпеки, а також використання методичних рекомендацій і міжнародних стандартів дозволяють мінімізувати наслідки надзвичайних ситуацій і забезпечити безперервність функціонування критично важливих інформаційних систем.

4.3 Висновки до четвертого розділу

У розділі розглянуто основні вимоги охорони праці та безпеки в надзвичайних ситуаціях під час розробки й експлуатації програмної системи з використанням обчислювальної техніки та хмарної інфраструктури. Проаналізовано нормативно-правові вимоги щодо організації безпечних умов праці, пожежної безпеки та цивільного захисту.

Отже, дотримання чинних законодавчих і нормативних документів, а також впровадження організаційних і технічних заходів безпеки забезпечує захист персоналу та стійке функціонування програмної системи, у тому числі в умовах надзвичайних ситуацій і воєнного стану.

ВИСНОВКИ

У межах даної магістерської кваліфікаційної роботи було розв’язано науково-технічну задачу розробки програмної системи класифікації голосової пошти на основі малої мовної моделі, адаптованої для роботи в умовах обмежених обчислювальних ресурсів. Запропоноване рішення орієнтоване на практичне використання в інформаційних системах та забезпечує поєднання високої якості класифікації, масштабованості та економічної доцільності.

У роботі проведено аналіз предметної області та існуючих підходів до використання великих і малих мовних моделей для задач обробки природної мови. На основі цього виконано порівняння великої мовної моделі gpt-35-turbo-0125 як представника LLM та малої мовної моделі Phi-4-mini-instruct, тонко налаштованої на керованих обчислювальних ресурсах, як представника SLM. Отримані результати підтвердили, що за умови адаптації до предметної області малі мовні моделі можуть демонструвати якість, співставну з великими мовними моделями.

За результатами експериментального оцінювання тонко налаштована модель Phi-4-mini-instruct досягла таких показників якості класифікації:

Accuracy – 96.45 %, Precision – 96.64 %, Recall – 96.44 %, F1-score – 96.38%.

Зазначені значення свідчать про високу стабільність та узгодженість роботи моделі при розв’язанні задачі класифікації голосової пошти та підтверджують ефективність використаного підходу тонкого налаштування.

Для оцінювання продуктивності системи було проведено навантажувальне тестування з використанням різної кількості інстансів endpoint. Встановлено, що при використанні одного інстансу система забезпечує пропускну здатність 78 запитів за секунду при середній затримці 0.4 с. Масштабування до двох інстансів дозволило підвищити пропускну здатність до 124 RPS із середньою затримкою 0.58 с, а при використанні трьох інстансів досягнуто показника понад 150 RPS при затримці близько 0.6 с. Отримані результати підтверджують ефективність горизонтального масштабування та стабільність роботи системи при зростанні навантаження.

Порівняльний аналіз експлуатаційних характеристик показав, що використання Phi-4-mini-instruct є економічно доцільнішим порівняно з gpt-35-turbo-0125. Зокрема, вартість хостингу тонко налаштованої малої мовної моделі є приблизно у два рази нижчою, ніж для великої мовної моделі, а витрати на обробку вхідних і вихідних токенів також зменшуються приблизно у два рази. Це робить запропоноване рішення більш придатним для довготривалої експлуатації та масштабування в реальних системах.

Достовірність отриманих результатів забезпечується використанням стандартизованих метрик оцінювання, відтворюваних експериментів та реального хмарного середовища Azure Machine Learning та Azure AI Foundry. Усі експерименти виконувалися з фіксованими параметрами, що дозволяє повторити дослідження та підтвердити отримані висновки.

Наукова новизна роботи полягає у практичному доведенні можливості використання малої мовної моделі як повноцінної альтернативи великій мовній моделі для задачі класифікації голосової пошти в умовах обмежених обчислювальних ресурсів, а також у комплексному порівнянні якості, продуктивності та економічних показників таких моделей.

Практичне значення одержаних результатів полягає у створенні готової до використання програмної системи, яка може бути інтегрована в реальні інформаційні платформи контакт-центрів та телекомунікаційних сервісів. Запропонований підхід дозволяє зменшити витрати на інфраструктуру, забезпечити стабільну продуктивність та ефективно масштабувати систему відповідно до навантаження, що робить його доцільним для промислового застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brown T. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS). 2020.

2. Bommasani R. et al. On the Opportunities and Risks of Foundation Models. Stanford: Center for Research on Foundation Models, 2021.

3. Beck K. et al. Manifesto for Agile Software Development. 2001. URL: https://agilemanifesto.org (дата звернення: 20.12.2025).

4. Sommerville I. Software Engineering. 10th Edition. Boston: Pearson, 2016. 768 p.

5. Zhang S. et al. MLOps: Continuous Delivery and Automation Pipelines in Machine Learning. IEEE Software. 2020.

6. Microsoft. Azure Machine Learning Documentation. 2024. URL: https://learn.microsoft.com/azure/machine-learning (дата звернення: 12.05.2025).

7. Microsoft. Azure AI Foundry Overview. 2024. URL: https://learn.microsoft.com/azure/ai-foundry (дата звернення: 20.12.2025).

8. Azure Machine Learning Architecture Overview. URL:

https://learn.microsoft.com/en-us/azure/machine-learning/concept-azure-machine-learning-architecture (дата звернення: 20.12.2025).

9. Azure AI Foundry Documentation. URL:

https://learn.microsoft.com/uk-ua/azure/ai-foundry (дата звернення: 20.12.2025).

10. Sculley D. et al. Hidden Technical Debt in Machine Learning Systems. Proceedings of the Neural Information Processing Systems Conference (NIPS). 2015.

11. Buyya R. et al. Cloud Computing: Principles and Paradigms. Hoboken: Wiley, 2019. 637 p.

12. Zhang S. et al. MLOps: Continuous Delivery and Automation Pipelines in Machine Learning. IEEE Software. 2020.

13. Fine-tune models using serverless APIs in Azure AI Foundry. URL:

https://learn.microsoft.com/en-us/azure/ai-foundry/how-to/fine-tune-serverless?view=foundry-classic&tabs=chat-completion&pivots=foundry-portal#prepare-data-for-fine-tuning (дата звернення: 14.05.2025).

14. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 775 p.

15. Fine-tune models using serverless APIs in Azure AI Foundry. URL:

https://learn.microsoft.com/en-us/azure/ai-foundry/how-to/fine-tune-serverless (дата звернення: 20.12.2025).

16. Configure fine-tuning task parameters in Azure AI Foundry. URL:

https://learn.microsoft.com/en-us/azure/ai-foundry/how-to/fine-tune-serverless#configure-task-parameters (дата звернення: 20.12.2025).

17. IBM. What is LoRA (Low-Rank Adaptation). URL:

https://www.ibm.com/think/topics/lora (дата звернення: 20.12.2025).

18. Hu E. et al. LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint arXiv:2106.09685. 2022.

19. Sommerville I. Software Engineering. 10th Edition. Boston: Pearson, 2016. 768 p.

20. Powers D. M. W. Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness. Journal of Machine Learning Technologies. 2011.

21. Microsoft Azure Pricing. URL:

https://azure.microsoft.com/en-us/pricing (дата звернення: 20.12.2025).

22. Grafana k6 Documentation. URL:

https://grafana.com/docs/k6/latest (дата звернення: 20.12.2025).

23. Про охорону праці: Закон України від 14.10.1992 № 2694-XII (зі змінами та доповненнями).

24. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» : Наказ Міністерства соціальної політики України від 14.02.2018 № 207 / База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/z0508-18 (дата звернення:20.12.2025)

25. Кодекс цивільного захисту України від 02.10.2012 № 5403-VI (чинна редакція).

26. Правила пожежної безпеки в Україні: затверджені наказом Міністерства внутрішніх справ України від 30.12.2014 № 1417 (чинна редакція).

27. ДСТУ EN 62305:2012. Захист від блискавки. Загальні положення.

28. Стручок В. С. Безпека в надзвичайних ситуаціях: методичний посібник для здобувачів освітнього ступеня «магістр». Тернопіль: ФОП Паляниця В. А., 156 с.

29. Стручок В. С. Техноекологія та цивільна безпека. Частина «Цивільна безпека»: навчальний посібник. Тернопіль: ФОП Паляниця В. А., 156 с.

30. Кодекс цивільного захисту України. Чинна редакція.

31. Методичні рекомендації ДСНС України з розроблення планів цивільного захисту на особливий період. Чинна редакція.

32. Правила пожежної безпеки в Україні. Чинна редакція.

33. ISO 22301:2019. Business continuity management systems — Requirements.

34. Рекомендації Держспецзв’язку України щодо захисту інформації в умовах воєнного стану.

35. Методичні вказівки до виконання кваліфікаційної роботи магістра [Електронний ресурс]. URL:

https://dl.tntu.edu.ua/content.php?cid=534814 (дата звернення: 16.05.2025).

ДОДАТКИ

УДК 004.8
Містерман П. – ст. гр. СПм-61; Петрик М. – д.ф.-м.н., професор.
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА МАЛОЇ МОВНОЇ МОДЕЛІ SLM ДЛЯ СИСТЕМ З ОБМЕЖЕНИМИ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ НА БАЗІ AZURE ML ТА AI FOUNDRY

Misterman P.; Petrik M., PhD-M.Sc., Professor
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A SMALL LANGUAGE MODEL (SLM) FOR RESOURCE-CONSTRAINED SYSTEMS BASED ON AZURE ML AND AI FOUNDRY

У сучасних умовах стрімкого розвитку штучного інтелекту все більшої уваги набуває використання малих мовних моделей (SLM), які здатні забезпечувати високу ефективність при суттєво нижчих вимогах до обчислювальних ресурсів порівняно з великими мовними моделями (LLM). Зростаючий попит на локальні та економічно ефективні рішення зумовлює необхідність адаптації SLM до різних прикладних задач — класифікації, екстракції даних, діалогових систем та автоматизації бізнес-процесів.

Платформа Azure Machine Learning у поєднанні з Azure AI Foundry надає можливості для навчання, керування та розгортання моделей різного масштабу, включно з оптимізованими SLM, що дозволяє створювати продуктивні системи навіть у середовищах з обмеженими ресурсами. Одним із ключових аспектів роботи є дослідження можливостей використання малої моделі замість LLM у задачах, де традиційно очікується висока якість мовного розуміння, а також оцінювання продуктивності, швидкодії, вартості та точності.

Під час розробки рішення передбачено використання технік донавчання моделей, таких як LoRA та QLoRA, а також застосування форматів оптимізації типу GGUF та інструментів прискорення інференсу, включно з vLLM. Це дозволяє забезпечити високу швидкість відповіді, зменшити споживання пам’яті та адаптувати модель до вузької предметної області в умовах реальних бізнес-сценаріїв.

Практичне значення роботи полягає у створенні прототипу системи, здатної виконувати типові завдання штучного інтелекту без значних апаратних витрат. Це відкриває можливості для впровадження інтелектуальних рішень у малих організаціях, на периферійних пристроях або в середовищах з підвищеними вимогами до приватності та локальності обробки даних. Результати дослідження демонструють потенціал SLM як життєздатної альтернативи LLM для широкого спектра задач, забезпечуючи баланс між точністю, продуктивністю та доступністю.

Література

1. Azure Machine Learning documentation | Microsoft Learn. URL: https://learn.microsoft.com/uk-ua/azure/machine-learning/?view=azureml-api-2 (дата звернення: 10.12.2025).

2. What is Microsoft Foundry? - Microsoft Foundry | Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/ai-foundry/what-is-azure-ai-foundry?view=foundry-classic (дата звернення: 10.12.2025).

3. Maninder S. *Practical Guide to Fine-tune LLMs with LoRA*. 13.10.2024. URL: https://medium.com/@manindersingh120996/practical-guide-to-fine-tune-llms-with-lora-c835a99d7593 (дата звернення: 10.12.2025).

Лістинг 2.1 – Програмний код функції очищення даних:

```
nlp = spacy.load("en_core_web_sm")

CUSTOM_NAMES = {"Corina", "Rita", "Jake", "Samantha", "Nicole",
"Pollyanna", "Eric", "Washi", "Rihanna", "Trisha", "Jessica",
"Dwight", "Art", "Heather",
                "Jim", "Leanne", "Marsha", "Debbie", "Caltrans",
"Kurt", "Cathy", "Vince", "Larnie", "Jen Crow", "Brooke", "Shauna",
"Gloria", "Zach", "Jerry",
                "Bailey", "Carol", "Aiden"}

def preprocess_sentence(sentence: str, nlp=nlp) -> str:
    phone_pattern = r"\b(\+?\d{1,3}?[-.\s]?(\(?\d{1,4}?\)?[-
.\s]?)?\d{1,4}[-.\s]?\d{1,4}[-.\s]?\d{1,9})\b"
    time_pattern = r"\b([0-2]?\d:[0-5]\d)\b"
    date_pattern = r"\b(\d{1,2}[/.-]\d{1,2}[/.-
]\d{2,4}|\b(?:January|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:
e)?|Jul(?:y)?|Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|
Dec(?:ember)?)\.?\s?\d{0,2})\b"

    doc = nlp(sentence)
    cleaned_sentence = sentence

    for ent in doc.ents:
        if ent.label_ == "PERSON":
            cleaned_sentence = re.sub(rf"\b{re.escape(ent.text)}\b",
"[NAME]", cleaned_sentence)

        if  ent.label_ == "ORG":
            cleaned_sentence = cleaned_sentence.replace(ent.text,
"[COMPANY]")

    for name in CUSTOM_NAMES:
        cleaned_sentence = re.sub(rf"\b{name}\b", "[NAME]",
cleaned_sentence, flags=re.IGNORECASE)

    cleaned_sentence = re.sub(phone_pattern, "[PHONE]",
cleaned_sentence)
    cleaned_sentence = re.sub(time_pattern, "[TIME]",
cleaned_sentence)
    cleaned_sentence = re.sub(date_pattern, "[DATE]",
cleaned_sentence)
    cleaned_sentence = cleaned_sentence.replace("&apos;", "'")
    cleaned_sentence = cleaned_sentence.replace("’", "'")
    cleaned_sentence = cleaned_sentence.replace("[[", "[")
    cleaned_sentence = cleaned_sentence.replace("]]", "]")
    cleaned_sentence = cleaned_sentence.replace("[Name]", "[NAME]")
    cleaned_sentence = cleaned_sentence.replace("[Company]",
"[COMPANY]")
```

```
    return cleaned_sentence
```

Лістинг 2.4 – Код функцій для збереження тренувальних даних на Azure AI Foundry

```
def write_to_blob_storage(data, file_name):
    """
    Save to Azure's blog storage
    """
    connection_string = os.getenv("CONNECTION_STRING")
    container_name = "vmdetectdataset"
    blob_path = f"unio/processed/{file_name}.jsonl"

    jsonl_content = "\n".join(json.dumps(record) for record in data)

    blob_client =
BlobClient.from_connection_string(connection_string, container_name,
blob_path)
    blob_client.upload_blob(jsonl_content, overwrite=True)
    print(f"File {blob_path} uploaded successfully to container
{container_name}")

    """
    Save to project's blog storage
    """
    timestamp = datetime.now(timezone.utc).strftime("%Y-%m-
%d_%H%M%S_UTC")

    sas_token = os.getenv("SAS_TOKEN")
    storage_name = os.getenv("STORAGE_NAME")
    storage_id = os.getenv("STORAGE_ID")
    sas_url =
f"https://{storage_name}.blob.core.windows.net/{storage_id}-azureml-
blobstore/UI/{timestamp}/{file_name}.jsonl?{sas_token}"

    blob_client = BlobClient.from_blob_url(sas_url)
    blob_client.upload_blob(jsonl_content, overwrite=True)

    print(f"File uploaded successfully to path {storage_id}-azureml-
blobstore/UI/{timestamp}/data.jsonl")

    return f"{timestamp}/{file_name}.jsonl"

def create_data_in_project(resource_name, path_to_file,
version="1"):

    credential = DefaultAzureCredential()

    ml_client = MLClient(
        credential=credential,
        subscription_id=os.getenv("SUBSCRIPTION_ID"),
```

```
        resource_group_name=os.getenv("RESOURCE_GROUP"),
        workspace_name=os.getenv("WORKSPACE_NAME"),
    )

    print("Successfully connected to Azure AI Foundry")

    data_asset = Data(
        name=resource_name,
        version=version,
        # description="",

path=f"azureml://datastores/workspaceblobstore/paths/UI/{path_to_fil
e}",
        type="uri_file",
    )

    ml_client.data.create_or_update(data_asset)

    print("Data asset registered successfully")
```

Лістинг 2.7 – Параметри тонкого налаштування з використанням LoRA

```
finetune_parameters = {
    "apply_lora": True,
    "merge_lora_weights": True,
    "lora_alpha": 128,
    "lora_r": 8,
    "lora_dropout": 0.0,
    "num_train_epochs": 3,
    "max_steps": -1,
    "per_device_train_batch_size": 2,
    "per_device_eval_batch_size": 1,
    "auto_find_batch_size": False,
    "optim": "adamw_hf",
    "learning_rate": 3E-05,
    "warmup_steps": 0,
    "weight_decay": 0.1,
    "adam_beta1": 0.9,
    "adam_beta2": 0.95,
    "adam_epsilon": 1e-05,
    "gradient_accumulation_steps": 1,
    "eval_accumulation_steps": 1,
    "lr_scheduler_type": "cosine",
    "precision": 16,
    "seed": 42,
    "enable_full_determinism": False,
    "dataloader_num_workers": 0,
    "ignore_mismatched_sizes": False,
    "max_grad_norm": 1.0,
    "evaluation_strategy": "epoch",
    "evaluation_steps_interval": 0.0,
    "eval_steps": 500,
    "logging_strategy": "steps",
```

```
    "logging_steps": 10,
    "metric_for_best_model": "loss",
    "resume_from_checkpoint": False,
    "save_total_limit": 1,
    "apply_early_stopping": False,
    "early_stopping_patience": 0,
    "early_stopping_threshold": 0.0,
    "apply_deepspeed": True,
    "deepspeed_stage": 3,
    "apply_ort": True,
}
```