

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Сервіс управління Docker контейнерами за допомогою
команд природніми мовами»**

Виконав студент групи ЗПЗм-21-1
КАДЬКАЛО Владислав Валерійович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
КРАВЧЕНКО Світлана Миколаївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії

програмного забезпечення

Тетяна ВАКАЛЮК

«05» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Кадькало Владислав Валерійович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Сервіс управління Docker контейнерами за допомогою команд природніми мовами»,**

затверджена Наказом закладу вищої освіти від «05» лютого 2024 р. № 49/с

Вихідні дані для роботи: **Розробка і аналіз методології для інтерпретації природних мов і перетворення їх в конкретні команди для управління Docker контейнерами; створення інтерфейсу, що дозволяє користувачам взаємодіяти з Docker за допомогою природньої мови для виконання операцій.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	05.02.2024р.	05.02.2024р.
2	Чижмотря О.В.	05.02.2024р.	05.02.2024р.
3	Чижмотря О.В.	05.02.2024р.	05.02.2024р.

Реферат

Кваліфікаційна робота освітнього ступеня «магістр» присвячена розробці підходу та реалізації прототипу системи управління Docker контейнерами через інтерфейс природної мови. Робота містить детальний опис концепції та створення API, що дозволяє вводити команди українською мовою, які за допомогою методів машинного навчання трансформуються в команди bash/docker CLI для подальшого виконання на хост-машині.

Унікальною характеристикою розробленого прототипу сервісу є вбудована система збору даних, що вдосконалює процес тренування основної моделі перекладу, через збільшення варіативності та точності даних шляхом додаткової обробки запитів користувача за допомогою ChatGPT API.

В пояснювальній записці, яка включає вступ, три розділи, висновки, список використаної літератури та додатки, детально розглянуто процеси розробки, валідації команд та методики навчання моделі.

Пояснювальна записка містить 68 сторінок, 15 ілюстрацій, 1 таблицю, 6 додатків та 8 посилань на джерела.

Метою даної кваліфікаційної роботи є розробка методу, який може бути використаний для побудови сервісу що спрощує управління Docker контейнерами через інтерфейс природної мови. Також увага приділена дослідженню та аналізу ефективності підходу, який полягає в використанні вбудованої системи збору даних.

КЛЮЧОВІ СЛОВА: Docker, машинне навчання, MarianMT, GPT, API, управління контейнерами, природна мова, Bash, CLI.

Abstract

The master's thesis is dedicated to developing an approach and implementing a prototype for managing Docker containers through a natural language interface. The work contains a detailed description of the concept and the creation of an API that allows entering commands in Ukrainian, which, through machine learning methods, are transformed into bash/docker CLI commands for further execution on the host machine. A unique feature of the developed prototype service is the integrated data collection system, which improves the training process of the main translation model by increasing the variability and accuracy of the data through additional processing of user requests using the ChatGPT API. The explanatory note, which includes an introduction, three sections, conclusions, a list of references, and appendices, thoroughly examines the development processes, command validation, and model training methodologies. The explanatory note contains 68 pages, 15 illustrations, 1 table, 6 appendices, and 8 references. The purpose of this qualification work is to develop a method that can be used to build a service that simplifies the management of Docker containers through a natural language interface. Attention is also given to the study and analysis of the effectiveness of the approach, which involves using an integrated data collection system. **KEYWORDS:** Docker, machine learning, MarianMT, GPT, API, container management, natural language, Bash, CLI.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ УПРАВЛІННЯ DOCKER КОНТЕЙНЕРАМИ З ВИКОРИСТАННЯМ ПРИРОДНИХ МОВ	9
1.1. Огляд існуючих рішень	9
1.2. Теоретичні аспекти використання природних мов для управління Docker контейнерами	13
1.3. Постановка задачі дослідження	17
Висновки до розділу 1	18
РОЗДІЛ 2. ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ПРОТОТИПУ СЕРВІСУ	20
2.1 Збір даних для навчання моделі	20
2.2. Вибір основної моделі перекладу	23
2.3. Архітектура сервісу	35
2.4. Адаптація моделі MarianMT та валідація і корекція команд з використанням ChatGPT	38
2.5 Майбутні напрямки дослідження	42
Висновки до розділу 2.....	46
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОТОТИПУ СЕРВІСУ	47
3.1. REST API та команди	47
3.2. Засоби реалізації	50
3.3. Приклад роботи системи	51
3.4. Вимоги та залежності	56
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CLI – (англ. Command Line Interface) – інтерфейс командного рядка

GPT – (англ. Generative Pre-trained Transformer) – генеративний попередньо навчений трансформер

ML – (англ. Machine Learning) – машинне навчання

API – (англ. Application Programming Interface) – інтерфейс програмного забезпечення для додатків

NLP – (англ. Natural Language Processing) – обробка природних мов

ВСТУП

Актуальність теми. В умовах швидкого розвитку технологій контейнеризації, зокрема з використанням Docker, виникає потреба у спрощенні та оптимізації процесів управління контейнерами. Впровадження інтуїтивно зрозумілих інструментів для управління Docker контейнерами може значно підвищити доступність технології для ширшого кола користувачів. Таким чином, використання природної мови для керування такими системами може відкрити нові перспективи для розробки інтерфейсів користувача та способів взаємодії з складними системами.

Незважаючи на широке поширення Docker, керування контейнерами часто вимагає спеціалізованих знань і навичок. Ця робота спрямована на розробку методу, який би дозволяв використовувати природню мову для формування команд управління Docker контейнерами, знижуючи тим самим бар'єр входження для користувачів без глибоких технічних знань. Підхід, обраний у цій роботі, включає комбіноване застосування адаптованої(fine-tuned) моделі машинного навчання для перекладу команд природньою мовою в рядки bash та Docker CLI, а також подальшу валідацію, та виправлення невірних результатів за допомогою ChatGPT API, та їх збереження для подальшого навчання моделі.

Мета роботи полягає у розробці прототипу сервісу, що дозволяє управляти Docker контейнерами за допомогою команд, введених природною (українською) мовою, з системою збору даних для покращення якості перекладу через покращення кількості та якості даних в датасеті.

Для досягнення поставленої мети були визначені наступні основні завдання:

- Аналіз існуючих методів управління Docker контейнерами.

- Вивчення потенціалу машинного навчання для перекладу команд природною мовою в команди Docker CLI.

- Розробка та тестування сервісу для демонстрації можливостей управління Docker контейнерами за допомогою природномовних команд.

Оцінка ефективності сервісу та його потенціалу для подальшого розвитку.

Об'єктом дослідження є процес управління Docker контейнерами.

Предметом дослідження є методи перекладу команд природною мовою в команди Docker CLI з використанням машинного навчання.

Наукова новизна даної роботи полягає у створенні унікального методу, що інтегрує можливості попередньо навчених моделей машинного навчання з ChatGPT API для перетворення та перевірки вірності команд управління Docker контейнерами, введених в натуральній мовній формі.

Практичне значення полягає у спрощенні процесу взаємодії з Docker контейнерами, розширенні доступності технології для неспеціалістів та в освітніх цілях.

Апробація. За темою що близька до висвітленої в кваліфікаційній роботі було опубліковано тезу [1] конференції.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ УПРАВЛІННЯ DOCKER КОНТЕЙНЕРАМИ З ВИКОРИСТАННЯМ ПРИРОДНИХ МОВ

1.1. Огляд існуючих рішень

Контейнеризація часто відіграє одну з ключових ролей у сучасному світі розробки програмного забезпечення, забезпечуючи стандартизоване середовище для розгортання та виконання програм. Docker, як лідер у цій галузі, дозволяє розробникам інкапсулювати додатки з усіма їхніми залежностями у стандартизовані середовища, відомі як контейнери. Використання Docker значно спрощує процедуру розповсюдження та розгортання застосунків, оскільки усуває типові проблеми, пов'язані з сумісністю середовищ, тим самим оптимізуючи процес розробки та впровадження програмного забезпечення на будь-якій платформі [6]. Ключові переваги, які може запропонувати технологія контейнеризації, охоплюють широкий спектр аспектів, таких як портабельність, що дозволяє легко переміщати додатки між різними середовищами, модульність, яка сприяє розробці та управлінні окремими компонентами додатку, масштабованість, що забезпечує ефективну витрату ресурсів відповідно до потреб, а також ізоляцію, яка гарантує, що кожен контейнер працює незалежно.

Хоча контейнеризація з Docker пропонує ряд значних переваг, процес управління контейнерами може представляти певні складнощі для користувачів. Основним з цих викликів є потреба в глибокому розумінні команд Docker CLI та скриптових мов, зокрема Bash, що може стати серйозним бар'єром для осіб, які лише починають працювати з Docker, або для тих, хто не має високого рівня технічних знань. Ця потреба в глибоких знаннях команд Docker CLI та скриптових мов, таких як Bash, може стати значним бар'єром для тих, хто тільки починає знайомство з Docker, або для користувачів, які не мають розширеного технічного фону. Така вимога до рівня знань і навичок може ускладнити доступність та зручність використання Docker для широкого кола користувачів, зацікавлених в використанні контейнеризованих додатків.

Управління Docker контейнерами здійснюється переважно через два типи інтерфейсів: командний рядок (CLI) та графічні інтерфейси користувача (GUI). Docker CLI є потужним інструментом, який надає користувачам широкий спектр команд для управління контейнерами, образами та мережами. Незважаючи на свою потужність, інтерфейс командного рядку може бути складним для освоєння новачками, оскільки вимагає знань мов оболонки командного рядка та синтаксису Docker CLI.

З іншого боку, графічні інтерфейси, такі як Docker Desktop та Portainer, надають більш інтуїтивно зрозуміле середовище для управління контейнерами. Ці інструменти дозволяють користувачам виконувати більшість операцій з контейнерами через графічний інтерфейс, зменшуючи потребу в безпосередньому використанні інтерфейсу командного рядку. Однак, навіть з графічним інтерфейсом, користувачі все одно стикаються з обмеженнями у функціональності та гнучкості порівняно з повним набором можливостей що доступний в командному рядку.

Аналіз існуючих рішень управління Docker контейнерами виявляє декілька прогалин у доступності та ефективності. Перш за все, існує значний бар'єр для входження для новачків через складність використання командного рядка та обмежену функціональність графічних інтерфейсів, багато графічних інтерфейсів не надають повного набору можливостей, доступних у командному рядку, обмежуючи досвідчених користувачів у використанні усієї потужності Docker.

Також, хоча графічні інтерфейси спрощують деякі задачі, вони часто не забезпечують достатнього рівня деталізації або гнучкості для складних сценаріїв використання. Це вказує на необхідність розробки нових рішень, які б поєднували легкість використання з потужністю та гнучкістю, дозволяючи користувачам управляти Docker контейнерами більш ефективно та інтуїтивно.

Актуальність розробки інтуїтивно зрозумілих і доступних методів управління контейнерами не може бути переоцінена. Спрощення управління

контейнерами може відкрити цю технологію для ширшого кола користувачів, у тому числі для малодосвідчених розробників та інших фахівців, не залучених безпосередньо у процеси розробки програмного забезпечення. Це може значно прискорити процеси розгортання та тестування додатків, роблячи інноваційні технології доступнішими для експериментів та освітніх цілей. Враховуючи ці аспекти, розробка сервісу для управління Docker контейнерами за допомогою команд природними мовами може вважатись важливим кроком у напрямку демократизації технологій контейнеризації та спрощення їх використання.

Природні мови, завдяки своїй гнучкості та доступності, можуть суттєво спростити взаємодію людини з комплексними технологічними системами, не обмежуючись лише Docker контейнерами.

Потенціал природних мов як інтерфейсу має наступні переваги:

- Використання природних мов дозволяє користувачам взаємодіяти з системою в звичному форматі, без необхідності вивчення спеціалізованих команд або інтерфейсів.
- Природні мови можуть бути застосовані для управління широким спектром ІТ-систем, включаючи обчислювальні ресурси, бази даних, мережеві конфігурації тощо.
- Взаємодія з системами через природні мови робить технології більш доступними для неспеціалістів, сприяючи ширшому впровадженню та використанню.

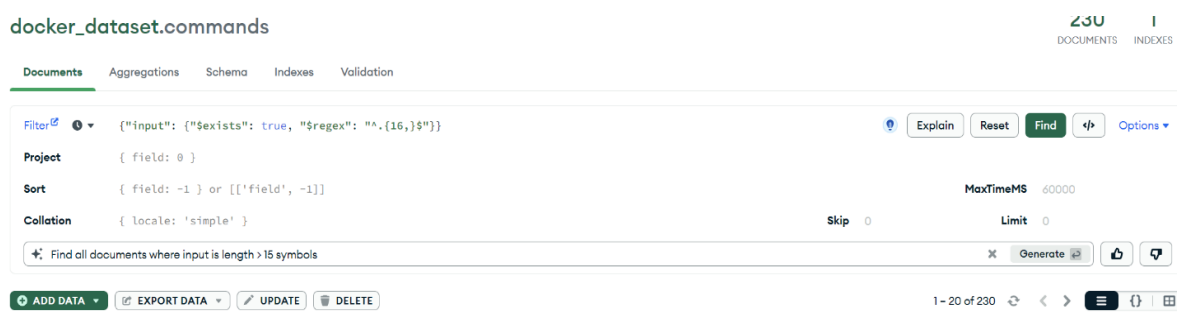


Рис 1.1. Інструмент «Generate query» в застосунку «MongoDB Compass»

Проте поряд з цими перевагами ми бачимо і суттєві обмеження які обов'язково потрібно враховувати, а саме:

- Природні мови характеризуються багатозначністю та невизначеністю, що може призводити до неправильного тлумачення команд системою.
- Системи, що використовують природні мови, повинні вміти розпізнавати та адаптуватися до контексту запитів, що вимагає розвинених алгоритмів обробки мови та машинного навчання.
- Взаємодія через природні мови може включати передачу конфіденційної інформації, що ставить питання про забезпечення безпеки даних та захисту приватності користувачів. В зв'язку з чим перезд на окремі самостійні моделі є пріоритетним в майбутніх дослідженнях.
- Адаптація природних мов як інтерфейсу для вже існуючих ІТ-систем може виявитися складним завданням, що вимагає значних зусиль та ресурсів для інтеграції та тестування.

Також можна згадати про виклики розуміння контексту що створюють значні вимоги до систем, які використовують природні мови для взаємодії з користувачами. Адекватне розуміння інструкцій вимагає від систем не тільки базового аналізу тексту, а й глибокого занурення в контекст, у якому ці інструкції надаються. Це може включати в себе розгляд історії попередніх запитів користувача, що допомагає зрозуміти його поточні наміри, а також аналіз специфіки роботи конкретної ІТ-системи, щоб коректно інтерпретувати команди. Наприклад, вказівка на "видалення файлу" може вимагати від системи розуміння, чи користувач має на увазі тимчасове переміщення файлу в кошик або його остаточне видалення. Також важливим є аналіз контексту виконання, такий як поточний стан системи чи доступні ресурси, що може вплинути на виконання запиту. Ці аспекти підкреслюють важливість розвитку розуміння

контексту в системах обробки природної мови, що є ключем до створення більш інтуїтивно зрозумілих та ефективних інтерфейсів.

Інтеграція природних мов як інтерфейсів знаходить застосування у широкому спектрі ІТ-сфер, відкриваючи нові можливості для взаємодії з технологіями. Управління хмарними сервісами, наприклад, може стати значно зручнішим, коли користувачі мають змогу виконувати складні операції, такі як масштабування інфраструктури або розгортання сервісів, за допомогою простих мовних команд. У сфері Інтернету речей (ІоТ), можливість керувати розумними пристроями через природні мовні запити значно підвищує комфорт і доступність систем для кінцевих користувачів. Автоматизація бізнес-процесів та обслуговування клієнтів також виграє від використання природних мов, оскільки системи можуть обробляти запити, резервації, навіть складні запитання щодо продуктів або послуг у звичайному діалозі. Ці приклади демонструють, як природні мови розширюють можливості ІТ-систем, роблячи технології більш інтуїтивно зрозумілими та доступними для широкого кола користувачів.

Використання природних мов як інтерфейсу для управління ІТ-системами відкриває нові можливості для роботи з технологіями, роблячи їх більш доступними та зрозумілими для широкого кола користувачів. Проте, досягнення цієї мети вимагає ретельного врахування викликів та активного розвитку технологій та методологій, що дозволять максимально використати потенціал природних мов як зручного та ефективного інструменту для взаємодії з комплексними технологічними системами.

1.2. Теоретичні аспекти використання природних мов для управління Docker контейнерами

Обробка природної мови (NLP) є галуззю машинного навчання, яка займається розробкою алгоритмів та систем, здатних розуміти, інтерпретувати та генерувати людську мову в її найширшому розумінні. Ця дисципліна об'єднує

знання з комп'ютерних наук, штучного інтелекту та лінгвістики, щоб створювати інструменти, які можуть спілкуватися з людьми на природній мові, аналізувати текстові дані, розуміти запитання та надавати відповіді. Завдяки прогресу в області машинного навчання та збільшенню доступних даних, NLP демонструє значні досягнення у розпізнаванні мови, автоматичному перекладі, семантичному аналізі та генерації тексту, відкриваючи нові можливості для розвитку інтерактивних систем, що можуть слугувати в освіті, медицині, бізнесі та багатьох інших сферах.

У контексті інтерфейсів користувача, NLP може зробити технології більш доступними та зрозумілими для широкого кола користувачів. Наприклад, функція "Generate query" у застосунку MongoDB Compass, що являє собою клієнтську програму для бази даних MongoDB, дозволяє користувачам формувати запити за допомогою природньої мови, значно спрощуючи роботу з базами даних. Таке використання NLP підкреслює потенціал для створення більш інтуїтивних та доступних інструментів управління технологіями.

Для перекладу команд з використанням технік обробки природньої мови можна застосувати широкий спектр моделей машинного навчання, серед яких особливе місце займають нейронні мережі. Ці моделі навчаються на великих обсягах даних і здатні точно трансформувати висловлювання з однієї мови на іншу. Серед різноманіття підходів до машинного перекладу вирізняються декілька ключових архітектур.

Рекурентні нейронні мережі (RNN) [10] були одними з перших, хто знайшов широке застосування в задачах машинного перекладу. Вони ефективно обробляють послідовності даних, здатні зберігати інформацію про попередні елементи в послідовності. Однак RNN стикаються з проблемами довготривалої залежності та важко масштабуються через високі вимоги до обчислювальних ресурсів.

Мережі з довгою короткочасною пам'яттю (LSTM) [10] і Шлюзові рекурентні одиниці (GRU) [11] є вдосконаленими RNN, які краще справляються

з довгими текстами завдяки механізмам вентилів, що дозволяють контролювати потік інформації.

Трансформери що вважаються революційною архітектурою, що базується на механізмі уваги, дозволяючи моделям ефективніше працювати з відносинами між словами в довгих текстах. Вони вирішують проблеми рекурентних мереж, пропонуючи кращу масштабованість і швидкість обробки. Трансформери стали основою багатьох сучасних мовних моделей, включаючи BERT [9], GPT, та MarianMT [12].

Моделі, засновані на архітектурі трансформерів, демонструють виняткову здатність до перекладу, обробки та генерації тексту, завдяки чому вони мають ідеально підходити під задачу перекладу команд природною мовою в Docker CLI команди. Вони можуть забезпечити високу точність перекладу, також вони дозволяють налаштовувати (fine-tune) модель під конкретні завдання, враховуючи специфіку та контекст використання.

Застосування комбінованого методу, що включає використання спеціалізованих моделей для перекладу та сторонніх високотехнологічних API, зокрема ChatGPT, для додаткової перевірки та вдосконалення команд, відкриває широкі перспективи для збільшення точності та ефективності системи. У контексті нашої задачі з керування Docker контейнерами цей підхід дозволить покращити якість команд, навіть якщо переклад основної моделі матиме помилки. В такій ситуації ChatGPT має виступити як інструмент для доопрацювання та перевірки команд, що не тільки підвищить валідність команд, але й забезпечить нас матеріалами для додаткового тренування та вдосконалення основних моделей перекладу. Цей метод дозволить нам створити систему з позитивним зворотнім зв'язком, що буде сприяти неперервному поліпшенню продуктивності застосунку. Отже, розділяючи функції основних моделей перекладу від функцій ChatGPT, ми не тільки підвищуємо точність перекладу команд, але й збагачуємо навчальний датасет для майбутнього покращення основних моделей. Врешті-решт, застосування обробки природної мови до

технологій контейнеризації може радикально змінити підхід до взаємодії з контейнеризованими програмами, роблячи цей процес більш інтуїтивним та ефективним.

Основною науковою новизною є створення методу керування Docker контейнерами за допомогою інтерфейсу природної мови. Цей метод буде базуватись на використанні заздалегідь натренованих моделей машинного навчання налаштованих під конкретну задачу перекладу разом з моделями загального використання такими як ті що пропонує ChatGPT. Завдяки цьому, команди, введені користувачами звичайною мовою, можуть бути точно перетворені на команди Docker CLI, а згодом виконані на хост машині. Водночас, система має забезпечувати активну перевірку та вдосконалення результатів перекладу. Такий процес не лише має на меті покращити якість перекладу, але й розширює навчальний датасет, що сприяє автономному розвитку системи. Таким чином, розробка представленої методології має потенціал суттєво трансформувати підходи до управління Docker контейнерами, роблячи процес більш доступним та ефективним для широкого кола користувачів.

Запропонована методологія може знаходити застосування у широкому спектрі областей, де застосовуються технології контейнеризації. Серед таких сфер можна виділити розробку програмного забезпечення, девопс, тестування, наукові дослідження та освіту. Даний метод може стати корисним при розробці інструменту для новачків і студентів, які тільки починають своє знайомство з Docker та контейнеризацією, оскільки вона така система дозволить виконувати складні команди без глибокого розуміння синтаксису Docker CLI. Для досвідчених розробників та спеціалістів з адміністрування комп'ютерних систем такий метод може служити ефективним засобом для поліпшення автоматизації рутинних завдань. Крім того даний метод можна застосувати у сферах не пов'язаних з технологією контейнеризації, він може спростити будь яку задачу що зводиться до використання інструментів командного рядку.

Дана робота може встати в основу для багатьох напрямків подальших досліджень і розвитку. Одна з можливостей полягає у додаванні підтримки додаткових мов, що робитиме сервіс доступним для користувачів, які спілкуються різними мовами. Інша сфера, де можливий прогрес, – це інтеграція з додатковими інструментами для оркестрації контейнерів, наприклад Kubernetes, що відкриє нові можливості для керування інфраструктурою. Крім того, важливим напрямком є загальний розвиток галузі машинного навчання для розробки та подальшого вдосконалення моделей що здатні точно розпізнавати і розуміти контекст команд, які надаються користувачами. В кінцевому підсумку, застосування таких технологій може вийти за рамки управління Docker контейнерами та знайти застосування в інших сферах IT і програмування, відкриваючи широкі перспективи для інновацій у цих областях.

1.3. Постановка задачі дослідження

Метою дослідження є створення прототипу сервісу що продемонструє REST API для управління Docker контейнерами за допомогою команд, введених звичайною мовою. В процесі розробки мають бути викорисані сучасні технології у галузі машинного навчання та аналізу природної мови для точного перетворення мовних команд у відповідні технічні інструкції для Docker CLI. Завдання передбачає розробку системи, здатної не лише точно інтерпретувати мовні запити, але й оптимізувати процес взаємодії з контейнеризованими середовищами.

Цілі дослідження включають пошук та доопрацювання моделі, заснованої на машинному навчанні, яка здатна перетворити команди, введені на звичайній мові, в інструкції Docker CLI. Ця модель має ефективно обробляти мовні запити та перетворювати їх на відповідні дії в контексті управління контейнерами. А також розробку прототипу системи, що використовує комбінований підхід із застосуванням передових моделей машинного навчання, таких як ChatGPT, для перевірки та валідації перекладених команд. Що забезпечить перевірку точності

команд, введених звичайною мовою та перетворених на технічні інструкції Docker CLI. А також забезпечить додаткову перевірку для гарантування високої точності та надійності результатів. Таким чином, ефективність взаємодії з Docker має стати значно вищою, ніж при використанні кожного з методів окремо.

Завдання цього дослідження включають:

- Визначення потреб користувачів та визначення основних способів використання сервісу.
- Підбір і налаштування моделі машинного перекладу, щоб вона могла точно виконувати завдання, пов'язані з керуванням Docker контейнерами, з урахуванням специфіки цієї галузі.
- Розробка методів взаємодії з сторонніми API для валідації та корекції перекладених команд.
- Розробка та перевірка роботи ключових API ендпоінтів та консольних команд, які є необхідними для демонстрації роботи методу.

Основна увага приділяється розробці та впровадженню комбінованого підходу для створення мінімального варіанту сервісу. Комбінований підхід базується на використанні методів машинного навчання та технік обробки природної мови за допомогою сторонніх API для точного перекладу команд введених на звичайній мові в інструкції Docker CLI. Окрім розробки алгоритмів перекладу, великий акцент робиться на інтеграції системи валідації команд з використанням передових моделей, таких як ChatGPT, для забезпечення високої точності та надійності результатів.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи розглянуто актуальність управління Docker контейнерами за допомогою природних мов, та описано комбінований підхід, що включає застосування спеціалізованих моделей машинного перекладу разом з сторонніми API, зокрема ChatGPT, для перевірки

та оптимізації команд. Така система не тільки виправляє помилки у первісному перекладі, але й дозволяє удосконалювати команди за допомогою додаткової валідації, що буде сприяти безперервному поліпшенню продуктивності та якості сервісу.

Висновки підкреслюють важливість розробки прототипу сервісу з використанням описаного підходу. Окрім підтвердження ефективності цього методу, в майбутньому планується дослідження можливостей його поліпшення та розширення. Це може включати розробку більш адаптованих моделей машинного навчання, що краще розуміють контекст користувача, і розширення мовної підтримки, щоб сервіс став доступним для більш широкої аудиторії.

Також, передбачається подальший аналіз з ціллю визначення можливості інтеграції з додатковими інструментами оркестрації контейнерів, як-от Kubernetes, що відкриє нові можливості для керування інфраструктурою та поліпшить інтеграцію в різноманітні середовища розробки. Впровадження глибших досліджень у сфері машинного навчання дозволить вдосконалити точність та розуміння перекладених команд, підвищуючи загальну надійність системи.

Завдяки включенню в дослідження різних аспектів від перекладу команд до валідації та оптимізації сервісу, ми відкриваємо двері до нових можливостей у спрощенні роботи з складними ІТ-системами, тим самим збагачуючи інструментарій розробників та кінцевих користувачів.

В підсумку, перший розділ має закласти міцний фундамент для подальших досліджень, визначаючи ключові напрямки майбутніх поліпшень, що не тільки збільшать ефективність взаємодії з Docker контейнерами, але й зроблять технологію контейнеризації більш доступною для ширшого кола користувачів.

Розділ 2. ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ПРОТОТИПУ СЕРВІСУ

2.1 Збір даних для навчання моделі

Для навчання нашої основної моделі перекладу, необхідно провести роботу по створенню початкового датасету. Цей датасет складається з команд для управління Docker контейнерами, описаних звичайною мовою, та їх відповідників у форматі Docker CLI. Елементи датасету зберігаються у базі даних MongoDB зі структурою, що включає поля "input" та "output" які в свою чергу відображають приклад вхідних даних у вигляді команди українською мовою, та цієї ж команди перекладеної на виклик bash/docker cli.

```
_id: ObjectId('65e336356a84a5c00009fd92')
input: "Створи контейнер на основі образу Ubuntu та встанови Python 3.7"
output: "docker run -d --name ubuntu-python3.7 -it ubuntu bash -c "apt-get upda..."
```

```
_id: ObjectId('65e336426a84a5c00009fd93')
input: "Створи новий контейнер з образу Node.js і встанови в нього express"
output: "docker run -d --name my_node_container node:latest /bin/bash -c "npm i..."
```

```
_id: ObjectId('65e336486a84a5c00009fd94')
input: "Створи новий контейнер з образу MySQL і налаштуй його на порт 3306"
output: "docker run -d -p 3306:3306 --name some-container-name -e MYSQL_ROOT_PA..."
```

Рис 2.1. Приклад елементів датасету

Було використано наступні способи збору даних:

- Ручне додавання: Деякі записи було створено вручну, використовуючи знання Docker. Цей метод забезпечує високу точність та валідність команд, але є часозатратним і обмеженим у кількості даних, які можна згенерувати за короткий час.
- Генерація за допомогою ChatGPT та плагіна Dataset Creator: Для розширення датасету також було використано можливості ChatGPT разом з плагіном Dataset Creator, задавши запит для створення 20 записів, за запит, пов'язаних з управлінням Docker контейнерами.

Цей метод дозволив автоматизувати процес генерації даних, отримуючи різноманітні сценарії використання Docker.

Варто зауважити переваги та недоліки кожного з методів.

Ручне додавання забезпечує високу якість даних, але вимагає значних часових витрат та обмежене кількістю даних, які можна обробити.

Генерація за допомогою ChatGPT підвищує швидкість та різноманітність датасету, проте вимагає перевірки на валідність виконання команд та може призводити до додаткових витрат через обмеження на кількість запитів та необхідність оплати за використання плагінів.

Також було створено спеціалізовану консольну команду, яка автоматично використовує ChatGPT API для створення нових записів датасету. Ця команда дозволяє більш ефективно масштабувати процес збору даних, мінімізуючи ручні втручання та оптимізуючи використання ресурсів.

Для розширення та підвищення ефективності розробленого методу при подальших дослідженнях необхідно зосередитись на більш глибокому аналізі та розвитку датасетів, які використовуються для тренування моделей. Одним з ключових аспектів такої роботи є не тільки постійне збагачення існуючих датасетів новими даними, але й розвиток нових методів підготовки та обробки даних.

Для підвищення якості перекладу і розширення функціоналу сервісу під час майбутніх досліджень, планується застосувати більш глибокий підхід до підготовки та аналізу даних. Враховуючи необхідність адаптації сервісу до специфічних задач і контекстів, передбачається розробка додаткових датасетів для тренування спеціалізованих моделей. Ось деякі напрямки, які розглядаються:

- Створення датасетів з пар «команда-результат валідації», де результат може включати інформацію про валідність команди, потенційні помилки та рекомендації щодо виправлення.

- Розробка моделей, здатних оцінювати правильність та безпечність команд на основі контексту виконання та історії взаємодії з системою.
- Постійне доповнення і оновлення датасету новими командами та їх перекладами для підтримки розширення.
- Вдосконалення алгоритмів підготовки даних для забезпечення вищої якості перекладу, враховуючи специфічні технічні терміни та контекст використання.
- Розробка методів та алгоритмів для аналізу поточного стану системи, на якій виконується команда, щоб уникнути виконання потенційно небезпечних або несумісних дій.
- Впровадження контекстно-чутливих механізмів, які дозволять сервісу адаптувати команди з урахуванням специфіки операційної системи, наявних контейнерів та їх станів.

Враховуючи ці напрямки, ми плануємо провести ретельний аналіз існуючих датасетів та визначити потреби в додаткових даних. Очікується, що це дозволить нам покращити якість та розширити можливості нашого методу та сервісів побудованих на його основі, забезпечуючи користувачам більш точні та безпечні інструменти для взаємодії з Docker контейнерами через природну мову.

Для досягнення цієї мети будуть встановлені такі цілі:

- Ідентифікація найбільш частих проблем і випадків використання, що потребують покращення.
- Розробка та впровадження нових алгоритмів для підвищення якості перекладу, особливо з урахуванням технічної специфіки та контексту виконання команд.
- Тестування адаптованих моделей у різних середовищах, щоб забезпечити їх стабільну та ефективну роботу незалежно від особливостей системи користувача.

- Налагодження процесу безперервного навчання моделей на основі зібраних даних та зворотного зв'язку, щоб сервіс постійно вдосконалювався та адаптувався до змін у технологіях та потребах користувачів.

Вдосконалення та розширення датасетів є критично важливим для подальшого розвитку нашого сервісу. Це включає не тільки збільшення обсягу та різноманітності даних, але й розробку нових методів їх обробки та аналізу. Розуміння та адаптація до контексту виконання команд, забезпечення безпеки, інтеграція підтримки додаткових мов – всі ці аспекти вимагатимуть значних зусиль з нашого боку, але в кінцевому результаті дозволять створити більш ефективні, адаптивні та безпечні інструменти для управління Docker контейнерами через природну мову.

2.2. Вибір основної моделі перекладу

У пошуку оптимальної моделі для нашої задачі необхідно проаналізувати різні підходи машинного навчання, щоб визначити, яка з них найкраще впорається з цим завданням. Виходячи з теоретичного огляду, ми розглянемо рекурентні нейронні мережі (RNN), мережі з довгою короткочасною пам'яттю (LSTM), Шлюзові рекурентні одиниці (GRU), та трансформери.

Рекурентні моделі (RNN) були одними з перших інструментів машинного навчання, задіяними для обробки текстів. Вони здійснюють проходження через текст крок за кроком, зберігаючи інформацію про попередній контекст у своїй "пам'яті". Однак, RNN часто стикаються з викликом "затухання градієнта", коли втрачається інформація про дуже ранні входні дані в довгих послідовностях, що обмежує їхню здатність до точного перекладу складних команд.

Проблема затухання градієнту у рекурентних моделях (RNN) виникає, коли модель намагається проаналізувати інформацію з довгих текстів. У таких випадках, інформація про початок тексту "забувається" до того часу, як доходить до кінця. Це може зробити використання RNN проблемним для завдань, де

важлива кожна деталь великого тексту, наприклад, при перекладі довгих команд разом з контекстом. Коли дані містять багато деталей, які мають зберігатися в пам'яті моделі від початку до кінця, RNN може "втратити" частину цієї інформації, що може призвести до помилок у перекладі та виконанні команд. Хоча на даному етапі роботи ми не потребуємо обробки великих масивів інформації, проте в майбутньому планується модифікувати процес перекладу команд таким чином щоб сервіс перекладу розумів контекст виконання, так разом з командою ми будемо відправляти також багато додаткової інформації (стан контейнерів, мереж, логи etc.) що потенційно може викликати проблеми.

LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Units) – це типи моделей, що вдосконалюють звичайні рекурентні нейронні мережі, вирішуючи деякі з їх основних проблем. Вони мають унікальну особливість – «вентилі», які можна уявити як маленькі "ворота" всередині моделі. Ці ворота вирішують, яку інформацію слід зберегти для майбутнього використання, а яку – забути. Це дозволяє моделям краще впоратися з дуже довгими текстами, зберігаючи важливі деталі, навіть якщо вони з'явилися на початку тексту і потрібні значно пізніше.

Наприклад, уявімо, що ми читаємо довгий текст, де на початку згадується важлива деталь, яка знадобиться для розуміння закінчення. У випадку з LSTM і GRU, ці «ворота» допомагають «пам'яті» моделі зберегти цю важливу деталь до самого кінця. Вентилі контролюють потік інформації, що дуже корисно для зберігання і використання довготривалої інформації.

Однак, хоча ці моделі і ефективніше справляються з довготривалою інформацією, їх послідовний спосіб обробки даних може сповільнити обробку великих обсягів даних і збільшити вимоги до обчислювальних ресурсів. Це означає, що великі датасети вимагають більше часу та потужності комп'ютера для обробки.

Трансформери [2] – це відносно новий клас моделей у машинному навчанні, який відомий завдяки своєму унікальному механізму, відомому як

"увага" [5]. Ми можемо уявити це як здатність одразу зосередитися на найважливіших словах читаючи великий текст, не читаючи його слово за словом. Трансформери роблять саме так, швидко виявляючи важливі зв'язки між словами, навіть якщо вони далеко одне від одного у тексті. Це робить їх неймовірно швидкими та точними при роботі з великими обсягами інформації.

Цей механізм особливо корисний, коли мова йде про переклад команд. В контексті перекладу, кожне слово важливе, і зміна одного слова може радикально змінити значення всієї команди. Трансформери добре підходять для таких завдань, бо вони можуть врахувати весь контекст команди, забезпечуючи, щоб переклад був якомога точнішим.

Завдяки трансформерам ми можемо швидко та ефективно обробляти команди, перекладаючи їх з природної мови на технічні інструкції, не втрачаючи при цьому важливих деталей. Тому ми можемо вважати що трансформери є хорошим вибором в контексті нашого проекту, де швидкість, точність і здатність розуміти складні зв'язки в командах є ключовими.

Трансформери мають кілька додаткових переваг, які роблять їх особливо цінними для нашого проекту з управління Docker контейнерами:

- Масштабованість: Трансформери ефективно обробляють великі обсяги даних, що дозволяє нам розширювати датасети без суттєвого зниження продуктивності. Це особливо важливо, коли ми хочемо додавати більше прикладів команд для покращення точності перекладу.
- Гнучкість у навчанні: Завдяки механізму уваги трансформери здатні краще адаптуватися до різноманітності датасетів, що дозволяє ефективно тренувати модель на специфічних для нашого проекту прикладах команд.
- Покращення з розумінням контексту: Трансформери вирішують проблему затухання градієнту, що дозволяє їм краще розуміти контекст команд, навіть якщо ключова інформація з'являється на

великій відстані від відповіді. Це забезпечує високу точність перекладу навіть для складних та довгих команд.

- Покращення точності для маловикористовуваних мов: Трансформери, завдяки своїй високій ефективності в розпізнаванні мовних зв'язків, можуть краще працювати з різними мовами, такими як українська. Це може надати нам можливість розширювати можливості нашого сервісу для різних користувачів.
- Ефективна робота зі змінним контекстом: Трансформери ефективно впораються з командами, контекст яких може змінюватися в залежності від попередніх дій користувача або стану системи. Це може забезпечити високу адаптивність сервісу до реальних сценаріїв використання. Особливо це може бути корисним при подальших дослідженнях і розробці, коли ми додамо аналіз контексту системи на якій має бути виконана команда.

Враховуючи порівняльний аналіз різних архітектур моделей машинного навчання, було вирішено обрати трансформери як оптимальну архітектуру для проекту. Їх здатність ефективно обробляти великі послідовності даних, висока точність перекладу, а також гнучкість і масштабованість роблять їх гарним вибором для виконання нашої задачі. Далі ми зосередимось на аналізі конкретних моделей трансформерів, щоб визначити, яка з них найкраще відповідає потребам проекту.

Для забезпечення найкращого перекладу команд Docker з природної мови, необхідно розглянути декілька варіантів моделей трансформерів, серед яких особливу увагу приділено GPT [3] та фреймворку MarianMT [4].

GPT це модель з потужними генеративними здібностями, відома здатністю створювати змістовний текст. Ця модель могла б стати інструментом для генерації відповідей на запитання або автоматичного формування тексту на основі вхідних даних. Проте GPT вимагає значних обчислювальних ресурсів та додаткового навчання для спеціалізованих задач.

GPT є передовою архітектурою нейронних мереж, яка здатна генерувати текст, наближений до людського письма. Ця модель відома своїми вражаючими високими здібностями, здатністю відповідати на питання, проводити резюмування текстів, переклади, а також виконувати багато інших завдань пов'язаних з обробкою природної мови. Основною відмінністю GPT від інших моделей є її здатність до навчання з мінімальною кількістю керівництва (few-shot learning), що дозволяє їй адаптуватися до широкого спектру завдань.

Модель GPT базується архітектурі трансформерів, яка використовує механізм уваги [5] для аналізу взаємозв'язків між словами в тексті. Ця особливість дозволяє моделі враховувати контекст кожного слова, незалежно від його положення в тексті, що забезпечує високу точність генерації та розуміння тексту. Процес навчання GPT включає два етапи: спочатку модель навчається на величезному корпусі тексту, а потім адаптується до конкретного завдання з невеликою кількістю прикладів.

Одним з найбільших досягнень GPT є її здатність до генерації змістовного тексту, що може включати створення вигаданих історій, автоматичне написання коду, формулювання відповідей на питання у форматі, що наближений до людського. Ця здатність робить використання GPT надзвичайно корисним для розробки чат-ботів, систем автоматизації відповідей на запитання клієнтів, а також інструментів, що спрощують роботу з текстовим контентом. Це добре вписується в контекст нашого дослідження.

Ще одним аспектом використання GPT є необхідність додаткового навчання моделі під конкретні завдання. Хоча GPT має великі можливості "з коробки", максимальна ефективність в конкретних застосуваннях може бути досягнута лише після її адаптації та тонкої настройки. Це може включати додавання специфічного для домену тренувального набору даних або коригування параметрів моделі, щоб краще відповідати особливостям завдання.

Незважаючи на значний потенціал, використання GPT пов'язане з деякими потенційними проблемами. Перш за все, його використання вимагає значних

обчислювальних ресурсів, особливо для великих моделей, таких як GPT-3. Висока вартість та потреба в спеціалізованому апаратному забезпеченні можуть обмежити доступність GPT для деяких дослідників та розробників. Крім того, точність та надійність генерованого тексту можуть варіюватися в залежності від вхідних даних і завдання, для якого застосовується модель. Це вимагає ретельного тестування і адаптації моделі до конкретного застосування.

MarianMT, з іншого боку, є фреймворком, спеціалізованим на машинному перекладі. Його ефективність, швидкість та відносно низькі вимоги до обчислювальних ресурсів роблять його привабливим для нас в контексті нашого дослідження. MarianMT містить моделі що підтримують українську мову та здатен обробляти великий спектр інших мов, що розширює можливості його застосування.

Для проведення дослідження можливостей MarianMT було обрано попередньо треновану модель Helsinki-NLP/opus-mt-uk-en [8].

Однією з основних переваг MarianMT Helsinki-NLP/opus-mt-uk-en є її зосередженість на українсько-англійському перекладі. В контексті дослідження, де основна мова - українська, важливість точного та ефективного перекладу на англійську не можна недооцінювати. Технічні команди та інструкції часто формулюються англійською мовою, і забезпечення високоякісного перекладу є критичним для успішної взаємодії користувача з системою.

MarianMT відома своєю високою швидкістю перекладу та оптимізацією для паралельних обчислень, що робить її ідеальною для використання в реальному часі та у системах з великою кількістю запитів. Це особливо важливо для веб-сервісів та інтерактивних застосунків, де час відгуку системи має вирішальне значення для забезпечення зручності користувача.

На відміну від більших моделей, таких як GPT, MarianMT потребує менше обчислювальних ресурсів для тренування та використання. Це робить модель більш доступною для дослідників та розробників, особливо тих, хто працює з

обмеженими ресурсами або в умовах, де важлива оптимізація використання апаратного забезпечення.

MarianMT дозволяє легко адаптувати модель до специфічних завдань перекладу, включаючи технічні команди та інструкції, завдяки можливості тонкої настройки та додавання спеціалізованих тренувальних даних. Це означає, що MarianMT може бути ефективно налаштована для забезпечення високої точності перекладу в контекстах, де необхідно розуміння та переклад специфічної технічної термінології. Така адаптація критично важлива для сервісів, що надають користувачам можливість взаємодіяти з системами управління контейнерами через природну мову.

Завдяки великій кількості тренувальних даних та оптимізації моделі під пару мов українська-англійська, MarianMT Helsinki-NLP/opus-mt-uk-en забезпечує високу точність перекладу. Це особливо важливо у ситуаціях, коли від точності перекладу залежить правильність виконання команд системою. Точний переклад забезпечує, що команди, введені користувачем українською мовою, будуть коректно інтерпретовані та виконані без помилок.

MarianMT має велику підтримку різноманітності мовних даних, що дозволяє моделі ефективно працювати з текстами різного стилю та жанру. Це робить модель гнучкою та адаптивною до широкого спектру завдань перекладу, включаючи не тільки стандартний переклад текстів, але й специфічні завдання, такі як переклад технічних документацій, інструкцій чи команд.

Вибір MarianMT Helsinki-NLP/opus-mt-uk-en для дослідження, яке зосереджено на взаємодії з Docker контейнерами через природну мову, є виправданим завдяки вищеописаним перевагам. Модель не тільки забезпечує високу точність перекладу між українською та англійською мовами, але й відповідає потребам проекту у швидкості, ефективності та мінімальних вимогах до ресурсів. Крім того, здатність моделі до адаптації під специфічні завдання та її гнучкість роблять її ідеальним інструментом для створення ефективних та інтуїтивно зрозумілих інтерфейсів взаємодії з технологіями. Завдяки MarianMT,

розробники можуть створювати системи, які дозволяють користувачам виконувати складні технічні дії, використовуючи звичайну природну мову, тим самим розширюючи доступність та зручність технологічних рішень.

Таблиця 2.1

Модел ь	Спеціаліза ція	Швидкість обробки	Вимоги	Точність	Підтримка української мови
GPT	Генерація тексту, розуміння природної мови	Висока для простих завдань, знижується для складних	Високі, особливо для великих моделей	Висока залежно від завдання і навчального датасету	Залежить від тренувальн их даних
Marian MT	Машинний переклад	Висока	Помірні, оптимізована для паралельних обчислень	Висока	Так

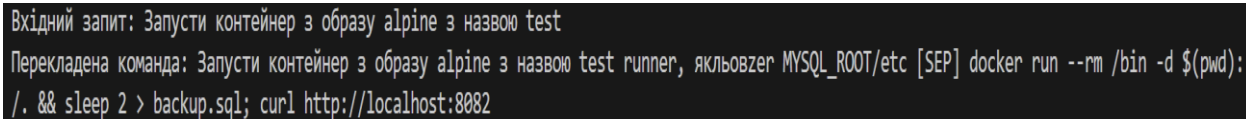
У процесі навчання моделей для перекладу команд Docker з природної мови ми зіштовхнулися з необхідністю вибору найбільш ефективної моделі. Аналізуючи результати, можна побачити, що обидві розглянуті моделі – GPT та MarianMT – продемонстрували хороші результати, але з деякими відмінностями в процесі їх тренування.

Модель GPT, яка відома своїми генеративними здібностями, показала стабільне зниження втрати з часом, що свідчить про ефективне навчання. Цей процес відбувався зі збільшенням швидкості навчання, що допомагало уникнути потрапляння в локальні мінімуми та покращувати здатність моделі до точного

передбачення. Навчання тривало близько 3 епох, демонструючи, що GPT може адаптуватися до специфіки наших даних за відносно короткий час.

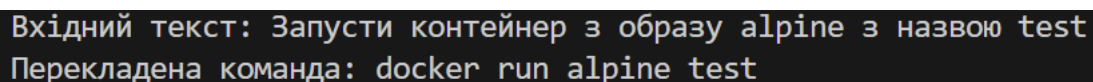
MarianMT, натомість, почала тренування з вищою початковою втратою порівняно з GPT, але згодом досягла схожого рівня кінцевої втрати. Це вказує на високу ефективність моделі у вивченні специфіки перекладу команд Docker, навіть попри складність завдання. Важливим фактором стала більш висока продуктивність MarianMT у плані швидкості обробки даних, що свідчить про її придатність для використання в умовах обмежених обчислювальних ресурсів.

Загалом, обидві моделі показали хороші результати в процесі навчання, але MarianMT виявилася більш вигідною у контексті наших завдань. Вона не тільки ефективно зменшувала втрату, але й продемонструвала вищу швидкість навчання та краще використання ресурсів, що є ключовими факторами для нашого проекту. Таким чином, MarianMT стає нашим вибором як основної моделі для перекладу команд з природної мови в інструкції, зрозумілі для Docker.



```
Вхідний запит: Запусти контейнер з образу alpine з назвою test
Перекладена команда: Запусти контейнер з образу alpine з назвою test runner, якльовзер MYSQL_ROOT/etc [SEP] docker run --rm /bin -d $(pwd):
/. && sleep 2 > backup.sql; curl http://localhost:8082
```

Рис 2.2. Результат тесту моделі GPT



```
Вхідний текст: Запусти контейнер з образу alpine з назвою test
Перекладена команда: docker run alpine test
```

Рис 2.3. Результат тесту моделі MarianMT

У процесі тестування навчених моделей MarianMT та GPT було проведено простий експеримент, щоб перевірити, як кожна з моделей впорається з перекладом команди на природній мові в команду Docker CLI. Результати тесту показали цікаві особливості обох моделей.

Для моделі MarianMT, вхідною командою було "Запусти контейнер з образу alpine з назвою test". Перекладена команда "docker run alpine test" не була повністю коректною, але вона відображала частково правильне розуміння завдання і здатна створити контейнер з вказаним образом. Це демонструє, що

MarianMT має здатність до точного перекладу, хоча і потребує подальшого вдосконалення.

В той же час, результат, отриманий від моделі GPT, значно відрізнявся. З вхідною командою "Запусти контейнер з образу alpine з назвою test", GPT генерувала довгий і складний вихід, який містив елементи схожі на команди Docker CLI, але в цілому був некоректним та не відповідав заданій інструкції. Переклад від GPT містив низку додаткових інструкцій, що не мали відношення до поставленого запиту, що свідчить про недостатнє розуміння контексту та завдання перекладу в цьому конкретному випадку.

Ці результати підтверджують важливість вибору спеціалізованих моделей для конкретних завдань. MarianMT, будучи спеціалізованою на машинному перекладі, показала кращі результати у розумінні та перекладі команд Docker, навіть з урахуванням деяких неточностей. GPT, хоча й демонструє здатність до генерації складних текстів, не змогла адекватно впоратися з завданням перекладу команди в контексті цього дослідження.

Таким чином, на основі результатів тестування та проведеного аналізу, можна зробити висновок, що MarianMT з моделю helsinki-NLP/opus-mt-uk-en краще підходить для нашого проекту, оскільки забезпечує більш точний та релевантний переклад команд Docker, важливий для нашого дослідження.

Проте для повного розуміння важливо знати як працює механізм перекладу в MarianMT та чому вона здатна точно перекладати запити не просто на англійську мову, а на bash та Docker CLI після адаптації на невеликому датасеті.

MarianMT базується на архітектурі трансформер, яка використовує механізм самоуваги (self-attention) для моделювання взаємозв'язків між словами у вхідному тексті. Це дозволяє моделі враховувати контекст кожного слова в реченні, що підвищує точність перекладу.

Перед використанням для конкретного завдання перекладу, MarianMT навчається на великих масивах текстів, що містять приклади перекладу між

різними мовами. Після цього, модель може бути адаптована до специфічних завдань за допомогою тонкої настройки на менших датасетах, що містять спеціалізовану термінологію або конкретні види текстів, такі як команди `bash` та `Docker CLI`.

Під час адаптації на невеликому датасеті, який містить приклади команд на природній мові та їх відповідники на `bash` та `Docker CLI`, MarianMT вивчає специфічні шаблони і термінологію, які використовуються в цих командах. Це дозволяє моделі розпізнавати не тільки загальний зміст вхідного тексту, але й генерувати точні технічні команди, які відповідають вимогам задачі.

Точність перекладу забезпечується завдяки здатності MarianMT розуміти контекст вхідних фраз і правильно інтерпретувати технічні деталі, необхідні для формування коректних команд `bash` і `Docker CLI`. Механізм самоуваги дозволяє моделі зосереджуватися на ключових словах та виразах, що мають вирішальне значення для формування команди, ігноруючи менш важливі елементи. Таким чином, навіть якщо вхідний текст містить зайву або неоднозначну інформацію, модель здатна виділити та використати лише необхідну інформацію для створення точної команди.

Трансферне навчання відіграє одну з ключових ролей у процесі адаптації MarianMT до специфічних задач. Завдяки цьому підходу, знання, отримані моделлю під час попереднього навчання на великих датасетах, можуть бути ефективно застосовані та доповнені інформацією з набагато менших спеціалізованих датасетів. Це дозволяє моделі швидко адаптуватися до нових завдань, зберігаючи при цьому високу точність перекладу.

Вибір конкретної моделі MarianMT, такої як `Helsinki-NLP/opus-mt-uk-en`, спеціально оптимізованої для перекладу між українською та англійською мовами, є важливим чинником, що забезпечує високу якість перекладу. Спеціалізація на конкретних мовних парах дозволяє моделі краще враховувати мовні особливості та нюанси, що підвищує точність перекладу команд з природної мови на технічні.

Технічні тексти, зокрема команди `bash` та `Docker CLI`, часто містять специфічну термінологію та структуровані вирази, які важливо точно перекласти. Модель, навчена на даних, які включають технічні текстові дані, краще розуміє контекст і призначення технічних команд. Це дозволяє MarianMT адаптуватися до особливостей перекладу з української на англійську в контексті програмування та системного адміністрування.

Модель, оптимізована для конкретних мовних пар, легше адаптується під специфічні завдання, такі як переклад команд. Вона може бути додатково налаштована шляхом тренування на спеціалізованих датасетах, що містять приклади команд. Це дозволяє досягти високої точності перекладу навіть для складних або незвичайних команд.

Використання спеціалізованої конфігурації `Helsinki-NLP/opus-mt-uk-en` для MarianMT у дослідженні з перекладу української мови на команди `bash` і `Docker CLI` виявилось виправданим вибором з кількох причин. По-перше, забезпечення високої точності перекладу є критично важливим для правильної інтерпретації та виконання команд системою. Спеціалізована модель забезпечує глибше розуміння контексту і технічних деталей, що необхідно для ефективного перекладу команд.

По-друге, оптимізація моделі під українсько-англійський переклад сприяє кращому збереженню смислу технічних команд у процесі перекладу. Це особливо важливо, коли деталі команди впливають на її виконання та потенційні результати. Правильний переклад з української на англійську забезпечує, що команди будуть інтерпретовані точно, відповідно до їх призначення та очікуваного результату.

По-третє, здатність моделі до швидкої адаптації під специфічні задачі є великою перевагою. Завдяки гнучкості MarianMT та можливості тренувати модель на вибраних датасетах, її можна ефективно налаштувати для розпізнавання та перекладу навіть найбільш специфічних та складних команд.

Це дозволяє розширити можливості системи управління контейнерами, роблячи її більш гнучкою та зручною для користувачів.

У сукупності, ці фактори роблять MarianMT і, зокрема, конфігурацію Helsinki-NLP/opus-mt-uk-en, оптимальним вибором для дослідження перекладу команд з природної мови на технічні команди.

Окрім MarianMT та GPT, існує ряд інших моделей трансформерів, які також мають потенціал у контексті нашого дослідження. Серед них - BERT, T5 та інші спеціалізовані моделі, кожна з яких має свої унікальні особливості та можливості. Деякі з цих моделей можуть бути більш ефективними у певних аспектах перекладу або обробки мови, що робить їх вартими уваги в майбутніх дослідженнях.

При подальших дослідженнях важливо не тільки тестувати різні моделі з різноманітними параметрами налаштування, але й збирати більшу кількість високоякісних даних для навчання. Вдосконалення датасетів, зокрема додавання більшої кількості коректних прикладів команд, може суттєво підвищити якість перекладу та зробити процес використання мовних моделей для управління Docker контейнерами ще більш ефективним.

2.3. Архітектура сервісу

Сервіс управління Docker контейнерами через природні мови має бути побудовано з використанням передових методик машинного навчання та обробки природної мови. Архітектура даного сервісу організована навколо декількох ключових компонентів, що забезпечують ефективну трансформацію мовних команд користувачів у виконувани Docker CLI інструкції.

Однією з основних частин є модель MarianMT, адаптована для задачі машинного перекладу команд з природної мови в Bash команди. Дана модель була додатково оптимізована за допомогою процесу фін-тюнінгу на спеціалізованому датасеті, що включає приклади мовних команд та їх відповідності у форматі Docker CLI.

Для забезпечення високої точності перекладу та валідності генерованих команд в системі використовується API ChatGPT. Цей компонент відіграє роль рівня валідації, покращуючи якість та надійність перекладених інструкцій, а також може бути використаний для генерації правильної команди у випадку хибної відповіді від основної моделі.

Інформація про команди зберігаються у базі даних MongoDB. Це дозволяє ефективно використовувати зібрані дані для подальшого навчання моделі, сприяючи її удосконаленню.

Інтеграція з Docker відбувається через виконання перекладених команд, надаючи можливість користувачам взаємодіяти з контейнерами за допомогою природні мови.

Навчання та оптимізація моделі MarianMT здійснюється за допомогою спеціальних Python скриптів. На поточний момент ці скрипти функціонують окремо від основної системи, але при подальших дослідженнях та розробках має бути виконана їх інтеграція в сервіс для автоматизації процесу навчання та адаптації моделі.

Для ефективного масштабування при переході від прототипу до повноцінного сервісу, особливо з урахуванням зростаючої кількості одночасних запитів і користувачів, планується перехід від монолітної архітектури до мікросервісної. Мікросервісна архітектура дозволить розділити весь процес на незалежні сервіси, такі як переклад, валідація, виконання команд тощо, кожен з яких буде працювати автономно. Це сприятиме кращій масштабованості, збільшенню надійності та поліпшенню швидкодії системи в цілому [7].

Перехід на мікросервісну архітектуру дозволить додати гнучкості в розгортання окремих компонентів сервісу. Це полегшує розподіл навантаження, впровадження оновлень та розширення функціональності без впливу на інші частини системи.

Laravel [13] надає потужні інструменти для управління асинхронними завданнями, такими як черги і jobs. Використання цих інструментів дозволить

ефективно розподіляти обробку запитів та виконання команд, зменшуючи час відгуку та оптимізуючи ресурси сервера. Черги можуть використовуватися для тимчасового зберігання завдань, що мають бути виконані, забезпечуючи стабільну роботу системи під високим навантаженням.

Кожен мікросервіс буде відповідальний за конкретну ділянку логіки. Наприклад, сервіс перекладу може фокусуватися лише на перекладі команд, в той час як сервіс валідації перевірятиме коректність і безпеку виконання. Це забезпечує більшу спеціалізацію та покращення якості кожного аспекту сервісу.

Завдяки мікросервісній архітектурі, сервіс може легко масштабуватися шляхом додавання більшої кількості інстанцій для кожного з мікросервісів у відповідь на збільшення навантаження. Це дозволяє легше адаптуватися до змін у вимогах до обробки запитів і забезпечує стабільність роботи при піках навантаження.

Використання окремих мікросервісів для обробки команд дозволяє оптимізувати процес їх виконання. Наприклад, можна впровадити механізм кешування для часто використовуваних команд або пріоритезувати обробку критичних запитів. Це підвищує ефективність використання ресурсів та знижує час відгуку для кінцевих користувачів.

Застосування цих стратегій дозволить не тільки масштабувати прототип до повноцінного сервісу, але й оптимізувати його роботу, забезпечуючи високу швидкість обробки запитів, надійність і доступність для широкого кола користувачів. Перехід на мікросервісну архітектуру вимагає ретельного планування та впровадження, але переваги, які вона пропонує, значно перевищують витрати на її реалізацію.

Для доопрацювання архітектури при переході до повноцінного сервісу, одним із ключових аспектів є розробка і інтеграція фронтенду, який забезпечить кінцевим користувачам графічний інтерфейс для взаємодії з системою. Цей елемент архітектури відіграє критичну роль у забезпеченні інтуїтивно зрозумілого та зручного доступу до функціональності сервісу, дозволяючи

користувачам вводити команди природною мовою та отримувати відповіді в реальному часі.

Фронтенд сервісу може бути реалізований як веб-додаток, що дозволяє користувачам отримувати доступ до системи з будь-якого пристрою з підтримкою веб-браузера, включно з настільними комп'ютерами, ноутбуками, планшетами та смартфонами. Це рішення має забезпечити широкую доступність сервісу та його адаптивність до різних умов використання.

Основним завданням фронтенду є надання користувачам зрозумілого та зручного інтерфейсу для введення команд та перегляду результатів їх виконання. Це включає розробку текстових полів для введення команд, вікон відповідей для відображення результатів виконання команд, а також інтерфейсу управління станом Docker контейнерів, який може включати інформацію про статус, налаштування та логи контейнерів.

Додатково, фронтенд може включати елементи інтерактивної допомоги та підтримки користувачів, такі як інтерактивні посібники, підказки щодо використання команд, FAQ секції та функцію зворотного зв'язку. Це допоможе користувачам швидше адаптуватися до сервісу та ефективніше використовувати його можливості.

Інтеграція фронтенду з серверною частиною сервісу має бути забезпечена через використання API для обміну даними, що дозволить фронтенду відправляти команди на сервер для обробки та отримувати відповіді. Це вимагає ретельної проробки API, забезпечення його безпеки та ефективності для підтримки швидкого та надійного зв'язку між клієнтом та сервером.

2.4. Адаптація моделі MarianMT та валідація і корекція команд з використанням ChatGPT

В процесі адаптації моделі MarianMT до завдань перекладу команд для управління Docker контейнерами було проведено відбор даних. Для цього було створено датасет, який об'єднав як сформовані вручну приклади, так і записи,

згенеровані автоматично за допомогою API ChatGPT. Мета полягала у забезпеченні різноманітності та якості даних для охоплення широкого спектру команд Docker, представлених як природною мовою, так і відповідними їм командами Bash. Важливою частиною підготовки стало виправлення помилок та вилучення дублікатів, що допомогло підвищити якість датасету для подальшого навчання.

Для доопрацювання моделі MarianMT було розроблено та використано скрипт на Python, який забезпечив інтеграцію моделі з базою даних MongoDB для доступу до актуального датасету. Файн-тюнінг здійснювався протягом трьох епох з обраними параметрами навчання, що включали оптимізацію гіперпараметрів, таких як кроки розігріву та регуляризації, спрямовані на підвищення ефективності процесу навчання. Ефективність файн-тюнінгу оцінювалася на основі здатності моделі впоратися з перекладом команд. Спостерігалось покращення точності перекладів з кожним циклом навчання, коли до процесу були додані нові дані з реального використання сервісу.

Це дозволило моделі ефективніше адаптуватися до специфіки задач перекладу, забезпечуючи користувачам можливість використовувати більш складні та точні команди для керування Docker контейнерами. У контексті нашого дослідження ми також експериментували з тренуванням моделі GPT разом із MarianMT, щоб порівняти ефективність різних підходів.

Адаптація MarianMT показала обнадійливі результати, ілюструючи потенціал моделі для завдань перекладу в контексті управління Docker контейнерами.

Цей процес також виявив значну потребу в постійному оновленні датасету для забезпечення актуальності і релевантності моделі, що є ключовим для адаптації до нових версій Docker та змін у командному інтерфейсі. Крім того, результати навчання підкреслили важливість глибшої інтеграції з контекстуальними даними системи, що дозволить моделі більш точно інтерпретувати наміри користувачів та генерувати адекватні команди в різноманітних сценаріях використання.

```
{'loss': 0.3822, 'learning_rate': 2.8000000000000003e-05, 'epoch': 1.71}
{'loss': 0.2376, 'learning_rate': 2.9e-05, 'epoch': 1.77}
{'loss': 0.1307, 'learning_rate': 3e-05, 'epoch': 1.83}
{'loss': 0.2569, 'learning_rate': 3.1e-05, 'epoch': 1.89}
{'loss': 0.2489, 'learning_rate': 3.2000000000000005e-05, 'epoch': 1.95}
{'loss': 0.1705, 'learning_rate': 3.3e-05, 'epoch': 2.01}
{'loss': 0.1933, 'learning_rate': 3.4000000000000007e-05, 'epoch': 2.07}
{'loss': 0.1717, 'learning_rate': 3.5e-05, 'epoch': 2.13}
{'loss': 0.1935, 'learning_rate': 3.6e-05, 'epoch': 2.2}
{'loss': 0.1196, 'learning_rate': 3.7e-05, 'epoch': 2.26}
{'loss': 0.2447, 'learning_rate': 3.8e-05, 'epoch': 2.32}
{'loss': 0.1622, 'learning_rate': 3.9000000000000006e-05, 'epoch': 2.38}
{'loss': 0.1717, 'learning_rate': 4e-05, 'epoch': 2.44}
{'loss': 0.1223, 'learning_rate': 4.1e-05, 'epoch': 2.5}
{'loss': 0.2021, 'learning_rate': 4.2e-05, 'epoch': 2.56}
{'loss': 0.1453, 'learning_rate': 4.3e-05, 'epoch': 2.62}
{'loss': 0.1732, 'learning_rate': 4.4000000000000006e-05, 'epoch': 2.68}
{'loss': 0.134, 'learning_rate': 4.5e-05, 'epoch': 2.74}
{'loss': 0.1713, 'learning_rate': 4.6000000000000001e-05, 'epoch': 2.8}
{'loss': 0.1486, 'learning_rate': 4.7e-05, 'epoch': 2.87}
{'loss': 0.2493, 'learning_rate': 4.8e-05, 'epoch': 2.93}
{'loss': 0.1899, 'learning_rate': 4.9e-05, 'epoch': 2.99}
```

Рис 2.4. Логи адаптації моделі MarianMT

Подальші дослідження та розробка передбачають не лише постійне оновлення датасету, а й пошук нових моделей і оптимізацію параметрів для підвищення якості перекладу. Важливою є розробка і втілення новітніх методів збору та обробки даних, що включає автоматизацію процесів збору високоякісних даних і їх аналізу для постійного покращення моделі.

Для підвищення якості та точності перекладу команд, було використано API ChatGPT, який надає доступ до моделі GPT-4 – передової технології в області обробки природної мови, розробленої компанією OpenAI. Завдяки високій якості та глибокому розумінню мовних зв'язків, GPT-4 може генерувати виправлення та валідації для перекладених команд з вражаючою точністю, особливо коли ми надаємо чіткі та деталізовані інструкції.

Однак, використання API ChatGPT має свої обмеження. По-перше, доступ до GPT-4 не є безкоштовним, і вартість використання API може зростати з підвищенням кількості запитів. Це ставить бюджетні обмеження на масштабування сервісу, особливо при обробці великої кількості команд. По-

друге, взаємодія через API має технічні обмеження, такі як ліміти на кількість запитів за одиницю часу та затримки у відповідях, що може вплинути на швидкість обробки команд у реальному часі, особливо якщо сервіс буде активно експлуатуватись.

У світлі цих обмежень, при подальших дослідженнях необхідно продовжувати розвивати та вдосконалювати власні методи валідації та корекції перекладів. Це дозволить зменшити залежність від зовнішніх сервісів, оптимізувати витрати та покращити швидкість обробки. Адаптація окремої моделі машинного навчання для внутрішньої валідації команд є одним з ключових кроків у подальших дослідженнях, який допоможе забезпечити високу якість та ефективність управління Docker контейнерами за допомогою нашого методу.

Ось декілька ключових аспектів, на які слід звернути увагу:

- Для створення ефективного сервісу валідації необхідно розробити спеціалізований датасет, що включатиме широкий спектр команд: від простих до складних, з можливими помилками або неточностями. Цей датасет має містити як валідні команди, так і ті, що потребують корекції або виправлення, з детальними поясненнями помилок для кожного випадку.
- Введення етапу передвалідації дозволить виявляти потенційні помилки на ранньому етапі обробки запиту, що сприятиме зменшенню кількості невалідних команд, що доходять до етапу виконання. Це може бути реалізовано шляхом аналізу синтаксису та логічної структури команди з метою виявлення невідповідностей або неправильних викликів.
- Розвиток механізмів автоматичного виправлення дозволить сервісу не тільки вказувати на помилки, але й пропонувати варіанти їх виправлення, засновані на контексті використання команд та знаннях про типові помилки користувачів.

- Розвиток власного сервісу валідації зменшує залежність від ChatGPT та інших сторонніх API, що дозволяє краще контролювати витрати та час відгуку системи. Це також сприяє забезпеченню стабільної роботи сервісу навіть при великому навантаженні.

2.5 Майбутні напрямки дослідження

У межах даного дослідження ми розглянули ряд ключових аспектів і викликів, пов'язаних з розробкою сервісу управління Docker контейнерами через природні мови. Однак, перед нами стоять численні майбутні напрямки для подальшого розвитку та вдосконалення проекту. Нижче представлено підсумок основних аспектів, що вимагають додаткових досліджень та розробки:

- Вдосконалення моделей перекладу:
 - Розвиток та адаптація спеціалізованих моделей для точнішого перекладу мовних команд у технічні інструкції.
 - Вивчення можливості та реалізація включення додаткових мов.
- Оптимізація процесу валідації команд:
 - Розробка внутрішніх механізмів для автоматичної валідації та корекції перекладених команд.
 - Створення датасетів для тренування моделей валідації з прикладами невалідних команд та їх коректними варіантами.
- Масштабування та оптимізація сервісу:
 - Перехід від монолітної архітектури до мікросервісної для підвищення ефективності та масштабованості.
 - Впровадження системи керування чергами для оптимізації обробки запитів та команд.
- Розширення бази даних та оптимізація її структури:

- Удосконалення структури бази даних для підтримки різноманітності даних та оптимізації запитів.
- Аналіз та можлива інтеграція додаткових баз даних для зберігання інформації про команди, моделі та налаштування сервісу.
- Вивчення потенціалу природних мов як інтерфейсу:
 - Дослідження можливостей використання природних мов для управління іншими ІТ-системами та сервісами.
 - Аналіз викликів та обмежень, пов'язаних з інтеграцією мовних інтерфейсів у складні технологічні рішення.

Реалізація цих напрямків вимагатиме не тільки технічної розробки, але й глибокого аналізу потреб користувачів, вивчення передових практик у сфері машинного навчання та обробки природної мови, а також адаптації до змінюваних умов ринку ІТ-технологій.

- Розширення функціональності сервісу:
 - Додавання нових функцій, таких як підтримка більшої кількості Docker команд, інтеграція з іншими інструментами контейнеризації
 - Розробка інтерфейсу користувача для спрощення взаємодії з сервісом, зокрема через веб-інтерфейси та мобільні додатки.
- Покращення безпеки та надійності сервісу:
 - Впровадження розширених механізмів автентифікації та авторизації для захисту даних користувачів.
 - Розробка стратегій відновлення після збоїв та забезпечення високої доступності сервісу.
 - Впровадження просунутої валідації що унеможливить, або хоча б зменшить ситуації коли невірно перекладена команд задіює шкоду системі.

- Впровадження детальних повідомлень про переклад команд для того щоб користувач міг прогнозувати потенційні наслідки виконання цих команд.
- Дослідження способів зниження витрат на обчислювальні потужності, зокрема через оптимізацію алгоритмів та вибір ефективних обчислювальних платформ.
- Співпраця зі спільнотою та залучення користувачів

Майбутнє дослідження та розвиток проекту повинні зосередитися на інтеграції новітніх досягнень у сфері машинного навчання та штучного інтелекту, а також на створенні максимально зручного та безпечного інструменту для керування Docker контейнерами через природні мови. Систематичний підхід до розширення функціоналу, оптимізації продуктивності та вдосконалення інтерфейсу користувача дозволить створити універсальний та високоефективний сервіс, який задовольнить потреби широкого кола користувачів.

У розвитку повноцінного сервісу управління Docker контейнерами через природні мови, одним із ключових аспектів є розробка окремого модуля, який відповідатиме за безпосереднє виконання команд. Цей модуль має стати центральним елементом архітектури сервісу, що забезпечує взаємодію між перекладеними командами та Docker API.

Потенційно, це може вимагати переосмислення самого підходу до перекладу команд. Замість безпосереднього перекладу команд з природної мови на Bash або Docker CLI, може бути ефективнішим перекладати їх на набір внутрішніх інструкцій, які будуть безпосередньо інтерпретовані модулем виконання команд. Наприклад, команда для створення нового контейнера та встановлення в нього РНР може бути представлена як послідовність об'єктів з певними діями.

```
[
{
  "action": "createContainer",
  "name": "test",
  "image": "Ubuntu"
},
{
  "action": "execute",
  "shell": "bash",
  "commands": ["apt update", "apt install -y php"]
}
]
```

Рис 2.5. Структура пайплайну для створення контейнера та встановлення залежності

Ця теоретична концепція потребує значного доопрацювання та тестування, проте вона може стати ключовою у майбутніх дослідженнях з кількох причин:

- **Універсальність:** Набір внутрішніх інструкцій може бути адаптований під різні середовища виконання, не обмежуючись лише Docker. Це робить підхід більш гнучким та універсальним.
- **Безпека:** Використання стандартизованого набору інструкцій може підвищити рівень безпеки, оскільки кожна дія може бути точно перевірена та валідована перед виконанням.
- **Оптимізація:** Модуль виконання команд може бути оптимізований для ефективного виконання складних пайплайнів, зменшуючи затримки та покращуючи продуктивність сервісу.
- **Адаптивність:** Система, здатна інтерпретувати команди як набір дій, може легше адаптуватися до змін у технологічних стеках та інструментах управління контейнерами, забезпечуючи тривалу актуальність сервісу.

Цей підхід вимагає ретельного планування та розробки, включаючи створення детальних специфікацій для інструкцій, розробку алгоритмів

переклад. Та інтерпретації цих інструкцій модулем виконання команд. Окрім того, потрібно буде розробити механізми валідації та безпеки, щоб гарантувати, що виконання інструкцій не призведе до несанкціонованого доступу або змін у системі користувача.

Висновки до розділу 2

У розділі 2 було проведено аналіз різних моделей машинного навчання, зокрема рекурентних нейронних мереж, LSTM, GRU, та трансформерів, з акцентом на трансформери як найбільш ефективний інструмент для нашого завдання.

Було описано процес збору даних, використовуючи комбінацію ручного додавання та автоматизованої генерації через ChatGPT, щоб створити різноманітний та якісний датасет. Наша основна модель перекладу, MarianMT, була обрана після порівняння з іншими технологіями, завдяки її ефективності, швидкості та здатності працювати з малорозповсюдженими мовами.

Також було звернуто увагу на важливість валідації та корекції перекладених команд, використовуючи ChatGPT для досягнення високого рівня точності. Попри ефективність цього підходу, було визнано необхідність розробки власної системи валідації під час подальших досліджень для зменшення залежності від зовнішніх API та оптимізації витрат.

У підсумку, було виявлено значний потенціал використання трансформерів для перекладу команд управління Docker контейнерами. При подальших дослідженнях планується продовжити оптимізацію сервісу, розширюючи датасет, покращуючи моделі перекладу та розробляючи власні рішення для валідації та корекції команд, з метою підвищення доступності та ефективності управління контейнерами через природні мови.

Розділ 3. РЕАЛІЗАЦІЯ ПРОТОТИПУ СЕРВІСУ

3.1. REST API та команди

У рамках розробки прототипу сервісу було впроваджено декілька REST API ендпоінтів, які відповідають за переклад команд та управління моделями перекладу. Ендпоінт POST /execute є ключовим в інтерфейсі нашої системи, він здійснює обробку команди користувача перекладаючи її за допомогою активної моделі перекладу. У разі виявлення помилок або неточностей у перекладі, система вдається до використання API ChatGPT для корекції та валідації команди. Для додавання, оновлення, видалення та перегляду моделей перекладу розроблено декілька додаткових ендпоінтів.

POST	api/execute
DELETE	api/translation-model/{id} ..
POST	api/translation-model/{id} ..
GET HEAD	api/translation-model/{id} ..
GET HEAD	api/translation-models
POST	api/upload-model

Рис 3.1. Список розроблених ендпоінтів

Структура бази даних для зберігання інформації про моделі перекладу була спроектована з огляду на простоту та можливість адаптації при майбутніх дослідженнях. Таблиця translation_models включає поля для зберігання назви моделі, шляху до файлу моделі, статусу активності та опису, дозволяючи ефективно керувати доступними ресурсами.

Проте, з погляду масштабування проекту та переходу до повноцінного сервісу, існуюча структура бази даних MySQL потребуватиме значного розширення. Майбутнє розширення бази даних може включати додавання таблиць для зберігання детальної інформації про:

- Датасети: Таблиці для зберігання метаданих та статистики датасетів, використовуваних для навчання моделей, що дозволить ефективно управляти даними та оцінювати їх якість.

- **Логи перекладу:** Зберігання історії запитів перекладу, включаючи вхідні дані, вивід моделі та час обробки, для аналізу ефективності та виявлення потенційних помилок.
- **Конфігурації моделей:** Додаткові налаштування моделей, такі як параметри файн-тюнінгу, версії та залежності, що спростять адміністрування та оновлення моделей.
- **Конфігурація систем:** дані про конфігурації систем на яких виконується керування контейнерами
- **Конфігурація контейнерів:** зберігання інформації про контейнери, їх стан, встановлені пакети, etc.
- **Місця зберігання Dockerfile.** В майбутньому буде логічним додати підтримку операцій з Dockerfile та docker-compose. Для цього майже напевне може знадобитись зберігати інформацію про проекти які пов'язані з даними файлами.
- **Користувачі та ролі:** Управління доступом до системи через зберігання інформації про користувачів, їхні ролі та права доступу до різних функцій сервісу.

Основна логіка обробки команд українською мовою включає кілька кроків:

- Прийом команди від користувача через ендпоінт /execute.
- Переклад команди за допомогою вибраної моделі перекладу.
- Валідація перекладеної команди через ChatGPT API.
- Корекція неправильно перекладеної команди за допомогою ChatGPT API.
- Запис коригованої команди у базу даних як частини датасету для подальшого вдосконалення моделі.
- Надання відповіді з результатом роботи команди користувачеві або повідомлення про помилку, якщо таке трапилось.

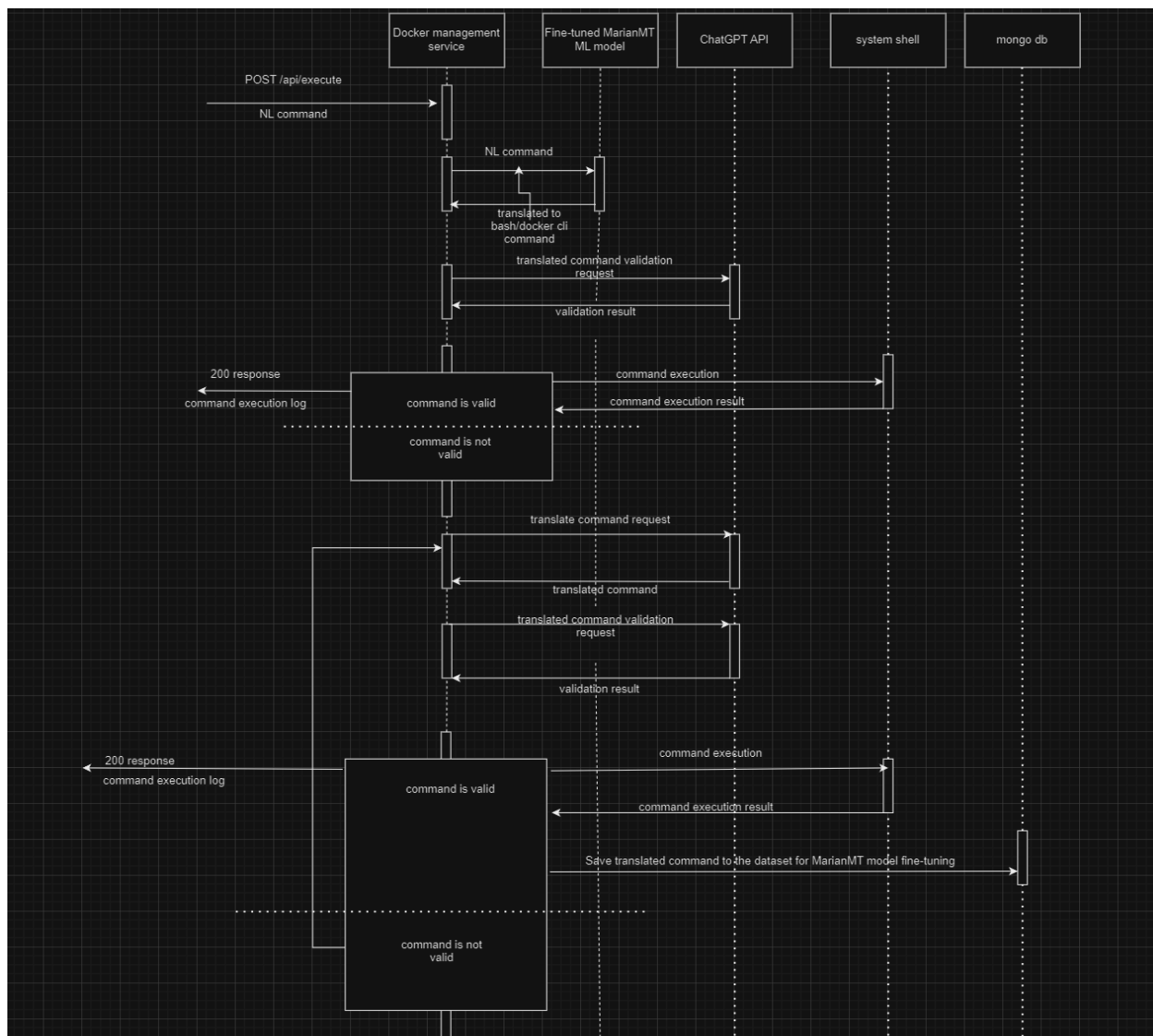


Рис 3.2. Діаграма процесів ендпоінту POST /execute

Таким чином використовується інноваційний комбінований підхід, залучається як власна, адаптована під завдання сервісу модель перекладу, так і спеціально сформовані запити до API ChatGPT від OpenAI. Цей метод дозволяє не тільки перекладати команди з природної мови у bash/docker CLI команди, але й забезпечує безперервне покращення якості перекладу шляхом генерації нових даних для навчання моделі.

Для підтримки та оптимізації процесу збору даних для навчання моделей, також було реалізовано дві консольні команди Laravel:

php artisan generate:dataset {inputDatabaseCollection} {outputDatabaseCollection}, що дозволяє генерувати нові записи датасету з використанням ChatGPT API.

```

sysadmin@DESKTOP-S1V4C76:~/projects/dockerManagementApi$ php artisan generate:dataset docker_dataset.commands docker_dataset.test
Attempt: 0 Input: Створи контейнер з ім'ям my_container, використовуючи образ docker/whalesay Output:
Attempt: 1 Input: Створи контейнер з ім'ям my_container, використовуючи образ docker/whalesay Output:docker run --name my_container -d docker/whalesay
Додано нову команду: docker run --name my_container -d docker/whalesay
Attempt: 0 Input: Запусти контейнер з ім'ям my_container в фоновому режимі із образу Ubuntu Output:
Attempt: 1 Input: Запусти контейнер з ім'ям my_container в фоновому режимі із образу Ubuntu Output:docker run -d --name my_container ubuntu
Додано нову команду: docker run -d --name my_container ubuntu
Attempt: 0 Input: Зупини роботу контейнера з ідентифікатором 123456789abc Output:

```

Рис 3.3. Приклад генерації команд за допомогою generate:dataset

```

_id: ObjectId('66043312a1e618c07902bbe2')
input : "Створи контейнер з ім'ям my_container, використовуючи образ docker/wha..."
output : "docker run --name my_container -d docker/whalesay"

_id: ObjectId('6604331aa1e618c07902bbe3')
input : "Запусти контейнер з ім'ям my_container в фоновому режимі із образу Ubu..."
output : "docker run -d --name my_container ubuntu"

```

Рис 3.4 Елементи датасету згенеровані за допомогою generate:dataset

3.2. Засоби реалізації

Використання фреймворку Laravel дозволило швидко розробити простий та ефективний API, що став основою прототипу. Для зберігання даних були обрані MySQL та MongoDB, завдяки чому вдалося ефективно організувати роботу зі структурованими та неструктурованими даними відповідно.

Особливу увагу в проекті було приділено роботі з моделями машинного навчання за допомогою мови програмування Python та бібліотеки Transformers. Використання моделі MarianMT для перекладу команд дозволило забезпечити базовий рівень взаємодії між користувачем та системою. Однак, зіткнувшись із викликами точності та адаптивності перекладу, було впроваджено використання валідації та корекції результатів з використанням API ChatGPT за допомогою

бібліотеки `openai-php/laravel`. При цьому валідація та корекція команд здійснюється завдяки спеціально підготованим контекстуальним повідомленням до яких перед відправкою до стороннього API конкатенуються команди які необхідно провалідувати чи відкоригувати. Важливо забезпечити повноту та точність інструкцій валідації та корекції за допомогою ChatGPT API, особливо на ранніх етапах коли особливо важливо зібрати коректні дані для подальшого покращення основної моделі, та розробок власних моделей валідації.

При проведенні подальших досліджень планується зменшити залежність від зовнішніх API, фокусуючись на розробці та впровадженні власної моделі для валідації та корекції команд. Це дозволить не тільки оптимізувати витрати, але й значно підвищити швидкість обробки запитів. Розробка власної моделі вимагатиме детального аналізу існуючих даних та вдосконалення алгоритмів машинного навчання, але в кінцевому результаті може зробити сервіс більш ефективним.

Планується, що окрема модель валідації буде адаптуватись на великому датасеті команд та їх перекладів, що накопичуватимуться в процесі роботи сервісу. Особлива увага буде приділена не тільки точності перекладу, але й здатності моделі адаптуватися до специфіки задач, що виникають у реальних сценаріях використання Docker.

Окрім розробки власної моделі для валідації, планується вдосконалення механізмів збору та аналізу даних, що дозволить не лише покращити якість перекладу, але й забезпечити систему необхідними даними для тренування моделі.

3.3. Приклад роботи системи

Розглянемо сценарій, де користувач бажає створити новий Docker контейнер на базі образу Alpine та встановити в нього пакет PHP.

Користувач формує запит до API на виконання цієї задачі, використовуючи українську мову. Система спочатку використовує модель

машинного навчання для перекладу цієї команди на відповідний bash та Docker CLI виклик. Після перекладу команда валідується та виконується на хост-машині в випадку якщо запит валідації повернув результат що вище або дорівнює певній границі. У разі, якщо переклад містить помилки, система коригує їх, використовуючи API ChatGPT, зберігає команду користувача разом з валідною перекладеною командою в базу даних та виконує команду на хост машині повертаючи користувачу дані про процес виконання.

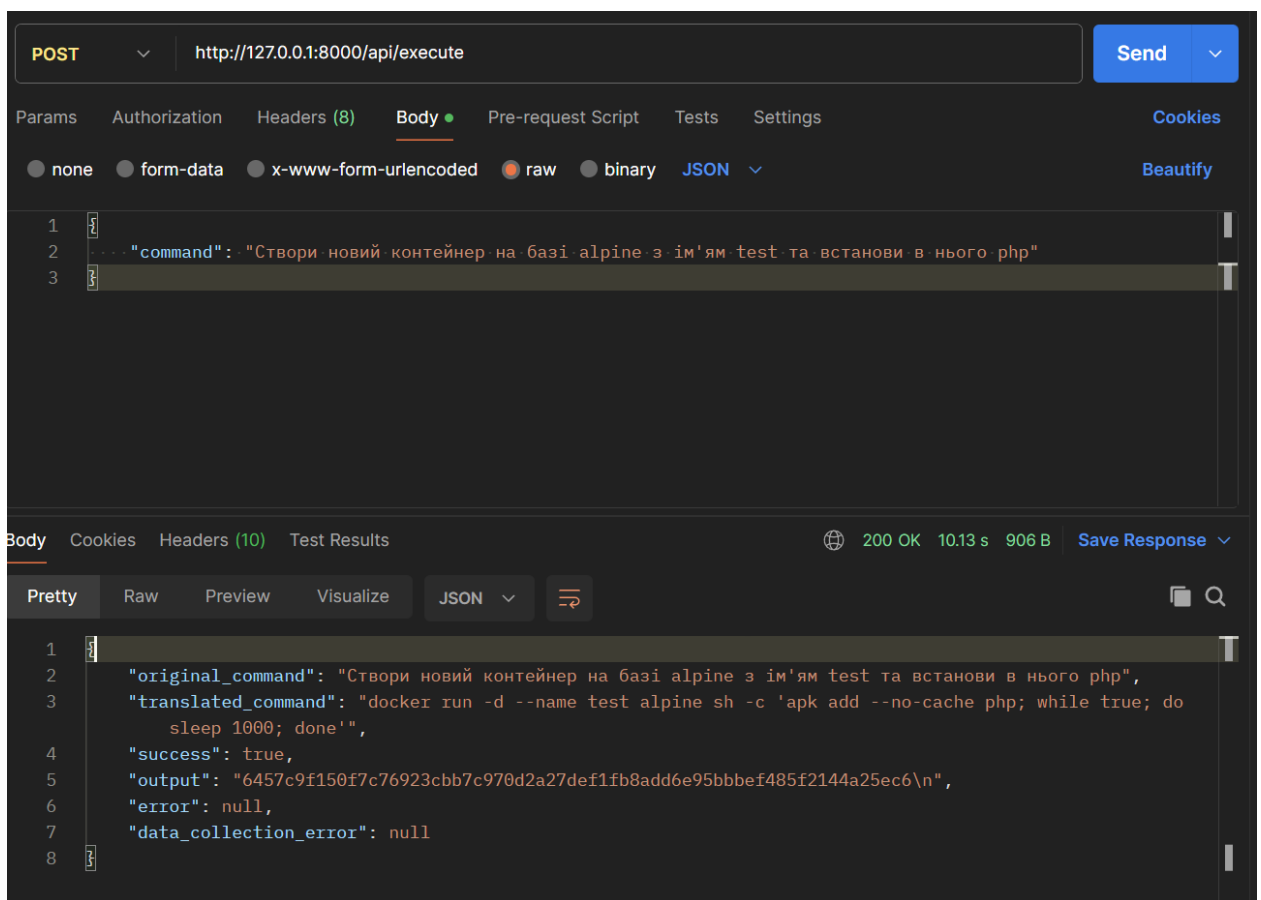


Рис 3.5. Успішне виконання команди з створення нового контейнера та встановлення пакету в цей контейнер



Рис 3.6. Контейнер створений за допомогою сервісу в програмі Docker Desktop

Цей результат підтверджує, що команда була успішно перекладена та виконана, а контейнер було створено без будь-яких помилок. Для кращого зрозуміння процесу, надається детальна інформація про перекладену команду та

її результат. Хоча команда не була перекладена ідеально, проте вона все одно валідна і виконує поставлену мету. Необхідно звернути увагу на неточність, і продовжувати роботу над покращенням моделі та алгоритмів для досягнення більшої точності у перекладі.

У майбутньому, при розробці повноцінного сервісу, планується впровадити додаткові перевірки для врахування стану контейнерів на хост-машині до виконання команд. Це дозволить забезпечити більшу надійність та безпеку виконання запитів, а також поліпшити інформативність відповідей API, роблячи їх більш зрозумілими та корисними для користувачів.

Також має бути приділена увага підтримці Dockerfile та docker-compose, оскільки ці інструменти є ключовими для ефективної роботи з Docker контейнерами. Розробка механізмів для генерації та використання Dockerfile і docker-compose файлів через природномовні інтерфейси значно розширить можливості сервісу, дозволяючи користувачам створювати та конфігурувати складні середовища з мінімальними зусиллями. Це не тільки спростить процес розгортання додатків і сервісів, але й зробить технологію контейнеризації більш доступною для розробників, які можуть не мати глибоких знань у роботі з Docker. Інтеграція з розширеними функціями Docker, такими як створення мультисервісних додатків за допомогою docker-compose, відкриє нові горизонти для автоматизації та оптимізації розробки та впровадження програмного забезпечення.

При переході від прототипу до повноцінного сервісу, важливим аспектом розробки буде оптимізація способу представлення інформації користувачам. Для того щоб забезпечити більш гнучке та розгалужене взаємодію з системою може виявитись необхідним розширення API, зокрема ендпоінту execute. Наприклад, можна включити детальне пояснення результатів виконання команд, статусів помилок, а також рекомендації щодо можливих дій у випадку виникнення проблем. Такий підхід не лише поліпшить досвід користувачів, але й сприятиме легшій інтеграції сервісу з фронтенд застосунками. Також важливо розглянути

потребу розробки комплексного API, яке включатиме ендпоінти для передвалідації команд, отримання статусів виконання та історії команд.

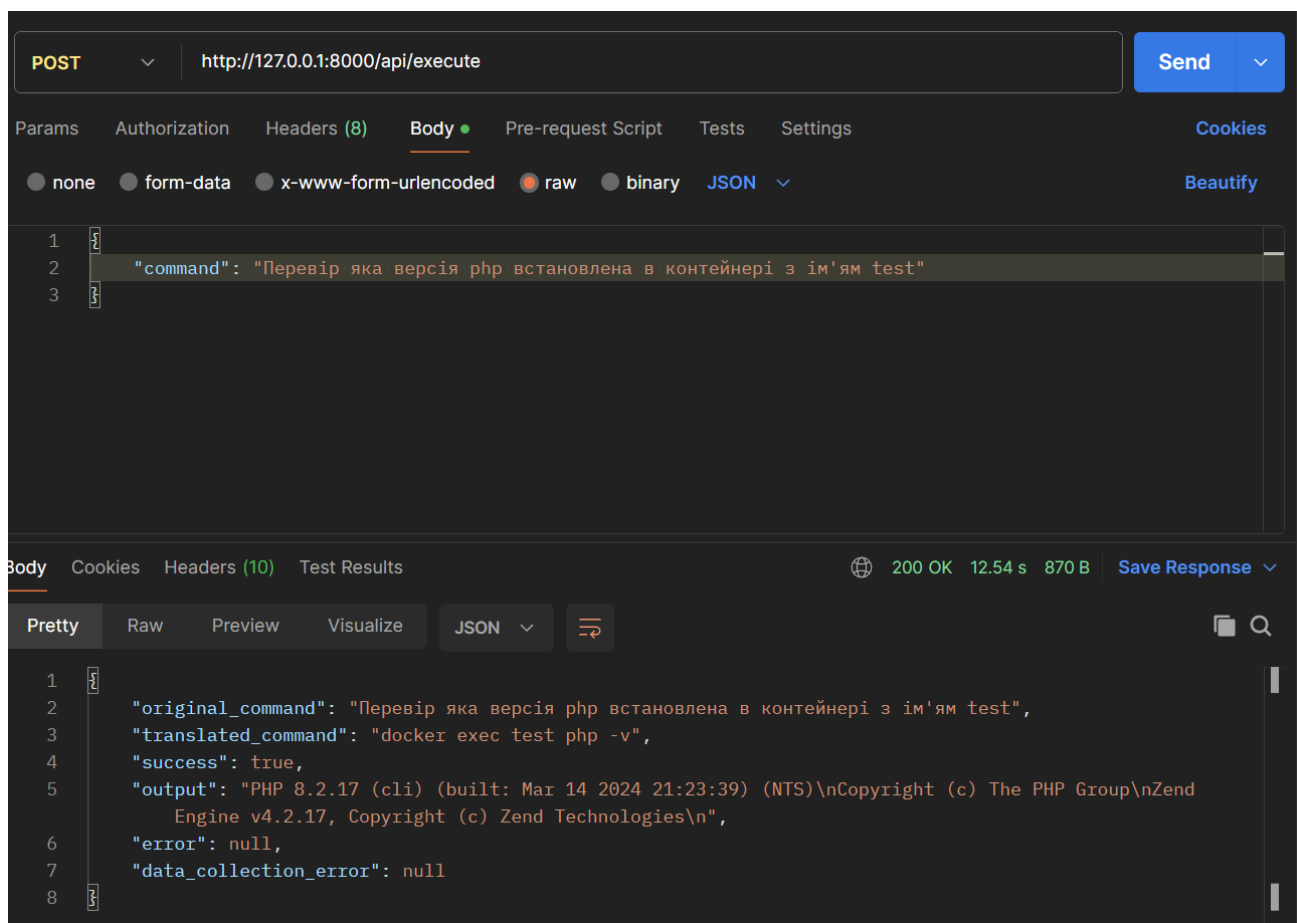


Рис 3.7. Результат успішного виконання команди з перевірки версії пакету php

```
_id: ObjectId('66007fd3cbae3901d907e2d4')
input: "Створи новий контейнер на базі alpine з ім'ям test та встанови в нього..."
output: "docker run -d --name test alpine sh -c 'apk add --no-cache php; while ...'"
updated_at: 2024-03-24T19:32:35.418+00:00
created_at: 2024-03-24T19:32:35.418+00:00
```

Рис 3.8. Новий запис в базі даних mongodb що доданий в процесі роботи ендпоінту execute

Виконання перекладених bash або Docker CLI команд, особливо коли виконання відбувається безпосередньо через `Symfony\Component\Process\Process`, вважається потенційно небезпечним підходом.

Перш за все, це пов'язано з ризиком ін'єкції шкідливих команд, якщо валідація не виявляє спроб експлуатації у вхідних даних. Такі вразливості

можуть дозволити атакуючому виконувати довільний код на хост-системі, що може призвести до несанкціонованого доступу, зміни даних або навіть виведення системи з ладу.

Також не можна виключати того що модель машинного навчання просто зробить помилку яка може призвести до потенційно небажаних наслідків.

```
if ($isValid) {
    try {
        $process = new Process(['bash', '-c', $command]);
        $process->setWorkingDirectory(storage_path( path: self::WORKING_DIRECTORY));
        $process->run();
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
```

Рис 3.9. Поточна логіка виконання команди на хост системі

Крім того, неконтрольоване виконання команд може спричинити помилкове конфігурування системи або її компонентів, що, у свою чергу, загрожує стабільності та безпеці оброблюваних даних. Це особливо актуально в контексті, де система використовується для управління критично важливими аспектами інфраструктури.

У зв'язку з вищевказаними ризиками, розвиток повноцінного сервісу вимагає розробки більш безпечного механізму виконання команд. Один із потенційних підходів полягає у використанні абстракції команд через визначений набір внутрішніх інструкцій, які потім інтерпретуються та виконуються спеціалізованим модулем сервісу. Цей підхід був описаний у другому розділі, і вважається пріоритетним в подальших дослідженнях. Такий підхід дозволить не тільки знизити ризики безпеки, але й забезпечити більш гнучке управління виконанням команд, адаптованих під специфіку сервісу.

3.4. Вимоги та залежності

Прототип було побудовано на основі фреймворку Laravel версії 10.10, що потребує встановлення PHP [14] версії 8.1 або вище та застосунку Composer [15] для керування залежностями PHP.

Необхідно встановити Docker для виконання команд на хост-системі.

Також необхідно встановити та налаштувати MongoDB для зберігання датасету з командами, та MySQL для зберігання даних про моделі.

Необхідно зауважити про залежності необхідні для адаптації та використання основної моделі перекладу MarianMT.

- Python версії 3.6 або новіше.
- PyTorch або TensorFlow: Бібліотека Transformers Hugging Face може працювати з будь-яким з згаданих бекендів, але PyTorch часто вважається таким що має перевагу через його динамічний граф обчислень.
- Бібліотека Hugging Face Transformers
- Для ефективного навчання вкрай корисно мати GPU, сумісний з CUDA, з відповідними драйверами і версією CUDA. При переході від прототипу до повноцінного сервісу ми вважаємо за важливе реалізувати простіше розгортання, тому додаток має бути упакований в контейнер і налаштований на автоматичне встановлення необхідних залежностей.

ВИСНОВКИ

У рамках даної магістерської роботи був розроблений метод та побудований прототип системи управління Docker контейнерами за допомогою команд, введених природною мовою.

Основним досягненням роботи є розробка прототипу системи для перекладу команд з української природної мови в bash/docker CLI команди, з подальшою їх валідацією та корекцією за допомогою API ChatGPT, що дозволяє безперервно покращувати якість перекладу під час експлуатації сервісу.

Результати роботи можуть сприяти спрощенню управління Docker контейнерами, особливо для користувачів без досвіду роботи з командним рядком.

Майбутні дослідження можуть охоплювати розвиток більш комплексних механізмів перекладу для обробки складніших інструкцій та розширення датасету для підтримки більшої кількості задач. Важливими аспектами залишаються оптимізація продуктивності та безпеки системи, у тому числі через вдосконалення процесів валідації та контролю доступу.

Розвиток програмного комплексу включає створення графічного користувацького інтерфейсу для ще ефективнішого спрощення взаємодії користувачів з Docker.

Особливу увагу слід приділити розробці альтернативних рішень для мінімізації залежності від зовнішніх API, та оптимізації витрат на їх використання. Пошук та навчання власних моделей для валідації та корекції команд може значно покращити самодостатність системи та знизити оперативні витрати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кадькало В.В., Чижмотря О.В. Перспективи та виклики інтеграції систем штучного інтелекту з рішеннями для управління контейнеризованими додатками, Житомир, Україна, Листопад 29–30, 2023. VI Всеукраїнська науково-технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення» Рр. 211-212 Vol. 3373. <https://conf.ztu.edu.ua/wp-content/uploads/2024/01/povnyj-tekst-2.pdf>
2. Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The Long-Document Transformer. arXiv:2008.02217.
3. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv:2005.14165.
4. Документація Huggingface MarianMT [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/docs/transformers/model_doc/marian
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. arXiv:1706.03762.
6. Kane, S. P., & Matthias, K. Docker: Up & Running, 2016. – 254 с. – (O'Reilly Media)
7. Newman, S. Building Microservices: Designing Fine-Grained Systems, 2015. – (O'Reilly Media).
8. Helsinki-NLP/opus-mt-uk-en [Електронний ресурс] – Режим доступу до ресурсу: <https://huggingface.co/Helsinki-NLP/opus-mt-uk-en>
9. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805v2.

10. Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. arXiv:1808.03314.
11. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:1412.3555.
12. Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, L., Seide, F., Hermann, U., Aji, A. F., Bogoychev, N., Martins, A. F. T., & Birch, A. (2018). Marian: Fast Neural Machine Translation in C++. arXiv:1804.00344.
13. Laravel Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/docs/11.x/readme>
14. PHP: Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/manual/en/>
15. Composer: Dependency Manager for PHP [Електронний ресурс] – Режим доступу до ресурсу: <https://getcomposer.org/doc/>

ДОДАТКИ

ДОДАТОК А

Скрипт адаптації моделі MarianMT

```
import pandas as pd
from transformers import MarianMTModel, MarianTokenizer, Trainer, TrainingArguments
from datasets import load_dataset, Dataset
from pymongo import MongoClient

# Параметри з'єднання з MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['docker_dataset']
collection = db['management']

# Завантаження даних з колекції (без змін)
data = pd.DataFrame(list(collection.find()))
data.drop('_id', axis=1, inplace=True)
dataset = Dataset.from_pandas(data)

# Підготовка датасету
def preprocess_data(examples):
    model_inputs = tokenizer(examples['input'], max_length=512, truncation=True,
padding="max_length")
    labels = tokenizer(examples['output'], max_length=512, truncation=True,
padding="max_length")
    model_inputs['labels'] = labels['input_ids']
    return model_inputs

# Завантаження моделі та токенизатора MarianMT
model_name = 'Helsinki-NLP/opus-mt-uk-en'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

tokenized_dataset = dataset.map(preprocess_data, batched=True)

# Розділення датасету на навчальний і тестовий
train_test_split = tokenized_dataset.train_test_split(test_size=0.1)
train_dataset = train_test_split['train']
test_dataset = train_test_split['test']

# Налаштування тренування
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
```

```
        logging_steps=10,  
    )  
  
    trainer = Trainer(  
        model=model,  
        args=training_args,  
        train_dataset=train_dataset,  
        eval_dataset=test_dataset,  
    )  
  
    # Тренування моделі  
    trainer.train()  
  
    # Збереження моделі  
    model.save_pretrained('./docker_command_model')  
    tokenizer.save_pretrained('./docker_command_model')
```

ДОДАТОК Б

Скрипт адаптації моделі GPT-2

```

import pandas as pd
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments
from datasets import load_dataset, Dataset
from pymongo import MongoClient

# Параметри з'єднання з MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['docker_dataset']
collection = db['management']

# Завантаження даних з колекції
data = pd.DataFrame(list(collection.find()))
data.drop('_id', axis=1, inplace=True)
dataset = Dataset.from_pandas(data)

def preprocess_data(examples):
    # Враховуємо, що examples['input'] і examples['output'] є списками
    examples['combined'] = [inp + " [SEP] " + out for inp, out in
zip(examples['input'], examples['output'])]
    model_inputs = tokenizer(examples['combined'], max_length=512, truncation=True,
padding="max_length", return_tensors="pt")

    model_inputs['labels'] = model_inputs['input_ids'].clone()
    return model_inputs

# Завантаження моделі та токенизатора GPT
model_name = 'gpt2'
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token

model = GPT2LMHeadModel.from_pretrained(model_name)

# Підготовка і тренування
tokenized_dataset = dataset.map(preprocess_data, batched=True)
train_test_split = tokenized_dataset.train_test_split(test_size=0.1)
train_dataset = train_test_split['train']
test_dataset = train_test_split['test']

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    warmup_steps=500,
    weight_decay=0.01,

```

```
        logging_dir='./logs',
        logging_steps=10,
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset,
    )

    trainer.train()

    # Збереження моделі
    model.save_pretrained('./docker_command_model1')
    tokenizer.save_pretrained('./docker_command_model1')
```

ДОДАТОК В

Константи з контекстуальними повідомленнями для валідації та виправлення команд за допомогою ChatGPT API

```
private const VALIDATION_CONTEXT_MESSAGE = <<<'EOD'
```

Ти є інтерфейсом для оцінки якості перекладу команд з природньої мови на команди bash та Docker CLI.

Твоя задача - аналізувати, наскільки точно оригінальна команда природною мовою перекладена на команду bash/docker-cli.

Перекладена команда має бути в неінтерактивному режимі та враховувати інші потенційні проблеми що можуть перешкоджати її запуску, так наприклад якщо команда має виконати дії в контейнері після його запуску, ми маємо бути впевнені що команда запущена.

Input має містити тільки валідну bash/docker cli команду без будь якої додаткової та зайвої інформації

Надайте оцінку від 1 до 10, де 10 означає ідеальний переклад, а 1 - повну відсутність відповідності.

Ти не повинен повертати нічого крім оцінки у форматі що відповідає регулярному виразу "/Оцінка: (\d+)/"

```
EOD;
```

```
private const CORRECTION_CONTEXT_MESSAGE = <<<'EOD'
```

Ти є інтерфейсом для виправлення невірно перекладених команд з природньої мови на команди bash та Docker CLI.

Твоя задача - аналізувати команди, надані українською мовою, і перекладати їх згідно з контекстом bash/docker-cli, забезпечуючи їх коректність та можливість виконання на хост-машині.

Важливо точно визначати назви пакетів та їх версії, особливо при роботі з дистрибутивами Linux, де пакети можуть мати різні назви чи бути замінені іншими версіями.

Наприклад, для встановлення Python використовуйте 'python3' замість 'python', оскільки більшість сучасних дистрибутивів Linux використовують Python 3 як стандарт.

Подібно, замість 'php' або 'ruby', вказуйте конкретну версію пакету, якщо це необхідно. Після аналізу наданої команди та її контексту, повертай лише виправлену команду, придатну для виконання, без додавання будь-яких обгортки чи форматів.

Кожна команда має бути сформульована для виконання в неінтерактивному режимі та гарантувати, що контейнер залишається в запущеному стані після завершення всіх дій. Перекладена команда має бути представлена чітко та лаконічно, включаючи всі необхідні дії для виконання задачі, вказаної в оригінальному запиті, з особливою увагою до точності назв пакетів та їх версій.

```
EOD;
```

ДОДАТОК Г

Константа з контекстуальними повідомленнями що використовується для генерації додаткових елементів датасету за допомогою ChatGPT API

```
private const GENERATE_CONTEXT_MESSAGE = <<<'EOD'
```

Створи 10 команд українською мовою для інтерфейсу, що призначений для генерації команд, які можна буде перекласти на bash та Docker CLI. Кожна команда повинна бути унікальною та мати певну складність, відмінну від простих і базових команд. Команди мають бути сумісні з можливостями bash та Docker API. Зверни увагу на наступні деталі:

1. Використовуй лише українську мову для формулювання команд.
2. Зосередься на контексті Docker, враховуючи різні аспекти роботи з контейнерами та образами.
3. Уникай додавання будь-яких пояснень чи додаткової інформації – потрібні тільки команди.
4. Вивід має бути у форматі JSON, де кожна команда є окремим елементом масиву.
5. Уникай операції з застосування Dockerfile та docker-compose
6. Конкретизуй команди. Приклад погано конкретизованої команди – Видалити певний локальний образ Docker. Приклад добре конкретизованої команди – Видалити локальний образ Docker з ім'ям тест.

Приклад оформлення виводу:

```
[
  "Команда 1",
  "Команда 2",
  ...
]
```

Згенеруй команди, дотримуючись вищевказаних вимог та умов.

```
EOD;
```

ДОДАТОК І

Застосування моделі для отримання перекладу

```
public function translateCommand(string $text): string
{
    // Отримання активної моделі перекладу
    $activeModel = TranslationModel::where('is_active', true)->firstOrFail();
    $modelPath = $activeModel->path; // Шлях до активної моделі
    $modelPath = storage_path("app/" . $modelPath);

    // Визначення шляху до Python скрипта
    $scriptPath = base_path('scripts/translate.py');

    // Створення процесу і передача шляху до моделі та тексту для перекладу
    $process = new Process(['python3', $scriptPath, $modelPath, $text]);
    $process->run();

    if (!$process->isSuccessful()) {
        throw new ProcessFailedException($process);
    }

    return trim($process->getOutput());
}
```

ДОДАТОК Д

Валідація команди за допомогою виклику ChatGPT API

```

public function validateCommand(string $originalCommand, string
$translatedCommand)
{
    $prompt = "\n\n### Оригінальна команда природною мовою:\n" .
    $originalCommand . "\n### Перекладена команда:\n" . $translatedCommand;

    $response = OpenAI::chat()->create([
        'model' => 'gpt-4',
        'messages' => [
            [
                'role' => 'system', 'content' =>
self::VALIDATION_CONTEXT_MESSAGE
            ],
            [
                'role' => 'user', 'content' => $prompt,
            ],
        ],
    ]);

    $validationResult = $response->choices[0]->message->content ?? 'error: Не
вдалося отримати оцінку';

    if (preg_match('/Оцінка: (\d+)/', $validationResult, $matches)) {
        $score = (int)$matches[1];
        if ($score >= 8) {
            return true;
        }
    }

    return false;
}

```