Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Використання малих мовних моделей для перекладу природної мови в команди Bash
(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-23-1
Платаний В.А.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник проф. Кулішова Н.Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ О.В. Золотухін
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Науки про дані (Data Science)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Платаному Вадиму Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Використання малих мовних моделей для перекладу природної мови в команди Bash
затверджена наказом університету від 22 листопада 20 24 р. № 1238Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23 січня 20 25 р.
3. Вихідні дані до роботи python 3.11, git, бібліотеки huggingface, transformers, datasets, evaluate, pandas, numpy

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі дослідження
2) Загальна схема процесу підготовки і навчання моделей
3) Навчання SML для задачі перекладу
4) Експериментальний розділ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	виконано
2	Дослідження предметної галузі	10.12.2024	виконано
3	Дослідження проблематики LM перекладу	15.12.2024	виконано
4	Постановка завдання	16.12.2024	виконано
5	Вибір моделей машинного навчання	20.12.2024	виконано
6	Тюнінг моделей	05.01.2025	виконано
7	Формування подальших впроваджень системи	07.01.2025	виконано
8	Написання пояснювальної записки	15.01.2025	виконано
9	Захист перед ЕК	23.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Кулішова Н.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 43 с., 10 рис., 1 дод., 8 джерел.

МОВНІ МОДЕЛІ, ОБРОБКА ПРИРОДНОЇ МОВИ, ПЕРЕКЛАД, ПРИРОДНА МОВА, BASH, SEQUENCE-TO-SEQUENCE.

Об’єкт дослідження – процес автоматичного перекладу команд з англійської мови у Bash-команди за допомогою сучасних мовних моделей.

Предмет дослідження – особливості роботи малих мовних моделей, спеціалізованих на обробці англійської мови та генерації програмного коду, у задачі автоматичного перекладу природної мови у Bash-команди.

Мета роботи – порівняння результатів роботи різних малих мовних моделей для автоматичного перекладу команд з англійської мови у Bash-команди, з акцентом на точність та ефективність виконання завдання.

Методи дослідження – моделювання, програмна реалізація, аналіз і аналітика, формалізація.

У роботі проведено порівняння малих мовних моделей для автоматичного перекладу команд з англійської мови у Bash. Для цього буде підготовлено датасет, проведено навчання моделей із оптимізацією гіперпараметрів та виконано їх тестування. Результати будуть оцінюватися за точністю та ефективністю виконання завдання, з урахуванням специфіки Bash-команд.

ABSTRACT

Master’s thesis contains: 43 pp., 10 fig., 1 ann., 8 references.

BASH, LANGUAGE MODELS, NATURAL LANGUAGE, NATURAL LANGUAGE PROCESSING, SEQUENCE-TO-SEQUENCE, TRANSLATION.

Object of research – the process of automatic translation of commands from English to Bash commands using modern language models.

The subject of the study is the peculiarities of the work of small language models specialized in English language processing and program code generation in the problem of automatic translation of natural language in Bash commands.

The purpose of the work is to compare the results of the work of different small language models for automatic translation of commands from English to Bash commands, with an emphasis on the accuracy and efficiency of the task.

Research methods – modeling, software implementation, analysis and analytics, formalization.

The paper compares small language models for automatic translation of commands from English in Bash. For this purpose, a dataset will be prepared, models will be trained with hyperparameter optimization and tested. The results will be evaluated by the accuracy and efficiency of the task, taking into account the specifics of the Bash commands.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NL – Natural Language – природна мова;

NLP – Natural Language Processing – обробка природної мови;

Bash – Bourne Again Shell – оболонка Bash;

UNIX – UNiplexed Information Computing System – операційна система UNIX;

Shell – Command-Line Shell – командна оболонка.

ВСТУП

Автоматичний переклад між природною мовою та мовою командного інтерпретатора є важливою задачею в галузі обробки природної мови (NLP) та генерації програмного коду. Це завдання знаходить застосування в багатьох сферах, зокрема у розробці інтерфейсів природної мови для систем автоматизації, навчанні користувачів роботі з командним інтерпретатором та спрощенні взаємодії з програмним забезпеченням. Особливий інтерес викликає проблема перекладу команд, сформульованих англійською мовою, у відповідні команди Bash – однієї з найпопулярніших мов командного інтерпретатора.

Розв’язання цієї задачі передбачає використання сучасних мовних моделей, зокрема архітектур типу sequence-to-sequence (seq2seq), які досягли значних успіхів у завданнях машинного перекладу та генерації тексту.

Водночас застосування цих моделей до перекладу між природною мовою та Bash-командами стикається з низкою викликів. До них належать специфічна структура команд Bash, обмежений контекст, а також необхідність забезпечення високої точності для уникнення помилкових інструкцій.

Метою даної роботи є порівняння результатів роботи різних малих мовних моделей для автоматичного перекладу команд з англійської мови у Bash. Зокрема, у фокусі дослідження перебувають моделі, спеціалізовані на роботі з англійською мовою, та моделі, оптимізовані для генерації програмного коду.

У рамках дослідження передбачено проведення експериментів, спрямованих на аналіз точності та ефективності цих моделей, з урахуванням специфіки завдання.

Одним із ключових етапів дослідження є підготовка якісного датасету, який забезпечить консистентність і коректність навчання моделей.

Крім того, важливе значення має пошук оптимальних гіперпараметрів, таких як швидкість навчання та параметри оптимізатора, що дозволить досягти максимальної продуктивності моделей.

Таким чином, це дослідження спрямоване на порівняння результатів оцінки моделей, котрі спрямовані на вирішення цієї специфічної проблеми перекладу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

Універсальність консольних команд полягає у створенні інтерфейсу, який дозволяє користувачам взаємодіяти з програмами та операційними системами через текстові інструкції незалежно від їхньої специфіки. Це досягається завдяки стандартним підходам до структури команд і аргументів, а також узгодженій семантиці, що дозволяє адаптувати консольні інтерфейси для різних програмних середовищ. Такий підхід робить консольні команди потужним і гнучким інструментом для автоматизації, налаштувань і управління програмами.

Ключовими аспектами універсальності є стандартизація синтаксису (наприклад, поділ команди на ім'я, аргументи та опції), підтримка інтуїтивно зрозумілих команд (наприклад, ls, cd, mkdir в UNIX-подібних системах) і можливість інтеграції команд різних програм. Крім того, такі інструменти, як оболонки (shell), дозволяють поєднувати команди у скрипти, передавати вихідні дані однієї команди як вхідні іншій (через конвеєр), або формувати складні робочі процеси. Цей рівень узгодженості забезпечує масштабованість і адаптивність, дозволяючи застосовувати універсальні навички роботи з консоллю для керування новими програмами чи середовищами без необхідності вивчення нових підходів.

Універсальність консольних команд робить їх потужним інструментом для автоматизації, налаштувань і управління програмами. Завдяки стандартизованій структурі, інтуїтивно зрозумілим командам і можливості інтеграції, консольні команди стають незамінним інструментом для будь-якого користувача комп'ютера. Їхня здатність працювати на різних програмних платформах, автоматизовувати повторювані завдання і управляти віддаленими системами робить їх невід'ємною частиною сучасного технологічного ландшафту.

1.1 Основна структура Bash команд

Bash команди мають чітку структуру, яка дозволяє виконувати різноманітні операції в командному рядку. Основна структура команди виглядає так: `<команда> [прапори] [аргументи]`.

Елементи структури:

– команда: це основна частина, що вказує, яку дію потрібно виконати (наприклад, `ls`, `cd`, `echo`);

– прапори: це додаткові параметри, які змінюють поведінку команди. Вони зазвичай починаються з дефіса (-) і можуть бути короткими (одна літера) або довгими (слово). Наприклад, у команді `ls -l`, `-l` є прапором, який вказує на детальний формат виводу;

– аргументи: це дані, які передаються команді для виконання. Наприклад, у команді `mkdir new_folder`, `new_folder` є аргументом, що вказує ім'я нової директорії.

Також існують складені команди, за технікою pip. У Bash pipe (|) дозволяє передавати вихідні дані однієї команди як вхідні для іншої, створюючи ланцюжок обробки даних. Це зручно для поєднання простих команд у складні операції без використання тимчасових файлів. Наприклад, команда `cat file.txt | grep "pattern" | wc -l` зчитує вміст файлу, шукає рядки, що містять «pattern», і рахує їх кількість. Pipe значно спрощує автоматизацію обробки даних у скриптах, роблячи їх більш ефективними та читаємими.

1.2 Задача перекладу з натурального мовлення

Задача перекладу натуральної мови у машинному навчанні передбачає автоматичне перетворення тексту з однієї мови на іншу, зберігаючи зміст і стиль оригіналу. Це складна задача, оскільки вимагає

врахування граматичних правил, ідіом, контексту й культурних особливостей мови.

Традиційно переклад здійснювався за допомогою статистичних методів, які використовували лінгвістичні правила та великі двомовні корпуси для знаходження ймовірнісних відповідностей між словами та фразами. Однак такі підходи мали обмежену здатність обробляти довгі або складні речення.

Сучасний підхід базується на використанні нейронних мереж, зокрема архітектури трансформерів. Моделі типу Seq2Seq (послідовність у послідовність) із механізмом уваги, такі як Neural Machine Translation (NMT), дозволяють розглядати увесь контекст речення для більш точного перекладу. Трансформери, наприклад, модель BERT або GPT, здатні зберігати значення складних структур завдяки механізму самоуваги, який фокусується на ключових частинах тексту. Це дозволило створювати системи перекладу, які демонструють майже людську якість, особливо в популярних мовних парах.

Використання нейронних мереж і трансформерів дозволяє ефективніше обробляти складні речення, враховуючи контекст і семантику мови. Це значно покращує якість перекладу порівняно зі статистичними методами. Проте, навіть із сучасними підходами, виникають виклики. Однією з ключових проблем є переклад малопоширених мов через нестачу навчальних даних. Щоб вирішити цю проблему, дослідники розвивають технології перенесення знань і багатомовного навчання. Ці підходи дозволяють моделям використовувати знання, отримані від одного мови, для поліпшення перекладу іншої мови.

Додатковим викликом є переклад натуральної мови у мови програмування або команди для комп'ютера. Ця задача вимагає не лише розуміння мови, але й глибокого розуміння синтаксису мови програмування, логіки задачі та контексту, що описується текстом. Такі моделі, як OpenAI Codex (на основі GPT) або DeepMind AlphaCode,

навчаються на величезних обсягах вихідного коду та документації. Вони здатні генерувати програмний код на основі текстового опису, пояснювати функції існуючого коду, а також автоматизувати рутинні задачі програмування.

Іншою проблемою є обробка неоднозначних описів. Природна мова може бути багатозначною, а описи задач програмування можуть мати кілька вірних рішень. Щоб подолати це, моделі повинні бути здатними розуміти нюанси мови, враховувати контекст та генерувати код, який не лише технічно коректний, але й семантично правильний.

Нарешті, важливим викликом є вирішення проблем безпеки. Автоматично генерований код може містити помилки, вразливості або навіть шкідливий код. Щоб запобігти цьому, дослідники розробляють методи перевірки безпеки генерованого коду, використовують техніки статичного аналізу коду та розробляють механізми виправлення виявлених проблем.

Переклад натуральної мови у мови програмування є важливою і складною задачею, яка вимагає поєднання глибокого розуміння мови, програмування та контексту. Використання нейронних мереж і трансформерів дозволяє досягти значного прогресу в цій області, але виклики все ще залишаються. Щоб вирішити ці проблеми, дослідники продовжують працювати над вдосконаленням моделей, розвитком нових методів та технік, а також використанням додаткових даних та ресурсів для поліпшення точності, безпеки та загальної корисності систем перекладу коду.

2 ЗАГАЛЬНА СХЕМА ПРОЦЕСУ ПІДГОТОВКИ І НАВЧАННЯ МОДЕЛЕЙ

Дане дослідження спирається на результати, котрі були досягнені на NeurIPS 2020 NLC2CMD Challenge**.** Змагання було направлене на розробку моделей, котрі переводять натуральне мовлення у команди для командної строки. Однак значною перевагою у результатах було використання саме малих мовних моделей для їх легкого навчання, котре не потребувало б значних обчислювальних потужностей.

Основні етапи дослідження (рисунок 2.1) можна описати так:

– неопрацьовані пари мовлення та перекладу. На початковому етапі алгоритм працює із сирими даними, які складаються з пар текстів: фраз природною мовою (вихідний текст) та відповідних команд (цільовий текст). Ці дані можуть бути неструктурованими, містити помилки, дублікати або неточності;

– фільтрація невалідних даних, токенізація. На цьому етапі дані очищуються від некоректних або неповних записів (наприклад, команд, які не відповідають заданим стандартам). Виконується токенізація – поділ тексту на окремі елементи (слова, символи або токени), що необхідно для подальшої обробки та навчання моделей. Цей процес забезпечує узгодженість і покращує якість моделі;

– заміна літералів на шаблонні значення. Для стандартизації даних здійснюється заміна конкретних значень (наприклад, імен файлів, шляхів, чисел) на шаблонні позначення. Наприклад, шлях `/home/user/file.txt` може бути замінений на `{FILE_PATH}`. Це допомагає моделі краще узагальнювати та уникати перенавчання на конкретних даних;

– тренування моделей. Підготовлені дані використовуються для навчання мовних моделей. У роботі можуть застосовуватись претреновані seq2seq моделі, спеціалізовані на англійській мові чи генерації коду.

Навчання виконується за однаковими умовами, включаючи кількість епох, з метою забезпечення справедливого порівняння;

– оцінка ефективності моделі. Після завершення тренування моделі оцінюються на тестовому наборі даних. Використовуються метрики машинного перекладу, такі як BLEU або ROUGE, для визначення точності та якості перекладу. Це дозволяє визначити сильні та слабкі сторони кожної моделі;

– порівняння метрик. На основі оцінок ефективності проводиться порівняння моделей. Аналізуються результати метрик для визначення найбільш підходящої моделі для задачі. Порівняння допомагає зробити висновки про те, яка архітектура або конфігурація моделі краще справляється із перекладом.

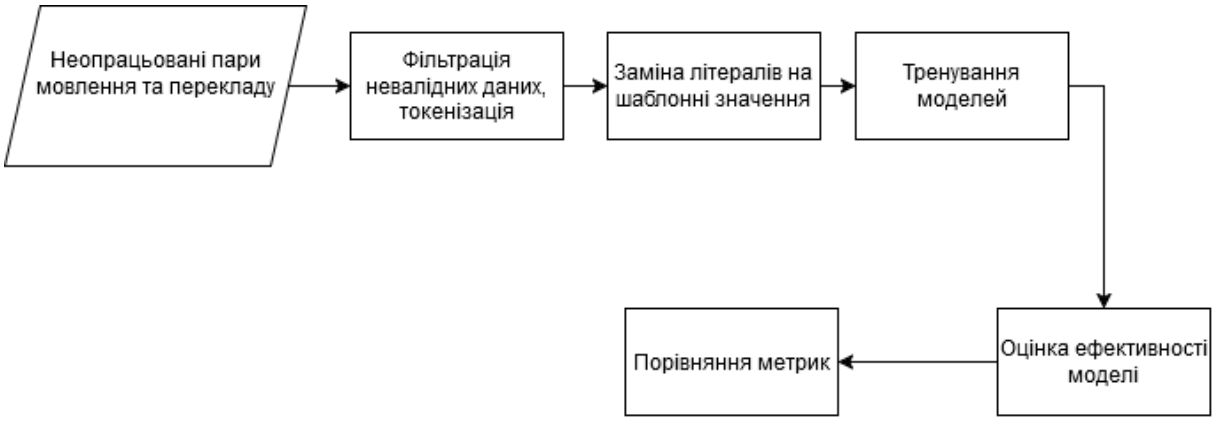

Рисунок 2.1 – Основні етапи дослідження

2.1 Підготовка даних до навчання

2.1.1 Пайплайн перетворення даних при задачі перекладу у команди bash

Необроблені дані піддаються розбору для перетворення англійських речень і команд Bash у дерева синтаксису. Дані, які не можуть бути

проаналізовані, відфільтровуються, забезпечуючи надходження лише дійсних синтаксичних дерев. Синтаксичні дерева зводяться, а параметри замінюються загальними представленнями. Це спрощує структуру для подальшої обробки та аналізу. Перетворені пари речень токенізуються. Створення словників для англійських речень та команд Bash, що полегшує структуровану обробку даних [4].

Даний пайплайн від Magnum-NLC2CMD був доповнений це декількома етапами перетворення тренувальних даних. По перше – у деяких команд можуть бути присутні вкладені команди, котрі можуть бути різноманітними і відрізнятись одне від одного. До цього вони замінялися шаблонами. Тепер, вони теж повністю присутні і відкриті після перетворення. Наприклад, `awk '{ if(/2012/)print;else exit }'`:

– `awk` – це мова обробки тексту, яка працює з рядками тексту або файлами;

– `if(/2012/)` – перевіряє, чи містить рядок підрядок `2012`;

– `/2012/` – це регулярний вираз, який знаходить будь-який текст, що містить `2012`;

– `print` – якщо в рядку знайдено `2012`, він виводиться;

– `else exit` – якщо в рядку немає `2012`, команда `awk` завершує роботу (вихід із програми).

Раніше увесь контекст внутрішньої програми замінювався на шаблон і перетворена команда виглядає як “awk COMMAND”. Таким чином втрачається контекст внутрішньої команди. Але важливо робити це до загальної заміни шаблонами, бо внутрішні команди можуть містити літерали, котрі повинні бути замінені.

Наступний етап – це нумерація замінених шаблонів. Замінені під час попередньої обробки літерали повинні бути замінені після постпроцесінгу. Але команди можуть містити по декілька літералів однакового шаблону, шо після генерації відповіді буде неможливо відрізнити одне від одного. Тому,

наприклад, команда `echo REGEX | grep -E -o REGEX` тепер виглядатиме як `echo REGEX0 | grep -E -o REGEX1`.

2.1.2 Навчальні дані

Навчальні дані представляють собою збірку пар перекладів у команди bash Magnum-NL2CMD. Ця вибірка є зібраною через різні інтернет ресурси автоматично або вручну. У 2023 році від цих розробників вийшов новий пайплайн збору даних для навчання з готовими даних, заснованих на генерації і валідації команд на основі існуючих з Linux manual і подальшого пояснення за допомогою ChatGPT [3]. У нашому випадку ми будемо використовувати старі навчальні дані, через менший розмір і більшу складність і розрідженість структур різних команд.

У ньому налічується 6621 пара для тренувань. Як можна побачити на рисунку 2.2, найбільш розповсюдженою першою командою є `find`. Додатково у свою чергу, ця команда найбільш розповсюджена у складених командах з механізмом `pipe` (рисунок 2.3).

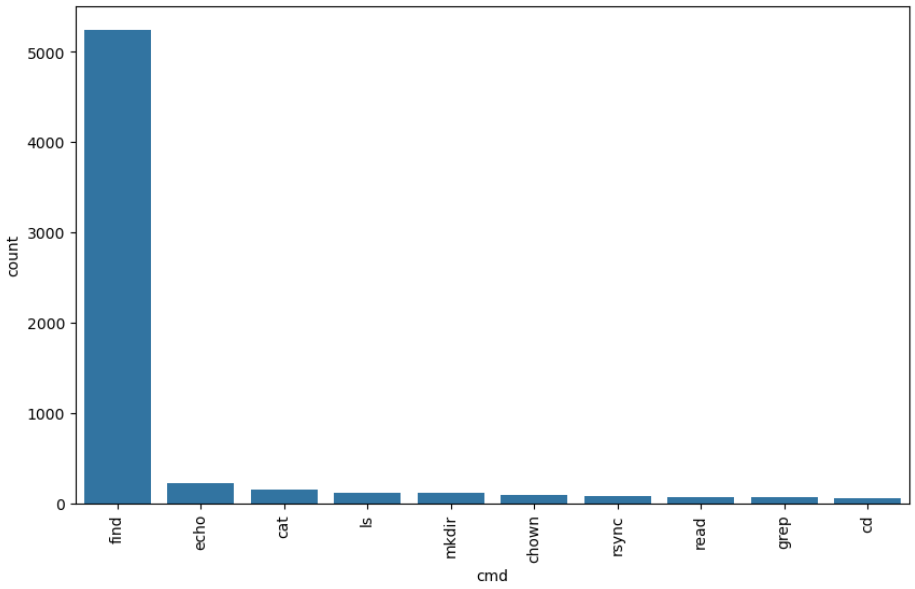

Рисунок 2.2 – 10 найрозповсюдженіших команд у тренувальному сеті та кількість команд котрі використовують їх як основну програму

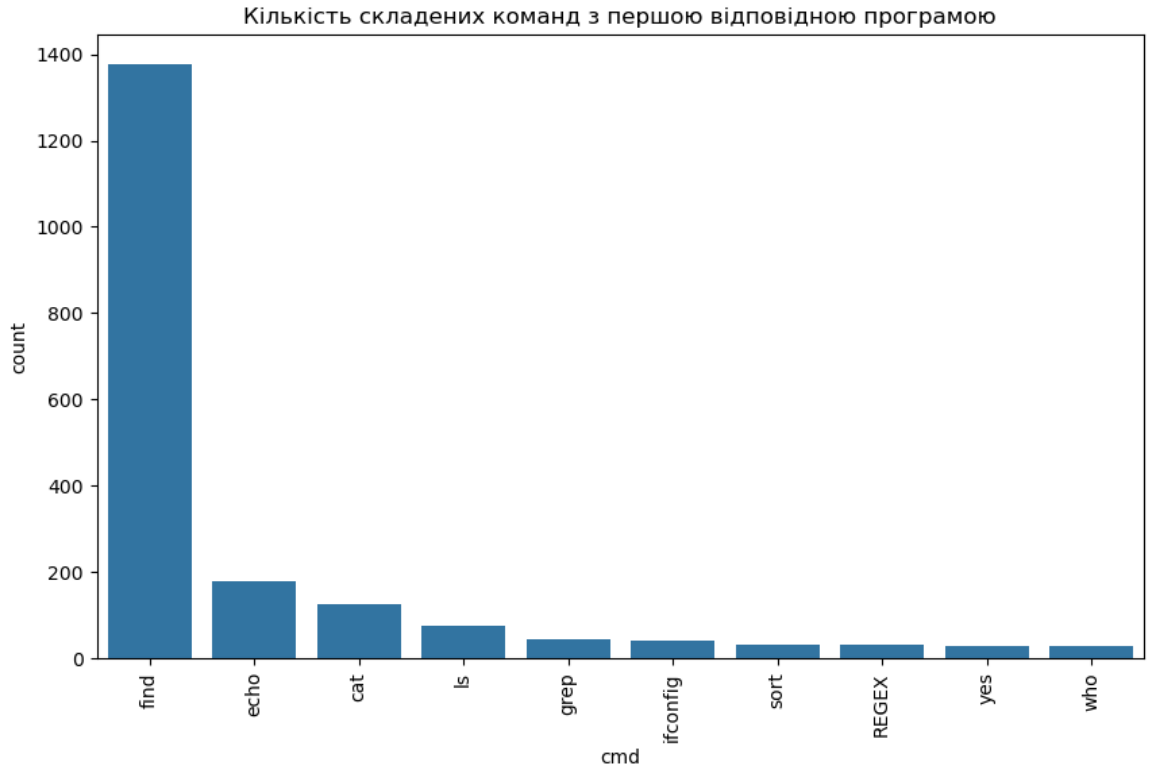

Рисунок 2.3 – Кількість складених команд з першою відповідною програмою

Але найважливіше для команди `find` те, що різний виклик може дати нам одну і ту саму команду шаблону. Наприклад: `find all REGEX0 file and directori under current directori` та `find all file in current directori tree whose name are FILE0` має той самий результат шаблону – `find PATH0 -name REGEX0`.

Цей приклад також дав нам іншу проблему ручної попередньої обробки природної мови. Ці замінені літерали не завжди мають однакові літерали для заміни в команді. Це критично важливо для кінцевого користувача системи перекладу, але не проблема для нашої мети оцінити SML.

Як можна бачити з рисунку 2.4, короткі речення домінують у датасеті, що типово для багатьох задач перекладу, особливо у прикладах із програмуванням чи командними рядками.

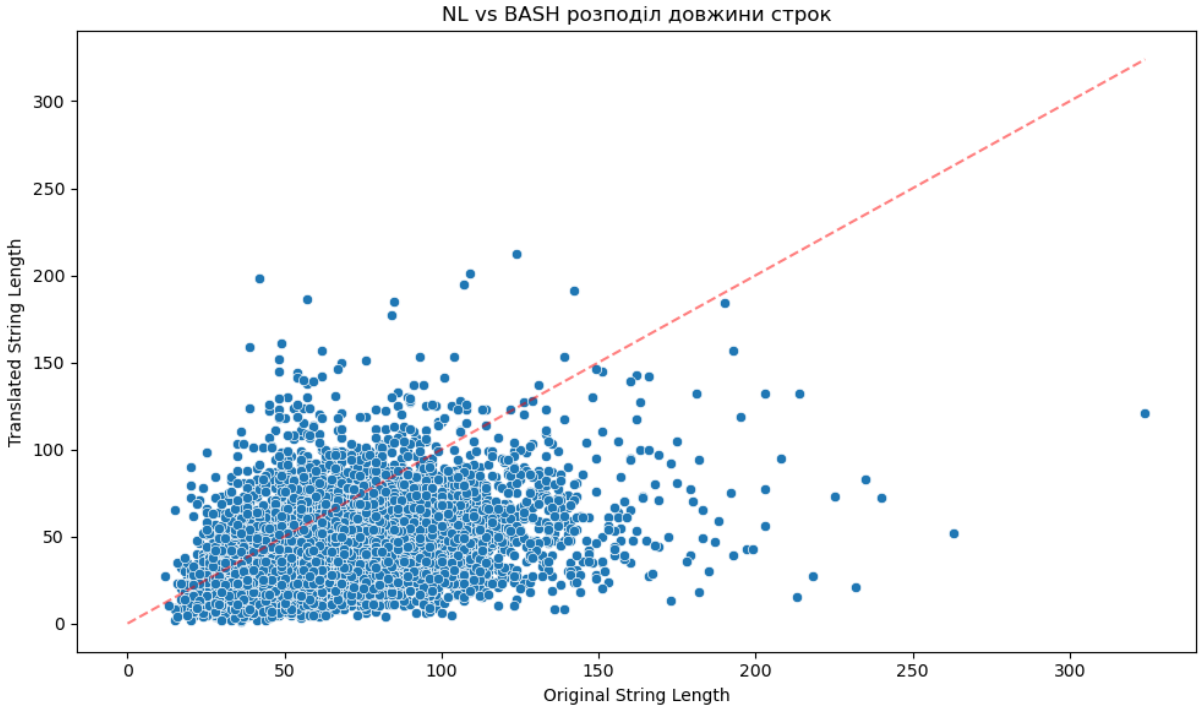

Рисунок 2.4 – Розподіл довжин строк у тренувальному наборі

Цей розподіл підтверджує, що модель перекладу потребуватиме ефективного механізму узгодження контексту між короткими і довшими текстами.

3 НАВЧАННЯ SML ДЛЯ ЗАДАЧІ ПЕРЕКЛАДУ

3.1 Що таке SLM і їх різниця від LLM

SLM – це мовні моделі меншого розміру, які зазвичай мають від кількох мільйонів до кількох мільярдів параметрів. Часто оптимізовані для специфічних завдань, таких як аналіз настроїв або розпізнавання іменованих сутностей, вони краще справляються з вузькоспеціалізованими задачами. Вимагають менше обчислювальних ресурсів, що робить їх доступнішими для використання на пристроях з обмеженими ресурсами. LLM містять мільярди параметрів, наприклад, GPT-3 має 175 мільярдів параметрів. Малі мовні моделі призначені для вирішення складних мовних завдань і можуть генерувати зв'язний текст з глибоким розумінням контексту. Вимагають значних витрат на навчання та експлуатацію, що може бути недоступним для малих організацій. Вибір між SLM і LLM залежить від конкретних вимог задачі [7]. Якщо потрібна висока швидкість і економічність, SLM можуть бути кращим вибором. Якщо ж важливо досягти високої якості генерації тексту з глибоким розумінням контексту, LLM будуть більш підходящими.

У контексті перекладу натурального мовлення у команди Bash великі мовні моделі справляються з великою точністю, однак потребують чіткого вказання способів виводу команди. І звичайно, через свої розміри не можуть бути локально інтегровані у різні застосунки, котрі потребують такої генерації.

Однак, сучасний підхід генерації команд базується на віддаленому зверненні до великих мовних моделей, таких як ChatGPT від OpenAI. І результати показують досить високі показники. ChatGPT досягає показника точності більше 80% на тестовому наборі в zero-shot умовах [5]. Хоча існують побоювання щодо можливості витоку даних у перекладі на основі LLM через величезну кількість інтернет-тексту в попередніх тренувальних

даних, можна бути впевненим в продуктивності ChatGPT, враховуючи його стабільну здатність досягати балів 80% або вище за всіма наборами даних навчання, тестування та оцінки. Однак окремим шляхом при використанні великих мовних моделей можна назвати використання моделей для генерації коду. Наприклад, рішення Codex-CLI від Microsoft використовує для цього модель Codex від OpenAI на базі GPT3. Використання малих мовних моделей для цієї задачі все ще залишається на етапі дослідження [4].

3.2 Ефективний Finetuning, PEFT

Тренування будь-якої моделі, і конкретно її розмір і тривалість тренування завжди має залежність від обчислювальних потужностей, на яких саме проходить цей процес. Навіть тренування невеликих мовних моделей може потребувати багато часу. На щастя, існують алгоритми, котрі можуть значно вплинути як на швидкість, так і на якість тренування мовної моделі.

Ефективне тонке налаштування параметрів (PEFT) – це ефективна техніка НЛП, яка вміло адаптує попередньо підготовлені мовні моделі для різних програм із надзвичайною ефективністю. Методи PEFT налаштовують лише невелику підмножину додаткових параметрів моделі, зберігаючи більшість попередньо підготовлених параметрів LLM замороженими, тим самим значно зменшуючи витрати на обчислення та зберігання. Такий підхід пом’якшує проблему катастрофічного забування, явища, коли нейронні мережі втрачають раніше набуті знання, та відчути значне зниження продуктивності раніше вивчених завдань під час навчання на нових наборах даних [8].

Методи PEFT продемонстрували кращу ефективність порівняно з повним тонким налаштуванням, особливо в сценарії з низьким обсягом даних і краще узагальнення для контекстів поза доменом.

3.2.1 LoRa

Low-Rank Adaptation (LoRA) – це техніка розроблена для тонкого налаштування великих мовних моделей, вона модифікує процес тонкого налаштування шляхом заморожування початкових вагових коефіцієнтів моделі та застосування змін до окремого набору вагових коефіцієнтів, доданих до вихідних параметрів. LoRA перетворює параметри моделі в вимір нижчого рангу, зменшуючи кількість параметрів, які можна навчити, прискорюючи процес і знижуючи витрати. Цей метод особливо корисний у сценаріях, коли потрібні точно налаштовані моделі для різних сценаріїв, що дозволяє створювати конкретні ваги для кожного випадку використання без потреби в окремих моделях. Використовуючи методи апроксимації низького рангу, LoRA ефективно знижує вимоги до обчислень і ресурсів, зберігаючи адаптованість попередньо навченої моделі до конкретних завдань або областей.

Переваги використання LoRA:

– ефективність параметрів: LoRA значно зменшує кількість параметрів, які потрібно тренувати, зосереджуючись лише на матрицях низького рангу, що призводить до нижчих вимог до пам'яті та сховища порівняно з повним тонким налаштуванням;

– ефективне зберігання: зберігання навченої моделі є більш ефективним, оскільки вимагає зберігання лише матриць низького рангу замість повної ваги моделі. Зменшене обчислювальне навантаження: тренування з матрицями низького рангу вимагає менше обчислювальних ресурсів, що робить його швидшим і масштабованішим;

– менший обсяг пам'яті: оскільки оновлюється менше параметрів, обсяг пам'яті під час навчання зменшується, що дозволяє використовувати більші розміри партій або складніші моделі в межах тих самих апаратних обмежень;

– гнучкість: LoRA можна легко інтегрувати з існуючими попередньо навченими моделями без значних змін в архітектурі моделі;

– сумісність: його можна використовувати разом з іншими методами тонкого налаштування, такими як шари адаптера або швидке налаштування, для подальшого підвищення продуктивності;

– порівняльні результати: незважаючи на зменшення кількості параметрів, що тренуються, було показано, що LoRA досягає продуктивності, порівнянної з повним тонким налаштуванням у багатьох завданнях;

– адаптація до конкретного завдання: вона ефективно адаптує попередньо навчену модель до конкретних завдань, використовуючи знання, вже закладені в оригінальну модель;

– уникнення перенавчання: зосереджуючись на оновленнях низького рангу, LoRA може допомогти пом'якшити перенавчання, особливо коли ви маєте справу з меншими наборами даних для конкретного завдання [8].

Незважаючи на те, що LoRA демонструє значну потужність, вона також створює проблеми:

– область тонкого налаштування: LoRA може зіткнутися з труднощами при застосуванні до завдань, що вимагають значних змін у внутрішніх представленнях попередньо навченої моделі;

– оптимізація гіперпараметрів: налаштування параметра рангу 'r' вимагає ретельного коригування для оптимальної продуктивності;

– поточні дослідження: незважаючи на свою багатообіцянку, LoRA все ще знаходиться на активних стадіях досліджень, і її довгострокові наслідки ще належить повністю вивчити.

Для даного дослідження були вибрані наступні оптимальні параметри:

– r (rank) – 32. Це ранг матриць LoRA, який визначає розмірність простору низького рангу для апроксимації оновлень ваг. Значення є відносно високим і підходить для складних задач трансформації тексту;

– lora alpha – 32. Це коефіцієнт масштабування для LoRA оновлень;

– lora dropout – 0.1. Ймовірність відключення нейронів під час тренування. Значення 0.1 означає, що 10% з'єднань випадково відключаються при кожному проході.

3.3 Пошук гіперпараметрів та оптимізатор

Для спрощення витрат на пошук оптимального оптимізатора та гіперпараметрів, усі моделі попередньо будуть використовувати один і той самий оптимізатор – AdamW (Adam with Weight Decay**),** що є алгоритмом оптимізації, який зазвичай використовується в навчанні моделей глибокого навчання. Він є варіантом оптимізатора Адама з поліпшеною реалізацією нормалізації розпаду ваги. Для кожної моделі пошук оптимальних гіперпараметрів складається з 5 пробних запусків і шукає наступні параметри:

– learning_rate – початкова швидкість навчання. range: 0.00001 – 0.01;

– adam_beta1 – являє собою експоненціальну швидкість розпаду для оцінок першого моменту. Він контролює, як оптимізатор обчислює ковзне середнє градієнтів під час тренування. Range: 0.8 – 0.999;

– adam_beta2 – являє собою експоненціальну швидкість розпаду для оцінок другого моменту, які є ковзними середніми квадратичних градієнтів. Range: 0.9 – 0.999;

– adam_epsilon – це невелика константа, що додається до знаменника під час оновлення параметрів для запобігання ділення на нуль або надзвичайно малих значень, що може дестабілізувати тренувальний процес. Range: 0.00000001 – 0.0000001;

– max_grad_norm – максимальна норма градієнта (для відсікання градієнта). range: 1.0 – 5.0.

Метод баєсової оптимізації використовується спеціально для пошуку. Цей метод використовує імовірнісну модель для прогнозування продуктивності різних гіперпараметрів і відповідно вибирає найкращі. Це ефективний метод, який може краще обробляти великі простори параметрів і є менш ресурсомістким, ніж пошук по сітці [6]. Однак він складніший у налаштуванні і може бути менш надійним у виявленні оптимального набору гіперпараметрів у порівнянні з пошуком по сітці.

3.4 Метрики для оцінки точності моделі

BLEU (Bilingual Evaluation Understudy) – це метрика, яка використовується для оцінки якості тексту, згенерованого мовною моделлю, порівнюючи його з одним або кількома еталонними текстами. Її зазвичай застосовують у задачах машинного перекладу або текстової генерації, де важливо оцінити схожість між вихідним текстом і референсними прикладами.

BLEU вимірює точність на рівні н-грам (послідовностей слів), визначаючи частку н-грам згенерованого тексту, які зустрічаються в референсному. Для уникнення переваги коротших текстів використовують штраф за довжину. Хоча ця метрика проста й ефективна, вона має обмеження, оскільки не враховує семантичного значення тексту і чутлива до змін порядку слів. Тому BLEU найкраще працює для завдань із чіткими і добре структурованими відповідями. Незважаючи на це, чудово підходить для оцінки задач перекладу.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) – це набір метрик для оцінки якості автоматично згенерованого тексту, особливо в задачах автоматичного реферування текстів. ROUGE порівнює згенерований текст із еталонним, обчислюючи збіг між ними на рівні слів, фраз чи послідовностей. Найпоширенішими є варіанти ROUGE-1 (збіг окремих слів), ROUGE-2 (збіг біграм) і ROUGE-L (збіг найдовшої спільної

підпослідовності). Метрика враховує повноту (recall), точність (precision) і їхнє гармонійне середнє (F1-score), щоб оцінити, наскільки добре модель охоплює інформацію з референсного тексту [8]. ROUGE використовується для оцінки завдань, де важливо оцінити зміст, але має обмеження, оскільки залежить від лексичного збігу і не враховує глибокого семантичного значення. Через це часто у задачі перекладу може показувати завищені показники, котрі засновані тільки на окремих послідовностях. Тому максимально незрозумілі людині результати можуть видавати добрі результати.

Зважаючи на недоліки та переваги всіх метрик, ціллю є найти гармонічно схожі та близькі одне до одного результати, шо буде слугувати ознакою правильності згенерованих відповідей при оцінці моделі, як і семантично, так і орфографічно.

3.5 Sequence-to-sequence мовні моделі

Для задачі перекладу підходять моделі котрі можуть перетворювати вхідну послідовність і видавати повністю іншу, але базовану на вхідній послідовність. Це так звані Sequence-to-sequence (seq2seq) моделі. Це особливий тип нейронних мереж, які перетворюють одну послідовність елементів в іншу (рисунок 3.1). Вони складаються з двох основних частин: енкодера та декодера. Енкодер обробляє вхідну послідовність і створює її представлення, а декодер використовує це представлення для генерації вихідної послідовності.

Звичайно, більша частина цих моделей заснована на архітектурі трансформера, після успішного виходу GPT-1, котрий представляв собою тільки однонаправлений декодер, але дослідники також описали архітектуру трансформера з енкодером та декодером який забезпечує паралельну обробку даних та краще масштабування. Це дозволило створювати більш ефективні моделі, здатні обробляти довші послідовності

та генерувати більш якісні результати. Механізм уваги (attention mechanism) став важливим удосконаленням seq2seq моделей. Він дозволяє декодеру фокусуватися на різних частинах вхідної послідовності під час генерації кожного елемента вихідної послідовності. Це особливо корисно для довгих послідовностей, де важливо зберегти зв'язок між віддаленими елементами.

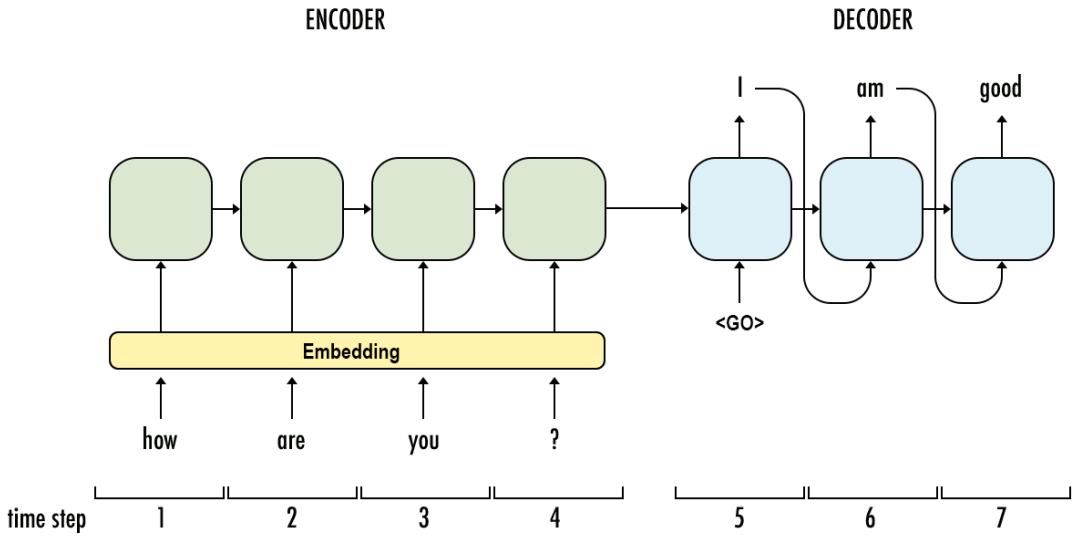

Рисунок 3.1 – Модель seq2seq

Seq2seq моделі використовуються не лише для перекладу, але й для багатьох інших завдань: узагальнення текстів, генерації відповідей на запитання, створення описів зображень, транскрипції мовлення в текст. У кожному випадку модель вчиться знаходити відповідності між вхідними та вихідними послідовностями, зберігаючи семантичний зміст та структуру даних.

3.6 Попередньо треновані моделі для дослідження

Попередньо треновані моделі (Pre-Trained Models) є ефективнішими для задач перекладу з кількох ключових причин. Попередньо треновані моделі, такі як T5, BART чи CodeT5, навчаються на великих корпусах

тексту, які включають різноманітні джерела даних (книги, статті, програмний код). Це дозволяє їм отримати глибоке розуміння синтаксису, семантики та загальних закономірностей природної мови. Завдяки цьому моделі краще розуміють контекст і можуть точніше перекладати складні речення. Сучасні попередньо треновані моделі використовують механізм «self-attention» (наприклад, у трансформерах), який дозволяє враховувати залежності між словами в реченні незалежно від їхньої відстані. Це особливо важливо для перекладу, де точність залежить від правильного врахування контексту.

Завдяки попередньому навчанню на великих датасетах, такі моделі можуть демонструвати хороші результати навіть при донавчанні на відносно невеликих наборах спеціалізованих даних. Це є критично важливим у задачах, де доступ до великих обсягів анотованих даних обмежений, як у випадку перекладу між природною мовою та командною мовою Bash. Попередньо треновані моделі можна ефективно адаптувати до задач перекладу шляхом донавчання на доменних або спеціалізованих наборах даних. Наприклад, моделі, оптимізовані для роботи з кодом, як CodeT5 або Codex, краще справляються із задачами генерації коду або перекладу тексту в команди. Попередньо треновані моделі з архітектурою sequence-to-sequence можуть працювати як із завданнями перекладу, так і з генерацією тексту. Це дозволяє їм гнучко адаптуватися до різних типів текстових входів і виходів, наприклад, англійських речень і команд Bash.

Використання попередньо тренованих моделей скорочує час і обчислювальні ресурси, необхідні для навчання моделей «з нуля». Ці моделі вже мають узагальнені мовні знання, тому для вирішення конкретної задачі потрібно лише виконати донавчання. Завдяки широкій попередній підготовці такі моделі краще справляються з даними, які можуть містити шум (наприклад, синтаксичні помилки чи незвичні конструкції). Це особливо важливо для задач перекладу команд, де помилки в Bash-командах можуть призвести до некоректного виконання.

Одними з ключових моментів при воборі моделі є кількість параметрів, від яких також і залежить розмір самої моделі. На даному етапі ми будемо розглядати найбільш легкі моделі, з кількістю параметрів меньше одного мільярда. Саме такі моделі дуже добро підходять під визначення «мобільності» моделі.

3.6.1 T5

T5 (Text-to-Text Transfer Transformer) – це мовна модель, розроблена компанією Google, яка була представлена в 2019 році. Вона є універсальною трансформерною моделлю, призначеною для виконання різноманітних завдань обробки природної мови, таких як машинний переклад, резюмування, запитання-відповіді та інші задачі, які можна сформулювати як перетворення тексту в текст.

T5 також відзначається своєю гнучкістю: вона може бути адаптована до специфічних завдань шляхом додаткового навчання на невеликих наборах даних. Це робить її потужним інструментом для дослідників і розробників у сфері обробки природної мови. T5 попередньо тренується на масивному корпусі англійського тексту під назвою C4 (Colossal Clean Crawled Corpus) [4].

T5 доступний у різних розмірах, починаючи від маленьких і закінчуючи надзвичайно великими. У тренуваннях буде використовуватися T5-base з 223M параметрів.

3.6.2 BART

BART (Bidirectional and Auto-Regressive Transformers) – це потужна мовна модель, створена дослідниками Facebook AI, яка є вдосконаленням трансформерної архітектури та спеціально розроблена для задач генерації

тексту, таких як машинний переклад, узагальнення текстів, заповнення пропусків, перефразування та інші подібні завдання.

Унікальність BART полягає в поєднанні властивостей двох підходів: двоспрямованого (bidirectional) контекстуального представлення, характерного для моделей типу BERT, і авто-регресивного підходу до генерації тексту, який використовується в моделях GPT. Це забезпечує моделі глибоке розуміння текстового контексту та можливість точного генерування тексту. Особливістю BART є підхід до попереднього тренування, у якому модель навчається відновлювати початковий текст із його пошкодженої версії. Пошкодження тексту може відбуватися різними способами, такими як маскування окремих слів чи фраз, їхнє вилучення, перестановка речень або введення шуму. Завдяки цьому BART здобуває здатність розуміти структуру тексту навіть у складних чи пошкоджених випадках, що робить її надзвичайно ефективною для завдань обробки природної мови [1].

Завдяки універсальності архітектури, BART добре адаптується до різних сценаріїв: наприклад, для задачі машинного перекладу вона може навчитися перетворювати текст однією мовою на іншу, тоді як для текстового узагальнення – створювати стислий і змістовний підсумок. Модель демонструє відмінні результати на багатьох задачах обробки тексту, часто перевершуючи інші архітектури.

BART має різні конфігурації (наприклад, BART-large і BART-base), що дозволяє збалансувати продуктивність і обчислювальні ресурси. Оскільки вона є двоспрямованою моделлю, то здатна краще враховувати контекст з обох боків маскованих або відновлюваних елементів тексту, що є важливим для задач із високими вимогами до розуміння контексту. Для тренування буде використано версію bart-large з 406 мільйонів параметрів.

3.6.3 CodeT5

І на останок, перевіримо модель з орієнтацією на генерацію коду, порівняно зі стандартною версією. CodeT5 – це модель трансформера, розроблена Salesforce Research для задач, пов'язаних із розумінням і генерацією програмного коду. Вона є частиною сімейства CodeT5, що базується на архітектурі Text-to-Text Transfer Transformer (T5), спеціально адаптованій для роботи з програмним кодом. CodeT5 поєднує потужність трансформерів із передтренуванням на великих наборах даних програмного коду, щоб ефективно виконувати різні завдання, такі як автодоповнення коду, рефакторинг, генерація коментарів, класифікація коду та навіть генерація коду з описів природною мовою.

Під час попереднього тренування застосовувався комбінований підхід до маскування токенів (denoising objectives), який охоплює як маскування токенів у тексті, так і маскування синтаксичних елементів у коді. Це дає змогу моделі навчатися відновлювати не лише окремі слова, але й складні синтаксичні конструкції. CodeT5-base попередньо тренована на великому корпусі даних із репозиторіїв GitHub, що включають коди, написані на популярних мовах програмування, таких як Python, Java, JavaScript, C++, Ruby тощо. Цей багатомовний підхід забезпечує гнучкість моделі, дозволяючи їй працювати з кількома мовами програмування одночасно. Модель також адаптована для завдань, які вимагають розуміння природної мови, наприклад, генерація коментарів до коду, переклад із однієї мови програмування на іншу чи пошук коду за текстовим описом [2].

Однією з важливих інновацій CodeT5 є її здатність поєднувати знання природної мови та програмного коду. Це особливо корисно для задач, пов'язаних із взаємодією між розробниками й інструментами програмування, таких як пошук коду за запитами природною мовою чи генерація кодових фрагментів із описів. Для цього CodeT5 використовує

методологію «text-to-code» і «code-to-text», що дозволяє легко переходити між текстом і кодом.

CodeT5-base, як середня модель у сімействі CodeT5, має 220 мільйонів параметрів і збалансована між точністю та обчислювальною ефективністю. Вона демонструє високу продуктивність на стандартних задачах обробки коду, перевершуючи інші моделі, такі як GPT-2 або базові версії T5, у задачах генерації, класифікації та перетворення коду. Модель також інтегрує в себе можливості до тонкого налаштування (fine-tuning) для конкретних завдань, що дозволяє її ефективно адаптувати до вузькоспеціалізованих потреб розробників або компаній.

4 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

У рамках даного дослідження ставиться задача порівняння результатів роботи різних малих мовних моделей для автоматичного перекладу команд, сформульованих природною англійською мовою, у відповідні команди командного інтерпретатора bash. Це завдання можна розділити на декілька ключових етапів.

Першим критичним етапом є підготовка якісного датасету. Наявні дані потрібно обробити та привести до уніфікованого шаблонного формату, що забезпечить консистентність при навчанні моделей. Це включає нормалізацію текстових команд, стандартизацію формату bash-команд та видалення потенційних невідповідностей чи помилок у даних.

Для вирішення поставленої задачі планується використовувати попередньо треновані sequence-to-sequence моделі двох типів: моделі, спеціалізовані на роботі з англійською мовою, та додатково, для порівняння, моделі для генерації програмного коду, так як семантика коду дуже схожа з семантикою команд Bash. Важливо відзначити, що вхідні дані обмежуються виключно англійською мовою, що дозволяє сфокусуватися на специфіці перекладу між природною мовою та командною мовою bash, не ускладнюючи задачу багатомовністю. Перед початком тренування кожної моделі необхідно провести ретельний пошук оптимальних гіперпараметрів. Цей процес має критичне значення для досягнення максимальної ефективності моделей та забезпечення якісного перекладу команд. Оптимізація гіперпараметрів включатиме підбір таких важливих характеристик як швидкість навчання та параметри оптимізатора AdamW. Після тренування проводиться з однаковою кількістю епох. Після цього проводиться оцінка на тестовому датасеті для отримання достовірних результатів.

4.1 Результати експерименту

Графік на рисунку 4.1 демонструє зміну значення BLEU (Bilingual Evaluation Understudy) для різних моделей під час тренування. Codet5-base стабільно має найвищі BLEU значення протягом усього тренування. Її продуктивність значно краща порівняно з іншими моделями. t5-base демонструє поступове зростання BLEU до певного моменту (приблизно після 1242 кроків), після чого стабілізується на певному рівні. bart-large спочатку має менше значення BLEU, але поступово наближається до t5-base, хоча й залишається трохи нижче.

З кожним збільшенням кількості тренувальних кроків всі моделі демонструють покращення, однак це зростання різниться між моделями. Після 1656 кроків більшість моделей показують стабілізацію, а їхній приріст зменшується.

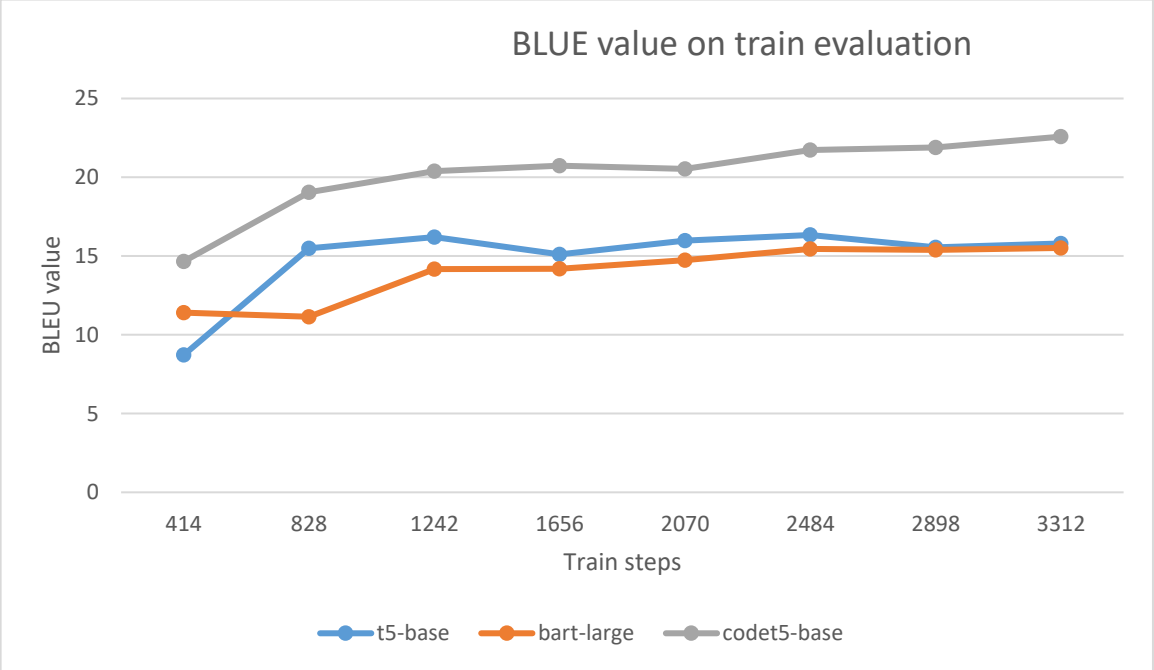

Рисунок 4.1 – Зміна значення метрики BLEU під час тренування

Codet5-base є найефективнішою моделлю для завдання, оскільки вона має найвищі значення BLEU на всіх етапах. t5-base демонструє кращу продуктивність, ніж bart-large, але не досягає рівня codet5-base. Вплив тренувальних кроків на BLEU зменшується після певного моменту, що свідчить про насичення навчання.

Графік на рисунку 4.2 демонструє зміну значень метрики Rouge1 під час тренування. Codet5-base стабільно демонструє найвищі результати серед трьох моделей, досягаючи значення Rouge1 ~55-58. Це свідчить, що codet5-base краще відтворює ключові елементи тексту. Вона, ймовірно, оптимізована для завдань, що потребують точного відповідності між прогнозами та еталонними відповідями. Bart-large починає із значення близько 42 та поступово покращується до 48 на середині тренувального процесу. Проте потім демонструє певну стагнацію. T5-base спершу має найгірші результати (~40), проте поступово наздоганяє bart-large. На пізніх етапах їхні результати майже однакові (~48-49). Відносно повільний початок може вказувати на те, що t5-base потребує більше часу для адаптації до задачі. Моделі показують зменшення приросту з часом, що може вказувати на потребу оптимізації навчальних параметрів або використання інших стратегій (наприклад, fine-tuning на специфічних даних).

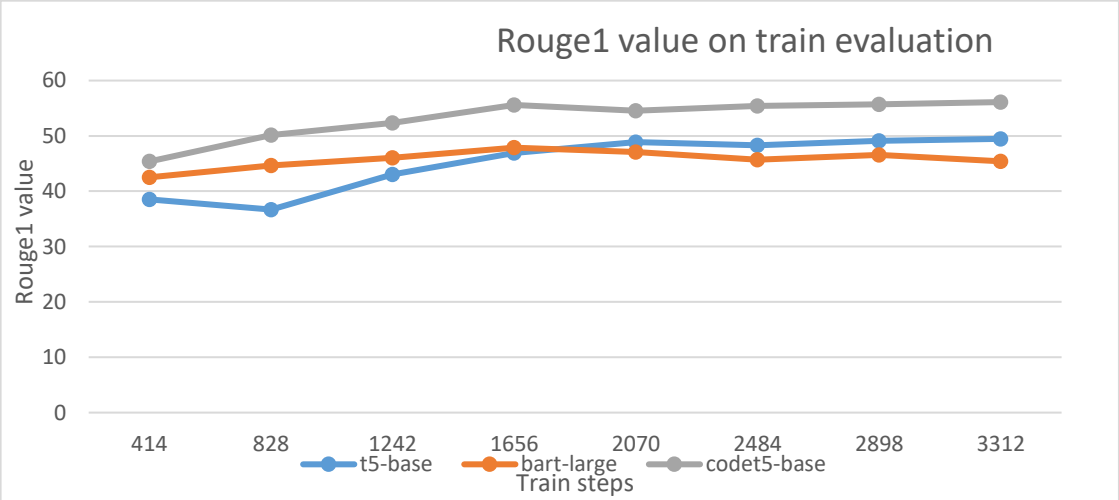

Рисунок 4.2 – Зміна значення метрики Rouge1 під час тренування

Графік на рисунку 4.3 демонструє значення метрики Rouge2. Codet5-base демонструє стабільно найвищі результати протягом усього тренування, досягаючи ~40 Rouge2 на завершальних етапах. Codet5-base спеціалізується на завданнях роботи з кодом і текстом, що вимагають точного збереження контексту та послідовності. Її високі значення Rouge2 свідчать про здатність моделі захоплювати складні шаблони та семантичні залежності.

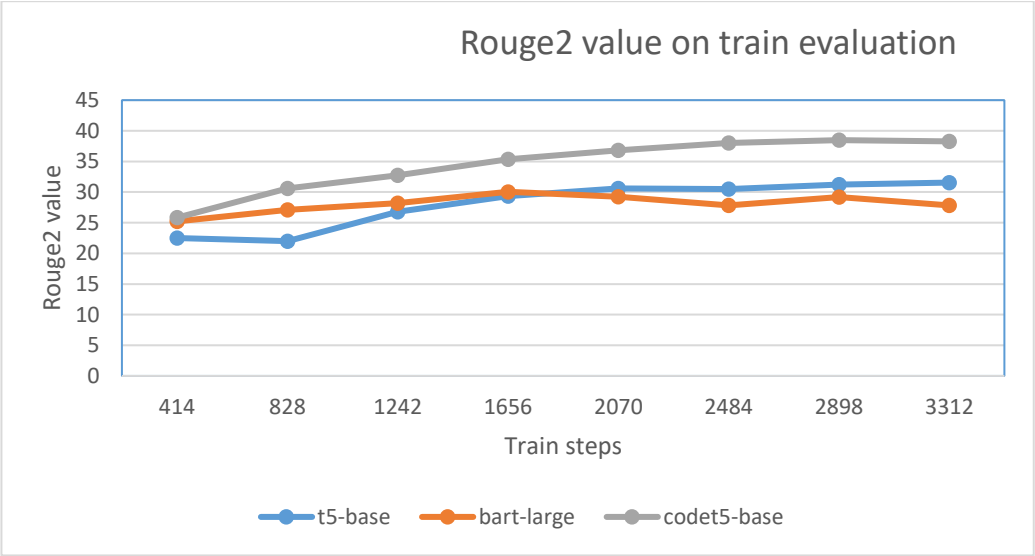

Рисунок 4.3 – Зміна значення метрики Rouge2 під час тренування

Для всіх моделей спостерігається уповільнення приросту значень після цієї точки, що може вказувати на досягнення рівноваги між навчанням та генерацією. Рівні Rouge2 значно нижчі, ніж Rouge1, що характерно для задач, де відтворення точних біграм складніше, ніж окремих слів.

Всі три моделі показують покращення RougeL score з часом. Найбільше зростання відбувається на перших 1242 кроках (рисунок 4.4). Після 2070 кроків криві виходять на плато. Вихід на плато після 2070 кроків може вказувати на досягнення межі можливостей моделей для даної задачі, потребу в збільшенні складності моделі або зміні гіперпараметрів або можливе перенавчання, хоча для підтвердження потрібно бачити метрики на валідаційному наборі.

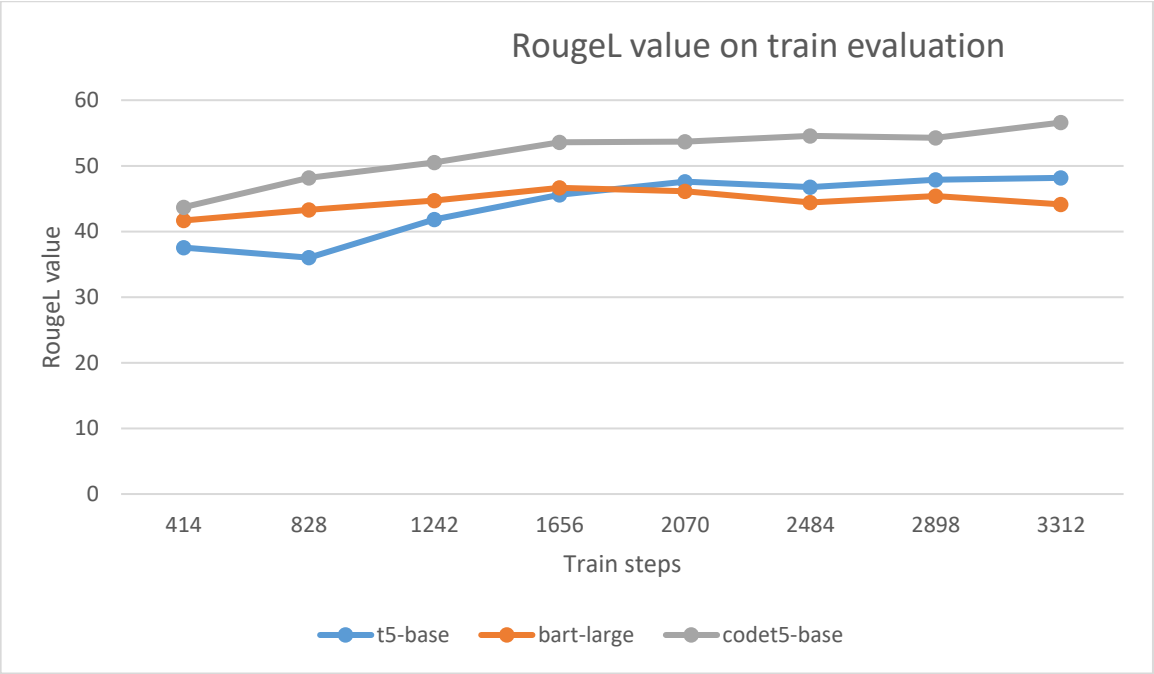

Рисунок 4.4 – Зміна значення метрики RougeL під час тренування

Codet5-base побудована на базі T5, ця модель адаптована для роботи з кодом, що вимагає глибокого розуміння структури даних. Її результати Rouge2 підтверджують, що вона краще передає локальні контексти та складні залежності. Bart-large орієнтована на узагальнення тексту. Хоча вона сильна у відтворенні загальних шаблонів, її результати нижчі через відносну неспеціалізованість. Базова версія T5 є універсальною, але менш оптимізованою для задач, де потрібна точність передачі складних шаблонів, що пояснює її середні результати.

Як показано на рисунку 4.5, на тестовому наборі даних моделі показують не гірші результати. CodeT5 демонструє найкращі результати за всіма метриками. Всі моделі краще справляються із збереженням окремих слів (Rouge1), ніж із точними фразами (Rouge2). Відносно низькі значення BLEU можуть вказувати на те, що завдання потребує певної перефразування або генерації нового тексту, а не точного копіювання.

Рисунок 4.5 – Значення метрик на тестовому наборі після тренування моделей

У висновку можна зазначити, що моделі орієнтовані на генерацію коду, дійсно мають перевагу у задачі перекладу у команди bash через схожість семантичних структур коду і команд, котрі потребують чіткості. Під час тренування усі моделі дуже швидко вийшли на плато. Найбільш вірогідно, шо причиною цьому став обмежений набір даних для тренування. Найбільше це спостерігається з bart моделлю, котра має складнішу архітектуру ніж t5, і тому набагато швидше знаходить усі можливі зв'язки під час тренування.

4.2 Подальші впровадження та можливий вектор дослідження

Подальший розвиток дослідження може відбуватися у декількох ключових напрямках

Удосконалення процесу навчання моделі:

– розширення навчального датасету шляхом включення більшої кількості прикладів природномовних команд та їх відповідників у Bash. Особливу увагу варто приділити збільшенню різноманітності синтаксичних конструкцій та складності команд;

– оптимізація тривалості навчання моделі з врахуванням кривих навчання та валідації для знаходження оптимального балансу між недонавчанням та перенавчанням;

– експериментування з різними параметрами LoRA, включаючи ранг адаптації (r), альфа масштабування, та dropout. Це дозволить знайти оптимальне співвідношення між розміром моделі та якістю перекладу.

Архітектурні вдосконалення:

– дослідження можливості використання інших архітектур малих мовних моделей, наприклад з decoder архітектурою, таких як BERT-tiny або DistilGPT, для порівняння ефективності різних підходів;

– впровадження механізму валідації згенерованих Bash-команд для перевірки їх синтаксичної коректності перед виконанням;

– розробка гібридного підходу, який би комбінував переваги різних архітектур для досягнення кращих результатів;

– додавання підтримки складних pipeline конструкцій та багатокомандних послідовностей;

– реалізація зворотного перекладу з Bash у природну мову для покращення зрозумілості складних команд.

Практичне застосування:

– розробка API для інтеграції з існуючими системами автоматизації та DevOps інструментами;

– створення інтерактивного навчального інструменту для початківців у вивченні Bash;

– дослідження можливостей застосування розробленого підходу до інших командних оболонок та скриптових мов.

Оптимізація ресурсів:

– дослідження методів квантизації та прунінгу для подальшого зменшення розміру моделі без суттєвої втрати якості;

– оптимізація швидкодії для забезпечення можливості роботи в режимі реального часу навіть на обмежених обчислювальних ресурсах;

– розробка методів кешування та попередньої обробки для прискорення роботи системи.

Ці напрямки досліджень можуть суттєво покращити практичну цінність системи та розширити її можливості застосування в реальних умовах. Особливу увагу варто приділити балансу між складністю моделі та її ефективністю, оскільки це критично важливо для практичного використання в умовах обмежених ресурсів.

ВИСНОВКИ

У ході виконання даної роботи було досліджено 4 малі мовні моделі для перекладу природномовних команд у команди Bash. Проведено комплексний аналіз існуючих підходів до обробки природної мови та їх адаптації для специфічних задач, зокрема перекладу в команди командного рядка. Виявлено, що використання малих мовних моделей є перспективним напрямком для вирішення даної задачі, враховуючи баланс між обчислювальними вимогами та якістю результатів.

Розроблено архітектуру системи, що базується на малій мовній моделі з використанням техніки LoRA для точного налаштування. Такий підхід дозволив досягти компромісу між розміром моделі та якістю перекладу, забезпечуючи при цьому можливість роботи на пристроях з обмеженими ресурсами. Модифіковано спеціалізований датасет для навчання моделі, що містить пари природномовних запитів та відповідних їм Bash-команд. Експериментальним шляхом визначено оптимальні параметри навчання та конфігурації моделі, що забезпечують найкращі результати перекладу. Виявлено основні обмеження та виклики у використанні малих мовних моделей для даної задачі, включаючи складності з обробкою складних командних конструкцій та необхідність балансування між розміром моделі та її ефективністю.

Результати дослідження підтверджують практичну цінність запропонованого підходу та демонструють можливість створення ефективних систем перекладу природної мови в команди Bash з використанням малих мовних моделей. Розроблені у подальшому системи можуть стати корисним інструментом як для початківців у вивченні командного рядка, так і для досвідчених користувачів, які прагнуть підвищити свою продуктивність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv.org*. URL: https://arxiv.org/abs/1910.13461 (date of access: 03.01.2025).

2. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. *arXiv.org*. URL: https://arxiv.org/abs/2109.00859 (date of access: 10.01.2025).

3. Enhancing SLM via ChatGPT and Dataset Augmentation. *arXiv.org*. URL: https://arxiv.org/abs/2409.12599 (date of access: 07.01.2025).

4. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv.org*. URL: https://arxiv.org/abs/1910.10683 (date of access: 29.12.2024).

5. NL2CMD: An Updated Workflow for Natural Language to Bash Commands Translation. *arXiv.org*. URL: https://arxiv.org/abs/2302.07845 (date of access: 29.11.2024).

6. Raghav P. Understanding Hyperparameters and its Optimisation techniques. *Medium*. URL: https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568 (date of access: 03.01.2025).

7. Small Language Models (SLMs) Can Still Pack a Punch: A survey. *arXiv.org*. URL: https://arxiv.org/abs/2501.05465 (date of access: 24.12.2024).

8. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *arXiv.org*. URL: https://arxiv.org/abs/2408.13296 (date of access: 02.12.2024).