МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ:

“Програмний інструментарій дистиляції знань длязадач обробки природної мови”

Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп’ютерні науки»**
Освітня програма **«Комп’ютерні науки»**
Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 41

Пагарський О.А.

Керівник:
асистент кафедри
к.т.н., с.д. Андрійчук О.В.

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № 13 від 05.06.2023 р.
зав. кафедри _____ доц. Іларіонов О.Є.

Київ - 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____ 2023 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Пагарському Олексію Артемовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)

«Програмний інструментарій дистиляції знань для задач обробки природної мови»

затверджена протоколом засідання кафедри від «11» листопада 2022 р. №

2. Термін здачі студентом закінченого проекту (роботи) 31 травня 2023 року

3. Вихідні дані до проекту (роботи)

Модель дистильована методом дистиляції знань

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

Аналітичний огляд, Проектні рішення, Реалізація, тестові приклади

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

1. Постановка задачі: мета, об'єкт, предмет, вимоги
2. Предметна область: огляд наявних рішень
3. Проектні рішення
4. Програмна реалізація

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 15 лютого 2023 року

Керівник _____ / Андрійчук О.В. /
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / Пагарський О.А. /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Опрацювання літератури	15.02 – 27.02	
2	Робота над розділом 1	27.02 – 08.03	
3	Робота над розділом 2	08.03 – 08.04	
4	Робота над розділом 3	08.04 – 13.05	
5	Робота над оформленням пояснювальної записки	13.05 – 25.05	
6	Робота над презентацією	25.05 – 29.05	

Студент-дипломник _____ / Пагарський О.А. /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / Андрійчук О.В. /
(підпис) (ПІБ)

Анотація

Дипломна робота викладена на 63 сторінках, містить 3 розділи, 28 ілюстрації, 32 джерела в переліку посилань.

Метою роботи є розробка програмного інструментарію (бібліотеки) для спрощення процесу дистиляції знань для моделей машинного навчання архітектури типу “трансформер”.

Об’єктом дослідження є процес дистиляції знань для моделей природної мови архітектури “трансформер”.

Предметом дослідження є методи та технології дистиляції знань для моделей природної мови.

У Розділі 1 буде досліджено метод дистиляції знань для генерації тексту, включаючи аналіз наявних рішень, опис моделей архітектури типу "трансформер", розгляд архітектури "Енкодер-Декодер", механізму уваги та моделі BERT, розглянуто проблему моделювання мови з маскою та підходи до дистиляції знань, а також оптимізації процесу дистиляції з використанням спеціальної функції втрат та проведено аналіз наявних бізнес процесів, пов’язаних з задачами обробки природної мови та сформовано вимоги до програмного інструментарію.

У Розділі 2 буде представлена архітектура системи дистиляції моделей для обробки тексту на високому рівні та на рівні окремих компонентів: trainer, data та tests.

У Розділі 3 буде реалізовано програмний інструментарій дистиляції моделей та натреновано зменшену версію моделі BERT, TinyBERT, для завдання моделювання мови з маскою.

Ключові слова: машинне навчання, дистиляція знань, нейронні мережі, трансформер, моделювання мови з маскою

Summary

The thesis is laid out on 63 pages, contains 3 chapters, 28 illustrations, 32 sources in the list of references.

The purpose of the work is to develop a software toolkit (library) to simplify the process of knowledge distillation for machine learning models of "transformer" type architecture.

The object of research is the process of knowledge distillation for natural language models of "transformer" architecture.

The subject of research is methods and technologies of knowledge distillation for natural language models.

Section 1 will explore the knowledge distillation method for text generation, including an analysis of existing solutions, a description of transformer-type architecture models, a review of the Encoder-Decoder architecture, the attention mechanism and BERT models, a review of the masked language modeling problem, and knowledge distillation approaches, as well as optimization of the distillation process using a special loss function, and an analysis of existing business processes related to natural language processing tasks, and will form requirements for software tool.

In Section 2, the architecture of the distillation system for text processing models will be presented at a high level and at the level of individual components: trainer, data, and tests.

In Section 3, a model distillation software toolkit will be implemented and a reduced version of the BERT model, TinyBERT, will be trained for the masked language modeling.

Keywords: machine learning, knowledge distillation, neural networks, transformer, masked language modeling

ЗМІСТ

ВСТУП

Однією з визначних проблем у обробці природної мови є задача розуміння природної мови (**natural language understanding, NLU**). В останні роки, завдяки росту потужності обчислювальної техніки, а особливо графічних, та спеціалізованих тензорних прискорювачів обчислень (GPU та TPU), у сфері обробки природної мови стався швидкий ріст у якості, кількості та розмірі текстових моделей, що використовуються для різноманітного спектру задач, таких як класифікація текстів за настроєм, генерація тексту, суммаризація текстів (підведення підсумків), логічне продовження тексту та мовне моделювання з маскою (masked language modeling). Прикладами таких моделей можуть служити сімейство моделей GPT[1] для задач генерації та суммаризації текстів від дослідницької команди компанії OpenAI, модель T5[2] для задач суммаризації, відповідей на питання і класифікації текстів та багато інших моделей.

Однак, цей розвиток створив проблему в використанні цих моделей у практичних сценаріях. Моделі стали надзвичайно дорогими для використання, а їх розмір став перешкодою для вбудованих і мобільних пристроїв. Таким чином, виникає потреба в стратегіях для вирішення цієї проблеми, а одним з перспективних підходів є метод дистиляції знань.

В даній роботі буде проведено створення програмного інструментарію дистиляції знань для задач обробки природної мови на прикладі проблеми моделювання мови з маскою.

У Розділі 1 буде досліджено метод дистиляції знань для генерації тексту, включаючи аналіз наявних рішень, опис моделей архітектури типу "трансформер", розгляд архітектури "Енкодер-Декодер", механізму уваги та моделі BERT. Крім цього буде розглянуто проблему моделювання мови з маскою та підходи до дистиляції знань, а також оптимізації процесу дистиляції з використанням спеціальної функції втрат. Також буде проведено

аналіз наявних бізнес процесів, пов’язаних з задачами обробки природної мови та сформовано вимоги до програмного інструментарію.

У Розділі 2 буде представлена архітектура системи дистиляції моделей для обробки тексту на високому рівні та на рівні окремих компонентів: trainer, data та tests.

У Розділі 3 буде реалізовано програмний інструментарій дистиляції моделей та натреновано зменшену версію моделі BERT, TinyBERT, для завдання моделювання мови з маскою. TinyBERT демонструватиме високу швидкість та набагато менший розмір порівняно з моделлю вчителем, що дозволить розширити спектр можливих застосувань моделі, включаючи вбудовані та мобільні пристрої.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Постановка задачі

Дипломна робота присвячена створенню програмного інструментарію дистиляції знань моделей машинного навчання для задач обробки природної мови.

Завдання дослідження:

1. Ознайомитися з архітектурою нейронних мереж типу “трансформер”.
2. Проаналізувати процес дистиляції знань підходом “вчитель-учень”.
3. Створити програмний інструментарій дистиляції знань моделей машинного навчання для задач обробки природної мови.
4. Натренувати зменшену версію моделі для завдання моделювання мови з маскою.
5. Провести аналіз результатів тренування.

1.1.1 Актуальність дослідження

Останні роки були дуже багаті на дослідження та розвиток моделей машинного навчання для обробки тексту, особливо з використанням глибокого навчання. Однак, збільшення кількості параметрів моделей привело до збільшення обчислювальних потреб та обсягу пам'яті, необхідного для розгортання моделей на промислових системах.

В результаті, процес навчання та розгортання таких моделей може бути досить складним та вимагати значних ресурсів, які можуть бути обмежені в мобільних та вбудованих пристроях. Тому, дистиляція моделей є одним зі способів зменшення розміру моделі та оптимізації її обчислювальних потреб для використання на пристроях з обмеженими ресурсами, таких як мобільні телефони та вбудовані системи.

Також можна зазначити, що однією з переваг дистиляції моделей є зменшення вимог до кількості даних для навчання моделі. Це особливо

важливо, коли навчальні дані обмежені або використовувані дані мають високу складність в обробці. Таким чином, дистиляція моделей може допомогти розв'язати проблему обмеженості навчальних даних, що з часом може стати ще більш актуальною проблемою.

Ці фактори підкреслюють важливість досліджень з дистиляції моделей та їх можливостей застосування в різних галузях, включаючи мобільні пристрої та вбудовані системи і прикладних задачах, таких як – текстові помічники, для автоматизації служби підтримки, виправлення помилок при написанні тексту, електронних листів, тощо, машинний переклад з однієї мови на іншу і т.д.

Метою роботи є розробка програмного інструментарію (бібліотеки) для спрощення процесу дистиляції знань для моделей машинного навчання архітектури типу “трансформер”.

Об’єктом дослідження є процес дистиляції знань для моделей природної мови архітектури “трансформер”.

Предметом дослідження є методи та технології дистиляції знань для моделей природної мови.

1.1.2 Аналіз наявних рішень

Розгляд сучасних підходів до розробки моделей машинного навчання пов'язаний з визначенням оптимальних алгоритмів та методів, що зменшують витрати ресурсів для рішення конкретних завдань. Одним із найважливіших аспектів є зменшення розміру моделі, що дозволяє знизити вимоги до обчислювальних ресурсів та збільшити її ефективність.

Окрім дистиляції знань, існують інші підходи для зменшення розміру моделей та покращення їх продуктивності. До таких підходів відносяться:

- Квантизація моделей: це метод зменшення кількості бітів, необхідних для збереження вагів моделі. Це дозволяє зменшити розмір моделі і споживання ресурсів для її виконання [27].

- Обрізання моделей: цей метод включає видалення частини моделі, які не впливають на її точність. Наприклад, можна видалити окремі шари або фільтри, які мають менший вплив на вихід моделі [28].
- Компресія моделей: цей метод використовується для стиснення моделі, щоб зменшити її розмір і споживання ресурсів [29].
- Архітектурна оптимізація: цей метод включає в себе зміну архітектури моделі, щоб зменшити її розмір і споживання ресурсів. Наприклад, можна використовувати більш прості шари або зменшувати кількість параметрів.
- Застосування простіших моделей: цей метод включає в себе використання менш складних моделей, які вимагають менше ресурсів для роботи, але можуть мати меншу точність. Наприклад, замість використання глибокої нейронної мережі можна використовувати більш просту лінійну модель.

Ці методи можна використовувати окремо або в комбінації для досягнення найкращих результатів у зменшенні розміру моделі та використанні ресурсів.

В порівнянні з цими методами дистиляція є більш сучасним та більш ефективним методом для певних задач, таких як обробка тексту, бо дозволяє використовувати менше даних для навчання (за умови навченого вчителя) і використання кращих моделей (більш підходящих для певної прикладної задачі) замість доступніших і простіших моделей [26].

1.2 Моделі архітектури типу “трансформер”

Трансформер (англ. Transformer) — це архітектура моделей глибинного навчання, яка імплементує механізм уваги, роздільно зважуючи важливість кожної частини даних входу. Її використовують переважно в області обробки природної мови та в комп'ютерному баченні.

Як і рекурентні нейронні мережі (РНМ), трансформери призначено для обробки послідовних даних входу, таких як природна мова, для таких задач як переклад та реферування тексту. Проте, на відміну від РНМ, трансформери оброблюють дані не обов'язково послідовно. Радше, механізм уваги забезпечує контекст для будь-якого положення в послідовності входу. Наприклад, якщо дані входу є реченням природної мови, то трансформерові не потрібно обробляти його початок, перш ніж взятися за обробку його кінця. Він, радше, визначає контекст, який надає значення кожному слову в цій послідовності. Ця властивість уможливлює набагато більше розпаралелювання, ніж РНМ, і відтак знижує тривалості тренування [11].

З моменту свого дебюту у 2017 році трансформери все частіше стають обраною моделлю для задач обробки природної мови замінивши рекурентні моделі, такі як довга короткочасна пам'ять. Додаткове розпаралелювання тренування уможливлює тренування на більших наборах даних, ніж це було колись можливим.

Архітектура трансформер (рис. 1.1) представляє собою нащадка архітектури “Енкодер-Декодер”[10], яка складається з двох відповідно названих частин і задача яких це перевести текст у певне векторне представлення (Енкодер) і потім відновити його у певну, звичну (текст, зображення тощо) форму (Декодер).

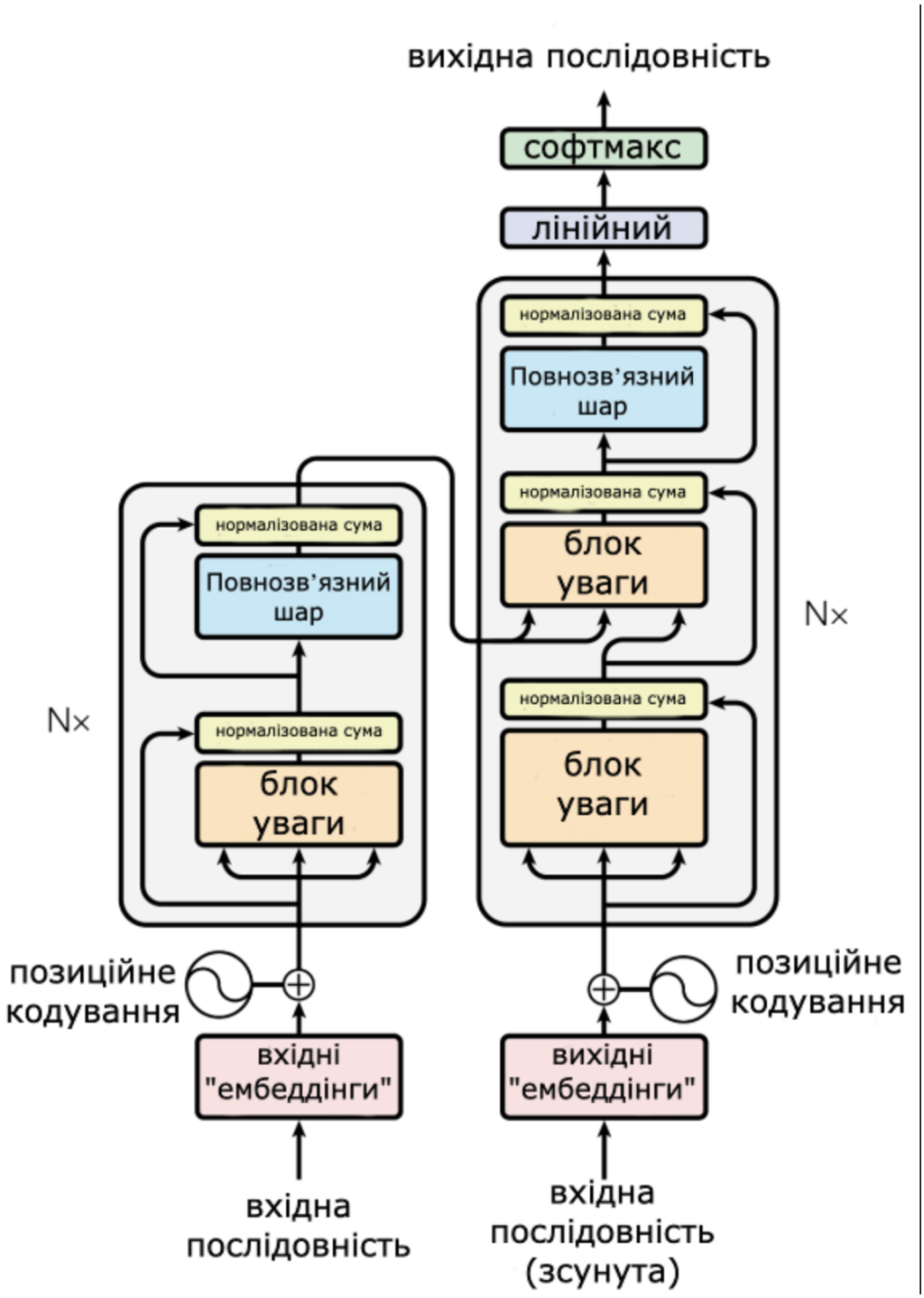

Рисунок 1.1 - Архітектура трансформер[11]

Трансформер відрізняється тим, що використовує блоки уваги, що дозволяють краще розуміти контекст слова чи речення і, за рахунок цього, може проводити обчислення паралельно, наприклад на графічному прискорювачі.

1.2.1 Архітектура “Енкодер-Декодер”

Моделі архітектури “Енкодер-Декодер” (рис. 1.2) складаються з двох частин - **Енкодер** та **Декодер** відповідно.

Рисунок 1.2 - Приклад архітектури типу “Енкодер-Декодер”

Енкодер відповідає за те, щоб представити вхідну послідовність, у вигляді певного векторного представлення, який називають **прихованим станом** (англ. hidden state) та складається з певного числа повторюваних елементів які, в свою чергу складаються з повторених задану кількість разів блоків, таких як LSTM-клітин, GRU-елементів[12] або блоків уваги**,** як у випадку архітектури трансформер, та шару повнозв’язної мережі для отримання вищезгаданого прихованого стану.

Отриманий прихований стан передається на вхід до Декодера разом з інформацією з виходу мережі з минулого токена послідовності, де токен - це нормалізоване представлення елементу послідовності. Наприклад, слово “будинок” може бути приведене до токену “будин”, що дозволить группувати до нього схожі форми цього ж слова як “будинки”, “будиночок” та ін. Звичайно, так втрачається певна частина інформації, проте без токенізації задача обробки текстів була обчислювально невиконанною.

Декодер, у випадку трансформерів, має таку саму структуру, як і Енкодер, проте на виході ми отримуємо ймовірності належності токену до

кожного класу з словника моделі. Словник представляє собою відображення токенів, що знає модель, на вектори, з якими вона може працювати.

1.2.2 Механізм уваги

У контексті нейронних мереж, увага (англ. attention) — це методика, що імітує когнітивну увагу. Це явище підсилює важливі частини даних входу, та пригнічує решту — вважається, що мережа повинна приділяти більше обчислювальної потужності цій маленькій, але важливій частині даних. Яка частина даних є важливішою за інші, залежить від контексту, й цього навчаються з тренувальних даних за допомогою градієнтного спуску.

Увагу використовують у різноманітних моделях машинного навчання, включно з обробкою природної мови та комп'ютерним баченням.

Важливою деталлю трансформерів є механізм уваги (рис. 1.3), що дозволяє моделі (Декодеру) брати до уваги інформацію з усієї послідовності, беручи зважену суму усіх попередніх станів Енкодера. Це дозволяє Декодеру призначати більшу вагу певним елементам вхідної послідовності ніж іншим, для кожного елементу на вихідній послідовності, що дає моделі кращу якість роботи.

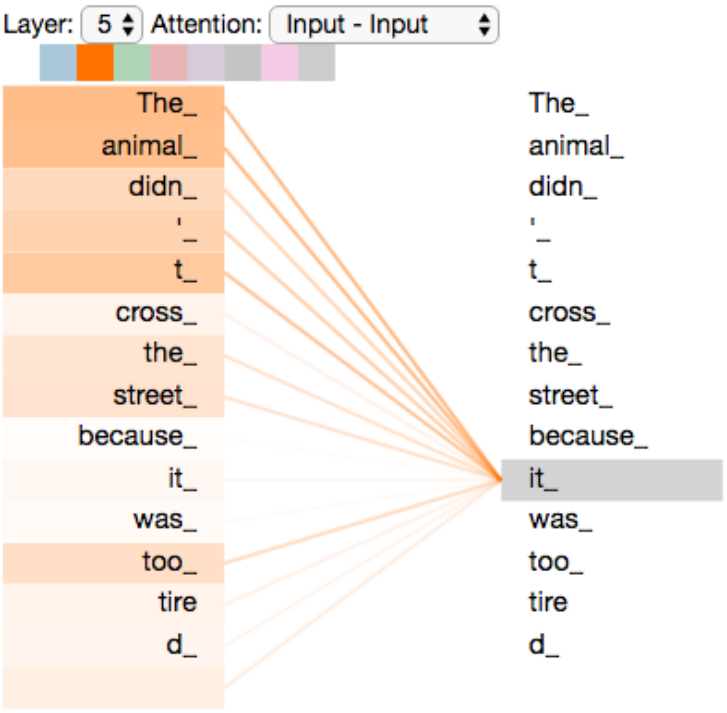

Рисунок 1.3 - Приклад роботи механізму уваги

Принцип роботи механізму уваги полягає в створенні для кожного вхідного значення трьох векторів **K**, **V**, **Q** (Keys, Values, Queries, англ - ключі, значення і запити). Вектори **K** та **Q**, зазвичай, є значеннями, що отримані на виході декодера на минулому кроці роботи, вектор **V** - це вектор прихованого стану, отриманого на виході енкодера. Результативна матриця вагів уваги розраховується за формулою (1.1):

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.1}$$

Отримані ваги множаться на прихований стан для отримання результуючого вектору на виході енкодера.

1.2.3 Модель BERT

Особливо цікавою для вивчення є сімейство моделей **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers, рис. 1.4)[3], яке, завдяки використанню архітектури типу трансформер, дозволило використовувати графічні та тензорні прискорювачі набагато повніше (оброблюючи тексти паралельно, а не послідовно, як, наприклад, моделі типу LSTM[4]).

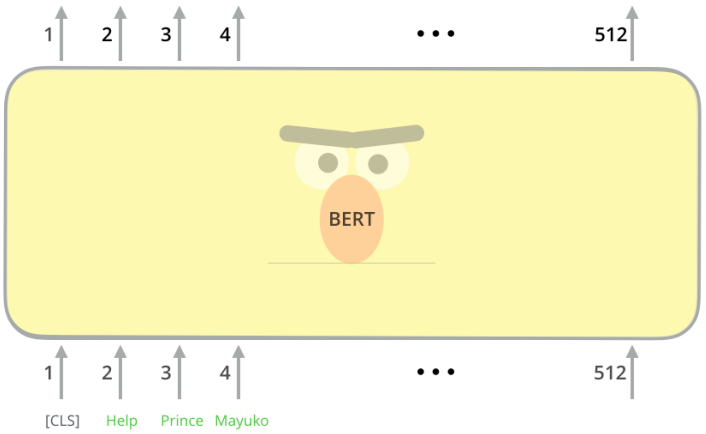

Рисунок 1.4 - Модель BERT

Оригінальна модель показала настільки високі результати, що породила за собою низку покращених моделей, таких як ALBERT[5], BORT[6] та RoBERTa[7]. Крім моделей, що якимось чином покращують BERT-а, також цікаві моделі, що зменшують розмір моделі, зберігаючи результативність. Зазвичай такі моделі носять приставку “distil”, що означає те, що модель була отримана в результаті методу дистиляції знань, наприклад distilBERT[8] або distilRoberta[9].

Головна частина архітектури BERT - це стек енкодерів, які використовуються для отримання контекстуальних представлень слів у вхідному тексті. Кожен енкодер в BERT є багатоголовим трансформером, що складається з багатьох шарів уваги та позиційно-залежних повнооб’ємних

мереж. Енкодери BERT використовують контекстуальне уявлення слова, що враховує контекст всього речення при обчисленні представлення кожного слова. Архітектура BERT має змінну кількість енкодерів, зазвичай варіюючи від 6 до 12, залежно від конфігурації моделі.

1.3 Проблема моделювання мови з маскою

Проблема моделювання мови з маскою складається з двох частин:

1. Модель отримує на вхід речення з закритим токеном певною маскою
2. Модель оптимізує свої ваги для отримання такого ж речення на виході

Тобто модель отримує речення і має зробити з нього таке саме речення, спробувавши здогадатись, який токен закритий маскою. Який від цього ефект? Модель має використати контекст слів навколо закритого токена, і це саме те, що від неї і потрібно. Наприклад:

“In the autumn the [MASK] fall from the trees.” (англ. Восени [МАСКА] падають з дерев.)

Коли ми бачимо слова “падають” та “дерев” - ми здогадуємося, що мова йде про *щось,* що падає з дерев. Багато речей підходять у це речення, такі як “жолуді”, “гілки”, “листя”, але в нас є ще одна підказка - “Восени”, що ще звужує наш пошук. Найвірогідніша річь, що може падати восени з дерев - листя.

Люди використовують суміш з загальновідомих знань та лінгвістичного розуміння, щоб прийти до цього висновку. Для текстових моделей, таких як BERT та TinyBERT - це знання походить від *дуже* великої кількості читання, та гарного вивчення лінгвістики мови. Модель може не знати, що таке “Листя”, “Дерева” чи “Осінь”, проте вона знає лінгвістичні схеми (часто відомі як *паттерни*), та контекст цих слів, з чого робить висновок про слово, якого не вистачає в цьому реченні - “листя”.

Отож, ми розуміємо, що відбувається у моделюванні мови з маскою, проте як це працює?

Перший крок - токенізувати текст. Це значить привести, або ж *нормалізувати* текст до стандартної форми. Другий - створити вектор ярликів (англ. labels), що являє собою просто копію вхідного вектору. Третій - замаскувати токен певною маскою, зазвичай для цього береться доволі очевидне слово в спеціальних лапках, що не дозволяє сплутати його зі звичайним, загальновживаним словом - “[MASK]” (англ. маска). Четвертий - порахувати функцію втрат, за результатами якої можна обрахувати градієнти для ваг моделі. Ця функція втрат (показана на 1.2 та 1.3) рахується як різниця між впевненістю моделі на виході та реальними значеннями.

$$L_{KD} = CE(z^T, z^S) \tag{1.2}$$

$$CE = -\sum_{i=1}^{K} = z^T * ln(z^S) \tag{1.3}$$

1.4 Розгляд підходу дистиляції знань

В останні роки моделі машинного навчання створюються все більшими і більшими, в 2021-му році, найбільша натренована модель M6-10T[13] була викладена у вільний доступ з 10-ма трильйонами параметрів. Чим більше стають натреновані моделі, тим більше проблем створює їх практичне використання, особливо на мобільних та вбудованих пристроях. Ці великі моделі часто не можуть бути використані на таких пристроях через обмеження у доступних ресурсах для таких пристроїв, як от потужність центрального процессора, кількість вільної постійної пам’яті або ж наявність (а точніше відсутність) графічного прискорювача. До того ж, більшість досліджень, спрямовані на навчання однієї великої моделі, яка має показати гарний результат на тестовому наборі даних, а не гарно показувати себе в експлуатації на кінцевому пристрої. Це протиріччя між навчанням і

розгортанням моделі призводить до розробки моделей, які є досить точними, але не відповідають показникам з швидкості роботи чи розмірам яких потребує використання у реальних системах. **Дистиляція знань**[14] це підхід до тренування моделей який спрямований на подолання таких труднощів за допомогою “дистиляції” знань великої і складної моделі та перенесення їх на меншу і швидшу модель. У сфері дистиляції знань більшу модель часто називають “мережа вчитель”, а меншу “мережа учень”. Моделлю вчителем може виступати як одна велика модель, так і багато різних моделей, зазвичай навчених на складних даних, включаючи дані поганої якості, аугментовані дані та дані для багатьох задач чи доменів. Після того як модель вчитель навчена процес дистиляції може виділити і перенести натреновані знання на меншу модель учня (рис. 1.5).

Модель учень навчається імітувати виходи моделі вчителя для досягнення такої самої, або навіть кращої точності. Як тільки результуюча модель, спроектована за вхідними вимогами, навчена - вона може бути розгорнута та використана на повільних мобільних пристроях чи вбудованих пристроях для прикладного використання.

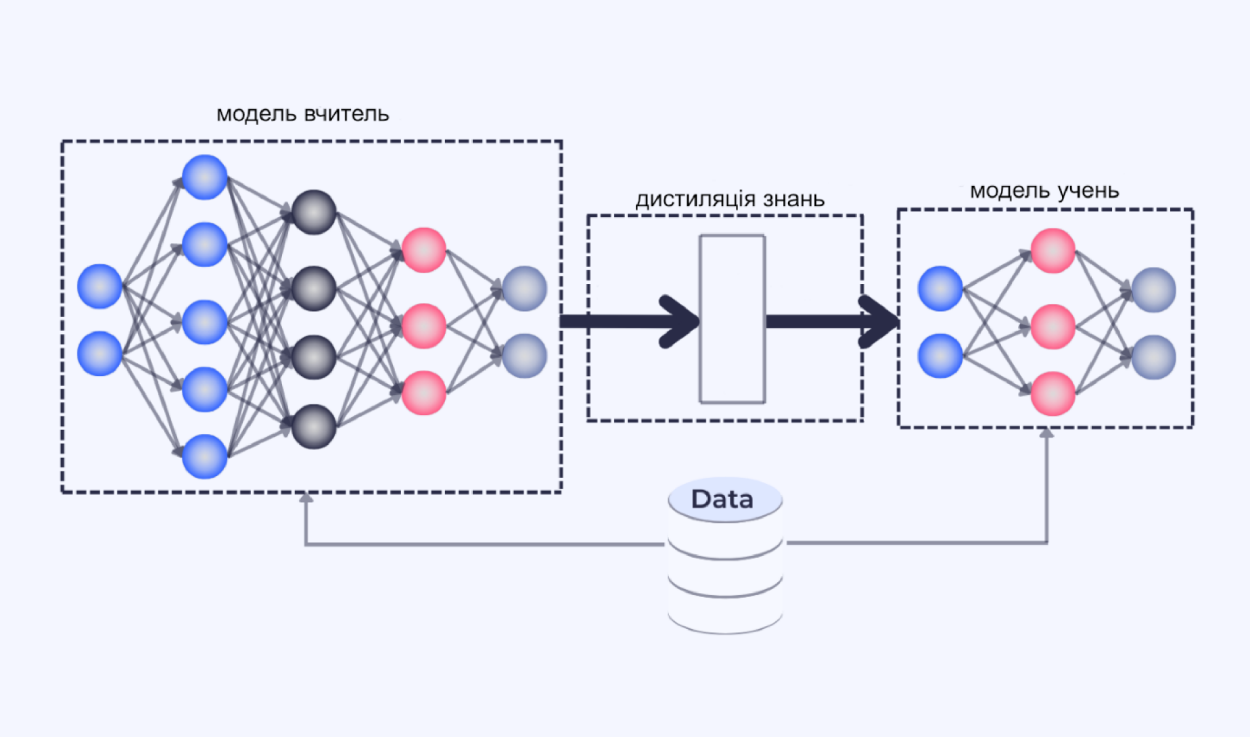

Рисунок 1.5 - Приклад підходу “вчитель-учень” у дистиляції знань

Існують три типи підходів до дистиляції знань, які базуються на тому, як знання переносяться на модель:

- Дистиляція знань на основі відповідей

  Дистиляція знань на основі відповідей вилучає та переносить на модель учня дані з вихідного шару (передбачення) моделі вчителя, та модель учень може напряму імітувати результуючі передбачення моделі мінімізуючи функцію втрат. Наприклад, у задачі класифікації модель вчитель створює на виході ймовірності належності об’єкту до певного класу, що зберігають цінну інформацію, яка може бути використана для оптимізації моделі учня.

- Дистиляція знань на основі особливостей (англ. feature)

  Даний тип дистиляції перехоплює інформацію з проміжних та вихідних шарів моделі вчителя. Проміжні шари зберігають інформацію про активацію шарів, які напряму імітовані моделлю учнем для того, щоб більш ефективно перенести інформацію без великих втрат.

- Дистиляція на основі відношень

  Даний тип дистиляції перехоплює не тільки інформацію з проміжних шарів та вихідного шару моделі вчителя. Він також досліджує відношення між різними навчальними даними та шарами. Існують три види такої дистиляції. Оффлайн дистиляція, у якій модель учень навчається після того, як модель вчитель була повністю навчена, онлайн дистиляція, у якій учня і вчителя навчають одночасно за допомогою паралельних обчислень та само-дистиляція (англ. self-distillation), у якій модель учень вивчає ті самі знання як і модель вчитель, бо має таку саму структуру, як і учитель (рис. 1.6).

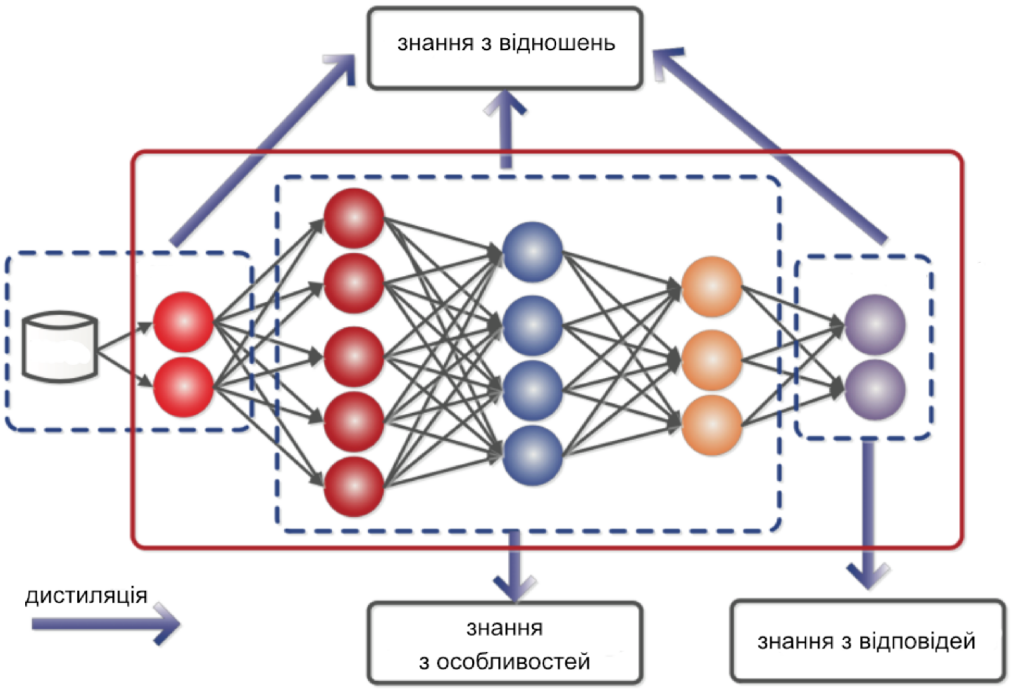

Рисунок 1.6 - Приклад різних видів дистиляції знань

Крім того, існують три основних види тренування моделей вчителя та учня, які називають офлайн, онлайн та само-дистиляцією (англ. offline, online, self-distillation). Це розділення походить від того, чи модифікується модель вчиталь в той же час, як і модель учень (рис. 1.7).

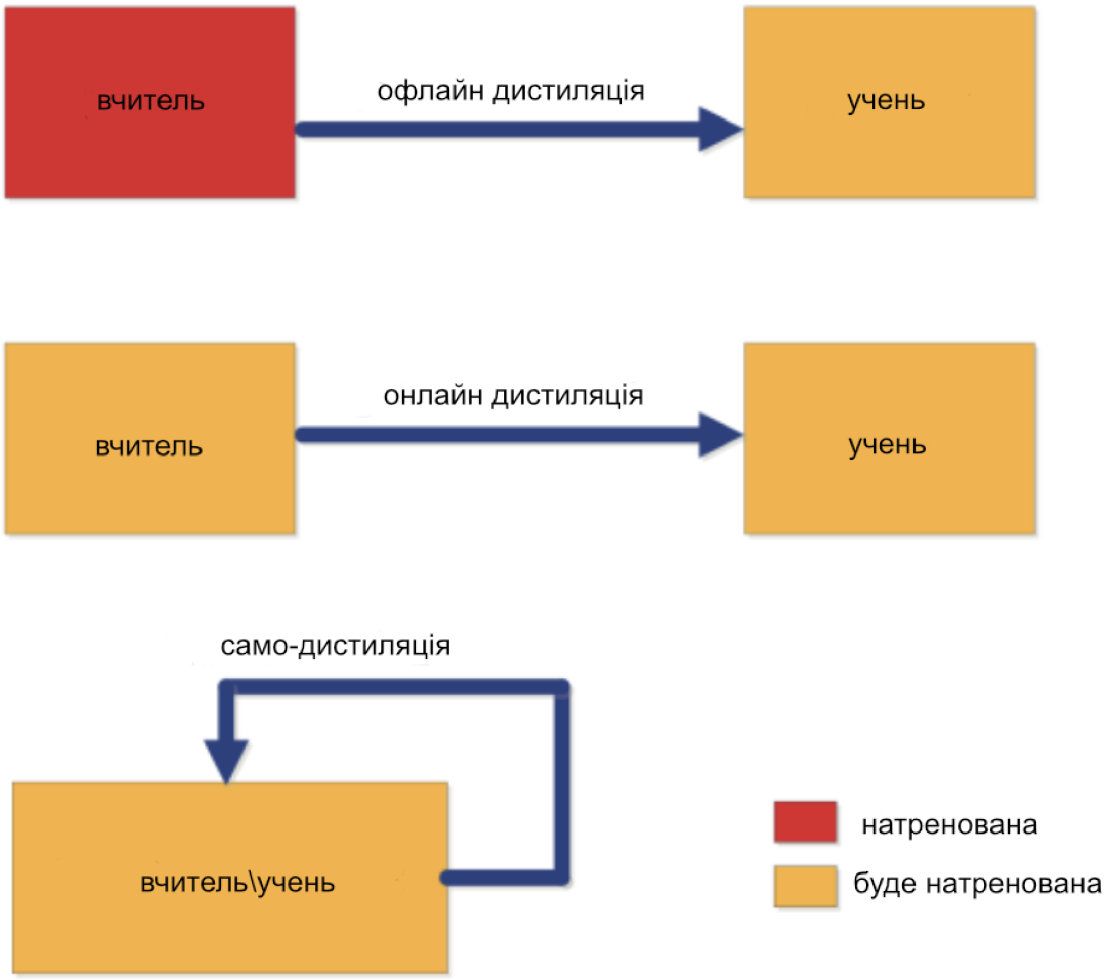

Рисунок 1.7 - Види дистиляції за способом тренування

- Офлайн дистиляція

  Офлайн дистиляція це найбільш розповсюджений метод, у якому модель вчитель використовується для навчання моделі учня. У цій схемі модель вчитель попередньо навчається на тренувальному наборі даних і вже потім знання вчителя дистилюються на модель учня. Враховуючи останні досягнення в глибокому навчанні, існує велика кількість навчених моделей які можуть служити в якості вчителя, залежно від необхідних задач. Крім того, такий підхід є найлегшим у реалізації.
- Онлайн дистиляція

У деяких випадках попередньо навчена модель вчитель не є доступною. Для таких випадків можна використовувати онлайн дистиляцію, у якій модель вчитель та учень навчаються одночасно в одному паралельному навчальному процесі.

- Само-дистиляція

  Іноді, одна й та сама модель може виступати в якості як вчителя, так і учня. Наприклад - знання з більш глибоких прихованих шарів можна використовувати для перенесення знань на шари ближче до вихідного шару моделі або для перенесення знань моделі з ранніх епох процесу тренування на більш пізніші. Цей тип дистиляції можна вважати особливим випадком онлайн дистиляції.

1.5 Оптимізація дистиляції використовуючи функцію втрат

В машинному навчанні функція втрат (англ. loss function) або функція витрат (англ. cost function) — це функція, яка відображує подію, або значення однієї чи декількох величин, на дійсне число, яке інтуїтивно представляє якісь «витрати», пов'язані з цією подією. Задача оптимізації намагається функцію втрат мінімізувати. Цільова функція (англ. objective function) є або функцією втрат, або протилежною їй (яку іноді називають функцією винагороди, функцією прибутку, функцією корисності, функцією допасованості тощо), в разі чого вона підлягає максимізації.

Зазвичай при дистиляції знань модель, яка використовується для задач класифікації використовує зважене середнє двох функцій втрат для оптимізації перенесення знань між моделю учнем і моделлю вчителем. Обидві функції втрат базуються на кросс-ентропії, яка мінімізує різницю між спражніми значеннями і значеннями, що видала модель. Перша функція використовує результати моделі вчителя для оптимізації правдивих міток, що

будуть використовуватися при навчанні моделі учня. Друга функція використовує ці мітки як правдиві значення і порівнює їх з результатами учня. Результуюча функція втрат обчислюється як зважене середнє цих двох функцій втрат. В даній роботі використовувалася функція перехрестної ентропії (англ. Cross-Entropy), що між двома розподілами ймовірності p та q над спільним простором подій вимірює середню кількість біт, необхідних для впізнання події з простору подій, якщо схема кодування, що використовується, базується на розподілі ймовірностей q, замість «істинного» розподілу p (формула 1.4).

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x) \tag{1.4}$$

1.6 Аналіз наявних бізнес-процесів, пов'язаних з задачами обробки природної мови

В сучасному світі обробка природної мови (NLP) стала важливим інструментом для багатьох бізнес-процесів. Вона допомагає компаніям аналізувати, розуміти та використовувати текстову інформацію для покращення своїх продуктів та послуг, зменшення витрат та поліпшення ефективності.

Цей розділ розглядає різні бізнес-процеси, які пов'язані з задачами обробки природної мови, такі як класифікація текстів, генерація тексту, автоматичне розпізнавання мовлення, машинний переклад та приклади покращення цих процесів за допомогою дистиляції текстових моделей, яка зменшує їх розмір, залишаючи якість на тому ж рівні.

Ось декілька прикладів бізнес-процесів, що пов’язані з обробкою природної мови:

- Класифікація текстів: багато компаній використовують моделі обробки природної мови для автоматичної класифікації текстової інформації.

Наприклад, класифікація повідомлень в соціальних медіа за тоном відгуку[15] (позитивний, негативний або нейтральний) або автоматичне визначення теми електронної пошти[16].

- Генерація тексту: іншим прикладом є генерація тексту[17], яку можна використовувати для створення автоматизованих звітів, новинних статей, продуктових описів та інших видів контенту. Такі системи можуть бути корисними для бізнесів, які хочуть зосередитися на більш високорівневих завданнях, замість витрачання часу на написання повсякденних текстів.
- Автоматичне розпізнавання усного мовлення: це ще один приклад, який використовується у бізнесі. Наприклад, банківські компанії можуть використовувати автоматичне розпізнавання усного мовлення для ідентифікації клієнта в телефонній розмові та автоматичної перенаправлення на відповідний відділ.
- Машинний переклад: компанії, які працюють з клієнтами або партнерами з інших країн, можуть використовувати машинний переклад[18], щоб допомогти з розумінням іноземних текстів, таких як листи або документація.

Щодо покращення цих процесів за допомогою дистиляції текстових моделей, можна розглянути наступні приклади:

- Зменшення розміру моделей для більш ефективного використання: дистиляція текстових моделей може допомогти зменшити розмір моделі без значного втрати якості, що може забезпечити більш ефективне використання моделей на обмежених пристроях або в умовах обмеженого ресурсу.
- Зменшення часу навчання моделей: дистиляція може зменшити час, необхідний для навчання текстових моделей, зменшуючи їх розмір та складність, а також покращуючи швидкість навчання.

- Покращення швидкості виконання: дистиляція може допомогти зменшити обчислювальні витрати для виконання завдань обробки природної мови, що може покращити швидкість виконання та знизити витрати на інфраструктуру.
- Покращення ефективності використання ресурсів: дистиляція може допомогти зменшити вимоги до обчислювальних ресурсів для роботи з текстовими моделями, що може допомогти бізнесам зекономити кошти на обчислювальних ресурсах та використовувати їх більш ефективно.

- Хмарні технології, що є надзвичайно популярними серед бізнесів, оскільки вони надають користувачам доступ до потужних обчислювальних ресурсів з погодинною оплатою. Це означає, що бізнеси можуть використовувати зовнішні хмарні ресурси замість забезпечення власної інфраструктури для виконання задач з обробки природної мови. Таким чином, бізнеси можуть зменшити свої витрати на обчислювальні ресурси, які можуть бути дуже дорогими, особливо для менших компаній. В той же час, використання дистильованих моделей може значно зменшити вимоги до обчислювальних ресурсів. За рахунок зменшення розміру моделі, вона може бути виконана на менш потужному обладнанні, що зменшує витрати на інфраструктуру. Саме цей приклад був врахований при розробці програмного інструментарію, який допомагає покращити бізнес-процеси з обробки природної мови. Дистилюючи моделі, що використовуються у цих процесах, можна значно зменшити розмір моделі та вимоги до ресурсів для її виконання, що забезпечує більш ефективне використання обчислювальних ресурсів. Таким чином, використання дистильованих моделей у бізнес-процесах обробки природної мови може бути економічно вигідним та допомогти компаніям заощадити гроші на інфраструктурі.

1.7 Формування вимог

Моделі архітектури типу “трансформер” навчаються на проблемі моделювання мови з маскою (Masked Language Modeling, MLM), яка є ключовою задачею у сфері обробки природної мови (NLP). Цей підхід допомагає моделі набути здатність розуміти, як слова використовуються в контексті і як їх можна замінити (проблему MLM було розглянуто детальніше у розділі 1.3). В даній роботі буде розглянуто процес дистиляції моделей саме для даної задачі, тому моделі мають відповідати наступним функціональним вимогам:

Функціональні вимоги

В рамках програмного інструментарію модель природної мови, отримана процессом дистиляції, повинна:

- приймати на вхід текстові дані, які можуть містити маски для слів.
- вміти передбачати наступне слово в тексті з масками на основі контексту.
- вміти генерувати текстовий вихід зі словами, які замінили маски у вхідних даних.

В рамках програмного інструментарію веб-застосунок для роботи з моделями природної мови повинен дозволяти користувачу робити передбачення наступного слова в тексті з масками на основі контексту.

Так як суть методу дистиляції знань в отриманні легшої і швидшої моделі, що майже не поступається в можливостях моделі-вчителю, тому задамо наступні нефункціональні вимоги для моделі, навченої методом дистиляції знань:

Нефункціональні вимоги

В рамках програмного інструментарію модель природної мови, отримана процессом дистиляції, повинна:

- бути швидкою та має працювати з достатньою швидкістю на масштабованих наборах даних.

- бути точною та має досягати задовільної точності (топ 5 - 55%) на тестових даних.
- бути ефективною з точки зору пам'яті та має використовувати мінімальну кількість пам'яті під час використання.
- бути стійкою до шуму в даних та має поводитись адекватно, коли вона отримує вхід з помилками або з шумом в даних.

В рамках програмного інструментарію веб-застосунок для роботи з моделями природної мови повинен мати зручний інтерфейс користувача.

1.8 Висновок до розділу 1

У Розділі 1 було проведено дослідження методу дистиляції знань для обробки тексту. Була сформована постановка задачі та визначена актуальність дослідження.

У рамках аналізу наявних рішень було розглянуто моделі архітектури типу "трансформер", архітектуру "Енкодер-Декодер", механізм уваги та модель BERT. Зокрема, було досліджено проблему моделювання мови з маскою, яка має важливе значення для задач обробки тексту.

В розділі було представлено підхід дистиляції знань, який передбачає передачу знань від однієї моделі (учителя) до іншої (учня) з метою покращення ефективності та швидкості роботи моделі-учня. Оптимізація процесу дистиляції знань була розглянута через використання спеціальної функції втрат, яка дозволяє забезпечити більш ефективний процес передачі знань.

Також було проведено аналіз наявних бізнес-процесів, пов’язаних з обробкою природної мови і сформульовано функціональні та нефункціональні вимоги до моделей машинного навчання, навчених методом дистиляції знань.

РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ ДЛЯ ПРОГРАМНОГО ІНСТРУМЕНТАРІЮ

У цьому розділі буде описано проектування загальної архітектури системи, включаючи опис компонентів, модулів, алгоритмів та методів, які будуть використовуватись у програмному інструментарії. Архітектура системи є важливим елементом проектування програмного забезпечення, оскільки вона визначатиме загальну структуру та функціональність програмного продукту. Також буде розглянута взаємодія компонентів між собою та зовнішніх програм та сервісів для кращого розуміння загальної структури системи.

2.1 Проектування програмного інструментарію

За наведеними в розділі 1 функціональними вимогами та оглянутими раніше бізнес-процесами можна визначити наступні функції, які має надавати система:

- Надавати змогу тренувати моделі машинного навчання на задачах моделювання мови з маскою
- Проводити перенесення знань з більшої моделі (моделі вчителя) на меншу (учня) методом дистиляції знань
- Дозволяти користувачеві задавати параметри тренування моделі
- Зберігати модель на хмарне сховище, для майбутнього використання

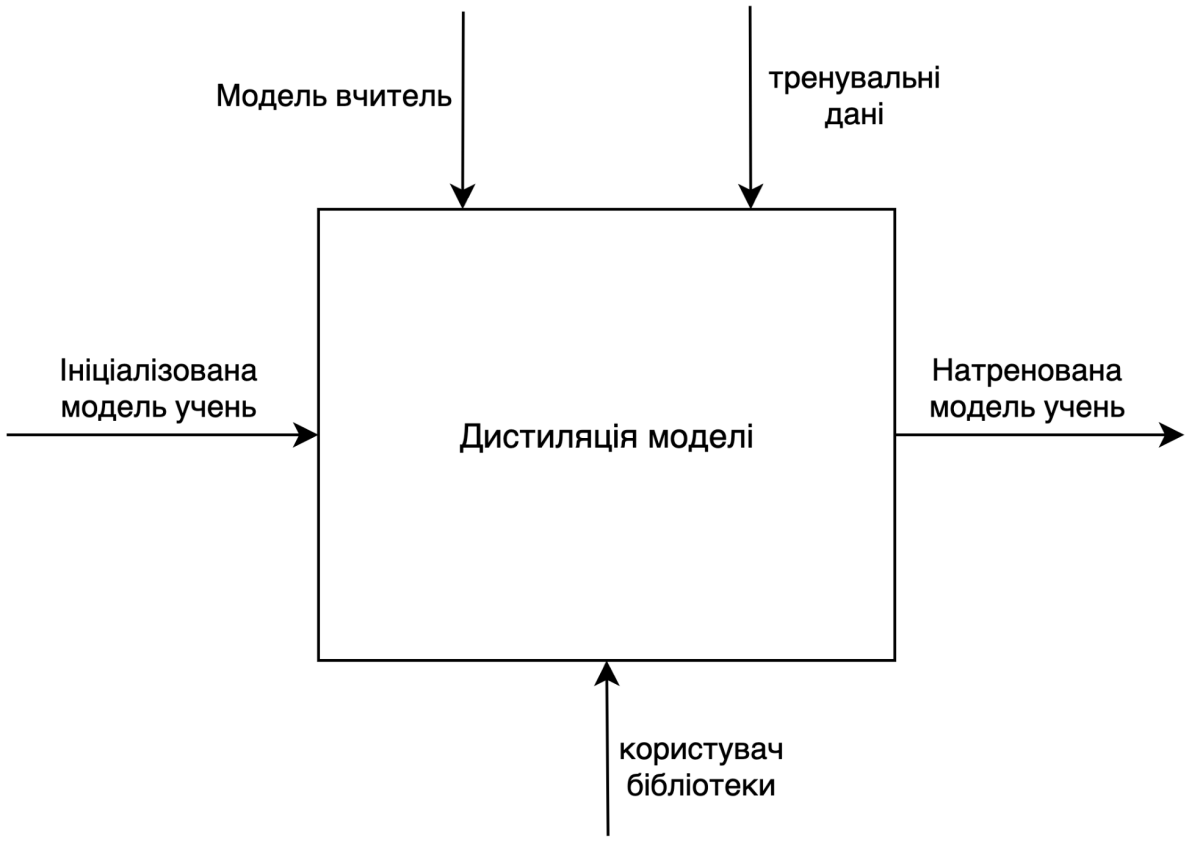

Рисунок 2.1 - Контекстна діаграма

Для спрощення проектування компонентів системи представимо систему у вигляді “чорного ящику” (рис. 2.1).

Опираючись на задані функції системи, та її представлення у вигляді “чорного ящику” розіб’ємо систему на логічні частини, кожна з яких буде відповідати за певну логічно-виділену частину:

- Компонент “trainer” - відповідатиме за тренування моделі та передачу аргументів для тренування
- Компонент “data” - відповідатиме за все, що пов’язано з даними, а саме попередню обробку даних для задачі моделювання мови з маскою та збереження натренованої моделі на хмарному сховищі
- Компонент “tests” - відповідатиме за тестування інших компонентів

2.1.1 Проектування високорівневої архітектури системи

У цьому розділі буде описана високорівнева архітектура програмного інструментарію. Бібліотека складатиметься з трьох основних компонентів: data, trainer та tests (наведено на рис. 2.2).

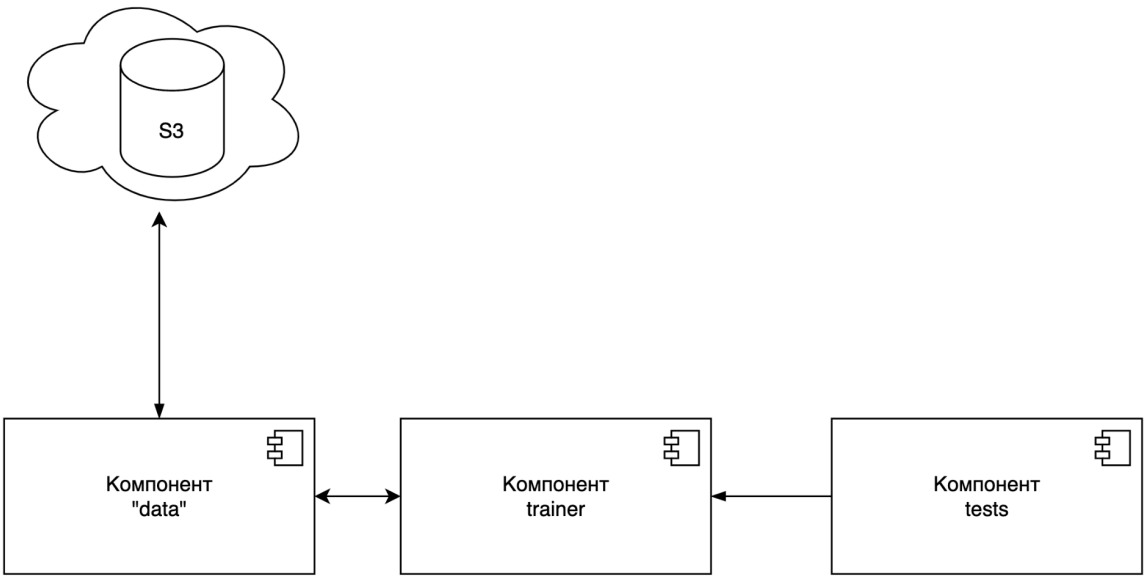

Рисунок 2.2 - Високорівнева архітектура програмного інструментарію

Основний сценарій використання програмного інструментарію передбачає використання класів з модулю trainer напряму, та, за бажанням, коду з модулю data для додавання зворотнього виклику в клас DistillationTrainer з можливістю зберігати модель на сховищі даних Amazon S3. Приблизний сценарій використання наведено на діаграмі послідовності на рисунку 2.3.

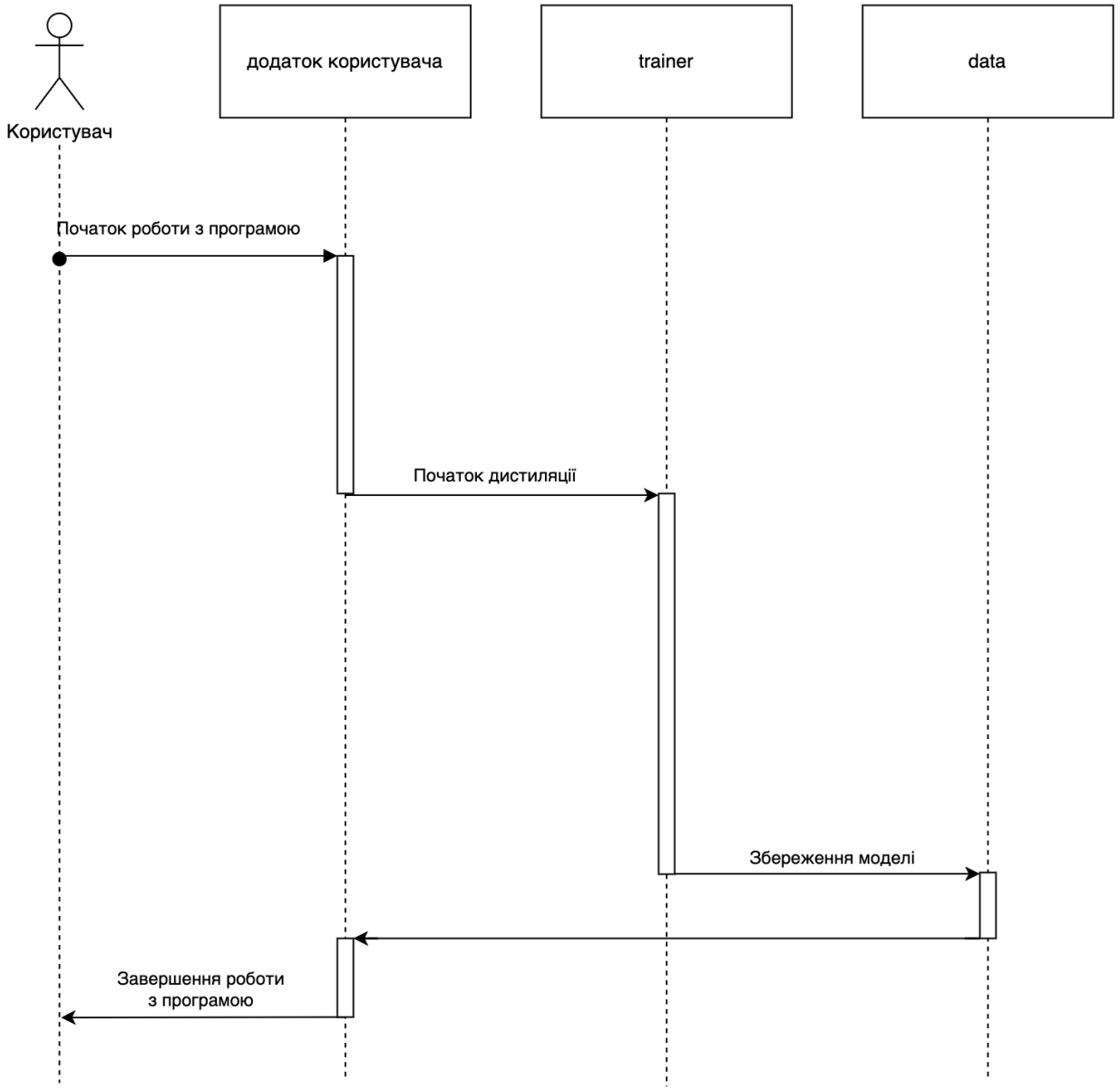

Рисунок 2.3 - Діаграма послідовності роботи з програмним інструментарієм

2.1.2 Проектування компоненту trainer

Компонент trainer буде відповідати за логіку дистиляції. Він буде зберігати у собі код двох основних класів - DistillationTrainer та DistillationTrainingArguments (рис. 2.4).

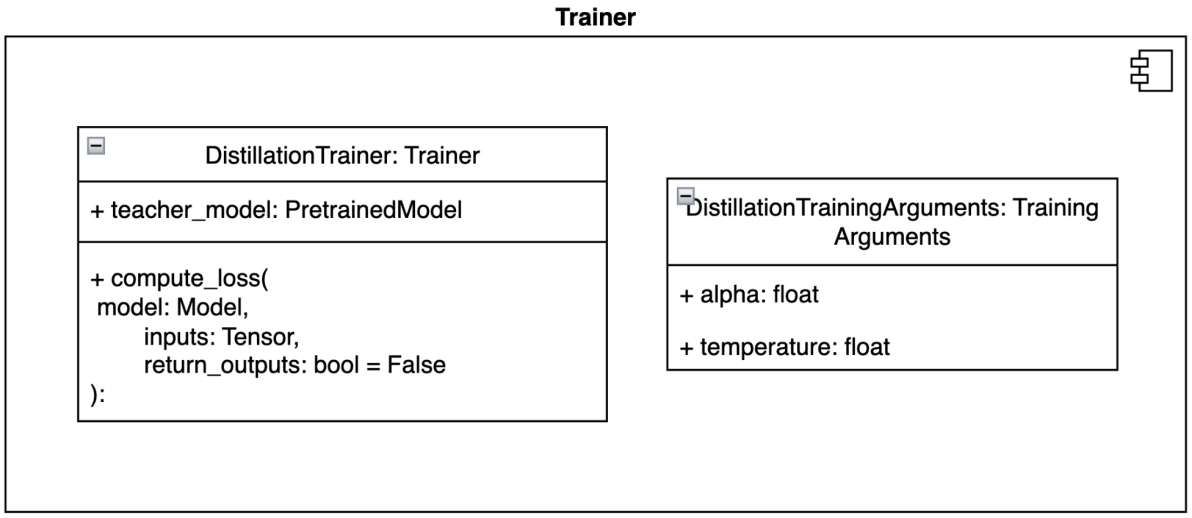

Рисунок 2.4 - Структура модулю trainer

Далі наведено короткий опис цих двох класів:

Клас **DistillationTrainingArguments** буде підкласом класу TrainingArguments з бібліотеки Hugging Face Transformers. Цей клас буде успадковувати всі атрибути та методи від TrainingArguments, а також додавати два додаткові параметри, які будуть специфічними для задачі дистиляції знань:

- alpha (значення за замовчуванням: 0.5): цей параметр буде визначати вагу втрати студента під час обчислення загальної втрати. Загальна втрата складатиметься з втрати студента, помноженої на alpha, та втрати дистиляції, помноженої на (1 - alpha).
- temperature (значення за замовчуванням: 2.0): цей параметр буде використовуватись для "пом'якшення" ймовірностей при обчисленні

втрати дистиляції. Він буде впливати на розподіл ймовірностей, згладжуючи різницю між найвищими та найнижчими значеннями.

Клас DistillationTrainingArguments може бути використаний для налаштування процесу навчання моделі студента з використанням знань, отриманих від моделі-вчителя.

Клас **DistillationTrainer** буде підкласом класу Trainer з бібліотеки Hugging Face Transformers. Цей клас буде успадковувати всі атрибути та методи від Trainer, а також додавати додатковий параметр teacher_model. Крім того, він перевизначатиме метод compute_loss.

- Параметр teacher_model буде використовуватись для передачі знань від моделі-вчителя до моделі-студента під час процесу дистиляції знань. Цей параметр має бути екземпляром класу PreTrainedModel або його підкласів.

- Метод compute_loss буде перевизначений, щоб обчислити функцію втрат дистиляції знань. Він отримуватиме вихідні дані моделі-студента та моделі-вчителя, пом'якшуватиме ймовірності з використанням заданого параметра temperature і обчислюватиме втрату дистиляції за допомогою функції втрат CrossEntropy. Загальна функція втрат обчислюватиметься як зважена сума функції втрат студента та функції втрат дистиляції з використанням параметра alpha.

Клас DistillationTrainer може бути використаний для навчання моделі-студента з використанням знань, отриманих від моделі-вчителя, на основі заданих параметрів дистиляції знань.

2.1.3 Проектування компоненту data

Модуль data буде складатися з двох підмодулів: process та s3 (рис. 2.5).

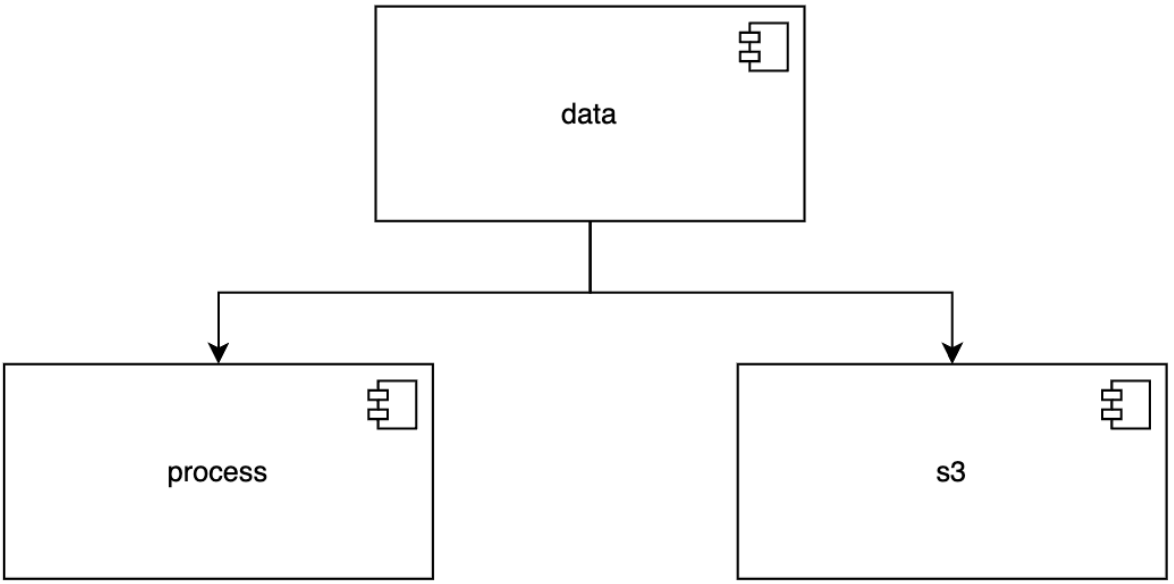

Рисунок 2.5 - структура модулю data

Розглянемо їх детальніше:

process: Цей підмодуль міститиме функцію tokenize_and_process_dataset, яка буде приймати два аргументи: dataset та teacher_tokenizer.

- dataset: це буде набір даних, який потрібно буде обробити.
- teacher_tokenizer: токенізатор, який буде використовуватись для токенізації вхідних даних.

Функція буде виконувати наступні кроки:

а) Токенізуватиме та обрізатиме вхідні речення до максимальної довжини 512.

б) Повертатиме токенізовані датасети, кількість міток та словники для перетворення міток на ідентифікатори і навпаки.

s3: Цей підмодуль міститиме клас S3UploadCallback, який буде підкласом TrainerCallback з бібліотеки Hugging Face Transformers. Цей клас дозволятиме зберігати файли моделі на Amazon S3 після кожного збереження під час навчання.

- bucket_name: буде назвою S3-бакета, в якому потрібно буде зберігати файли моделі.
- model_prefix: буде префіксом, який буде використовуватись для створення ключів S3 для файлів моделі.

Клас буде мати метод on_save, який буде викликатись під час навчання при збереженні моделі. Цей метод буде завантажувати файли моделі в S3-бакет з вказаним префіксом.

Використання цих підмодулів допоможе здійснити обробку та токенізацію наборів даних, а також збереження проміжних результатів навчання моделі в хмарному сховищі Amazon S3.

2.1.4 Проектування компоненту tests

Модуль tests буде складатися з трьох підмодулів: test_trainer, test_args та test_s3 (рис 2.6).

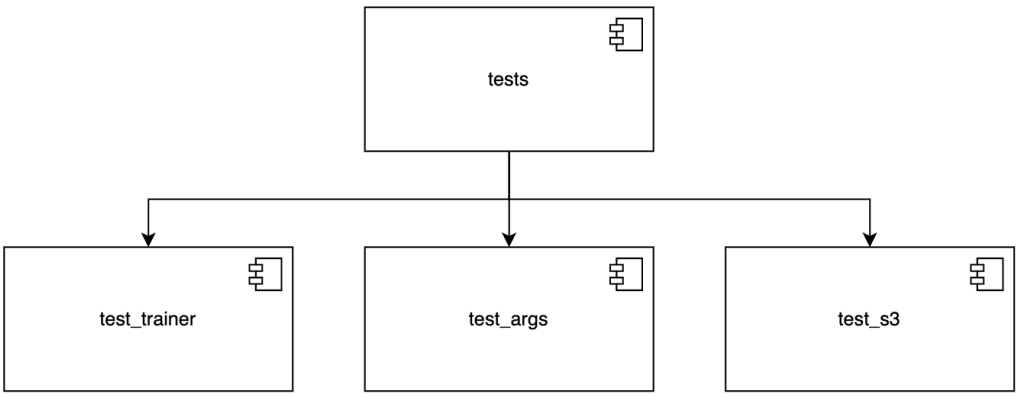

Рисунок 2.6 - Структура компоненту tests

Ці підмодулі будуть містити тестові класи для перевірки роботи класів DistillationTrainer, DistillationTrainingArguments та S3Callback відповідно.

test_trainer: Цей підмодуль буде містити тестовий клас, який буде перевіряти правильність роботи класу DistillationTrainer. Він буде виконувати різні тести для переконання в тому, що функціонал та логіка DistillationTrainer працюють вірно.

test_args: Цей підмодуль буде містити тестовий клас, який буде перевіряти правильність роботи класу DistillationTrainingArguments. Тестові методи будуть перевіряти, чи коректно працюють атрибути та методи цього класу, включаючи правильність задання параметрів дистиляції знань.

test_s3: Цей підмодуль буде містити тестовий клас, який буде перевіряти правильність роботи класу S3Callback. Тестові методи будуть переконуватися, що клас належним чином зберігає файли моделі на Amazon S3 під час навчання.

Загалом, модуль tests допомагатиме перевірити правильність роботи класів DistillationTrainer, DistillationTrainingArguments та S3Callback, а також виявити можливі проблеми та виправити їх перед використанням цих класів у реальних проектах. Виконання тестів допоможе переконатися, що функціонал цих класів працює належним чином і відповідає очікуванням.

2.1.5 Проектування модулю frontend

Для демонстрації можливостей дистильованої моделі спроектовано модуль frontend, що представляє собою простий односторінковий веб-додаток (англ. Single Page Application), що дозволяє протестувати модель-вчителя та модель-учня на довільних тестових даних.

Задача модуля - завантажувати модель і відображати користувачу результати роботи моделі, тому логічно зробити модуль максимально простим. Він складатиметься з двох основних компонент - app та templates, що зберігатимуть логіку обробки запитів та шаблони сторінок відповідно.

2.2 Вибір архітектури моделі

2.2.1 Модель вчитель

Для навчання обрано модель BERT, що була описана у розділі 1.2.3. Так як планується навчити модель учня з меншою кількістю шарів, ніж у базовому варіанті моделі BERT, то для моделі вчителя було обрано варіацію моделі **bert-base-uncased**, що має 12 шарів з архітектурою, описаною у розділі 1.2. Детальний огляд архітектури моделі представлено у [3], далі наведено перекладений уривок з джерела, що описує інші параметри моделі:

“*Ми використовуємо оптимізатор Адам зі швидкістю навчання 1e-4, β1 = 0,9, β2 = 0,999, ... Ми використовуємо регуляризацію “дропаут” з ймовірністю 0,1 на всіх рівнях.*”

2.2.2 Модель учень

Для дослідження методу дистиляції знань для задач обробки природної мови є сенс вибрати модель учня такої ж самої архітектури, як і модель вчитель. Для цього вибрано модель BERT, яка повторює параметри моделі DistillBERT[8], за винятком розмірності прихованого шару (DistillBERT - 768, TinyBERT - 312) та повнозв’язного прихованого шару (DistillBERT - 3072, TinyBERT - 1200). Даний вибір гіперпараметрів є комбінацією параметрів DistillBERT та варіацією $BERT_{Tiny}$ з [32]. Далі наведено гіперпараметри моделі учня:

"vocab_size": 30522 - розмір словника моделі, визначає кількість слів яку може вивчити модель

"hidden_size": 312 - розмірність прихованого стану

"num_hidden_layers": 4 - кількість прихованих шарів

"num_attention_heads": 12 - кількість блоків уваги

"intermediate_size": 1200 - розміри повнозв’язного прихованого шару

"hidden_act": "gelu" - функція активації прихованого шару

"hidden_dropout_prob": 0.1 - вірогідність регуляризації типу “дропаут” для прихованих шарів

"attention_probs_dropout_prob": 0.1 - вірогідність регуляризації типу “дропаут” для шарів уваги

"max_position_embeddings": 512 - максимальна кількість токенів, що може бути опрацьована моделлю

2.3 Висновок до розділу 2

У цьому розділі було представлено архітектуру системи дистиляції моделей для обробки тексту. Архітектура розглядається на високому рівні, а також на рівні окремих компонентів: trainer, data, та tests.

На високому рівні система складається з трьох основних компонентів: trainer, data, та tests. Компонент trainer відповідає за тренування моделі з використанням алгоритму дистиляції. Компонент data містить функції для обробки та підготовки даних для тренування моделі. Компонент tests включає тести для перевірки коректності реалізації компонентів системи.

Також у розділі наведено опис моделі вчителя та параметри моделі учня, які були вибрані для дистиляції знань.

У підсумку, архітектура системи дистиляції моделей для генерації тексту представляє собою гнучку та модульну структуру, що дозволяє легко розширювати та модифікувати окремі компоненти системи. Використання компонентів trainer, data та tests допомагає забезпечити рівень абстракції між різними частинами системи, сприяючи кращому розумінню та підтримці коду.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ, ТЕСТОВІ ПРИКЛАДИ

3.1 Вибір технічних засобів для реалізації програмного інструментарію

У даному розділі дипломної роботи розглянуто технічні засоби, обрані для реалізації програмного інструментарію з дистиляції знань для задач обробки природної мови. Вибір технологій та інструментів є важливим етапом у процесі розробки, оскільки від них залежить якість, продуктивність та масштабованість рішення.

Було обрано мову програмування Python[19] для реалізації програмного інструментарію, оскільки вона є однією з найпопулярніших мов у галузі штучного інтелекту та обробки природної мови. Python має простий синтаксис, велику кількість готових бібліотек та інструментів, що спрощують розробку та сприяють швидкому прототипуванню.

Основними бібліотеками, які були використані в роботі, є PyTorch та Transformers. PyTorch[20] - це відкрита платформа глибокого навчання, яка дозволяє легко проектувати та розгортати нейронні мережі. Transformers[21] - це бібліотека, розроблена компанією Hugging Face, яка надає зручний доступ до передових моделей трансформаторів та інших архітектур для обробки природної мови. Завдяки цим бібліотекам можливо швидко та ефективно розробити програмний інструментарій з дистиляції знань.

З метою забезпечення доступу до ресурсів обчислювального середовища, необхідних для тренування та виконання моделей, використано платформу Google Colab[22]. Вона надає безкоштовний доступ до високопродуктивних графічних прискорювачів, що було основною причиною з вибору цієї платформи.

Далі наведено більш детальний розгляд кожної використаної технології.

3.1.1 Мова програмування Python

Python[19] (найчастіше вживане прочитання — «Па́йтон», запозичено назву з британського шоу Монті Пайтон) — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією.

Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

3.1.2 Платформа PyTorch

PyTorch (рис. 3.1) — це платформа машинного навчання на основі бібліотеки Torch, що використовується для таких програм, як комп’ютерне бачення та обробка природної мови, спочатку розроблена Meta AI, а тепер є частиною групи Linux Foundation[20]. Це безкоштовне програмне забезпечення з відкритим кодом, випущене за модифікованою ліцензією BSD.

Рисунок 3.1 - Платформа PyTorch

3.1.3 Бібліотека Transformers

Transformers[21] (рис. 3.2) надає API та інструменти для легкого завантаження та навчання найсучасніших попередньо навчених моделей.

Рисунок 3.2 - Бібліотека Transformers

Використання попередньо підготовлених моделей може зменшити витрати на обчислення, викиди вуглецю та заощадити час і ресурси, необхідні для навчання моделі з нуля. Ці моделі підтримують загальні завдання в різних модальностях, наприклад:

- Обробка природної мови: класифікація тексту, розпізнавання іменованих об’єктів, відповіді на запитання, мовне моделювання, резюмування, переклад, множинний вибір і генерація тексту.
- Комп’ютерний зір: класифікація зображень, виявлення об’єктів та сегментація.

- Аудіо: автоматичне розпізнавання мовлення та класифікація аудіо.
- Мультимодальний: відповіді на запитання в таблиці, оптичне розпізнавання символів, витяг інформації зі сканованих документів, відеокласифікація та візуальні відповіді на запитання.

Transformers підтримують взаємодію між PyTorch, TensorFlow[24] і JAX[25]. Це забезпечує гнучкість використання різних фреймворків на кожному етапі життя моделі; можливість навчити модель у трьох рядках коду в одній структурі та завантажити її для висновку в іншій. Моделі також можна експортувати у такий формат, як ONNX і TorchScript, для розгортання у виробничих середовищах.

3.1.4 Платформа Google Colab

Для тренування моделі також було вибрано хмарну платформу Google Colaboratory[22] по декільком причинам:

1. Безкоштовність платформи

   Так як тренування особливо великих моделей може займати десятки годин на сотнях графічних прискорювачах безкоштовний доступ до середовища, яке дозволяє тренувати модель до 12-ти годин за одну сесію була вагомою перевагою перед тренуванням на персональному комп’ютері, чи арендою виділенного сервера, який би коштував би чималу суму грошей.
2. Тривалість безкоштовної сесії

   Google Colaboratory дозволяє користуватися середовищем до 12-ти годин за одну сесію на безкоштовному тарифному плані, що цілком достатньо для дистиляції текстової моделі TinyBERT, тренування якої зайняло приблизно 10 годин часу.
3. Доступ до графічних прискорювачів

   Навіть найдешевші графічні прискорювачі навіть на вторинному ринку можуть коштувати декілька тисяч гривень, не кажучи вже про більш

потужні чи сучасні прискорювачі, вартість яких може сягати декількох тисяч долларів. Завдяки безкоштовному доступу до обчислювалних ресурсів Google Colaboratory є ідеальним варіантом для тренування текстових моделей, що потребують багато обчислювальних потужностей (як графічних (GPU), так, і, за можливістю - тензорних (TPU)).

3.1.5 Сховище даних AWS S3

Amazon S3 (Simple Storage Service) - це облачне сховище об'єктів, що надається Amazon Web Services. Воно призначене для зберігання та організації великих обсягів даних з високим рівнем доступності та безпеки.

Amazon S3 (рис. 3.3) є сховищем даних типу ключ-значення, яке зберігає дані у вигляді об'єктів, кожен з яких має унікальний ключ для ідентифікації та доступу до нього. Це дає можливість зберігати інформацію будь-якого типу та формату, включаючи тексти, зображення, відео та інші дані.

Використання Amazon S3[23] для зберігання моделей має декілька переваг.

- По-перше, S3 є безпечним та надійним сервісом зберігання об'єктів, який забезпечує довготривалу збереженість даних і гарантує їх доступність у будь-який момент часу. Завдяки цьому можна бути впевненим, що збережені моделі будуть доступні для використання у програмному інструментарії в будь-який момент.
- По-друге, S3 є масштабовним сервісом, що дозволяє зберігати та обробляти великі об'єми даних. Це особливо важливо для задач обробки природної мови, де моделі можуть бути дуже великими та вимагати значних обчислювальних ресурсів.

- По-третє, S3 має простий та зрозумілий API, який дозволяє легко зберігати та отримувати дані з сервісу. Це дозволяє ефективно інтегрувати збереження моделей у програмний інструментарій.

Рисунок 3.3 - Amazon S3

Отже, використання Amazon S3 для зберігання моделей є гарним вибором, оскільки це забезпечує безпеку, доступність, масштабовність та простоту інтеграції.

3.2 Опис інструктивних матеріалів користувача

3.2.1 Встановлення програмного інструментарію

Програмний інструментарій дистиляції знань призначено для використання розробниками, інженерами даних та інженерами машинного навчання для тренування моделей, тому від користувача програмного інсрументарію необхідні базові навички програмування та знання мови Python і певний досвід з використання бібліотек PyTorch і HuggingFace, а також налаштована директорія (bucket) на хмарному сховищі Amazon S3.

Основний передбачений спосіб використання програмного інструментарію - встановлення і інтеграція в програмний код, написаний

користувачем. Для встановлення програмного інструментарію у програму користувача необхідно виконати наступну команду в директорії з кодом програми користувача (рисунок 3.4):

pip install distill-trainer

Рисунок 3.4 - виконання команди для встановлення в терміналі

Для використання бібліотеки у хмарному середовищі Google Colaboratory необхідно виконати наступну команду:

!pip install distill-trainer

Після виконання команд код бібліотеки буде доступний для використання.

Також, для доступності використання наведено приклад використання, у файлі example.py.

3.2.2 Опис інтерфейсу користувача

Оскільки даний програмний інструментарій представляє собою бібліотеку, яка написана для пришвидшення процесу тренування моделей розробниками - вона не має звичного графічного інтерфейсу користувача. Усі налаштування здійснюються програмно, через параметри класів. Далі наведено опис параметрів основних класів програми.

DistillationTrainer:

- model: Модель, яку потрібно навчити або відповідно покращити. Модель має наслідуватися від PreTrainedModel (з бібліотеки Transformers) або torch.nn.Module (з PyTorch).
- teacher_model: Модель, яка буде виступати вчителем. Має наслідуватися від PreTrainedModel (з бібліотеки Transformers) або torch.nn.Module (з PyTorch).
- training_args: Об'єкт TrainingArguments, який містить різні параметри, такі як шлях збереження, кількість епох, розмір пакета тощо.
- data_collator: Об'єкт DataCollator, який відповідає за обробку даних під час навчання. Якщо не вказано, використовується типовий збирач даних.
- train_dataset: Набір даних для навчання.
- eval_dataset: Набір даних для оцінки моделі. Можна вказати один набір даних або словник з кількома наборами даних, які будуть використовуватися під час оцінки.
- tokenizer: Токенізатор, який використовується для обробки тексту.
- model_init: Функція без параметрів, яка повертає новий екземпляр моделі. Це корисно для гіперпараметричного пошуку та оцінки різних моделей.
- compute_metrics: Функція, яка приймає об'єкт EvalPrediction і повертає словник з метриками. Ця функція використовується для обчислення метрик під час оцінки.

- callbacks: Список об'єктів TrainerCallback, які використовуються для налаштування роботи тренера.
- optimizers: Кортеж з двома оптимізаторами: один для параметрів моделі, а другий для швидкості навчання.
- preprocess_logits_for_metrics: Функція, яка приймає два тензори (logits та labels) і повертає модифікований тензор logits. Ця функція використовується для попередньої обробки logits перед використанням їх у функції compute_metrics. Це може бути корисним, наприклад, для відбору певних класів або застосування різних функцій активації перед обчисленням метрик.

Для класу DistillationTrainingArguments існує дуже багато параметрів, так як він наслідується від класу TrainingArguments з бібліотеки Transformers. Тому далі наведено лише основні з них:

- output_dir: Рядок, що вказує шлях до директорії, де будуть зберігатися файли моделі та результати.
- overwrite_output_dir: Логічне значення, яке вказує, чи слід перезаписувати вміст директорії output_dir.
- do_train: Логічне значення, яке вказує, чи слід навчати модель.
- do_eval: Логічне значення, яке вказує, чи слід оцінювати модель під час навчання.
- per_device_train_batch_size: Розмір пакета для навчання на пристрої (наприклад, GPU).
- per_device_eval_batch_size: Розмір пакета для оцінки на пристрої.
- gradient_accumulation_steps: Кількість кроків, на яких будуть накопичуватися градієнти перед виконанням оптимізації.
- learning_rate: Швидкість навчання для оптимізатора.
- weight_decay: Коефіцієнт згасання ваг моделі.
- num_train_epochs: Кількість епох для навчання моделі.

Також, для користувача доступний модуль frontend, що зберігає у собі логіку демонстрації роботи дистильованих моделей. На рисунку 3.5 можна побачити інтерфейс користувача. Він складається з трьох текстових полів для назви моделі-вчителя, моделі-учня та тестового прикладу відповідно. Під текстовими полями знаходиться кнопка “Подати” (англ. submit), що відправляє запит на обробку. Під нею виводяться результати роботи моделей вчителя та учня у відповідних колонках.

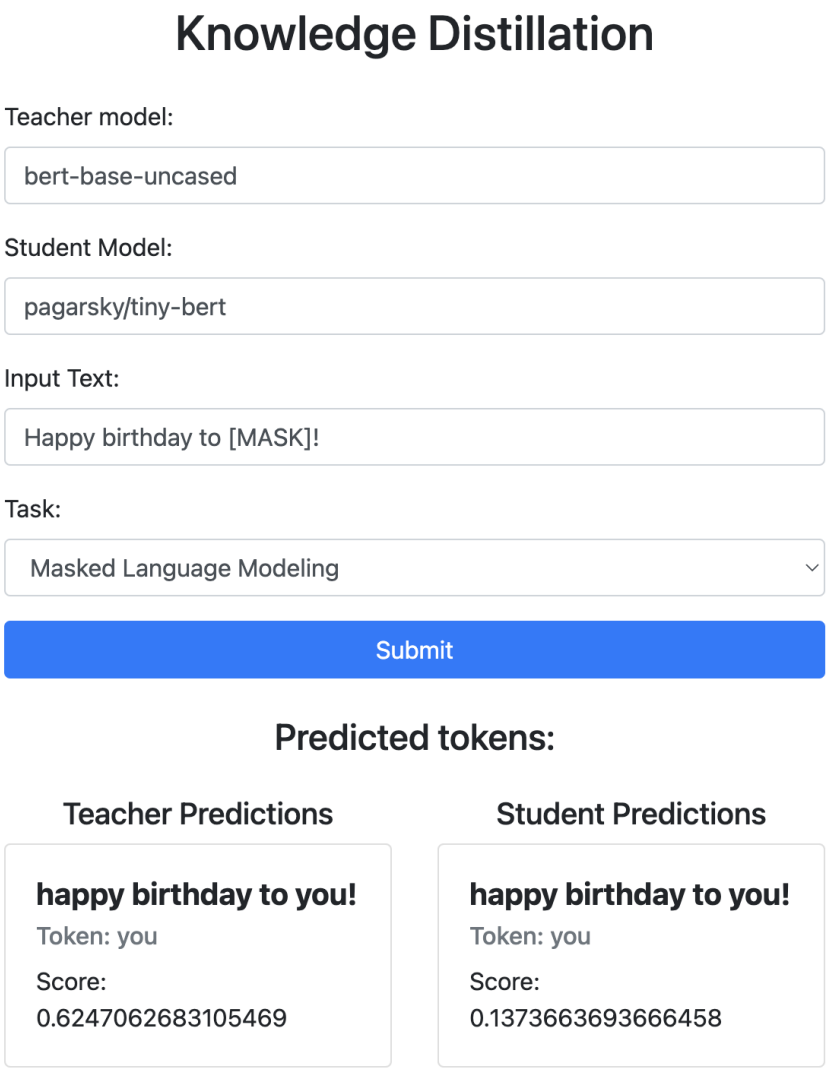

Рисунок 3.5 - Інтерфейс користувача модулю frontend

3.2.3 Огляд прикладу використання

Для спрощення використання програмного інструментарію, у файлі example.py наведено приклад тренування моделі TinyBERT. Повну версію коду можна подивитися у додатку Б, далі наведено короткий опис цього коду:

Цей код є основним скриптом для тренування зменшеної моделі TinyBERT за допомогою методу дистиляції знань від моделі-вчителя (BERT) для задачі класифікації тексту на датасеті SST-2. Розглянемо детальніше кожен з елементів коду:

Імпорт бібліотек:

- Імпортується S3UploadCallback з модуля data.s3, який відповідає за збереження моделі на S3 після кожної епохи тренування.
- Імпортується tokenize_and_process_dataset з модуля data.process, який обробляє датасет, токенізує його та виконує необхідні перетворення.
- Імпортуються DistillationTrainer та DistillationTrainingArguments з модуля trainer, які відповідають за тренування моделі з використанням алгоритму дистиляції.

Завантаження моделей та датасетів:

- Завантажуються токенізатори для моделі-вчителя та студента. Завантажується датасет SST-2. Застосовується функція tokenize_and_process_dataset для обробки датасету.

Налаштування аргументів тренування:

- Ініціалізується об'єкт DistillationTrainingArguments з параметрами тренування, що включають шлях до вихідної директорії, кількість епох, розміри пакетів, швидкість навчання, використання мультипроцесорної системи тощо.

Ініціалізація моделей та допоміжних компонентів:

- Ініціалізується об'єкт DataCollatorWithPadding, який допомагає об'єднати вхідні дані з пакетами. Ініціалізуються моделі-вчитель та студент для задачі класифікації тексту.

Ініціалізація тренера:

- Ініціалізується об'єкт DistillationTrainer з моделлю-студентом, аргументами тренування, моделлю-вчителем, навчальним та валідаційним датасетами, data_collator, токенізатором вчителя та колбеком для завантаження на Amazon S3.

Запуск тренування:

- Виконується метод train() на об'єкті DistillationTrainer для початку процесу тренування.

Загалом, цей код створює зменшену модель TinyBERT, яка навчається з моделі BERT (модель-вчитель) на датасеті SST-2 для задачі класифікації тексту. Він використовує дистиляцію знань для передачі інформації від моделі-вчителя до моделі-студента, з метою створення меншої та швидшої моделі зі збереженням якості передбачень.

3.3 Тестові приклади

Для тестування створенного програмного інструментарію були написані тести формату unittest і розміщені у директорії tests. Дані тести перевіряють поведінку класів DistillationTrainer, DistillationTrainingArguments і S3Callback. Далі наведено короткий опис цих класів:

Клас TestTrainer є класом модульного тестування для перевірки класу DistillationTrainer. Він включає такі методи:

- setUp(): Цей метод викликається перед кожним тестом. Він ініціалізує фіктивні об'єкти моделі, які імітують поведінку студентської та вчителівської моделей. Також створюється фіктивний об'єкт токенізатора та екземпляр аргументів DistillationTrainingArguments.

- test_compute_loss() (рис. 3.6): Цей тестовий метод перевіряє, чи правильно працює метод compute_loss класу DistillationTrainer. Спочатку створюється екземпляр класу DistillationTrainer з передачею моделі студента, моделі вчителя та відповідних аргументів, які були визначені в setUp. За допомогою цього екземпляра викликається метод compute_loss, який розраховує втрату на основі моделі студента та вхідних даних. Отриманий результат зберігається у змінній loss. Далі виконуються ряд перевірок для забезпечення коректності роботи функції compute_loss. Спочатку перевіряється, що loss не є None, щоб впевнитись, що функція compute_loss дійсно повернула результат. Тоді виконується перевірка, чи є об'єкт loss екземпляром torch.Tensor, щоб переконатися, що функція compute_loss повертає правильний тип даних. Також перевіряється, що розмірність тензора loss є нульовою, що означає, що він є скаляром. Нарешті, перевіряється, що тензор loss не вимагає обчислення градієнтів. Це важливо, тому що при виконанні backpropagation PyTorch не має відстежувати історію операцій для цього тензора.

```
def test_compute_loss(self):
    loss = DistillationTrainer(
        model=self.student_model,
        teacher_model=self.teacher_model,
        args=self.args,
    ).compute_loss(self.student_model, self.inputs)


    self.assertIsNotNone(loss)
    self.assertTrue(isinstance(loss, torch.Tensor))
    self.assertTrue(loss.dim() == 0)
    self.assertFalse(loss.requires_grad)
```

Рисунок 3.6 - Імплементіція функції test_compute_loss

Клас TestDistillationTrainingArguments є класом модульного тестування для перевірки класу DistillationTrainingArguments. Він містить один тестовий метод:

- test_init(): Цей тестовий метод перевіряє, чи правильно працює ініціалізація класу DistillationTrainingArguments. Він створює екземпляр DistillationTrainingArguments з вказаними значеннями alpha та temperature, а також іншими параметрами. Потім він перевіряє, чи екземпляр успадковує від класу TrainingArguments та чи встановлені правильні значення атрибутів alpha та temperature. Наостанок, він перевіряє, чи правильно встановлюються значення за замовчуванням для alpha та temperature.

Клас TestS3UploadCallback є класом модульного тестування для перевірки класу S3UploadCallback. Він містить один тестовий метод:

- test_on_save(): Цей тестовий метод перевіряє, чи правильно працює метод on_save класу S3UploadCallback. Він використовує mock_s3 з бібліотеки moto для створення фіктивного S3 середовища. У цьому тесті створюється S3 кошик, а також каталог output_dir, в якому будуть зберігатися файли моделі. Після цього створюється екземпляр Trainer з S3UploadCallback та викликається метод on_save, що завантажує файли моделі в S3 кошик. Тест перевіряє, чи всі очікувані файли були завантажені до S3 кошика, та їх імена відповідають очікуваним іменам файлів. В кінці тесту файли та каталоги видаляються.

3.4 Процес тренування

Тренування моделі зайняло приблизно 10 годин часу, на графічному прискорювачі **Nvidia K80,** що доступний безкоштовно у середовищі Google Colab. Графік тренування наведено на рисунку 3.7 (вісь X - кроки тренування, вісь Y - значення функції втрат):

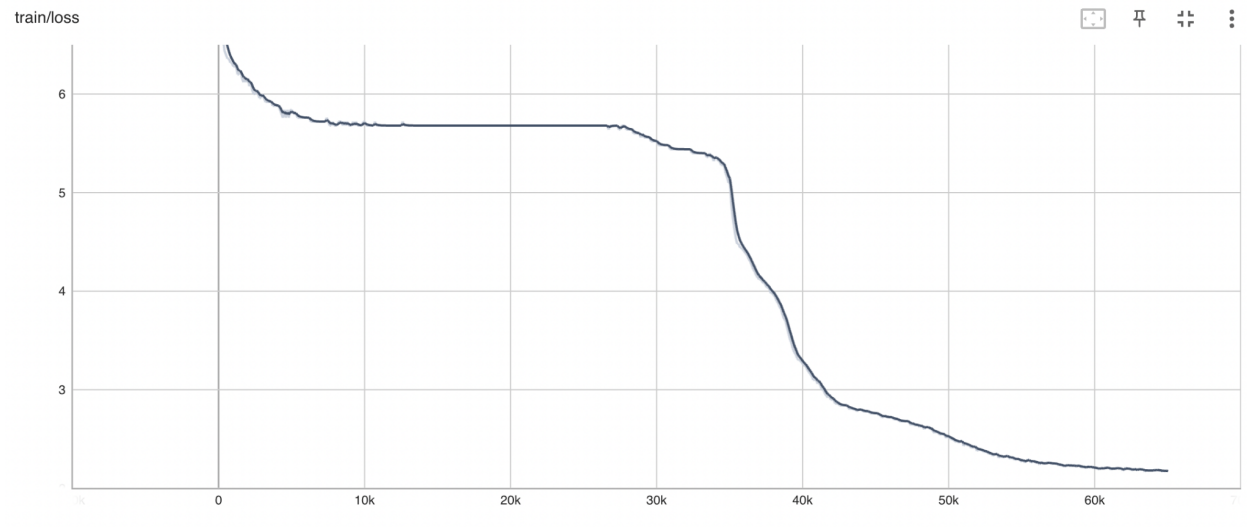

Рисунок 3.7 - Графік процесу тренування моделі

Тренування проводилося на корпусі текстових даних, що складається з копії вікіпедії англійською мовою[30], та book corpus dataset[31], що складається з різноманітних книг класичної літератури англійською мовою.

3.5 Аналіз результатів

Натренована модель TinyBERT важить у **двадцять два рази** менше (17.8 МБ проти 400 МБ) та є швидшою у більш ніж **одинадцять разів** (11 секунд на 100 речень проти 0.8 секунди) ніж модель учитель (рис 3.8, 3.9).

```
Execution time is 11.3929 seconds
Execution time is 0.8281 seconds
```

Рисунок 3.8 - Знімок екрану що демонструє швидкість відпрацювання моделей на 100 реченнях

```
400  MB
17.8 MB
```

Рисунок 3.9 - Знімок екрану, що демонструє вагу збережених на диску моделей BERT та TinyBERT

Тестування моделі на простих реченнях англійською мовою, таких як “I wish you a merry [MASK]!” (англ. Бажаю вам щастливого [МАСКА]!) показало, що модель TinyBERT гарно виконує задачку моделювання мови з маскою на простих реченнях. Якщо дати моделі більш складне речення, наприклад “Wow! TinyBERT is really [MASK] at what it does!.” (англ. Вау! TinyBERT дуже [МАСКА] у тому, що він робить!) показало, що моделі може не вистачати знань для правильного доповнення речення.

Також, процес тестування продемонстрував, що навчена модель TinyBERT хоч і дає непогані результати, але є менш впевненою у своїх передбаченнях. Наприклад, у реченні “Happy birthday to [MASK]!” модель хоч і передбачує правильне слово - “you”, проте робить це зі значно меншим числом впевненості (~13% проти ~62% у більшої моделі), що наведено на рисунку 3.10.

```
Happy birthday to [MASK]!
====================
happy birthday to you!  # 0.6247044801712036, took 0.16 s
happy birthday to you!  # 0.13736645877361298, took 0.02 s



We wish you a merry [MASK]!
====================
we wish you a merry christmas!  # 0.933159351348877, took 0.16 s
we wish you a merry christmas!  # 0.06594400107860565, took 0.01 s



The animal didn't cross the road because [MASK] was too tired.
====================
the animal didn't cross the road because it was too tired.  # 0.7269637584686279, took 0.21 s
the animal didn't cross the road because it was too tired.  # 0.430359810590744, took 0.02 s
```

Рисунок 3.10 - Знімок екрану тестування моделі

Також окремо було перевірено, як модель поведе себе отримавши на вхід дані з помилками (одруківки, тощо), що показало, що модель хоч і помиляється (рис. 3.11), проте працює правильно і не призводить до помилок.

```
Wow! TinyBERT is realy [MASK] at wat it does!
====================
wow! tinybert is realy good at wat it does!  # 0.16036053001880646, took 0.27 s
wow! tinybert is realy, at wat it does!  # 0.10147744417190552, took 0.02 s
```

Рисунок 3.11 - Тестування моделі на “шумних” даних

Окремо було протестовано модель на валідаційному наборі даних, що складався з 1000 перших речень з набору даних bookcorpus[31]. Для порівняння отриманої моделі були вибрані моделі distillbert-base та bert-base-uncased. Далі наведено отримані результати тестування.

Абсолютна точність (топ 1) - порівнює чи співпадає перший токен з передбачень моделі з правильним токеном (рис. 3.12).

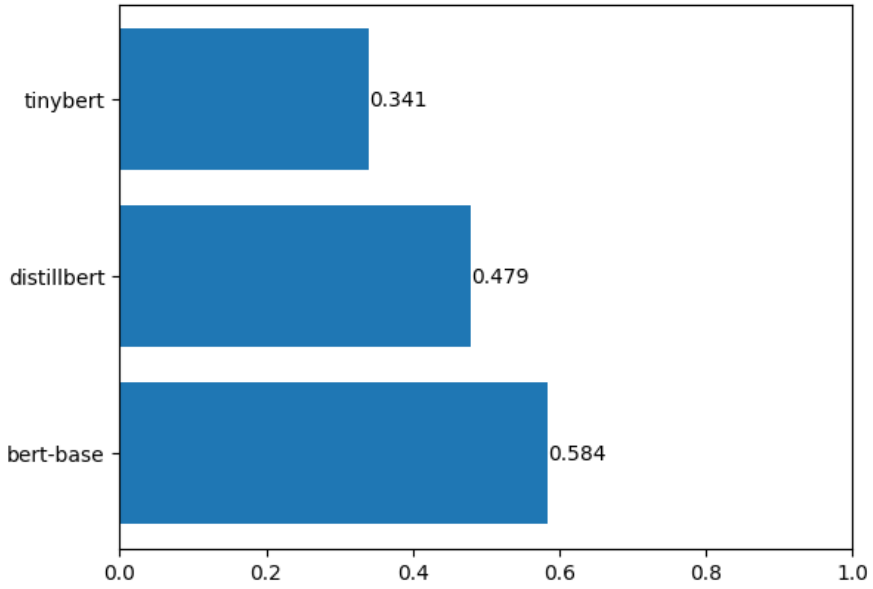

Рисунок 3.12 - Абсолютна точність (топ 1)

Абсолютна точність (топ 5) - порівнює чи співпадає хоча б один токен з передбачень моделі з правильним токеном (рис. 3.13).

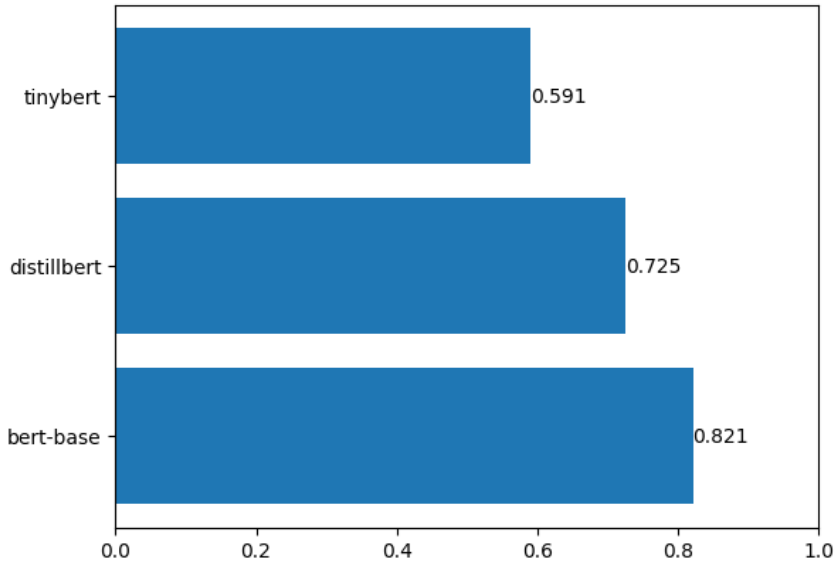

Рисунок 3.13 - Абсолютна точність (топ 5)

Відносна точність (топ 1) - порівнює чи співпадає перший токен tiny-bert-a з першими токенами вибраних раніше моделей (рис. 3.13).

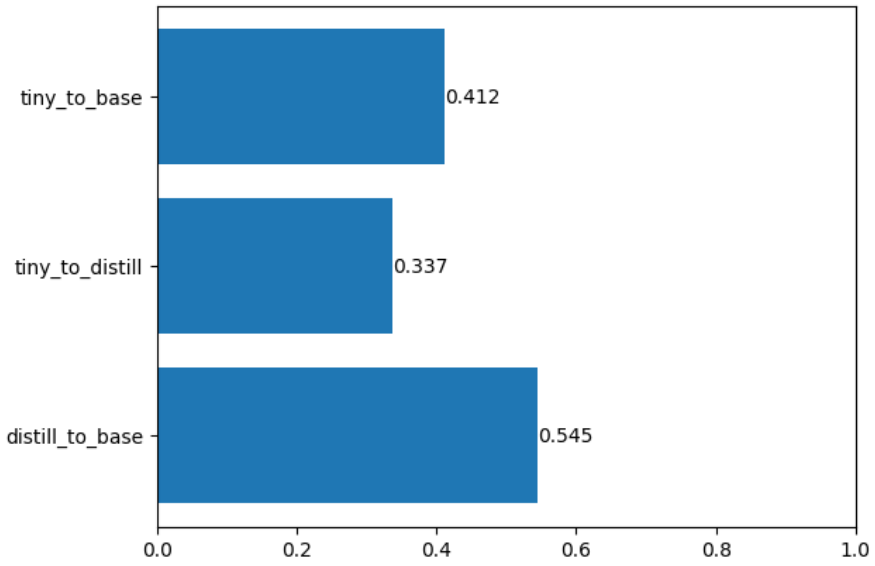

Рисунок 3.13 - Відносна точність (топ 1)

Відносна точність (топ 5) - порівнює чи співпадає хоча б один токен tiny-bert-a з токенами вибраних раніше моделей (рис. 3.14).

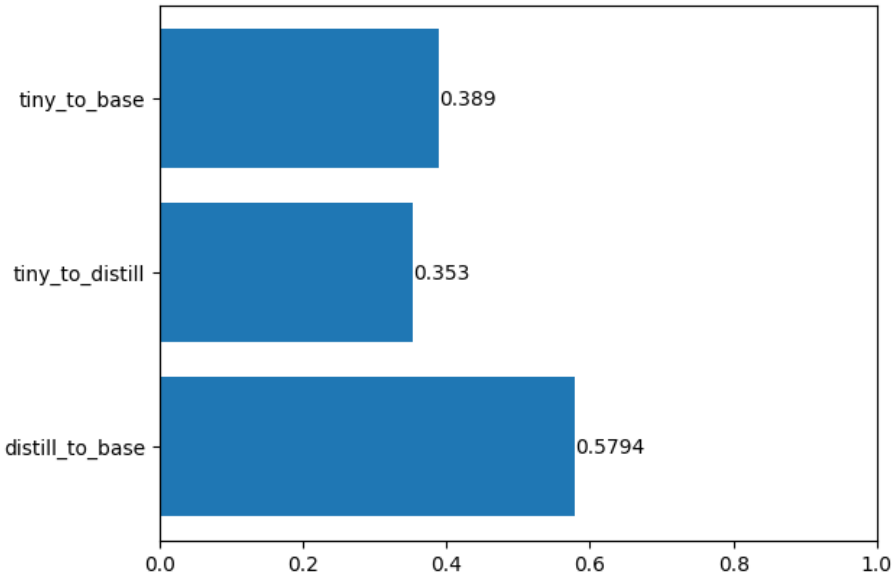

Рисунок 3.14 - Відносна точність (топ 5)

Отримані результати демонструють успішність тренування, так як вдалося отримати більшу точність (59% топ 5) від моделі, що менша в 22 рази і швидша в 11 разів.

3.6 Тестування на відповідність вимогам

Модель, отримана в результаті процесу дистиляції, має потенціал відповідати більшості зазначених вимог. Однак, важливо розуміти, що конкретна ефективність моделі залежить від якості даних, які використовуються для її тренування, а також від підбору оптимальних гіперпараметрів. Розглянувши отримані результати у минулому розділі можна провести аналіз отриманої моделі на відповідність вимогам:

Відповідність функціональним вимогам:

- приймати на вхід текстові дані, які можуть містити маски для слів - **відповідає.**
- вміти передбачати наступне слово в тексті з масками на основі контексту - **відповідає**.
- вміти генерувати текстовий вихід зі словами, які замінили маски у вхідних даних - **відповідає.**

У розділі 3.4 показано, що модель здатна приймати та обробляти дані з масками, передбачати наступне слово та замінювати маски у такому тексті. Тому можна сказати, що модель повністю відповідає поставленим функціональним вимогам.

Відповідність Нефункціональним вимогам:

- бути швидкою та має працювати з достатньою швидкістю на масштабованих наборах даних - **відповідає**. У розділі 3.4 продемонстровано, що модель є швидшою за вчителя в ~11 разів.
- бути точною та має досягати задовільної точності (топ 5 - 55%) на тестових даних - **відповідає**. У розділі 3.4 продемонстровано, що на 3-х тестових прикладах модель отримала такі ж відповіді, як і модель-вчитель та побудовано метрики, що показують точність моделі на валідаційних даних.
- бути ефективною з точки зору пам'яті та має використовувати мінімальну кількість пам'яті під час використання - **відповідає**. У розділі 3.4 продемонстровано, що модель є легшою за вчителя в ~22 рази.
- бути стійкою до шуму в даних та має поводитись адекватно, коли вона отримує вхід з помилками або з шумом в даних - **відповідає**. У розділі 3.4 продемонстровано, що модель не призводить до помилок, якщо на вхід подати дані з помилками.

Загалом, модель, отримана в результаті процесу дистиляції задовольняє усі поставлені на початку проектування вимоги. Однак, залежно від конкретного варіанту використання, можливо, доведеться провести додаткове налаштування моделі, наприклад, вибір оптимальних гіперпараметрів або покращення якості вхідних даних.

3.7 Висновок до розділу 3

В цьому розділі було проведено вибір технічних засобів для реалізації програмного інструментарію та описано інструктивні матеріали користувача.

Також, в результаті дослідження було натреновано зменшену версію моделі BERT для завдання моделювання мови з маскою під назвою TinyBERT яка за своєю результативністю наближалася до моделі вчителя на коротких текстах і показувала гарні результати на довших текстах, проте помилялася на

складних реченнях, що очікувано. Також, модель TinyBERT виявилася швидшою в одинадцять разів, та важила у двадцять два рази менше, ніж модель учитель, щодемонструє те, що массивні текстові моделі зберігають велику кількість надлишкової інформації, яку можна відкинути для отримання зрівнянних результатів на певних типах задач і розширення спектру можливих застосувань моделі доповнивши його застосуванням у вбудованих та мобільних пристроях.

ВИСНОВКИ

В даній роботі було проведено дослідження та розробка програмного інструментарію дистиляції моделей для задач обробки тексту на основі природної мови.

Розроблений в рамках дипломної роботи програмний інструментарій дистиляції моделей дозволяє покращити ефективність, швидкість та розмір моделей для задач обробки тексту на основі природної мови, забезпечуючи високу якість передбачень та можливість застосування у різноманітних областях, включаючи вбудовані та мобільні пристрої.

Використання методу дистиляції знань дозволяє створювати ефективніші, легші та швидші моделі, не втрачаючи при цьому значної якості передбачень. Архітектура системи дистиляції моделей для генерації тексту відрізняється гнучкістю та модульністю, що сприяє легкій розширеності та модифікації окремих компонентів системи.

В рамках програмного інструментарію веб-застосунок для роботи з моделями природної мови дозволяє користувачу робити передбачення наступного слова в тексті з масками на основі контексту.

Таким чином, дана робота є важливим кроком у розробці систем дистиляції моделей для обробки тексту на основі природної мови. Результати дослідження можуть стати основою для подальших розробок та покращень в цій області, що сприятиме використанню компактних та ефективних моделей в багатьох застосуваннях, пов'язаних з обробкою природної мови.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. [Електронний ресурс] - Режим доступу до ресурсу: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf
2. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. DOI: 10.48550/arXiv.1910.106831
3. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. DOI: 10.18653/v1/n19-14232
4. Sepp Hochreiter, Jürgen Schmidhuber; Long Short-Term Memory. Neural Comput 1997; 9 (8): 1735–1780 doi: 10.1162/neco.1997.9.8.1735
5. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. DOI: 10.48550/arXiv.1909.11942
6. Perry, D. J. (2020). Optimal Subarchitecture Extraction For BERT. arXiv preprint arXiv:2010.10499
7. Liu, Z., Lin, W., Shi, Y., & Zhao, J. (2021). A Robustly Optimized BERT Pre-training Approach with Post-training. In Proceedings of the 20th Chinese National Conference on Computational Linguistics. DOI: 10.48550/arXiv.1907.11692
8. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019. DOI: 10.48550/arXiv.1910.01108
9. HuggingFace: distilroberta-base [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/distilroberta-base
10. Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. DOI: 10.3115/v1/D14-1179
11. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. DOI: 10.48550/arXiv.1706.03762
12. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555
13. Lin, J., Yang, A., Bai, J., Zhou, C., Jiang, L., Jia, X., Wang, A., Zhang, J., Li, Y., Lin, W., Zhou, J., & Yang, H. (2021). M6-10T: A Sharing-Delinking Paradigm for Efficient Multi-Trillion Parameter Pretraining. arXiv preprint arXiv:2110.03888

14. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. DOI: 10.48550/arXiv.1503.02531
15. Getting Started with Sentiment Analysis [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/blog/sentiment-analysis-twitter
16. BuzzTrack: Topic Detection and Tracking in Email [Електронний ресурс] — Режим доступу до ресурсу: https://tik-db.ee.ethz.ch/file/84a6412bdf68c844ca21ed63a6eb2b31/iui07.pdf
17. Text Generation [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/tasks/text-generation
18. Translation [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/docs/transformers/tasks/translation
19. Python [Електронний ресурс] — Режим доступу до ресурсу:https://www.python.org/
20. PyTorch [Електронний ресурс] — Режим доступу до ресурсу: https://pytorch.org/
21. HuggingFace [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/
22. Google Colaboratory [Електронний ресурс] — Режим доступу до ресурсу: https://colab.research.google.com/
23. Amazon S3 [Електронний ресурс] — Режим доступу до ресурсу: https://aws.amazon.com/s3/
24. Tensorflow [Електронний ресурс] — Режим доступу до ресурсу: https://www.tensorflow.org/
25. JAX [Електронний ресурс] — Режим доступу до ресурсу: https://github.com/google/jax
26. Abdolmaged Alkhulaifi, Fahad Alsahli, Irfan Ahmad (2021) Knowledge Distillation in Deep Learning and its Applications DOI: 10.7717/peerj-cs.474
27. Quantization [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/docs/optimum/concept_guides/quantization
28. Model Compression via Pruning [Електронний ресурс] — Режим доступу до ресурсу: https://towardsdatascience.com/model-compression-via-pruning-ac9b730a7c7b
29. Model Compression Techniques for Edge AI [Електронний ресурс] — Режим доступу до ресурсу: https://embeddedcomputing.com/technology/software-and-os/simulation-modeling-tools/model-compression-techniques-for-edge-ai
30. Datasets: Wikipedia [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/datasets/wikipedia/viewer/20220301.en/train
31. Datasets: Book Corpus [Електронний ресурс] — Режим доступу до ресурсу: https://huggingface.co/datasets/bookcorpus

32. Turc, I., Chang, M., Lee, K., & Toutanova, K. (2019). Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. arXiv: Computation and Language.

ДОДАТКИ

Додаток А

Програмний код основних модулів:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from transformers import Trainer, TrainingArguments, PreTrainedModel


class DistillationTrainingArguments(TrainingArguments):
    def __init__(self, *args, alpha=0.5, temperature=2.0, **kwargs):
        super().__init__(*args, **kwargs)

        self.alpha = alpha
        self.temperature = temperature


class DistillationTrainer(Trainer):
    def __init__(self, *args, teacher_model=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.teacher: PreTrainedModel = teacher_model  # type: ignore

        self._move_model_to_device(self.teacher,self.model.device)
        self.teacher.eval()

    def compute_loss(self, model, inputs, return_outputs=False):
        labels = inputs.get('labels')

        # compute student output
        outputs_student = model(**inputs, labels=labels)
        student_loss=outputs_student.loss
        # compute teacher output
        with torch.no_grad():
          outputs_teacher = self.teacher(**inputs)

        # Soften probabilities and compute distillation loss
        loss_function = nn.KLDivLoss(reduction="batchmean")
        loss_logits = (loss_function(
            F.log_softmax(outputs_student.logits / self.args.temperature, dim=-1),
            F.softmax(outputs_teacher.logits / self.args.temperature, dim=-1)) *
(self.args.temperature ** 2))
```

```
        # Return weighted student loss
        loss = self.args.alpha * student_loss + (1. - self.args.alpha) * loss_logits
        return (loss, outputs_student) if return_outputs else loss
from pathlib import Path

import boto3
from transformers import (
    TrainingArguments,
    TrainerCallback,
    TrainerControl,
    TrainerState
)


class S3UploadCallback(TrainerCallback):
    def __init__(self, bucket_name, model_prefix):
        self.bucket_name = bucket_name
        self.model_prefix = model_prefix
        self.s3 = boto3.client("s3")

    def on_save(
        self,
        args: TrainingArguments,
        state: TrainerState,
        control: TrainerControl,
        **kwargs
    ):
        if state.is_world_process_zero:
            model_dir = Path(args.output_dir)
            for file_path in model_dir.glob('*'):
                s3_key = f"{self.model_prefix}/{file_path.name}"
                self.s3.upload_file(str(file_path), self.bucket_name, s3_key)
```

```
def tokenize_and_process_dataset(dataset, teacher_tokenizer):
    def process(examples):
        tokenized_inputs = teacher_tokenizer(
            examples["sentence"], truncation=True, max_length=512
        )
        return tokenized_inputs

    tokenized_datasets = dataset.map(process, batched=True)
    tokenized_datasets = tokenized_datasets.rename_column("label", "labels")

    labels = tokenized_datasets["train"].features["labels"].names
    num_labels = len(labels)
    label2id, id2label = dict(), dict()

    for id, label in enumerate(labels):
        label2id[label] = str(id)
        id2label[str(id)] = label

    return tokenized_datasets, num_labels, label2id, id2label
```

```
from flask import Flask, render_template, request
from transformers import pipeline

app = Flask(__name__)

pipelines_cache = {}

@app.route('/', methods=['GET', 'POST'])
def home():
    teacher_predictions, student_predictions = [], []
    teacher_model = student_model = input_text = task = ''

    if request.method == 'POST':
        teacher_model = request.form['teacher_model']
        student_model = request.form['student_model']
        input_text = request.form['input_text']
        task = request.form['task']

        teacher_predictions = _predict(teacher_model, task, input_text)
        student_predictions = _predict(student_model, task, input_text)

    return render_template(
        'index.html',
        teacher_predictions=teacher_predictions,
        student_predictions=student_predictions,
        teacher_model=teacher_model,
        student_model=student_model,
        input_text=input_text,
        task=task,
    )


def _predict(model_path, task, input_text):
    if model_path in pipelines_cache:
        nlp = pipelines_cache[model_path]
    else:
        nlp = pipeline(task, model=model_path)
        pipelines_cache[model_path] = nlp

    return nlp(input_text)

if __name__ == "__main__":
    app.run(debug=True)
```

Додаток Б

Код файлу example.py:

```
from data.s3 import S3UploadCallback
from data.process import tokenize_and_process_dataset
from trainer import DistillationTrainer, DistillationTrainingArguments

import datasets
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification, DataCollatorWithPadding


student_id = "google/bert_uncased_L-2_H-128_A-2"
teacher_id = "textattack/bert-base-uncased-SST-2"
repo_name = "tiny-bert-sst2-distilled"
dataset_id="glue"
dataset_config="sst2"

teacher_tokenizer = AutoTokenizer.from_pretrained(teacher_id)
student_tokenizer = AutoTokenizer.from_pretrained(student_id)

dataset = datasets.load_dataset(dataset_id, dataset_config)
tokenized_datasets, num_labels, label2id, id2label = tokenize_and_process_dataset(dataset, teacher_tokenizer)

training_args = DistillationTrainingArguments(
    output_dir=repo_name,
    num_train_epochs=7,
    per_device_train_batch_size=128,
    per_device_eval_batch_size=128,
    learning_rate=6e-5,
    seed=33,
    # logging & evaluation strategies
    logging_dir=f"{repo_name}/logs",
    logging_strategy="epoch",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    report_to="tensorboard",
    # distilation parameters
```

```
    alpha=0.5,
    temperature=4.0,
    use_mps_device=True,
)


data_collator = DataCollatorWithPadding(tokenizer=teacher_tokenizer)

teacher_model = AutoModelForSequenceClassification.from_pretrained(
    teacher_id,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id,
)
student_model = AutoModelForSequenceClassification.from_pretrained(
    student_id,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id,
)

trainer = DistillationTrainer(
    student_model,
    training_args,
    teacher_model=teacher_model,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=teacher_tokenizer,
    callbacks=[S3UploadCallback('somebucket', 'somekey')]
)

trainer.train()
```

Додаток В

Код юніт тестів:

```
import unittest
from unittest import mock

import torch

from src.distill_trainer.trainer import DistillationTrainer, DistillationTrainingArguments as Args


class TestTrainer(unittest.TestCase):

    def setUp(self):
        self.student_model = mock.MagicMock()
        self.teacher_model = mock.MagicMock()
        self.tokenizer = mock.MagicMock()
        dummy_logits = torch.tensor([[0.2, 0.8], [0.1, 0.9]])
            self.student_model.return_value = mock.MagicMock(logits=dummy_logits, loss=torch.tensor(1.0))
        self.teacher_model.return_value = mock.MagicMock(logits=dummy_logits)
        self.args = Args(
            output_dir='./',
            temperature=1.5,
            alpha=0.5
        )
        self.inputs = self.tokenizer("This is a test sentence.", return_tensors="pt")

    def test_compute_loss(self):
        loss = DistillationTrainer(
            model=self.student_model,
            teacher_model=self.teacher_model,
            args=self.args,
        ).compute_loss(self.student_model, self.inputs)

        self.assertIsNotNone(loss)
        self.assertTrue(isinstance(loss, torch.Tensor))
        self.assertTrue(loss.dim() == 0)
        self.assertFalse(loss.requires_grad)


if __name__ == "__main__":
    unittest.main()
```

```
import os
import unittest
from pathlib import Path

import boto3
from moto import mock_s3

from transformers import (
    BertConfig,
    BertForSequenceClassification,
    TrainingArguments,
    Trainer
)

from data.s3 import S3UploadCallback

class TestS3UploadCallback(unittest.TestCase):

    @mock_s3
    def test_on_save(self):
        bucket_name = "my-test-bucket"
        model_prefix = "test_model"
        s3 = boto3.client("s3")
        s3.create_bucket(Bucket=bucket_name)

        output_dir = "test_output"
        Path(output_dir).mkdir(parents=True, exist_ok=True)

        config = BertConfig(num_labels=2)
        model = BertForSequenceClassification(config)
        args = TrainingArguments(output_dir=output_dir)
        trainer = Trainer(
            model=model,
            args=args,
            callbacks=[
                                        S3UploadCallback(bucket_name=bucket_name,
model_prefix=model_prefix)
            ]
        )
        trainer.callback_handler.on_save(trainer.args, trainer.state, trainer.control)

        uploaded_files = s3.list_objects(Bucket=bucket_name).get("Contents", [])
        expected_files = list(Path(output_dir).glob('*'))
```

```
            uploaded_files_set = set([os.path.basename(obj["Key"]) for obj in
uploaded_files])
    expected_files_set = set([f.name for f in expected_files])

    self.assertSetEqual(uploaded_files_set, expected_files_set)

    for f in expected_files:
       f.unlink()
    Path(output_dir).rmdir()
```

```
import unittest

from transformers import TrainingArguments

from src.distill_trainer.trainer import DistillationTrainingArguments


class TestDistillationTrainingArguments(unittest.TestCase):

    def test_init(self):
        args = DistillationTrainingArguments(
            output_dir="./test_output",
            overwrite_output_dir=True,
            alpha=0.7,
            temperature=3.0,
        )

        self.assertIsInstance(args, TrainingArguments)

        self.assertEqual(args.alpha, 0.7)
        self.assertEqual(args.temperature, 3.0)

        default_args = DistillationTrainingArguments(output_dir="./test_output")

        self.assertEqual(default_args.alpha, 0.5)
        self.assertEqual(default_args.temperature, 2.0)

if __name__ == '__main__':
    unittest.main()
```