Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інженерії програмного забезпечення

Сиром'ятніков Микита Валерійович,
студент групи АС-172

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Програмна система для прототипування, передбачення та аналізу якості багатомовної мовної моделі на базі багатозадачного підходу

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Рувінська Вікторія Михайлівна
канд. техн. наук, професор

Одеса – 2022

ЗМІСТ

Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут комп'ютерних систем

Кафедра інженерії програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Комлева Н. О.

«___» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сиром'ятнікова Микити Валерійовича, група АС-172

1. Тема роботи: Програмна система для прототипування, передбачення та аналізу якості багатомовної мовної моделі на базі багатозадачного підходу

Керівник роботи: Рувінська Вікторія Михайлівна, канд. техн. наук, професор

затверджені наказом ректора від «_20_» _____жовтень_____ 2022 р. № _399-в_

2. Зміст роботи: аналіз проблеми, формування технічного завдання на розробку, огляд та вибір методів вирішення задачі, проектування, програмна реалізація та тестування програмної системи, випробування програмної системи

3. Перелік ілюстративного матеріалу: згідно зі слайдами презентації

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Дата видачі завдання: « 29 » ____серпня____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз предметної області	08.09.2022	Виконано
2	Формування технічного завдання на розробку	15.09.2022	Виконано
4	Дослідження існуючих рішень	22.09.2022	Виконано
5	Проектування архітектури системи	03.10.2022	Виконано
6	Розробка компонентів системи	05.11.2022	Виконано
7	Тестування компонентів системи	10.11.2022	Виконано
8	Випробування програмної системи	15.11.2022	Виконано
9	Оформлення записки	25.11.2022	Виконано

Здобувач вищої освіти _____ М. В. Сиром’ятніков

Керівник роботи _____ В. М. Рувінська

АНОТАЦІЯ

Робота присвячена розробці програмної системи для прототипування, тренування та аналізу якості багатомовної мовної моделі на базі багатозадачного підходу. Метою роботи є збільшення точності та зменшення часу тренування мовних моделей за рахунок удосконалення методу багатозадачного тренування. В результаті шляхом поєднання задач маскованого мовного моделювання, передбачення наступної послідовності та контрастного навчання вдосконалено метод багатозадачного навчання. На цій основі створено повноцінну програмну екосистему, що містить сервіси моделювання, розмітки текстових даних, а також оповіщення щодо прогресу у тренуванні і розмітці. У роботі використано мову програмування Python, веб-фреймворк Flask, сховища даних MongoDB та AWS S3, фреймворк глибинного навчання Pytorch та сервіс перекладу GCP Cloud Translation.

Ключові слова: обробка природної мови, мовна модель, багатозадачне навчання, глибинне навчання, багатомовність.

ABSTRACT

The work is devoted to developing a software system for prototyping, training, and evaluating a multilingual language model based on a multi-task approach. The work aims to increase the accuracy and reduce the training time of language models. As a result, the method of multi-task learning was improved by combining the tasks of masked language modeling, next sequence prediction, and contrasting learning. On this basis, a complete software ecosystem was created, including modeling services, text data annotation, and training notifications. Python, Flask web framework, MongoDB and AWS S3 storages, Pytorch deep learning framework, and GCP Cloud Translation service were used for the development phase.

Keywords: natural language processing, language model, multi-task learning, deep learning, multilingualism.

ВСТУП

Експоненційний ріст обчислювальних потужностей та кількості текстового контенту за останні 10 років стали основними рушійними силами «революції» у галузі обробки природної мови (NLP): було опубліковано багато фундаментальних алгоритмів [1] та архітектур на базі глибинного навчання, а також проведено значну ітеративну роботу над покращенням точності і узагальнюючих здібностей до рівня людини у багатьох задачах [2]. Однак, проблема низькоресурсних (з недостатньою кількістю тренувальних даних) мов [3] не втратила свою актуальність: для більшості мов Східної Європи так і не були побудовані загальнодоступні рішення основних NLP-задач. У цього є дві основні причини:

- Відсутність економічної доцільності: значна частина open-source рішень у галузі публікується багатомільярдними корпораціями, що мають достатні прибутки [4] для інвестування у розробку високоточних NLP-моделей. В той же час, у Східній Європі подібні компанії поки що на стадії розвитку.
- Недостатня кількість якісних тренувальних даних: для реалізації моделей із задовільним рівнем узагальнюючої здібності потребуються текстові корпуси з десятками та сотнями мільйонів текстів і в той час, коли для англійської чи китайської мов проблем з цим завданням не виникає завдяки значній кількості різноманітного текстового контенту [5], для української мови, наприклад, - це надскладна задача для переважної більшості організацій.

Одним із найважливіших аспектів розквіту галузі NLP став розвиток напряму мовного моделювання (language modeling). І хоча цей напрям і не можна назвати новим, адже розробки алгоритмів велися і у минулому сторіччі [6] і на початку нового тисячоліття [7], за останні кілька років попередньо навчені мовні моделі набули величезної популярності у сфері обробки природної мови завдяки значному покращенню розуміння, генерації тексту та великій здатності до узагальнення. Поява багатомовних моделей мови покращила становище для низькоресурсних задач та мов: завдяки зменшенню необхідної кількості даних для тренування провести тонке налаштування готового рішення на конкретній задачі стало значно

дешевше ніж тренувати модель повністю з нуля [8]. Отже, попередньо треновані багатомовні моделі здатні вирішити проблему недостатньої кількості даних, але високу точність при цьому вдається досягти здебільшого лише на простих задачах, таких як класифікація тексту, в той же час, у комплексних задачах, що потребують повноцінного розуміння контексту послідовності, таких як відповіді на запитання та розпізнавання іменованих сутностей, не вдається отримати задовільні результати [9]. Для досягнення високого рівня узагальнюючої здібності моделей мови на низькоресурсних задачах все ж необхідно проводити тренування з нуля, однак це породжує вже відому проблему – необхідність довготривалої процедури навчання, а отже і відповідний за розміром набір даних. Саме тому метою роботи є збільшення точності та зменшення часу тренування мовних моделей за допомогою удосконаленого методу багатозадачного навчання на основі задач маскованого мовного моделювання, передбачення наступної послідовності і контрастного навчання та побудови повноцінної програмної системи для прототипування, тренування та оцінювання багатомовних моделей мови.

Для реалізації кваліфікаційного проекту було поставлено та вирішено наступні завдання:

1. Аналіз існуючих рішень у галузі NLP для роботи з моделями глибинного навчання, а також інструментів для багатозадачного навчання мовних моделей.
2. Розробка технічного завдання на побудову програмної системи для прототипування, тренування та аналізу якості моделей мови.
3. Дослідження існуючих архітектур моделей мови, а також задач для їх тренування, включаючи багатозадачні підходи.
4. Реалізація вдосконаленого методу багатозадачного навчання багатомовних моделей мови з покращенням точності та зменшенням часу тренування.
5. Специфікація вимог та проектування архітектури розроблюваної системи.
6. Програмна реалізація усіх компонентів продукту відповідно до сформованого технічного завдання.
7. Тестування продукту, проведення випробувань і оцінювання метрик та показників роботи реалізованої програмної системи, визначених у меті роботи.

Під час виконання кваліфікаційної роботи було отримано нові наукові результати: вдосконалений метод багатозадачного навчання мовних моделей, який, на відміну від існуючих, ґрунтується на синтезі задач маскованого мовного моделювання, передбачення наступного речення та контрастного навчання, що дозволяє зменшити необхідну кількість тренувальних ітерацій без погіршення у точності для моделей низькоресурсних мов на базі архітектури Transformer при роботі з багатомовними наборами даних.

Структурно роботу поділено на шість частин. У першому розділі наведено результати аналізу предметних областей систем побудови NLP-моделей та відкритих програмних рішень для багатозадачного навчання. Другий розділ містить деталізоване технічне завдання на розробку програмної системи. У третьому розділі розглядаються основні існуючі підходи до мовного моделювання, їх недоліки та переваги, а також пропонується вдосконалений метод багатозадачного навчання моделей мови. В четвертому розділі наведено специфікацію вимог до програмного продукту, а також деталізований проект архітектури розробки. П’ятий розділ містить опис інструментів та технік, що були використані для реалізації системи, інструкції для інсталяції компонентів системи та подальшої роботи з ними, а також сценарії тестування продукту. У останньому, шостому, розділі описано підготовку даних та специфікацію експериментів для оцінювання розроблених рішень, також отримані результуючі показники були інтерпретовані з наданням їм відповідних оцінок.

Матеріали кваліфікаційної роботи, присвячені процедурам тренування багатомовних моделей мови, були використані у тезах “Increasing the Performance of the Multilingual Language Model with FNet Architecture” [10], що були представлені на конференції “Сучасні інформаційні технології 2022” 19-20 травня 2022 року. Практичні аспекти тонкого налаштування попередньо натренованої моделі мови, а також варіації її архітектури для конкретних задач зі сфери обробки природної мови наведено у тезах “Natural Language Processing for Intelligent Virtual Assistant System” [11], що були представлені на конференції “Інформаційні управляючі системи та технології” 23-25 вересня 2021 року.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз рішень для роботи з NLP-моделями

1.1.1 Основні поняття

Моделювання – процес, що узагальнює процедури прототипування, тренування та оцінювання моделей машинного навчання [12].

Попередньо навчена модель – модель машинного навчання, що була натренована на великому наборі даних та має достатню узагальнюючу здібність для проведення процедур її дотренування під конкретну задачу [13].

Узагальнююча здібність – здатність моделі вирішувати раніше небачені завдання без донавчання або після незначного тонкого налаштування [14].

Тонке налаштування або тонка настройка – пристосування попередньо навченої моделі машинного навчання під вирішення нової задачі шляхом дотренування її на незначному (у порівнянні з набором попереднього тренування) наборі даних [15].

Розмітка даних – процедура, в якій анотатор анотує (розмічує або проставляє експертну оцінку) надані йому нерозмічені приклади відповідно до узгодженого технічного завдання з метою формування набору даних для тренування або тестування алгоритмів навчання з учителем (у кожного прикладу з наборів даних для тренування є правильна мітка) [16]. Прикладом задачі на розмітку даних за напрямком комп’ютерного зору може бути визначення об’єкту на зображенні для побудови тренувального набору даних під задачу класифікації. У сфері обробки природної мови аналогічним прикладом може бути визначення тональності текстового відгуку за встановленою експертною шкалою. Ще однією можливою задачею розмітки текстових даних є визначення подібних пар речень, що може знадобитися при формуванні наборів даних для задачі ранжування.

Функціонал оповіщення – здатність програмної системи інформувати користувача про прогрес виконання запущених ним процедур за допомогою будь-якого зручного інтерфейсу взаємодії (браузерні сповіщення, пошта, корпоративний месенджер) [17]. У сфері машинного навчання такими процедурами здебільшого є

операції, що потребують значних часових витрат: тренування моделі або будь-яка трансформація великого набору даних.

Багатомовність рішень – здатність програмної розробки підтримувати дві або більше мови одночасно [18]. При цьому рішення не реалізується окремо для кожної мови, а є спільним.

Підтримка мови – наявність готових наборів даних та попередньо навчених моделей машинного навчання для основних задач обробки, розуміння та генерації тексту у вказаній мові [3].

1.1.2 Огляд наявних інструментів моделювання у галузі NLP

В ході аналізу предметної області було виділено такі існуючі рішення для моделювання у галузі обробки природної мови, що мають значну аудиторію та активно підтримуються розробниками:

1. Transformers – високорівневий фреймворк для машинного навчання та основний компонент екосистеми Hugging Face [19]. Надає API та інструменти для завантаження та тренування сучасних моделей на базі архітектури Transformer. Фреймворк дозволяє будувати рішення за напрямками комп’ютерного зору (класифікація зображень, виявлення об'єктів і сегментація), обробки аудіо (автоматичне розпізнавання мови), суміжними сферами (оптичне розпізнавання символів, відеокласифікація та візуальні відповіді на запитання), однак основним напрямом розвитку залишається обробка природної мови (класифікація тексту, розпізнавання іменованих сутностей, відповіді на запитання, мовне моделювання, переклад і генерація тексту). До переваг Transformers можна віднести велику кількість реалізованих архітектур, доступ до значного набору датасетів завдяки інтеграції з іншим компонентом екосистеми Hugging Face – datasets, а також відносно просту конвертацію моделей між основними бібліотеками глибинного навчання (PyTorch, Tensorflow, JAX). Недоліками є низький рівень гнучкості при побудові багатозадачних рішень, слабка підтримка української мови (датасети для тестування мовних моделей доступні здебільшого для англійської мови) та відсутність функціоналу для розмітки даних.

2. Flair – NLP-фреймворк, що підтримується Берлінським університетом імені Гумбольдтів та Zalando Research. Розробка дозволяє прототипувати та тренувати моделі для задач розмітки послідовності (ідентифікація іменованих сутностей, визначення частин мови) та текстової класифікації. Фреймворк побудовано на базі бібліотеки глибинного навчання Pytorch, що дозволяє комбінувати високорівневі об’єкти Flair з низькорівневим функціоналом тренування Pytorch для формування гнучких конфігурацій експериментів [20]. За допомогою розширення flair-lms фреймворк також може завантажувати попередньо натреновані мовні моделі, але вони наявні здебільшого для мов Західної Європи.

3. AllenNLP – побудована на PyTorch дослідницька бібліотека для розробки найсучасніших моделей глибокого навчання з широким спектром лінгвістичних завдань, що підтримується організацією AI2 (Allen Institute for AI) [21]. Бібліотека дозволяє прототипувати, тренувати та оцінювати моделі обробки природної мови на основі сучасних архітектур зі сфери глибинного навчання за допомогою зручного конфігураційного API, в тому числі підтримуються і рішення на базі мовних моделей. Також екосистема AI2 містить репозиторій з попередньо підготовленими моделями і фреймворки для використання рішень у production-середовищі та оптимізації гіперпараметрів – усі ці компоненти можна використовувати і у AllenNLP як зовнішні динамічні плагіни, що розширюють функціональні можливості бібліотеки.

4. SpaCy – це бібліотека для розширеної обробки природної мови. Вона поставляється з попередньо підготовленими рішеннями та наразі підтримує понад 60 мов, в тому числі і українську [22]. SpaCy містить також production-ready систему навчання та спрощену модель пакування, розгортання та управління робочим процесом. Основним недоліком утиліти є недостатня гнучкість при роботі з багатозадачними моделями та слабка підтримка напрямку мовного моделювання.

Результати порівняння аналогів з програмною системою, що розробляється у рамках виконання кваліфікаційної роботи (у таблиці під назвою MTFormer, або Multi-Task transFormer), наведені у таблиці 1.1.

Таблиця 1.1 – Порівняння розробки з інструментами моделювання

	MTFormer	Transformers	Flair	Allen NLP	SpaCy
Гнучка конфігурація	Так	Так	Так	Так	Ні
Функціонал розмітки даних	Так	Ні	Ні	Ні	Ні
Функціонал оповіщення	Так	Ні	Ні	Ні	Ні
Набір попередньо навчених моделей	Так	Так	Частково	Так	Так
Багатомовність рішень	Так	Так	Частково	Ні	Так
Підтримка мовних моделей	Так	Так	Частково	Так	Частково
Підтримка української мови	Так	Частково	Ні	Ні	Частково

З порівняльної таблиці можна побачити, що часткова підтримка української мови забезпечується лише інструментами моделювання з багатомовністю рішень та імплементованими мовними моделями (Transformers та SpaCy). Це, в свою чергу, дозволяє на практиці підтвердити коректність вибору технології багатомовних моделей мови у якості основного рішення проблеми реалізації моделей розуміння природної мови для низькоресурсних мов, до яких, безумовно, входить і українська.

1.2 Аналіз рішень для багатозадачного тренування моделей мови

1.2.1 Основні поняття

Багатозадачне тренування – підхід до тренування алгоритмів машинного навчання, при якому модель одночасно навчається не на одній задачі, а одразу на декількох [23]. Одним з прикладів такого підходу з галузі обробки природної мови

є спільне тренування моделі на задачі текстової класифікації для визначення тональності повідомлення та розмітки послідовності для визначення ключових словосполучень. Теоретично, перевагами багатозадачного тренування є зниження ризику перенавчення моделі та покращення її узагальнюючої здібності завдяки тому, що модель фокусується на декількох різних рівнях представлення вхідної послідовності. Однак, на практиці усе залежить від якості наборів даних, стратегій їх міксування та сумісності обраних тренувальних задач.

Діагностичний набір даних – наявний у програмній системі підготовлений датасет, що дозволяє оцінити якість натренованих моделей та порівняти їх на конкретній задачі. Одним з прикладів з галузі NLP є тест GLUE [24] з наборами даних для оцінки систем розуміння природної мови.

1.2.2 Огляд інструментів для багатозадачного тренування моделей мови

В ході аналізу предметної області було виділено такі існуючі рішення для багатозадачного тренування з підтримкою моделей мови:

1. M3TL - невелика бібліотека для багатозадачного навчання на основі фреймворку Transformers, імплементує задачі класифікації, маскованого мовного моделювання та регресії [25]. Її перевагами є можливість гнучкої конфігурації моделі, до недоліків же можна віднести відсутність готових моделей та діагностичного набору даних, а також засобу для зручного маніпулювання даними.

2. PALM (PArallel Learning from Multi-tasks) — це швидка, гнучка та розширювана система багатозадачного навчання у сфері обробки природної мови, що підтримується китайським пошуковиком Baidu [26]. Високорівневий фреймворк реалізує зручний доступ до попередньо навчених моделей та поширених архітектур на базі Transformer. В той же час, тестові датасети для оцінювання якості моделей наявні лише для китайської мови, а допоміжні інфраструктурні компоненти поширені здебільшого у китайському інтранеті та не мають значну підтримку ззовні.

3. Multi-task-NLP – це інструмент, який дозволяє розробникам легко будувати та тренувати моделі глибинного навчання на декількох завданнях обробки

природної мови одночасно [27]. Розробка підтримує більшість задач розуміння тексту та реалізує основні архітектури глибинних нейронних мереж з допомогою фреймворку Transformers. Гнучкість конфігурації підтримується завдяки можливості налаштувань тренувальних завдань у YAML-форматі. До недоліків Multi-task-NLP можна віднести відсутність діагностичних датасетів для оцінювання підготовлених рішень та непристосованість до багатомовних наборів даних.

Результати порівняння аналогів з програмною системою, що розробляється у рамках кваліфікаційної роботи (у таблиці під назвою MTFormer, або Multi-Task transFormer), наведені у таблиці 1.2.

Таблиця 1.2 – Порівняння розробки з рішеннями для багатозадачного тренування

	MTFormer	M3TL	PALM	Multi-task-NLP
Гнучка конфігурація	Так	Частково	Так	Так
Набір готових моделей	Так	Ні	Так	Так
Діагностичні набори даних	Так	Ні	Частково	Ні
Багатомовність рішень	Так	Частково	Частково	Ні
Підтримка української мови	Так	Ні	Ні	Ні

З результатів порівняння можна побачити, що на даний момент жоден інструмент для багатозадачного тренування не задовольняє ключової потреби – підтримка низькоресурсної української мови, а це є прямим наслідком слабкого рівня багатомовності розглянутих систем. Отже, аналіз продемонстрував актуальність створення програмної системи для прототипування, тренування та оцінювання багатомовних моделей мови з удосконаленим методом багатозадачного тренування.

1.3 Висновки до розділу

Огляд існуючих програмних систем для побудови NLP-моделей та інструментів для багатозадачного навчання моделей мови дозволив виділити такі важливі аспекти: наявність репозиторію готових моделей та діагностичних датасетів для порівняння реалізованих рішень, можливість гнучкої конфігурації архітектур нейромереж та процесу їх тренування чи оцінювання, наявність функціоналу для розмітки тренувальних даних і оповіщення про прогрес запущених операцій та підтримка роботи з низькоресурсними мовами. Порівняння програмних систем за цими аспектами в свою чергу підкреслило відсутність рішень, що б мали повноцінну підтримку української мови та власний функціонал для розмітки даних і оповіщення користувача щодо перебігу процедур тренування моделей.

2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ

2.1 Мета проведення роботи

Підставою для розробки програмного продукту є завдання на дипломне проектування. Тема кваліфікаційної роботи – “Програмна система для прототипування, передбачення та аналізу якості багатомовної мовної моделі на базі багатозадачного підходу”. Метою роботи є збільшення точності та зменшення часу тренування мовних моделей шляхом реалізації програмної системи для прототипування, тренування та оцінювання моделей мови разом з удосконаленим методом багатозадачного навчання.

2.2 Призначення розробки

Задачею розробки є створення програмного продукту, що реалізує функціонал прототипування, тренування та оцінювання багатомовних моделей мови, а також надає можливість проводити процедури розмітки текстових даних і отримувати інформацію щодо перебігу поточних процедур через зручні для користувача інтерфейси (месенджери).

Компонент моделювання інсталюється на локальні обчислювальні пристрої користувачів з операційними системами Windows або Linux у якості програмного пакету-розширення для мови програмування Python, а з сервісами розмітки текстових даних та оповіщення, розміщеними на серверному обладнані продукту, взаємодія відбувається за допомогою http-запитів.

Для виконання процедур тренування та оцінювання моделей прикладного програмного інтерфейсу сервісу моделювання користувачу спочатку необхідно сформувати конфігурацію, що представляється у вигляді JSON-файла або Python-словника, або ж скористатися однією з наявних у локальному середовищі, програмному пакеті чи хмарному репозиторії. Гнучкість конфігурації сервісу на практиці трансформується у можливість для кейсів тренування та оцінювання встановлювати усі можливі кардинальності відношень між моделями та наборами

даних: один-до-одного (одна модель та один набір), один-до-багатьох, багато-до-одного та багато-до-багатьох. Вибір моделей та тренувальних задач можна здійснювати з фіксованого набору реалізованих архітектур, проте сервіс моделювання надає можливість комбінувати конфігурації для розширення функціональних можливостей та побудови комплексних рішень.

Перед запуском тренування моделі користувачу також надається можливість встановити налаштування оповіщення, що дозволить отримувати інформацію про завершення процедур валідацій або тренувальних епох через інтерфейс сервісу сповіщень у месенджері Telegram. Аналогічний інтерфейс сервісу розмітки текстових даних користувач може використовувати для формування тренувального або тестового набору даних завдяки налагодженій взаємодії між сервісами моделювання та розмітки.

Для імплементації методу багатозадачного тренування багатомовних моделей мови необхідно спочатку провести аналіз існуючих підходів до багатозадачного навчання та попереднього навчання мовних моделей, а далі вже розробити власне рішення з урахуванням специфіки багатомовності.

2.3 Вимоги до програми

Реалізований у ході виконання кваліфікаційної роботи програмний продукт має відповідати наступним функціональним вимогам:

- надає можливість проводити процедуру тренування моделі машинного навчання з опцією інформування користувача щодо проміжних результатів;
- дозволяє завантажувати модель машинного навчання за наданою конфігурацією;
- надає можливість завантажувати дані та проводити їх обробку відповідно до вказаних налаштувань;
- дозволяє проводити оцінювання та порівняння підготовлених моделей на вказаних наборах даних;

- надає можливість підготувати модель для подальшого використання у production-режимі;
- дозволяє запускати процедури розмітки текстових даних та використовувати отримані дані для тренування чи тестування моделей машинного навчання.

Також програмна розробка має відповідати наступним нефункціональним вимогам:

- збій одного компоненту системи не призводить до збоїв у інших;
- на кожному етапі роботи з даними та тренування моделей відбувається формування контрольних точок, з яких можна відновити роботу у випадку збою;
- час побудови моделі за конфігурацією залежить від обчислювальної платформи, але не перевищує 20 секунд;
- час підготовки одного тренувального пакету залежить від обчислювальної платформи, але не перевищує 0.5 секунд;
- взаємодія з Telegram Bot API відбувається за допомогою техніки Long Polling.

2.4 Вимоги до програмної документації

Документація до розробленого у ході виконання кваліфікаційної роботи програмного продукту має містити наступні артефакти:

- специфікація функціональних вимог до системи (діаграма варіантів використання, таблиця пріоритетності, деталізований опис кожної функціональної вимоги, таблиця з деталізацією нефункціональних вимог);
- опис реалізованого багатозадачного методу тренування мовних моделей;
- опис архітектури системи (високорівнева схема взаємодії компонентів, діаграма класів кожного компоненту, ER-діаграма сховища даних);
- інструкція з інсталяції та використання.

2.5 Технічно-економічні показники

Моделі мови після попереднього тренування з використанням реалізованої програмної системи завдяки задовільній узагальнюючій здібності та спрощеній процедурі тонкої настройки можна використовувати у широкому колі застосунків, що потребують впровадження рішень зі сфери обробки природної мови, до них відносяться пошукові та рекомендаційні сервіси, чат-боти та віртуальні асистенти з текстовими і голосовими інтерфейсами взаємодії, системи автоматичного реферування та машинного перекладу, помічники з правопису та рішення для генерації програмного коду. Окрім того, підсистему розмітки даних можна використовувати у якості незалежного сервісу для побудови датасетів для великої кількості задач розуміння тексту.

2.6 Технічні вимоги

Реалізований у ході виконання кваліфікаційної роботи програмний продукт має відповідати наступним технічним вимогам:

• усі компоненти програмної системи мають підтримуватися операційною системою Windows не нижче 7-ї версії та Linux-дистрибутивом Ubuntu не нижче версії 16.04;

• процедура тренування моделей машинного навчання має підтримувати паралелізацію для рівномірного використання декількох обчислювальних пристроїв.

2.7 Стадії та етапи розробки програмної системи

Стадії, основні етапи, їх зміст та терміни розробки програмної системи представлено у таблиці 2.1.

Таблиця 2.1 — Стадії та етапи розробки програмної системи

Стадія	Етап	Зміст робіт	Строки
Огляд існуючих рішень	Аналіз існуючих інструментів моделювання у галузі NLP.	Дослідження функціоналу та особливостей реалізації інструментів для побудови, тренування та NLP-моделей.	29.08.22 – 04.09.22
	Аналіз рішень для багатозадачного навчання мовних моделей у галузі NLP	Дослідження функціоналу та особливостей реалізації рішень для багатозадачного тренування моделей мови.	04.09.22 – 09.09.22
Технічне завдання	Розробка документу специфікації вимог до програмної системи.	Побудова документу SRS з таких артефактів як нефункціональні вимоги до програмної системи та варіанти використання з деталізацією основних потоків подій та їх розширень.	09.09.22 – 15.09.22
Технічний проект	Реалізація методу для багатозадачного навчання моделей мови	Проектування тренувальних задач та їх поєднання для імплементації методу багатозадачного тренування моделей мови	30.08.22 – 15.10.22
	Проектування сервісу моделювання	Побудова діаграми класів сервісу для прототипування, тренування та тестування моделей мови з детальним описом усіх компонентів.	15.09.22 – 24.09.22
	Проектування сервісу оповіщення	Побудова діаграми класів сервісу оповіщення з детальним описом усіх компонентів.	24.09.22 – 27.09.22
	Проектування сервісу розмітки даних	Побудова діаграми класів сервісу розмітки даних та ER-діаграми сховища даних з детальним описом усіх компонентів.	27.09.22 – 03.10.22
Робочий проект	Реалізація програмної системи	Реалізація функціоналу системи, перевірка роботи варіантів використання та формування інструкцій, проведення тестування.	03.10.22 – 10.11.22

Продовження таблиці 2.1

Стадія	Етап	Зміст робіт	Строки
Готовий проект	Випробування системи	Підготовка даних, побудова та тренування моделей, проведення експериментів і аналіз результатів.	10.11.22 – 15.11.22

В результаті за допомогою таблиці було представлено результати декомпозиції процесу розробки програмної системи на стадії та основні етапи з їх детальним описом та зазначенням термінів виконання.

2.8 Висновки до розділу

У даному розділі було сформовано технічне завдання на розробку програмної системи, деталізовано вимоги до продукту, його призначення, описано вимоги до супроводжуючої документації, а також план роботи поділено на стадії та розплановано кожний етап його виконання.

3 РОЗРОБКА ВДОСКОНАЛЕНОГО МЕТОДУ БАГАТОЗАДАЧНОГО ТРЕНУВАННЯ МОДЕЛЕЙ МОВИ

3.1 Мовне моделювання в NLP

У загальному випадку, задача моделі мови полягає у визначенні ймовірності появи символів, слів чи n-грам (послідовності n елементів) враховуючи попередні елементи послідовності. Перемноження усіх ймовірностей елементів сформує ймовірність появи в мові усієї послідовності, а отже, можна представити модель мови у вигляді розподілу ймовірностей за послідовностями слів чи n-грам [28].

На даний момент сфера застосування моделей мови включає у себе більшість існуючих задач галузі розуміння природної мови (NLU) та її генерації (NLG). І якщо ще 10 років тому рішення на базі language modeling застосовували переважно як складову частину системи автоматичного розпізнавання мови або ж у якості компонента інтелектуальних клавіатур з функціоналом автовиправлення помилок і пропозицією можливих відповідей, то зараз попередньо навчені моделі застосовують для задач класифікації, пошуку, генерації, сумаризації та для витягнення важливих сутностей з послідовності (рис. 3.1).

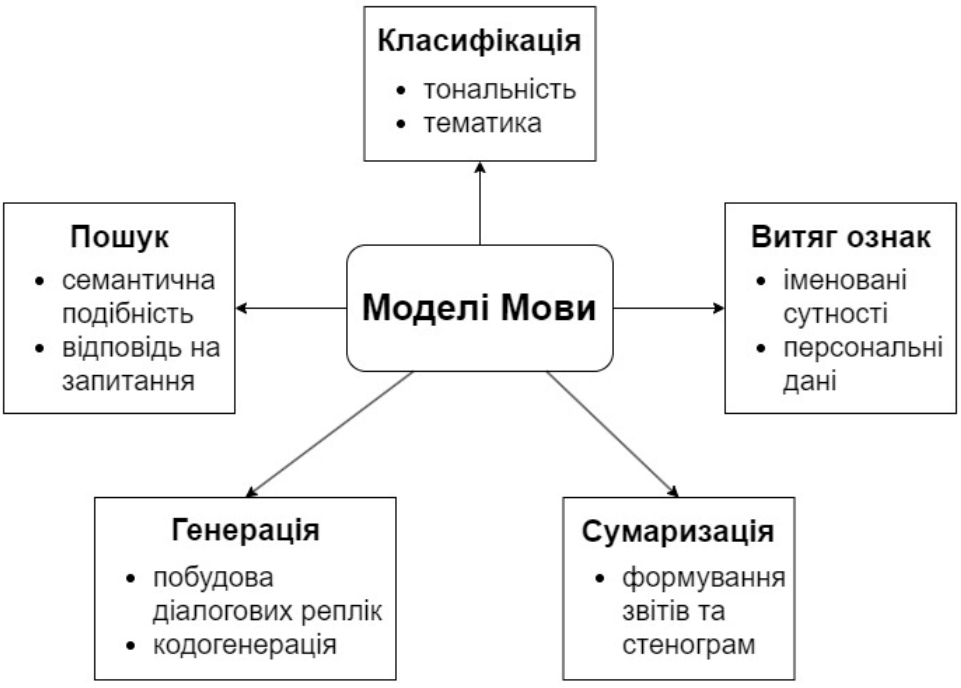

Рисунок 3.1 – Практичне застосування мовного моделювання

В задачах класифікації існує два основних варіанти використання моделей мови: трансформація текстових послідовностей у контекстні ознаки з їх використанням у інших алгоритмах, а також генеративний підхід на базі few-shot навчання [29]. Поширеними прикладами класифікації є визначення тональності (емоцій) для повідомлення, а також класифікація текстів новин за темами. Активно застосовуються моделі мови і у пошукових задачах, наприклад, Google використовує їх для контекстного ранжування [30] і для функціоналу відповіді на запитання. Найважливішим напрямом мовного моделювання є генерація тексту: в останні роки генеративні рішення все частіше застосовуються у віртуальних асистентах, окрім того, розвивається кейс кодогенерації [31]. Сумаризація – один із варіантів генерації, при якому генерується скорочена версія вхідної текстової послідовності (формування звітів, стенограм та переказів). Ще одним варіантом використання моделей мови є витягнення ознак, таких як іменовані сутності (імена, назви, числа та дати) або персональні дані.

3.2 Аналіз підходів до моделювання мови

Загалом виділяють такі підходи до моделювання мови:

- статистичний
- нейромережевий

Статистичний підхід базується на оцінюванні розподілу ймовірностей послідовності слів шляхом моделювання ймовірності наступного слова з урахуванням попередніх:

$$P(w_0, \dots, w_N) = P(w_0) \prod_{i=1}^{N} P(w_i | w_0, \dots, w_{i-1}) \tag{3.1}$$

де w_i - окремі індекси слів у словнику.

В той же час, для n-грамних моделей (поширений варіант статистичного підходу, де оцінюється ймовірність n слів чи символів замість одного елемента) при n > 1 (біграмна та триграмна моделі) існує проблема: у послідовностях великої довжини деякі елементи могли не зустрічатися раніше в одному контексті і відповідно обчислення ймовірності для усієї послідовності може бути неможливим. Рішенням цього може бути обчислення наближеної ймовірності використовуючи марковську властивість n-го порядку [32]: ймовірність появи і-го елемента залежить лише від контексту з попередніх n – 1 елементів. Тоді формула для обчислення наближеної ймовірності має наступний вигляд:

$$P(w_1, \dots, w_N) = \prod_{i=1}^{N} P(w_i | w_1, \dots, w_{i-1}) \approx \prod_{i=1}^{N} P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) \qquad (3.2)$$

Окрім того, в n-грамних мовних моделях для вирішення проблеми відсутності n-грам у словнику використовуються методи на основі згладжування, що призначує небаченим елементам якусь частку загальної маси ймовірності. Прикладами таких методів є модель відступання (використати меншу розмірність для невідомого контексту, біграми для триграм чи уніграми для біграм) та лінійна інтерполяція (зважене міксування ймовірностей уніграм, біграм та триграм).

Будучи доволі простим інструментом та “прозорою коробкою”, статистичні мовні моделі мають і недоліки: кількість параметрів експоненційно зростає зі збільшенням порядку n-грам, а також статистичні рішення не мають узагальнюючої здібності. Ці проблеми вирішує нейромережевий підхід до мовного моделювання завдяки проеціюванню токенів (символів чи слів) у безперервний простір, в якому токени зі схожим контекстом мають подібні представлення.

Першу нейронну модель мови, побудовану на основі нейромережі прямого поширення, було представлено в 2001 році [33]. Запропонований Йошуа Бенжіо підхід (рис. 3.2) полягав у тому, що для кожного слова зі словника формувався вектор Сі і використовувалася функція спільної ймовірності послідовності слів у вигляді векторів. Ця функція ітеративно змінює параметри збільшення

логарифмічної правдоподібності даних і одночасно проводить навчання векторних представлень і параметрів функції ймовірності.

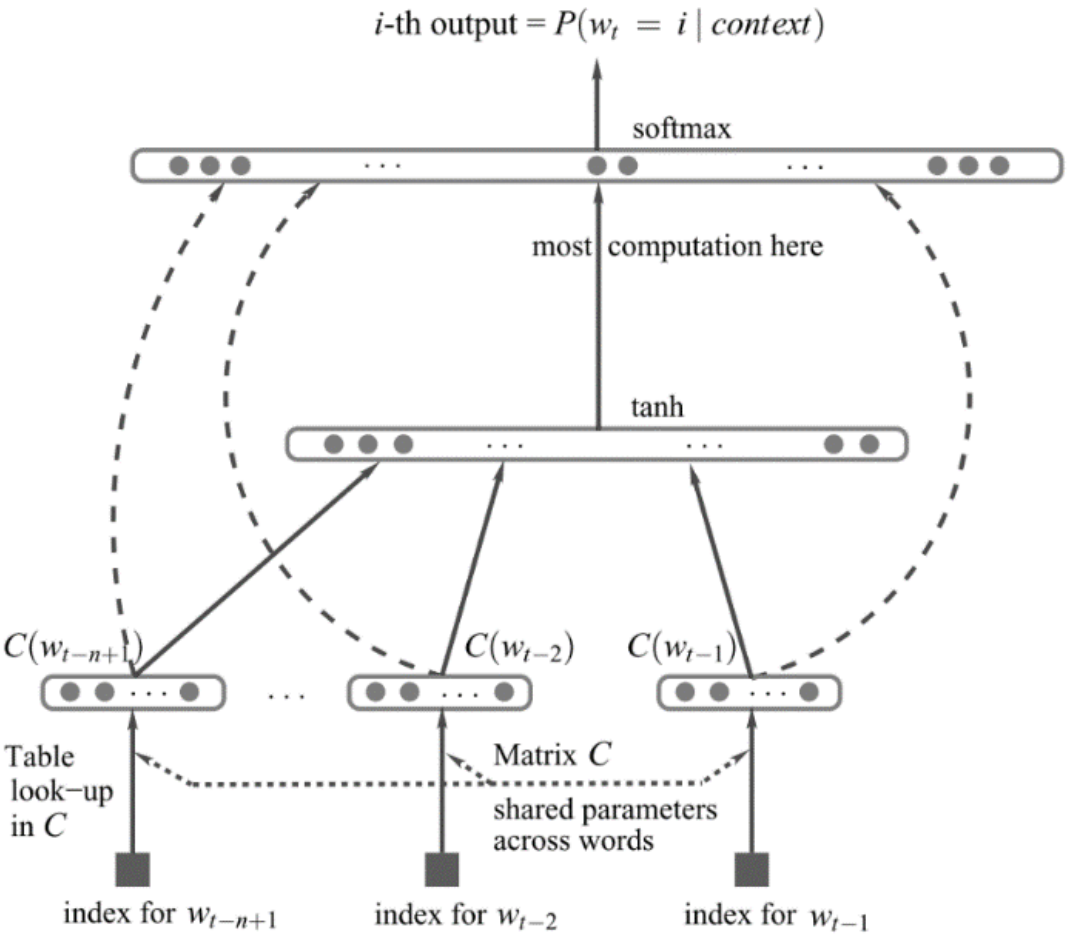

Рисунок 3.2 – Нейронна ймовірнісна модель мови. Джерело: «A Neural Probabilistic Language Model», NIPS-2000, proceedings.neurips.cc//paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

Після цього вектори слів об'єднуються і передаються в прихований шар з гіперболічним тангенсом у якості функції активації, вихід якого потім передається в softmax-шар для формування розподілу ймовірностей за словами зі словника.

Теоретично, багатошаровий персептрон (мережа прямого поширення) здатний апроксимувати будь-яку функцію з бажаною точністю, однак, на практиці це не завжди можливо. Найбільші проблеми виникають при роботі з даними, що підпорядковуються законам часу (текст, звук чи часові ряди): багатошарові персептрони потребують фіксованої довжини послідовності, тоді як текстовий ввід

може містити як декілька слів, так і цілий параграф, також мережі прямого поширення не здатні зберігати у пам’яті довготривалі залежності, що досить важливо для задач розуміння природної мови. Зазначені проблеми можуть бути вирішені рекурентними нейронними мережами.

Рекурентна нейронна мережа (RNN) – архітектура, що реалізує концепцію “пам’яті”: для кожного елемента послідовності виконуються одні й ті ж обчислення, але при цьому враховується інформація з попередніх часових кроків за допомогою імплементації спеціального прихованого стану, що передається між часовими кроками.

$$h_t = \tanh(W_{hh}h_{t-1} + W_{ih}x_t) \tag{3.3}$$

де W_{hh} та W_{ih} - ваги для прихованого стану та введеної інформації,

h_{t-1} – прихований стан з попереднього часового кроку,

x_t – введена інформація на поточному часовому кроці.

RNN теоретично можуть зберігати у пам’яті інформацію про послідовності довільної довжини, але на практиці розмір контексту обмежується декількома часовими кроками. Це зумовлено проблемами вибухаючого (exploding gradient) чи згасаючого градієнту (vanishing gradient): у рекурентній мережі довжини n буде проведено перемноження похідних n разів і якщо похідні більші за 1, то градієнт буде експоненційно зростати та сильно впливати на ваги мережі, у іншому випадку отримаємо градієнт близький до 0 і мережа не буде змінюватися.

Проблему вибухаючого та згасаючого градієнта вирішує архітектура LSTM (Long Short-Term Memory), що базується на рекурентній мережі, але додатково реалізує компоненти для підтримки довготривалої пам’яті: до прихованого стану додаються ще стан ячейки та три “гейти” або вентилі для роботи з ним (видалення, занесення та виведення інформації). Це дозволяє тримати у пам’яті інформацію протягом тривалого часу, що особливо важливо для формування коректного контексту усієї послідовності. І хоча архітектура LSTM була опублікована у 1997 році, вона і досі залишається потужним інструментом для вирішення багатьох

задач у галузі NLP. Так, наприклад, модель мови ELMo, що дозволяє будувати двонаправлений контекст, складається з декількох шарів комірок довгої короткочасної пам'яті [34]. Окрім того, існує ще одна архітектура на базі рекурентної нейронної мережі - вентильний рекурентний вузол (Gated Recurrent Unit). GRU містить менше параметрів через відсутність вихідного вентиля, що дозволяє використовувати архітектуру на пристроях з обмеженою кількістю оперативної пам’яті, при цьому ефективність рішення у задачах акустичного моделювання та розпізнавання мови близька до LSTM.

Недоліком архітектури RNN є фіксований розмір входу та виходу, тобто довжина результуючої послідовності повинна співпадати з довжиною вхідної послідовності. В той же час, у деяких задачах, таких як переклад, різниця між довжинами послідовностей у різних мовах може бути суттєвою. Окрім того, в рекурентних мережах результат поточного кроку формується з урахуванням лише попередніх станів, а іноді для формування результату потрібно спочатку отримати увесь контекст вхідної послідовності.

В 2014 році Google опублікувала архітектуру Sequence-to-Sequence, що вирішує вищезазначені проблеми RNN, завдяки чому значно покращується точність у задачах машинного перекладу та сумаризації тексту [35]. Seq2seq модель складається з кодувальника (Encoder), що трансформує вхідну послідовність у контекстний вектор фіксованого розміру, а також декодера (Decoder), що використовує цей вектор для генерації результуючої послідовності. При цьому у якості компонентів кодувальника та декодера використовуються варіації рекурентних комірок (LSTM або GRU). Підхід до моделювання мови у варіаціях Encoder-Decoder близький до концепцій, що були розглянуті вище, але у той час, як стандартні моделі мови обчислюють безумовну ймовірність послідовності $p(y)$, ця архітектура будує умовну ймовірність $p(y|x)$, де x – вхідна послідовність. На практиці така умовна модель мови працює наступним чином: кодувальник формує контекстний вектор з вхідної послідовності, а декодер на кожному часовому кроці отримує на вхід цей вектор і вихід попереднього часового кроку, який формується

за допомогою подачі виходу декодера у шар softmax для формування розподілу ймовірності за усіма токенами словника і вибору найбільш ймовірного токена.

Sequence-to-Sequence демонструє високу точність при роботі з послідовностями малої та середньої довжини (до 20 слів), але при роботі з довшими послідовностями точність значно падає через те, що контекстний вектор може зберігати лише обмежену кількість інформації та сама інформація має різну ступінь важливості в залежності від часового кроку декодера [36]. Обійти обмеження фіксованого розміру контекстного вектора кодувальника дозволяє спеціальний механізм уваги, що має досить просту інтерпретацію: на кожному кроці декодування модель фокусується на різних складових вхідної послідовності. На кожному кроці декодера механізм уваги вирішує, які вихідні частини важливіші, при цьому кодувальнику не потрібно стискати усю вхідну послідовність в єдиний контекстний вектор, він формує представлення для всіх вхідних токенів [37].

Першим кроком побудови представлення є обчислення показників уваги, формула 3.4 описує метод багатошарового персептрона з оригінальної статті [37], але також можна використовувати і скалярний добуток чи білінійну функцію:

$$score(h_t, s_k) = W_2^T * \tanh(W_1[h_t, s_k]) \tag{3.4}$$

де h_t – стан декодера на часовому кроці t,

s_k – стан кодувальника на вхідному токені k,

W_1, W_2 – ваги першого та другого шару персептрона.

Після обчислення показників уваги для кожного вхідного токена необхідно побудувати розподіл впливу вхідних токенів на стан декодера t:

$$a_k^t = \frac{\exp\big(score(h_t, s_k)\big)}{\sum_{i=1}^{m} \exp\left(score(h_t, s_i)\right)} \tag{3.5}$$

де a_k^t – ваги уваги для кожного вхідного токена k на часовому кроці декодера t,

m – довжина вхідної послідовності.

Вихід механізму уваги для стану декодера на часовому кроці t обчислюється як зважена сума вхідних станів кодувальника:

$$c^t = a_1^t s_1 + a_2^t s_2 + \cdots + a_m^t s_m = \sum_{k=1}^{m} a_k^t s_k \tag{3.6}$$

де c^t – вихід механізму уваги для стану декодера на часовому кроці t.

Реалізація механізму уваги стала дуже важливим аспектом розвитку галузі обробки природної мови, однак справжня революція відбулася дещо пізніше, у 2017 році, коли Google Brain презентувала архітектуру Transformer. Після початку активного використання Encoder-Decoder архітектур з механізмом уваги залишалася одна основна проблема: кількість даних та обчислювальні можливості швидко зростали, але здебільшого компонентами мереж були рекурентні блоки, що погано масштабуються та непридатні до паралельних обчислень, це призводило до непропорційно великого часу тренування рішень. Ці проблеми були вирішені у Transformer завдяки значній оптимізації та розпаралелюванню архітектури Encoder-Decoder: на відміну від рекурентних нейронних мереж, мережеві блоки Transformer не є послідовними, що дозволяє обробляти всю вхідну послідовність одночасно [38]. При цьому передача інформації між станами кодувальника відбувається за рахунок удосконаленого механізму уваги під назвою самоувага (self-attention), в той же час декодер використовує стандартний механізм уваги. Механізм самоуваги можна описати так: кожний токен за допомогою механізма уваги “запитує” інші токени в послідовності, збирає контекст та оновлює своє представлення. Математично це виражається наступним чином:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.7}$$

де Q або $query$ – вектор токена, що “питає” інші токени,

K або key – вектор токена, який запитується query,

V або $value$ – зазвичай той же вектор, що і k,

d – розмір вектора k.

У декодері самоувага дещо відрізняється від наведеної вище. В той час як кодувальник опрацьовує всі токени одночасно, у декодері генерується один токен за один часовий крок і тому під час генерації декодер не повинен мати інформацію про наступні токени. Щоб не “заглядати у майбутнє”, використовується маскований механізм самоуваги: перед softmax-активацією необхідно виконати поелементне множення результату на матрицю одиниць, верхній трикутник якої містить максимальні від’ємні числа. Результатом цієї операції стане нівелювання впливу наступних токенів, так як їх коефіцієнти будуть наближені до нуля.

Кожний токен може бути учасником декількох типів відносин, наприклад, синтаксичних та лексичних. Для виявлення цих відносин в архітектурі Transformer використовується концепція багатоголової уваги (multi-head attention): замість одного механізму уваги багатоголова увага має декілька “голів”, які працюють незалежно та проводять обчислення паралельно. Реалізується механізм за допомогою розділення запитів, ключей та значень на декілька проекцій розміру $\frac{\text{розмір вектора ключа}}{\text{кількість голів}}$, після цього обчислюється увага для кожної голови окремо та результати конкатенуються. Таким чином, багатоголова увага не збільшує розмір моделі.

$$MultiHead(Q,K,V) = Concat\big(Attention\big(QW_Q^i, KW_K^i, VW_V^i\big), \dots\big)W_o \qquad (3.8)$$

де W_Q^i – вектор проекції токенів запиту,

W_K^i – вектор проекції токенів ключа,

W_V^i – вектор проекції токенів значення,

W_0 – вектор проекції у вихідну розмірність,

$Attention$ – функція обчислення уваги для кожної голови,

$Concat$ – операція горизонтальної конкатенації.

Детальна схема оригінальної архітектури Transformer продемонстрована на рисунку 3.3.

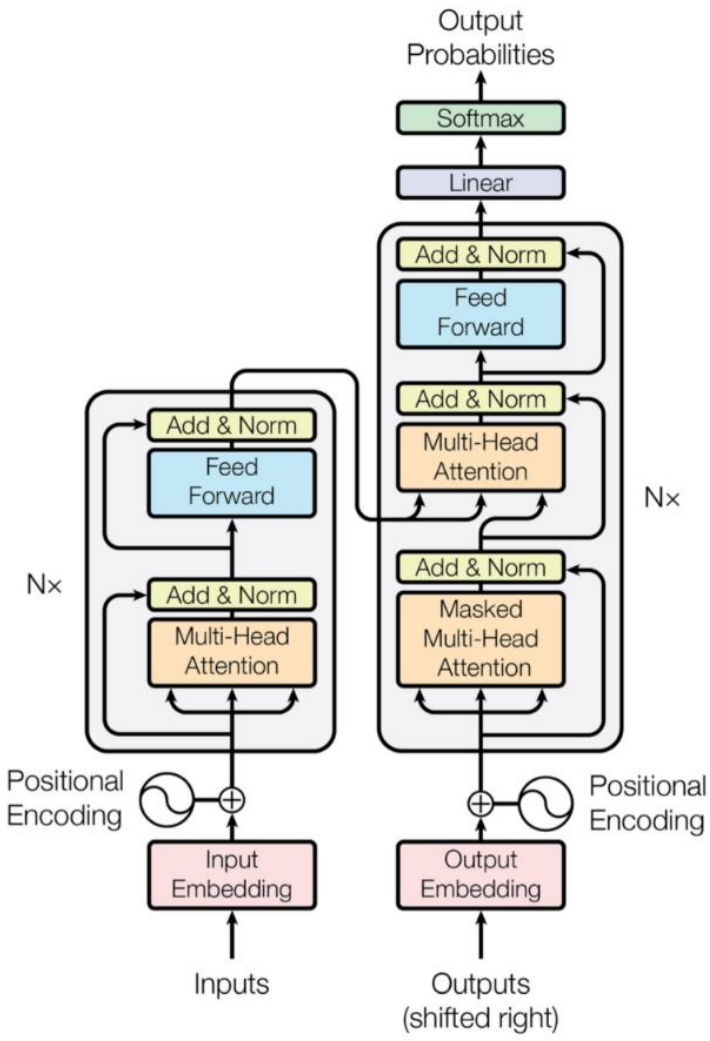

Рисунок 3.3 – Архітектура Transformer. Джерело: «Attention Is All You Need», NIPS-2017, arxiv.org/pdf/1706.03762.pdf

Одним із перших і найпопулярніших варіантів Transformer є архітектура BERT (двоспрямовані кодувальні представлення з трансформерів), опублікована в 2018 році. Ця мовна модель, що містить лише шари кодувальника, розроблена для попереднього багатозадачного навчання без учителя глибоких двонаправлених представлень із тексту шляхом спільного використання лівостороннього та правостороннього контексту на всіх рівнях [39]. Після публікації попередньо навчена модель BERT з різними вихідними шарами тонкого налаштування продемонструвала найкращі результати точності для одинадцяти завдань розуміння природної мови.

Також існують реалізації архітектури Transformer, що складаються тільки з шарів декодера, найпопулярнішим прикладом такої структури є модель

генеративного попередньо навченого трансформера або GPT. На відміну від BERT, вона авторегресивна, тобто для генерації поточного стану використовуються лише результати попередніх часових кроків. На даний момент вийшло 3 версії моделі, остання з яких (GPT-3) містить 175 мільярдів параметрів та завдяки своїй узагальнюючій здібності може бути використана для вирішення будь-якого завдання [40]. Однак, частіше за все генеративні трансформери у галузі обробки природної мови використовують для вирішення задач генерації статей, відповіді на запитання, побудови діалогових систем та сумаризації.

T5 (Трансформер для трансферу тексту в текст) — ще одна модель мови, але вже на базі Encoder-Decoder. що підходить до усіх задач обробки природної мови у форматі трансформації тексту в текст [41]. Алгоритм попереднього навчання моделі полягає у синтезі тренування на величезному текстовому корпусі без учителя та багатозадачного формату з наявними мітками. T5 “з коробки” добре працює з різними завданнями, необхідно лише додавати спеціальний префікс до вводу, що відповідає кожному завданню, наприклад, для перекладу “translate English to German”, для сумаризації – “summarize …”.

Великі мовні моделі демонструють раніше небачені результати точності в більшості завдань обробки природної мови, однак вони здебільшого містять сотні мільйонів (BERT), мільярди (mT5, GPT) або навіть трильйони (Switch Transformer) параметрів. Це означає, що для навчання або використання таких моделей потрібна величезна кількість обчислювальних ресурсів. Окрім того, будучи ключовим елементом архітектури Transformer, механізм самоуваги, однак, має квадратичну обчислювальну складність $O(N^2)$, що суттєво впливає на швидкість обробки довгих послідовностей.

Одним із можливих рішень для зменшення розміру моделі без значної втрати якості є дистиляція знань. DistilBERT, дистильована версія BERT, показує, що можна зменшити розмір моделі BERT на 40%, зберігаючи 97% її можливостей розуміння мови та на 60% швидше [42]. При цьому варто зазначити, що для дистиляції потребується спочатку провести процедуру тренування моделі-вчителя, а це не завжди може бути виправданим з точки зору фінансових витрат. Також

проблему обчислювальної складності можна частково вирішити шляхом заміни підрівня уваги на більш простий, такий як лінійний шар (MLP-Mixer) [43] або перетворення Фур'є (FNet) [44].

FNet — це варіація Transformer, у якій ми замінюємо підрівень самоуваги кожного шару кодувальника на підрівень Фур'є, який застосовує двовимірне дискретне перетворення Фур'є (DFT) до свого входу кодування – один 1D DFT уздовж довжини послідовності і один 1D DFT вздовж прихованого виміру [44]. Модель FNet поступається BERT на 8% у тесті General Language Understanding Evaluation (GLUE), тоді як навчання відбувається в 1,8 раза швидше на GPU.

Також існують підходи на основі гіпотези, що для обчислення уваги для кожного токена немає сенсу розглядати усі інші елементи послідовності, а можна розділити послідовність на декілька незалежних блоків за допомогою локально-чутливого хешування і обчислювати показники самоуваги у цих блоках незалежно (Reformer) [45] або для кожного токена враховувати лише сусідні та спеціальні токени послідовності (LongFormer) [46].

Загалом же можна простежити, що у напрямку моделювання мови за останні 20 років відбувся поступовий перехід зі статистичних рішень до монозадачних нейромережевих, а в останні 5 років – до попередньо навчених багатозадачних підходів, що чудово узагальнюються на усі існуючі задачі, архітектура Transformer при цьому стала де-факто стандартом для галузі обробки природної мови та багатьох інших, включаючи напрям комп'ютерного зору [47].

3.3 Огляд обраних для подальшого дослідження моделей та методів

3.3.1 Багатозадачне навчання моделі мови BERT

З огляду існуючих рішень для мовного моделювання можна зробити висновок, що більшість актуальних нейромережевих підходів, що демонструють високий рівень узагальнюючої здібності, ґрунтуються на підході багатозадачного навчання, коли мережа одночасно вчиться вирішувати декілька незалежних завдань. Одним із найяскравіших прикладів є алгоритм багатозадачного навчання для

попереднього тренування мережі BERT: модель одночасно вчиться на задачах передбачення наступної послідовності (NSP) та маскованого мовного моделювання (MLM).

3.3.2 Метод тренування моделі мови BERT на базі задачі передбачення наступної послідовності

В задачі NSP модель визначає, чи є друге речення продовженням першого у форматі бінарної класифікації (так чи ні). Візуалізація моделі передбачення наступного речення продемонстрована на рисунку 3.4. У реалізації BERT для позначення початку послідовності використовується спеціальний токен <CLS> (використовується для отримання результату класифікації), а для закінчення першого речення - <SEP>.

Рисунок 3.4 – Задача передбачення наступного речення

3.3.3 Метод тренування моделі мови BERT на основі задачі маскованого мовного моделювання

В задачі MLM випадковим чином маскуються (замінюються на спеціальний токен <MASK>) токени вхідної послідовності і модель намагається передбачити які реальні токени знаходяться за маскою. Приклад роботи маскованого мовного моделювання зображено на рисунку 3.5.

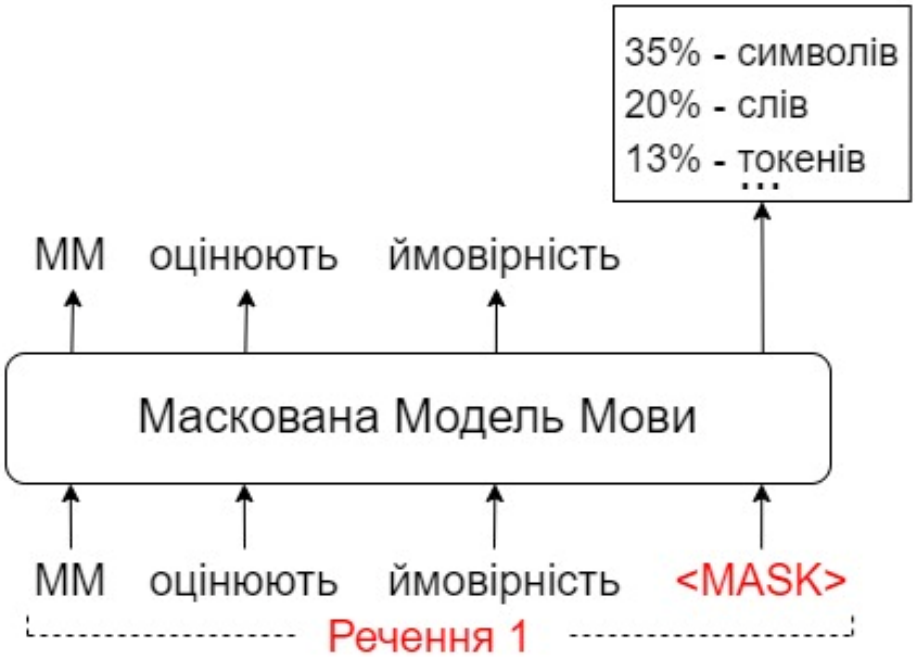

Рисунок 3.5 – Задача маскованого мовного моделювання

Попереднє навчання моделі на задачі MLM допомагає краще зрозуміти мову на рівні слів, а також спрощує подальше тренування у задачах розпізнавання іменованих сутностей та розмітки частин мови. У свою чергу задача NSP взаємодіє на рівні речень та спрощує тренування для вирішення задачі відповіді на запитання.

В той же час, проведені дослідження демонструють, що конструкція BERT робить його непридатним для пошуку семантичної подібності, а також для завдань навчання без учителя, таких як кластеризація [48]. У якості вирішення цієї проблеми автори моделі Sentence-BERT пропонують попереднє тренування на задачі контрастного навчання (Contrastive learning).

3.3.4 Контрастне навчання

Контрастне навчання – це варіація Similarity learning, мета якого полягає в тому, щоб вивчити функцію подібності, яка вимірює, наскільки схожі два об’єкти. Його особливістю є те, що для кожного тренувального об’єкта використовується не один об’єкт, а два: подібний та неподібний. Увесь процес навчання полягає у мінімізації відстані між оригіналом та подібним об’єктом і максимізації – між

оригіналом і неподібним. Схема моделі контрастного навчання представлена на рисунку 3.6.

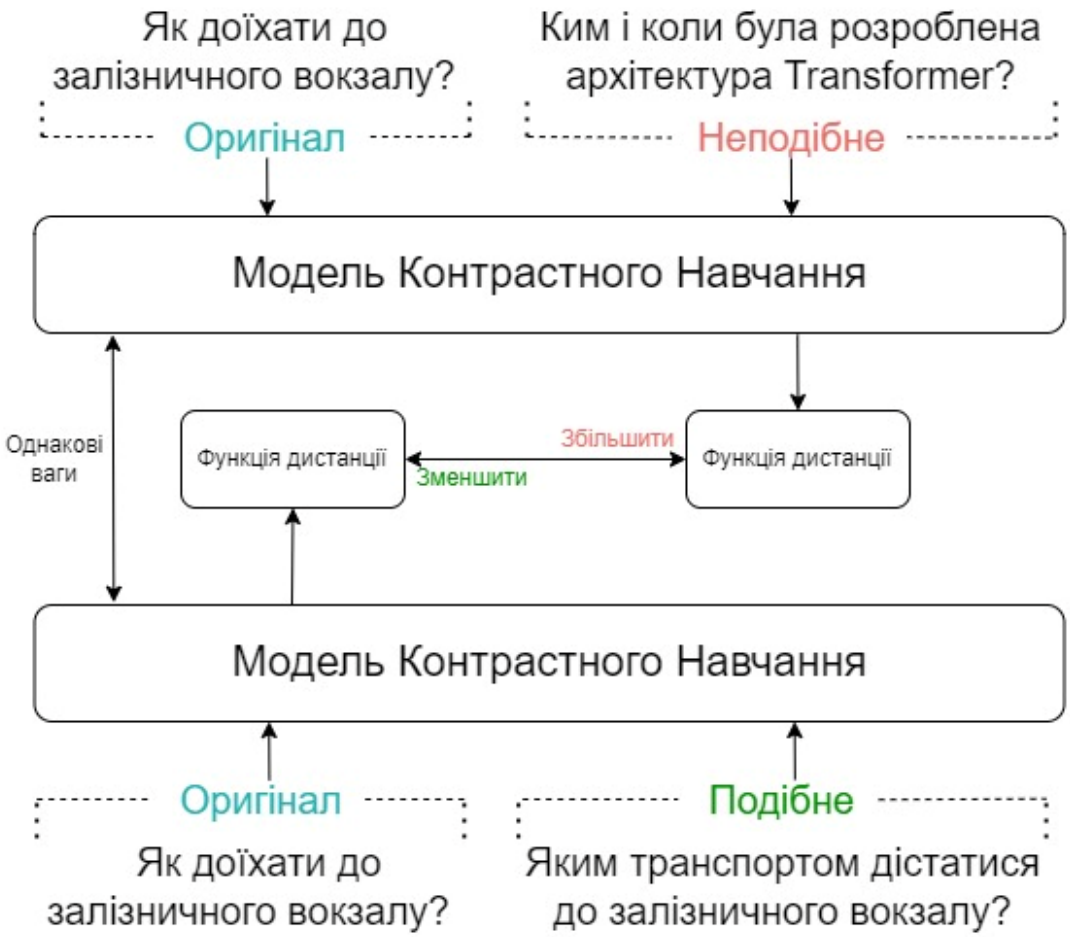

Рисунок 3.6 – Задача контрастного навчання

3.4 Вдосконалений метод багатозадачного тренування

У цій роботі пропонується покращений алгоритм багатозадачного тренування, що полягає у одночасному тренуванні моделі на задачах маскованого мовного моделювання, передбачення наступного речення та контрастного навчання. Багатозадачне тренування моделі реалізується шляхом встановлення по одній окремій тренувальній “голові” для кожної задачі після основних шарів моделі, що імплементують кодувальник з архітектури Transformer. Послідовні шари кодувальника трансформують вхідну послідовність та передають закодоване представлення у голови задач, кожна з яких формує незалежні передбачення.

Метод багатозадачного тренування моделі мови до та після вдосконалення зображено на рисунках 3.7-3.8 (внесені зміни виділено пунктирною рамкою).

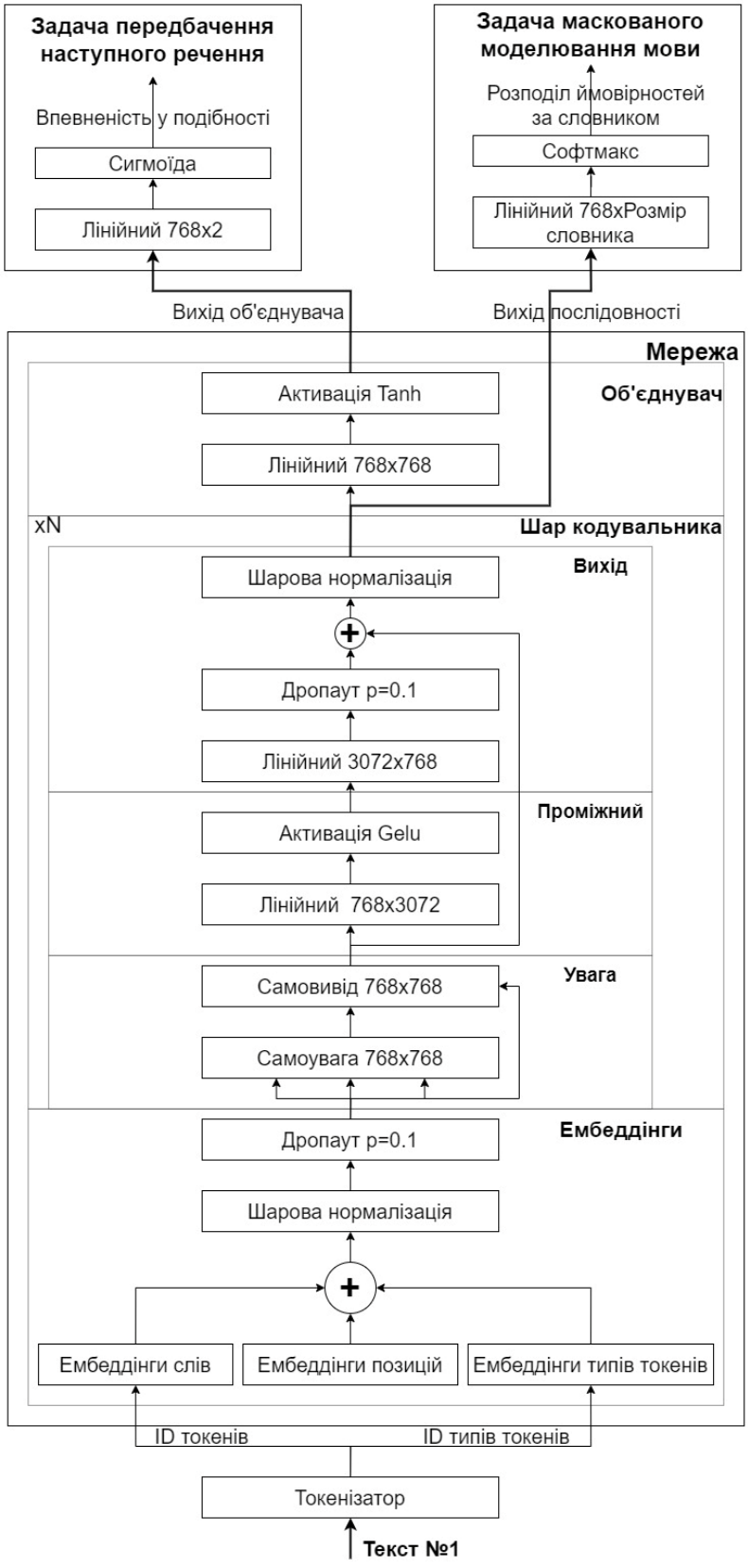

Рисунок 3.7 – Метод багатозадачного тренування моделей мови

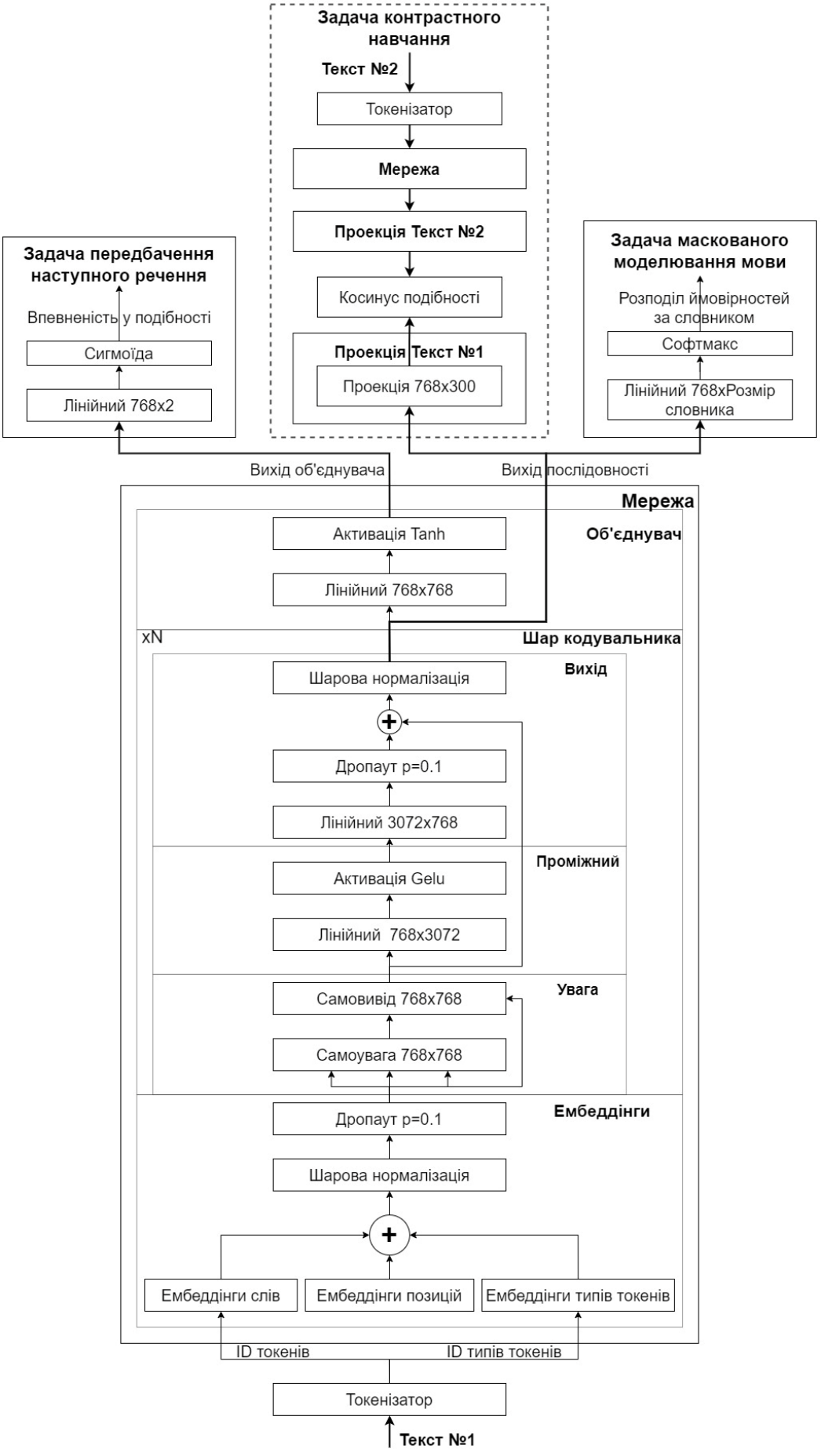

Рисунок 3.8 – Вдосконалений метод багатозадачного тренування моделей мови

Ембеддінг-шар використовує вихід токенізатора для створення представлення вхідної послідовності, що подається до шару кодувальника. Шар ембеддінгів слів використовується для побудови векторизованої форми вхідних токенів, а ембеддінги позицій вводять таку важливу властивість, як порядок токенів у послідовності [49]. Шар ембеддінгів типів токенів допомагає моделі зрозуміти, де закінчується одна послідовність і починається інша. Для цього використовується спеціальний токен <SEP>. Також для нормалізації активності нейронів і скорочення часу навчання використовується техніка шарової нормалізації [50]. Метод регуляризації “дропаут” [51] з імовірністю 0.1 деактивує приховані стани та використовується для зменшення перенавчання моделі.

Шар кодувальника містить підшар самоуваги, проміжний підшар з GELU-активацією для обробки інформації механізму уваги, а також вихідний – для компресії результатів та їх нормалізації. Кілька послідовних застосувань шару кодування з різними вагами дозволять моделі сформувати представлення сенсу, граматики чи інших аспектів вхідного вектора за допомогою інших елементів текстової послідовності. Кількість шарів кодування для цієї роботи була встановлена на рівні 12, розмір прихованого стану дорівнює 768, кожний шар самоуваги містить по 12 голів. Окрім того, з метою збільшення швидкості тренування у перших 10-ти шарах кодувальника замінили механізм самоуваги на дискретну трансформацію Фур’є з реалізації моделі FNet.

У Encoder-частинні моделі існує принаймні два можливих варіанти вибору виходу. Перший — це вихід послідовності, який формується у результаті прямого проходження через усі шари кодувальника кожного вхідного токена послідовності. Цей вихід під час навчання використовується для завдання маскованого мовного моделювання та для контрастного навчання. Другий — вихід об’єднувача з єдиним вектором для токена <CLS>, що є першим елементом послідовності. Другий вихід використовується для задачі визначення того, чи є друге речення продовженням першого. Окрім того, вихід об’єднувача також можна використовувати для різних завдань класифікації тексту.

Голова маскованого мовного моделювання представляє собою лінійну проекцію послідовного виходу кодувальника у простір розміром довжина послідовності на розмір словника та подальше застосування softmax-активації для формування розподілу ймовірностей для кожного токена. Голова контрастного навчання складається зі стискання прихованого стану послідовності у вектор розміром 300 елементів та функції дистанції між двома векторами. Голова для задачі передбачення наступного речення складається з лінійної проекції у 2 класи та sigmoid-активації для отримання ймовірнісної інтерпретації класифікації.

Для задачі маскованого мовного моделювання у якості функції втрат використовується варіація перехресної ентропії [52]:

$$L_{MLM} = L_{CE} = \frac{1}{N}\sum_{n}^{N} -log\frac{exp\left(\frac{x_{n,y_n}}{T}\right)}{\sum_{c}^{C}\exp\left(\frac{x_{n,c}}{T}\right)} * 1\{y_n \neq ignore\_index\} \quad (3.9)$$

де N – розмір пакету (кількість тренувальних елементів в одній ітерації),

C – кількість класів,

x - ненормовані оцінки для кожного класу (передбачення моделі),

y - індекси класів у діапазоні [0, C),

$ignore\_index$ – вказує цільове значення, що ігнорується при обчисленні,

T – температура, додатковий параметр, що використовується для контролю м’якості розподілу ймовірностей.

На вхід функції втрат подаються передбачення голови MLM для усіх токенів, однак з них замаскованими є тільки 15%. Для того, щоб не враховувати помилку передбачення на токенах без маски, використовується спеціальне значення *ignore_index*: елементи цільового списку токенів, що мають таке ж значення, ігноруються під час обчислення помилки. Для часткового вирівнювання розподілу ймовірностей токенів з метою покращення процедури багатомовного тренування (додаткове зашумлення робить завдання складнішим та запобігає перенавчанню мережі на одній з мов) у роботі використовуються параметр T у діапазоні 0.9 - 1.1.

Для задачі передбачення наступного речення у якості функції втрат використовується бінарний варіант перехресної ентропії:

$$L_{NSP} = L_{BCE} = \frac{1}{N}\sum_{n}^{N} y_n * logx_n + (1 - y_n) * \log(1 - x_n) \tag{3.10}$$

де N – розмір пакету,

x – ймовірність класу у діапазоні [0, 1],

y – цільове значення класу (0 або 1).

Для контрастного навчання використовується триплетна функція втрат, у якій еталонний вхід (оригінал) порівнюється з відповідним входом (подібним) і невідповідним (неподібним) [53]:

$$L_{CL} = L_{TL} = \frac{1}{N}\sum_{n}^{N} max(dist(A_n, P_n) - dist(A_n, N_n) + \mathrm{E}, 0) \tag{3.11}$$

де N – розмір пакету,

$dist$ – функція дистанції між двома послідовностями,

A – оригінальна послідовність,

P – подібна послідовність,

N – неподібна послідовність,

E – запас, мінімальне значення, на яке подібна послідовність має бути ближче до оригіналу ніж неподібна.

У якості функції дистанції було обрано косинус подібності – коефіцієнт подібності двох не нульових векторів у просторі, який обчислюється як косинус кута між ними [54]. Косинус подібності дорівнює 1, якщо два вектори мають однакий напрямок і 0, якщо вони утворюють кут 90°. Обчислюється косинус подібності двох векторів наступним чином:

$$dist(X_1, X_2) = 1 - cos\big(f(X_1), f(X_2)\big) = 1 - \frac{f(X_1) * f(X_2)}{\|f(X_1)\| * \|f(X_2)\|} \tag{3.12}$$

де X_1, X_2 – текстові послідовності,

f – функція трансформації текстової послідовності у вектор.

Після обчислення значення кожної функції втрат вони формують загальну похибку моделі шляхом зваженого додавання:

$$L = w_1 * L_{MLM} + w_2 * L_{NSP} + w_3 * L_{CL} \quad (3.13)$$

де w_1, w_2, w_3 – коефіцієнти впливу функцій втрат,

L_{MLM}, L_{NSP}, L_{CL} – функції втрат для маскованого моделювання мови, передбачення наступного речення та контрастного навчання.

Експериментальним шляхом було встановлено, що значення коефіцієнтів $w_1 = 0.4, w_2 = 0.2, w_3 = 0.4$ впливу функції втрат демонструють найшвидше тренування. Також варто зазначити, що встановлення великих коефіцієнтів одночасно для функцій втрат контрастного навчання та передбачення наступного речення призводить до погіршення результатів тренування через націленість цих задач на один і той же рівень контексту – зв'язок речень між собою.

Загалом реалізований в ході виконання кваліфікаційного проекту метод багатозадачного тренування моделі мови містить наступні етапи:

1. Випадковим чином на вхід токенізатору подається оригінальний текст або його друга половина (після <SEP>) замінюється на випадкову послідовність з багатомовного тренувального набору.

2. З ймовірністю 0.15 кожний токен вхідної послідовності маскується (замінюється спеціальним токеном <MASK>).

3. Маскована послідовність подається у модель та проходить через шари кодувальника.

4. Послідовний вихід кодувальника подається у голову маскованого мовного моделювання для формування передбачень, після чого обчислюється функція втрат задачі маскованого мовного моделювання.

5. Вихід об’єднувача подається у голову передбачення наступного речення для класифікації, далі обчислюється функція втрат задачі передбачення наступної послідовності.

6. Послідовний вихід кодувальника подається у голову контрастного навчання для формування вектору послідовності.

7. Для перекладеного тексту на іншу мову формується вектор послідовності. Випадковим чином обирається будь-який інший екземпляр з тренувального набору, для нього також будується вектор послідовності.

8. Обчислюються дистанції між оригінальним та подібним і неподібним текстами на різних мовах, а також функція втрат контрастного навчання за допомогою триплетної функції втрат з параметром E = 1.

9. Функції втрат формують загальну похибку передбачень шляхом зваженого додавання.

10. Модель виконує алгоритм зворотнього поширення помилки та оновлює ваги.

3.5 Висновки до розділу

У даному розділі описано розроблений метод багатозадачного навчання моделей мови, що представляє собою синтез існуючих алгоритмів маскованого моделювання мови, передбачення наступної послідовності та контрастного навчання. У якості архітектури моделі глибинного навчання було обрано гібридну версію FNet з використанням механізму самоуваги мережі BERT у останніх двох шарах кодувальника. Такий підхід дозволяє підтримувати баланс між швидкістю тренування та узагальнюючою здібністю моделі. Окрім того, з метою спрощення тренувального процесу на багатомовному наборі даних в обрані методи було внесено незначні модифікації: для обчислення загальної функції втрат використовується зважене додавання, а також у функцію втрат задачі маскованого моделювання мови додано параметр для вирівнювання розподілу ймовірностей токенів і зменшення перенавчання мережі на одній з мов.

4 ПРОЕКТУВАННЯ СИСТЕМИ

4.1 Опис функціональних вимог

Функціональні вимоги допомагають описати поведінку системи та зобразити основний функціонал, який має бути реалізований у програмній розробці. Для опису функціональних вимог скористаємося діаграмою варіантів використання (рис. 4.1) та деталізуємо їх сценаріями використання (табл. 4.1-4.8).

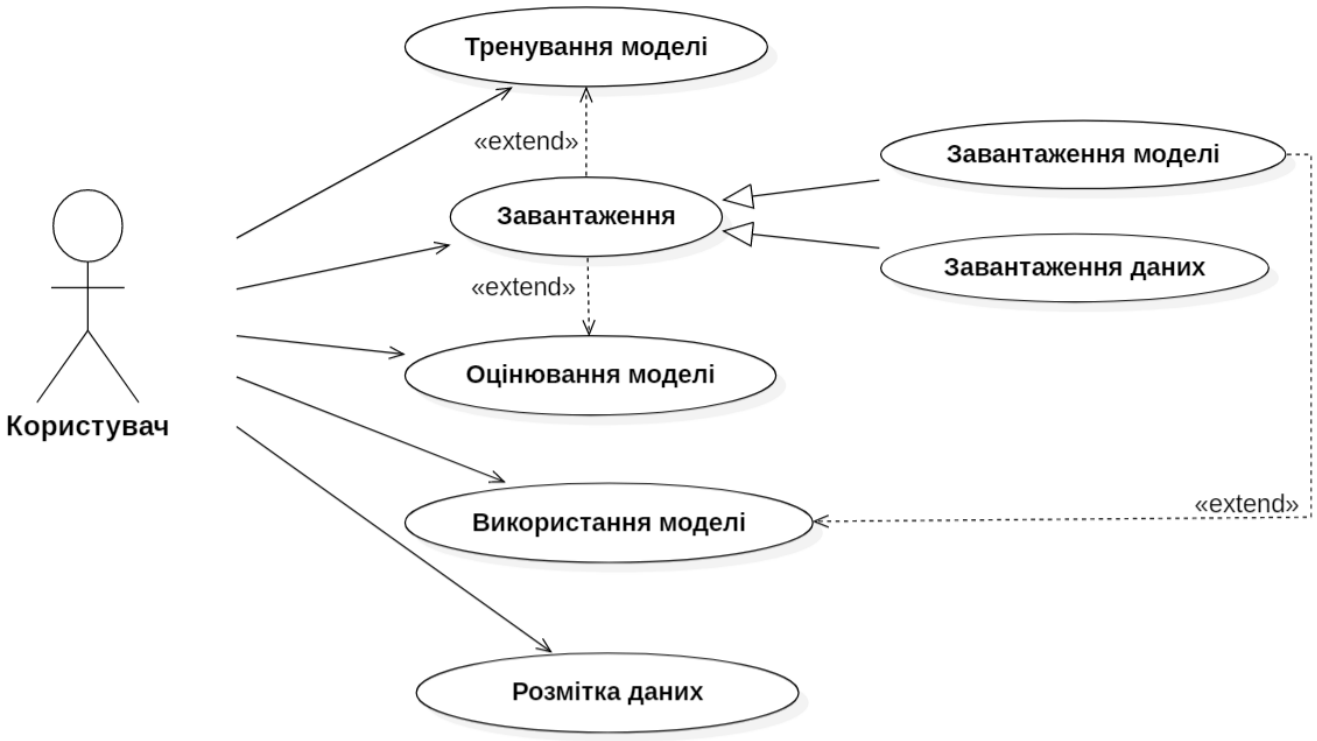

Рисунок 4.1 – Діаграма варіантів використання

Таблиця 4.1 – Варіанти використання

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог	
		Пріоритет	Складність
UC-01	Тренування моделі	Обов’язково	Висока
UC-02	Завантаження	Обов’язково	Середня
UC-03	Завантаження моделі	Обов’язково	Середня
UC-04	Завантаження даних	Обов’язково	Середня
UC-05	Оцінювання моделі	Бажано	Середня

Продовження таблиці 4.1

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог	
		Пріоритет	Складність
UC-06	Використання моделі	Бажано	Низька
UC-07	Розмітка даних	Рекомендовано	Висока

UC1 Тренування моделі та сценарій (табл. 4.2)

Система: підсистема моделювання.

Основна діюча особа: Користувач, Система.

Мета: виконати тренування моделі.

Тригер: користувач підготував тренувальний набір, визначився з конфігурацією моделі та викликає тренувальний метод з API підсистеми моделювання.

Результат: об’єкт моделі після тренування повертається користувачу, збережені ваги моделі, конфігурація та відомості щодо перебігу процедури.

Таблиця 4.2 – UC1 Тренування моделі

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає тренувальний метод з API підсистеми моделювання	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Завантажує модель за конфігурацією	
4	Система	Завантажує тренувальний набір	БД чи файл
5	Система	Виконує підготовку даних до тренування	
6	Система	Проводить контрольну ітерацію	
7	Система	Проводить процедуру тренування	Pytorch
8	Система	Оцінює якість моделі	Валідація
9	Система	Зберігає контрольну точку моделі	
10	Система	Повертає користувачу навчену модель	

Розширення:

2а. Користувач не надав жодних конфігураційних параметрів.

2а. 1) Система виводить помилку.

2а. 2) Повернення до дії №2.

2b. Користувач надав тільки тип моделі.

2b. 1) Система завантажує стандартний конфігураційний файл для вказаного типу моделі.

2b. 2) Система попереджає користувача, що завантажено стандартну конфігурацію.

2b. 3) Перехід до дії №3.

2c. Користувач одночасно надав декілька типів конфігурації (файл, об’єкт, директорія).

2c. 1) Система формує одну конфігурацію з таким пріоритетом: конфігураційний об’єкт > конфігураційний файл > конфігураційна директорія.

2c. 2) Система попереджає користувача про можливий конфлікт.

2c. 3) Перехід до дії №3.

3a. Заданий тип моделі не співпадає зі списком доступних.

3a. 1) Система виводить помилку.

3a. 2) Прецедент завершено.

3b. Задані параметри несумісні з типом моделі.

3b. 1) Система виводить помилку.

3b. 2) Прецедент завершено.

4a. Вказаний шлях до даних не існує.

4a. 1) Система виводить помилку.

4a. 2) Прецедент завершено.

4b. Розмір даних перевищує доступний об’єм оперативної пам’яті.

4b. 1) Система виводить помилку.

4b. 2) Система переходить у режим лінивого завантаження даних.

4b. 3) Перехід до дії №5.

6a. Не вистачає оперативної або відео- пам’яті для тренування.

6a. 1) Система виводить помилку.

6a. 2) Система пропонує користувачу змінити параметри тренування.

6a. 3) Прецедент завершено.

8a. Валідаційний набір відсутній.

8a. 1) Система виводить попередження.

8a. 2) Перехід до дії №9.

9a. Директорія для збереження не вказана або її не існує.

9a. 1) Система виводить попередження.

9a. 2) Система зберігає модель у поточній директорії.

9a. 3) Перехід до дії №10.

9b. Для збереження моделі недостатньо дискового простору.

9b. 1) Система виводить помилку.

9b. 2) Система пропонує користувачу змінити параметри збереження моделі.

9b. 3) Прецедент завершено.

UC2 Завантаження та сценарій (табл. 4.3)

Система: підсистема моделювання.

Основна діюча особа: Користувач, Система.

Мета: завантажити бажаний об’єкт.

Тригер: користувач сформував конфігурацію об’єкта та бажає завантажити його.

Результат: користувач завантажив бажаний об’єкт для подальшої роботи з ним.

Таблиця 4.3 – UC2 Завантаження

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає метод завантаження з API підсистеми моделювання	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Завантажує об’єкт за параметрами	
4	Система	Повертає користувачу завантажений об’єкт	

Розширення:

2a. Користувач не надав жодних конфігураційних параметрів.

2a. 1) Система виводить помилку.

2a. 2) Система завершує прецедент.

3a. Вказаний об’єкт не входить до списку підтримуваних.

3a. 1) Система виводить помилку.

3а. 2) Система виводить користувачу список підтримуваних об’єктів.

3а. 3) Система завершує прецедент.

UC3 Завантаження моделі та сценарій (табл. 4.4)

Система: підсистема моделювання.

Основна діюча особа: Користувач, Система.

Мета: отримати підготовлений об’єкт моделі за наданою конфігурацією.

Тригер: користувач формує конфігурацію для моделі та бажає виконати процедуру її завантаження.

Результат: користувач отримав об’єкт моделі, завантаженої за сформованою конфігурацією.

Таблиця 4.4 – UC3 Завантаження моделі

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає метод побудови моделі з API підсистеми моделювання	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Отримує клас моделі за допомогою репозиторія моделей	
4	Система	Ініціалізує об’єкт моделі за допомогою підготовленої конфігурації	
5	Система	Ініціалізує об’єкт токенізатора та надає моделі доступ до нього	
6	Система	Ініціалізує ваги моделі	
7	Система	Повертає користувачу навчену модель	

Розширення:

2a. Користувач не надав жодних конфігураційних параметрів.

2a. 1) Система виводить помилку.

2a. 2) Прецедент завершено.

2b. Користувач надав тільки тип моделі.

2b. 1) Система завантажує стандартний конфігураційний файл для вказаного типу моделі.

2b. 2) Система попереджає користувача, що завантажено стандартну конфігурацію.

2b. 3) Перехід до дії №3.

2c. Користувач одночасно надав декілька типів конфігурації (файл, об’єкт, директорія).

2c. 1) Система формує одну конфігурацію з таким пріоритетом: конфігураційний об’єкт > конфігураційний файл > конфігураційна директорія.

2c. 2) Система попереджає користувача про можливий конфлікт.

2c. 3) Перехід до дії №3.

3a. Заданий тип моделі не співпадає зі списком зареєстрованих у репозиторії.

3a. 1) Система виводить помилку.

3a. 2) Система завершує прецедент.

4a. Параметри конфігурації не співпадають з налаштуваннями моделі.

4a. 1) Система видаляє з конфігурації усі несумісні параметри, замість них використовуються параметри моделі за замовчуванням.

4a. 2) Перехід до дії №5.

4b. Клас для ініціалізації об’єкту не відноситься до підтримуваних типів моделей.

4b. 1) Система виводить помилку.

4b. 2) Система завершує прецедент.

4c. Не вистачає оперативної пам’яті для ініціалізації моделі.

4c. 1) Система виводить помилку.

4c. 2) Система завершує прецедент.

5a. Конфігурацію токенізатора не надано.

5a. 1) Система виводить попередження користувачу.

5a. 2) Перехід до дії №6.

6a. Шлях до файлу вагів моделі не надано.

6a. 1) Система ініціалізує ваги моделі випадковим чином.

6a. 2) Перехід до дії №7.

UC4 Завантаження даних та сценарій (табл. 4.5)

Система: підсистема роботи з даними.

Основна діюча особа: Користувач, Система.

Мета: завантажити необхідний набір даних.

Тригер: користувач формує конфігурацію для набору даних та бажає виконати процедуру його завантаження.

Результат: користувач отримав бажане представлення набору даних.

Таблиця 4.5 – UC4 Завантаження даних

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає метод завантаження даних з API підсистеми роботи з даними	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Завантажує набір даних за вказаним у конфігурації джерелом	Файл чи БД
4	Система	Проводить трансформацію даних за вказаною конфігурацією	
5	Система	Повертає користувачу представлення завантаженого набору даних	

Розширення:

2a. Користувач не надав жодних конфігураційних параметрів.

2a. 1) Система виводить помилку.

2a. 2) Система завершує прецедент.

3a. Не вдалося завантажити дані з вказаного у конфігурації джерела.

3a. 1) Система виводить помилку.

3a. 2) Система завершує прецедент.

3b. Для табличних даних не вказано ключі стовбців.

3b. 1) Система самостійно встановлює ключі стовбців.

3b. 2) Система повідомляє користувача про встановлені ключі.

3b. 3) Система завершує завантаження даних та переходить до дії №4.

3c. Для завантаження даних не вистачає пам’яті.

3c. 1) Система виводить помилку.

3c. 2) Система завершує прецедент.

4a. Параметри трансформації даних не вказано у конфігурації.

4a. 1) Система переходить до дії №5.

4b. У якості трансформації вказано послідовність дій.

4b. 1) Система проводить кожну трансформацію по черзі з використанням кешування даних на кожному етапі.

4b. 2) Система переходить до дії №5.

UC5 Оцінювання моделі та сценарій (табл. 4.6)

Система: підсистема моделювання.

Основна діюча особа: Користувач, Система.

Мета: оцінити підготовлену модель.

Тригер: користувач отримав підготовлену модель після завершення тренування та сформував конфігурацію і тестовий набір для процедури її оцінювання.

Результат: користувач отримав бажані метрики для наданої моделі.

Таблиця 4.6 – UC5 Оцінювання моделі

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає метод оцінювання моделі з API підсистеми моделювання	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Завантажує модель за конфігурацією	
4	Система	Завантажує тестовий набір даних	Файл чи БД
5	Система	Проводить оцінювання моделі на наборі даних	
6	Система	Повертає користувачу результати оцінювання	

Розширення:

2a. Користувач не надав жодних конфігураційних параметрів.

2a. 1) Система виводить помилку.

2a. 2) Система завершує прецедент.

3a. Заданий у конфігурації тип моделі не співпадає зі списком доступних.

3a. 1) Система виводить помилку.

3a. 2) Система завершує прецедент.

3b. У конфігурації вказано список моделей.

3b. 1) Система будує кожну модель, вказану у списку.

3b. 2) Система переходить до дії №4.

4a. Не вдалося завантажити дані з вказаного у конфігурації джерела.

4a. 1) Система виводить помилку.

4a. 2) Система завершує прецедент.

4b. У конфігурації вказано список тестових наборів даних.

4b. 1) Система завантажує кожний набір даних, вказаний у списку.

4b. 2) Система переходить до дії №5.

5a. Замість однієї моделі вказано список моделей.

5a. 1) Система для кожної моделі проводить процедуру оцінювання.

5a. 2) Система зберігає результати оцінювання у список.

5a. 3) Система переходить до дії №6.

5b. Замість одного тестового набору даних вказано список наборів.

5b. 1) Система для кожного тестового набору проводить процедуру оцінювання.

5b. 2) Система зберігає результати оцінювання у список.

5b. 3) Система переходить до дії №6.

5c. Вказано список моделей та список тестових наборів даних.

5c. 1) Система проводить процедуру оцінювання для кожної моделі зі списку на кожному наборі даних зі списку.

5c. 2) Система зберігає результати оцінювання у словник списків.

5c. 3) Система переходить до дії №6.

6a. Користувач вказав параметр повернення n найкращих результатів.

6a. 1) Система сортує результати оцінювання.

6a. 2) Система повертає топ-n результатів з кожного списку.

6a. 3) Прецедент завершено.

UC6 Використання моделі та сценарій (табл. 4.7)

Система: підсистема моделювання.

Основна діюча особа: Користувач, Система.

Мета: завантажити бажану модель для подальшого використання (режим передбачення або запуску у production-середовищі).

Тригер: користувач сформував конфігурацію моделі, провів її тренування та бажає завантажити її у production-режимі.

Результат: користувач завантажив бажану модель для подальшого використання.

Таблиця 4.7 – UC6 Завантаження моделі

№	Дійова особа	Крок	Коментар
1	Користувач	Викликає метод побудови моделі з API підсистеми моделювання	Jupyter Notebook
2	Система	Обробляє конфігураційні параметри	
3	Система	Завантажує модель	
4	Система	Переводить модель у режим використання	
5	Система	Повертає користувачу об’єкт моделі	

Розширення:

2а. Користувач не надав жодних конфігураційних параметрів.

2а. 1) Система виводить помилку.

2а. 2) Система завершує прецедент.

3а. Не вистачає оперативної або відео- пам’яті для завантаження моделі.

3а. 1) Система виводить помилку.

3а. 2) Прецедент завершено.

UC7 Розмітка даних та сценарій (табл. 4.8)

Система: підсистема розмітки.

Основна діюча особа: Користувач, Система.

Мета: отримати розмічений набір даних.

Тригер: користувач завантажує у сховище текстові дані та бажає їх розмітити.

Результат: до текстових даних було додано необхідну розмітку.

Таблиця 4.8 – UC7 Розмітка даних

№	Дійова особа	Крок	Коментар
1	Користувач	Запускає сервіс розмітки	Telegram

Продовження таблиці 4.8

№	Дійова особа	Крок	Коментар
2	Система	Система повертає список доступних задач	
3	Користувач	Обирає задачу для розмітки	
4	Система	Отримує запис зі сховища даних	Nosql
5	Система	Надсилає завдання користувачу	
6	Користувач	Вносить зміни до запису	Переклад, помилки
7	Система	Зберігає зміни	
8	Користувач	Підтверджує розмітку	
9	Система	Зберігає розмітку у сховищі	Nosql
10	Система	Надсилає користувачу наступне завдання	

Розширення:

1a. Сервіс розмітки недоступний.

1a. 1) Система повідомляє про помилку.

1a. 2) Прецедент завершено.

2a. Доступних типів задач не знайдено.

2a. 1) Система повідомляє про помилку.

2a. 2) Прецедент завершено.

4a. Сховище даних недоступне.

4a. 1) Система повідомляє про помилку.

4a. 2) Прецедент завершено.

4b. Немає доступних завдань.

4b. 1) Система повідомляє користувача про відсутність доступних завдань.

4b. 2) Система пропонує користувачу обрати іншу задачу для розмітки та переходить до дії №2.

9a. Сховище даних недоступне.

9a. 1) Система повідомляє про помилку.

9a. 2) Прецедент завершено.

10a. Немає доступних завдань.

10a. 1) Система повідомляє користувача про відсутність доступних завдань.

10а. 2) Система пропонує користувачу обрати іншу задачу для розмітки та переходить до дії №2.

4.2 Нефункціональні вимоги

Деталізувати умови функціонування та основні характеристики розроблюваної системи допоможе такий артефакт специфікації вимог до програмного забезпечення як нефункціональні вимоги, представлені у вигляді таблиці з полями ідентифікатору, назви, пріоритету, складності та контактної особи для опису кожної вимоги. Таблиця 4.9 також поділяється на розділи та підрозділи для класифікації вимог за наступними напрямками: інтерфейсні, апаратні і програмні та операційні.

Таблиця 4.9 – Нефункціональні вимоги

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог		
		Пріоритет	Складність	Контакт
1. Інтерфейси				
1.1 Інтерфейси ПЗ				
SI-01	Взаємодія системи з Amazon AWS S3.	Обов’язково	Середня	Розробник
SI-02	Взаємодія системи з ресурсом telegram.org.	Обов’язково	Низька	Розробник
1.2 Комунікаційні інтерфейси				
HI-01	Протокол передачі даних TCP.	Рекомендовано	Середня	Розробник
HI-02	Підтримка передбачених стандартами Fast Ethernet і встановленими ОС протоколів для обміну даними.	Рекомендовано	Середня	Розробник

Продовження таблиці 4.9

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог		
		Пріоритет	Складність	Контакт
HI-02	Підтримка передбачених стандартами Fast Ethernet і встановленими ОС протоколів для обміну даними.	Рекомендовано	Середня	Розробник
HI-03	Взаємодія з Telegram Bot API за допомогою техніки Long Polling	Рекомендовано	Середня	Розробник, Telegram Bot API
1.3 Інтерфейс користувача				
UI-01	Усі вікна та елементи інтерфейсу мають єдине стильове оформлення.	Обов’язково	Низька	Розробник
UI-02	Підтвердження або відхилення операцій за допомогою відповідних кнопок.	Обов’язково	Низька	Розробник
2. Апаратні та програмні вимоги				
SR-01	ОС Windows >= 7, ОС Ubuntu >= 16.04	Обов’язково	Низька	Користувач
SR-02	Не менше 4gb вільного дискового простору	Обов’язково	Низька	Користувач
SR-03	RAM > 16gb	Бажано	Низька	Користувач
SR-04	Для обчислень на GPU CUDA Compute capability >= 3.7	Обов’язково	Середня	Користувач, NVIDIA CUDA

Продовження таблиці 4.9

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог		
		Пріоритет	Складність	Контакт
SR-05	Для CPU-обчислень процесор не нижче Intel Core i3-8100	Обов’язково	Низька	Користувач
3. Операційні вимоги				
3.1 Безпека та конфіденційність				
SP-01	Блокуються усі недозволені з’єднання	Обов’язково	Низька	Системний адміністратор
SP-02	Усі персональні дані користувачів зберігаються у зашифрованому вигляді.	Бажано	Низька	Розробник
SP-03	Права на доступ до серверної частини розділено.	Обов’язково	Середня	Системний адміністратор
3.2 Надійність				
R-01	Продовження роботи системи у випадку збоїв.	Рекомендовано	Середня	Розробник
R-02	Збій одного компоненту системи не призводить до збоїв у інших.	Рекомендовано	Висока	Розробник, Системний адміністратор
3.3 Відновлюваність				
RE-01	Автоматичний повторний запуск у випадку помилок.	Обов’язково	Низька	Systemd/ Systemctl
RE-02	Формування контрольних точок на кожному етапі роботи з даними та тренування моделей.	Бажано	Висока	Розробник, Pytorch

Продовження таблиці 4.9

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимог		
		Пріоритет	Складність	Контакт
3.4 Продуктивність				
P-01	Час побудови моделі за конфігурацією залежить від обчислювальної платформи, але не перевищує 20 секунд.	Обов’язково	Середня	Розробник
P-02	Час підготовки одного тренувального пакету не перевищує 0.5 секунд.	Бажано	Висока	Розробник
3.5 Збереження даних				
DR-01	Система має підтримувати збереження файлів розміром до 8 gb.	Бажано	Висока	Розробник
DR-02	MongoDB у якості БД для неструктурованих даних.	Рекомендовано	Середня	MongoDB

В результаті за допомогою таблиці було наведено деталізований опис нефункціональних вимог до розроблюваної програмної системи, а також проведено їх розподіл за такими напрямками як інтерфейсні, апаратні і програмні та операційні вимоги.

4.3 Архітектура програмної системи

Архітектура розроблюваної програмної системи (рис. 4.2) використовує мікросервісний підхід, що реалізується на практиці у вигляді трьох автономних

сервісів на мові програмування Python: ModelingService, NotificationService та LabelingService. Окрім того, сервіс моделювання має власне сховище моделей та їх конфігурацій – ModelingRepository, сервіс розмітки – базу даних для роботи з наборами даних DatasetsStorage, а також LabelingService взаємодіє з зовнішнім сервісом перекладу Google Cloud Translation.

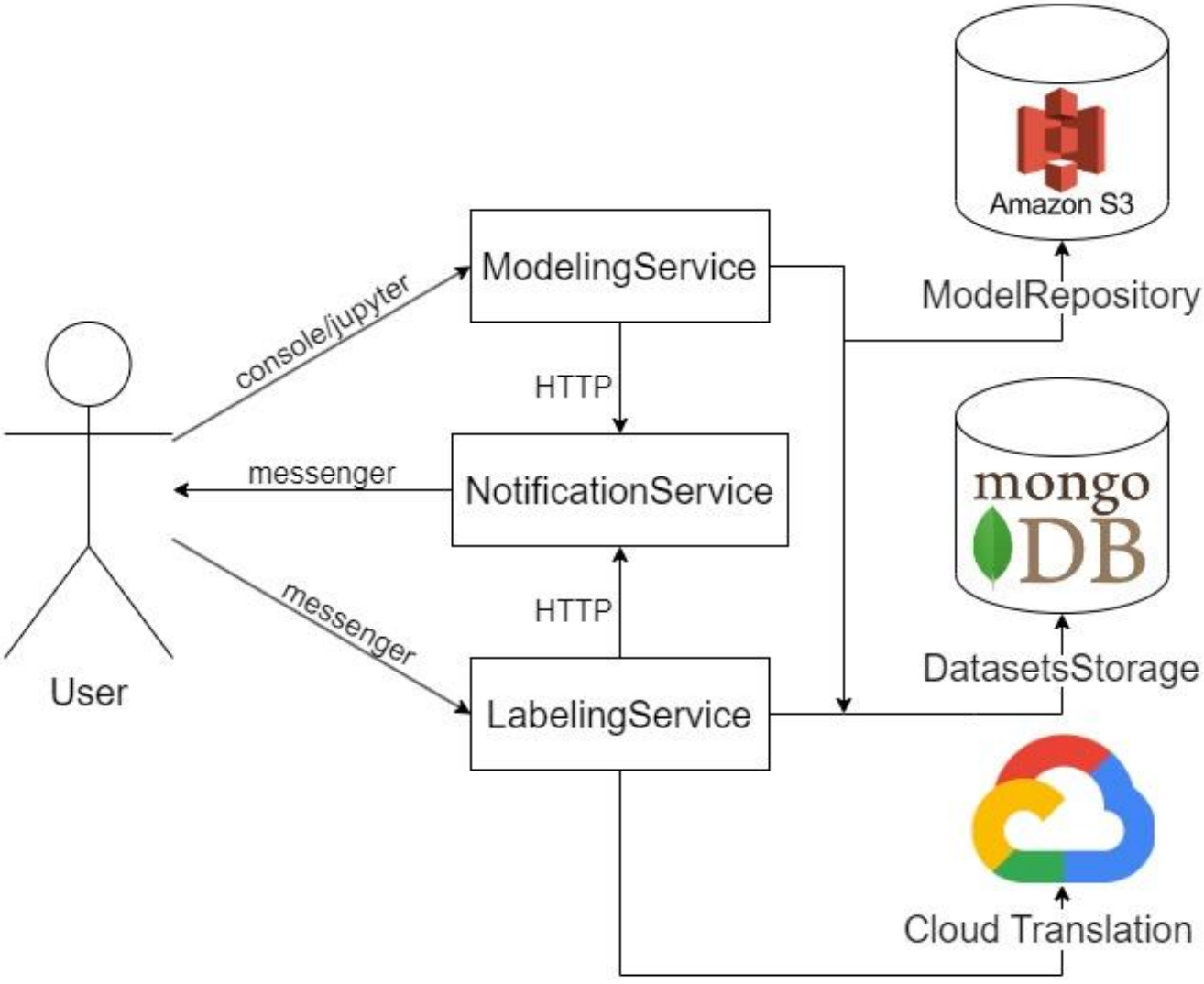

Рисунок 4.2 – Архітектура системи

ModelingService – сервіс моделювання з функціоналом прототипування, тренування та оцінювання моделей глибинного навчання, а також завантаження та підготовки тренувальних даних. Сервіс моделювання є локальним інструментом – завантажується і встановлюється користувачем на обчислювальну машину та не

потребує доступу до глобальної мережі через велику кількість імплементованих конвеєрів обробки даних для широкого кола задач та реалізованих архітектур моделей. Проте, за наявності інтернет-з'єднання стає доступним функціонал інформування щодо перебігу процедур тренування у месенджері Telegram за допомогою сервісу повідомлень, завантаження підготовлених моделей та їх конфігурацій з хмарного репозиторію Amazon S3, а також стає можливим завантаження даних для тренування чи тестування напряму зі сховища даних сервісу розмітки.

LabelingService – сервіс розмітки текстових даних для тренування, представлений у вигляді Telegram-бота, дозволяє проводити процедури маркування даних з залученням великої кількості користувачів-анотаторів. Сервіс має власне сховище даних на базі MongoDB, що підтримує роботу з ієрархічними та слабоструктурованими форматами – найпоширенішими варіантами збереження елементів текстових наборів даних. Додатково сервіс розмітки взаємодіє з сервісом сповіщення для інформування щодо прогресу у виконанні задач, а також з хмарним сервісом перекладу для реалізації функціоналу автоматичного перекладу контенту завдань при роботі з багатомовними наборами даних або задля спрощення їх локалізації.

NotificationService – сервіс сповіщення користувача про хід тренування/оцінки моделі чи статус розмітки завдання. Взаємодія з користувачами відбувається за допомогою Telegram-бота. Сервіси моделювання та розмітки даних надсилають повідомлення на відповідні точки доступу сервісу оповіщення, який виконує форматування вхідних даних для зручного перегляду та надсилає запит до точки доступу Telegram Bot API з метою відправки обробленого повідомлення користувачу.

ModelRepository – компонент програмної системи, що відповідає за збереження та організацію доступу до конфігураційних файлів та контрольних точок (ваги мереж, стан і параметри оптимізаційних алгоритмів) попередньо навчених моделей. Репозиторій моделей базується на продукті AWS S3, що представляє собою хмарну файлову систему для роботи зі статичними файлами

великої розмірності. Збереження моделей у репозиторію відбувається в архівованому вигляді з призначенням кожному об’єкту ключа, що формується шляхом хешування відповідного конфігураційного файлу. Завантаження моделей з репозиторія також відбувається в архівованому вигляді за призначеним ключем. Процедури завантаження і збереження моделей та їх конфігурацій користувач може здійснювати за допомогою програмного інтерфейсу (API) сервісу моделювання.

DatasetsStorage – база даних, що зберігає текстові набори даних для розмітки, розмічені приклади, а також відомості анотаторів у JSON-форматі. БД базується на NoSQL-СКБД MongoDB, а основна логіка для роботи зі сховищем та даними розміщена у LabelingService. Окрім того, до сховища даних також має доступ сервіс моделювання, що може використовувати DatasetsStorage для отримання розмічених даних для процедур тренування чи тестування моделей машинного навчання. Візуалізація спроектованого сховища даних та детальний опис кожного його компонента наведено у підрозділі 4.5.

Cloud Translation – зовнішній сервіс від провайдера хмарних послуг GCP (Google Cloud Platform), що використовується у системі для автоматичного машинного перекладу текстових наборів даних LabelingService. Взаємодія з Cloud Translation відбувається шляхом відправки сервісом розмітки даних HTTP-запитів з текстовим контентом, що необхідно перекласти, а також мовою, на яку необхідно виконати переклад. У якості відповіді на запит Cloud Translation надає JSON-об’єкт, що містить текст після перекладу на бажану мову.

Для взаємодії з сервісом моделювання користувач активує середовище у Jupyter Notebook або запускає утиліту командного рядка і вмикає інтерпретатор Python. Після цього необхідно імпортувати сервіс моделювання, що ініціалізує основні компоненти для подальшої роботи та сформує список доступних архітектур та шарів нейромереж. Якщо у користувача вже є готова конфігурація, то за допомогою спеціальної команди він може виконати побудову моделі, якщо ж її немає, то ініціювати модель можна з віддаленого репозиторію: у якості конфігураційного аргументу тоді вказується текстовий ідентифікатор, який сервіс

трансформує у відповідний конфігураційний файл, що потребується для завантаження попередньо підготовленої моделі зі сховища Amazon S3. У результаті виконання команди побудови користувачу повертається об’єкт моделі, який він може використовувати для тренування, оцінювання, або у режимі передбачення для production-середовища. Для тренування моделі необхідно у якості аргументів передати налаштування для завантаження та підготовки набору даних. Сервіс моделювання підтримує завантаження даних з файлової системи, а також зі сховища сервісу розмітки. Додатковою опцією є режим лінивого завантаження, при якому дані з джерела завантажуються та обробляються під час тренування, а не перед ним, що допомагає при роботі з надвеликими датасетами (чий розмір перевищує розмір доступної оперативної пам’яті).

З сервісом розмітки користувач взаємодіє через месенджер Telegram, здійснюючи запит до бота в текстовому вигляді у форматі повідомлення. З’єднання між LabelingService та Bot API Telegram реалізується у форматі long polling: сервіс розмітки здійснює запит на отримання нових повідомлень та не перериває його поки месенджер їх не надасть, далі сервіс аналізує кожне отримане повідомлення, отримує інформацію про поточний стан користувача та видане йому завдання, за потреби вносить необхідні зміни та відправляє http-запитом відповіді і повторно робить запит на отримання нових повідомлень. У кожній задачі користувач може виконати переклад тексту завдання, для цього сервіс надсилає http-запит до API системи перекладу Google Cloud Translation, отримана відповідь використовується для внесення змін до запису у таблиці розміток, після цього користувачу надсилається оновлене завдання.

4.4 Проектування структури та організація класів

На діаграмі класів (рис. 4.3) продемонстровано статичну декомпозицію сервісу розмітки даних на пакети, їх компоненти та методи, додатково зазначено зв’язки між класами та їх кратність.

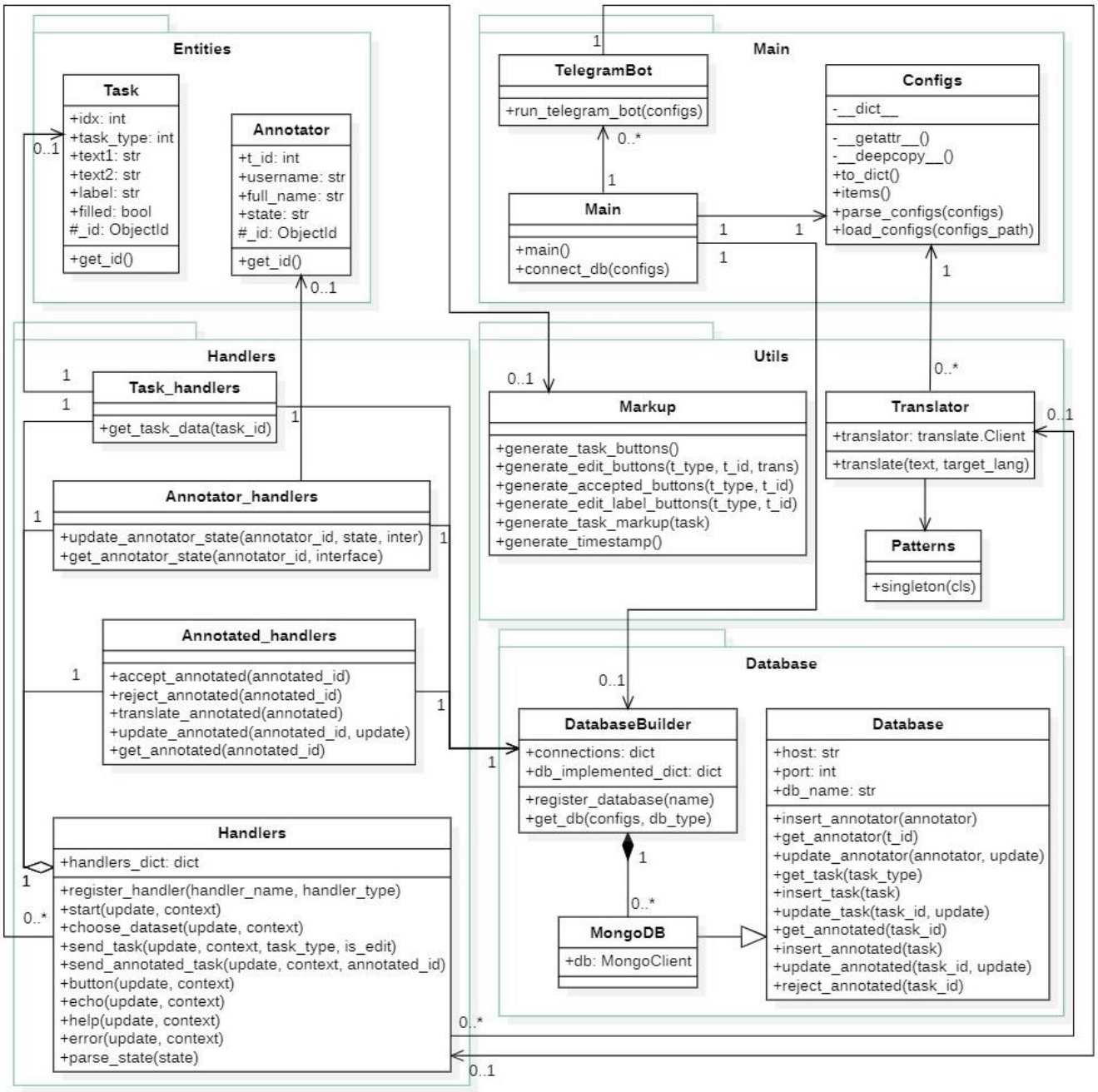

Рисунок 4.3 – Діаграма класів сервісу розмітки

Task – клас, що представляє собою сутність завдання.

- get_id() – повертає ідентифікатор задачі зі сховища.

Annotator – представляє собою сутність користувача, що розмічує дані.

- get_id() – повертає ідентифікатор користувача зі сховища.

Main – головний клас сервісу, конфігурує усі необхідні компоненти для роботи та запускає їх.

- main() – головний метод системи, запускає усі інші компоненти

- connect_db(configs) – метод для ініціалізації об’єкту взаємодії з БД.

Configs – клас, що формує ієрархічне представлення для конфігурації у форматі словника та організовує простий доступ до її параметрів.

- items() – повертає список пар ключ-значення
- parse_configs(configs) – розбирає конфігурацію та формує на її основі об’єкт
- load_configs(configs_path) – завантажує конфігураційний файл та трансформує його у json.

TelegramBot – клас, що конфігурує взаємодію з користувачами за допомогою Telegram Bot API.

- run_telegram_bot(configs) – запускає бота за параметрами.

Handlers – імплементує методи для обробки запитів користувача, що надходять у сервіс, а також реалізує шаблон проектування “Мультитон” для запуску методів обробки запитів об’єктів-нащадків. “Мультитон” реалізовано за допомогою метода-декоратора для реєстрації екземпляра та структури даних словник у якості реєстру, що дозволяє контролювати кількість створених екземплярів, а також дає можливість організовувати зручний доступ до об’єктів за допомогою складеного ключа (назва обробника разом з його конфігурацією).

- register_handler(handler_name, handler_type) – метод-декоратор для внесення методів обробки запитів до списку доступних
- start(update, context) – обробляє користувацьку команду “/start”, демонструє основні можливості сервісу
- choose_dataset(update, context) – обробляє запит на вибір типу задачі, що користувач обрав для розмітки
- send_task(update, context, task_type, is_edit) – надсилає користувачу завдання обраного типу
- send_annotated_task(update, context, annotated_id) – надсилає користувачу розмічене ним завдання
- button(update, context) – обробляє запити натискання на одну з кнопок керування

- echo(update, context) – обробляє текстові запити, що не є командами
- help(update, context) – обробляє користувацьку команду “/help”, демонструє основні можливі рішення проблем
- error(update, context) – обробляє користувацьку команду “/error”, що викликається для фіксації помилки
- parse_state(state) – обробити стан користувача у текстовому форматі та витягнути з нього необхідні елементи.

Annotated_handlers – клас, що реалізує шаблон проектування “Фасад”: надає методам обробки запитів, пов’язаним з розміткою завдань, простий інтерфейс для взаємодії з базою даних.

- accept_annotated(annotated_id) – підтвердити завершення розмітки задачі за її ідентифікатором
- reject_annotated(annotated_id) – відхилити підтвердження завершення розмітки задачі за її ідентифікатором
- translate_annotated(annotated) – перекласти текстовий контент завдання на поточну мову користувача
- update_annotated(annotated_id, update) – редагувати розмітку для завдання за ідентифікатором
- get_annotated(annotated_id) – отримати розмічене завдання за його ідентифікатором.

Annotator_handlers – клас, що виконує роль посередника між методами обробки запитів, пов’язаних з користувачем, та базою даних. Аналогічно реалізує шаблон проектування “Фасад”.

- update_annotator_state(annotator_id, state, interface) – оновити стан користувача за його ідентифікатором
- get_annotator_state(annotator_id, update) – отримати стан користувача за його ідентифікатором.

Task_handlers – клас, що виконує роль посередника між методами обробки запитів, пов’язаних із завданням, та базою даних. Також реалізує шаблон “Фасад”.

- get_task_data(task_id, update) – отримати завдання за його ідентифікатором.

Markup – клас з методами для генерації відповіді на користувацький запит.

- generate_task_buttons() – генерує меню для вибору задачі
- generate_edit_buttons(task_type, task_id, translated) – генерує меню для внесення змін до розмітки
- generate_accepted_buttons(task_type, task_id) – генерує меню для завдання, що вже було підтверджено
- generate_edit_label_buttons(task_type, task_id) – генерує меню для внесення змін до мітки завдання
- generate_task_markup(task) – генерує розмітку для відображення текстових даних запиту
- generate_timestamp() – генерує часову мітку.

Translator – клас для перекладу тексту.

- translate(text, target_lang) – метод для перекладу тексту.

DatabaseBuilder – клас, що також реалізує шаблон проектування “Мультитон”: створює обмежену кількість клієнтів баз даних та надає зручний доступ до них.

- register_database(name) – метод-декоратор для внесення клієнтів для роботи з базою даних до списку підтримуваних
- get_db(configs, db_type) – отримати клієнт для роботи з базою даних.

MongoDB – клас, що реалізує інтерфейс для взаємодії зі сховищем MongoDB.

Database – абстрактний клас, що описує методи для роботи з базою даних.

- insert_annotator(annotator) – метод для внесення нового користувача
- get_annotator(t_id) – отримати користувача за його ідентифікатором
- update_annotator(annotator, update) – оновити дані про користувача
- get_task(task_type) – отримати випадкове завдання за його типом
- insert_task(task) – внести завдання у базу даних
- update_task(task_id, update) – оновити завдання за його ідентифікатором
- get_annotated(task_id) – отримати розмічене завдання за ідентифікатором
- insert_annotated(task) – внести розмічене завдання у сховище даних

- update_annotated(task_id, update) – оновити розмічене завдання за його ідентифікатором
- reject_annotated(task_id) – відхилити підтверджене розмічене завдання.

На діаграмі класів (рис. 4.4) продемонстровано статичну декомпозицію сервісу сповіщень на пакети, їх компоненти та методи, додатково зазначено зв’язки між класами та їх кратність.

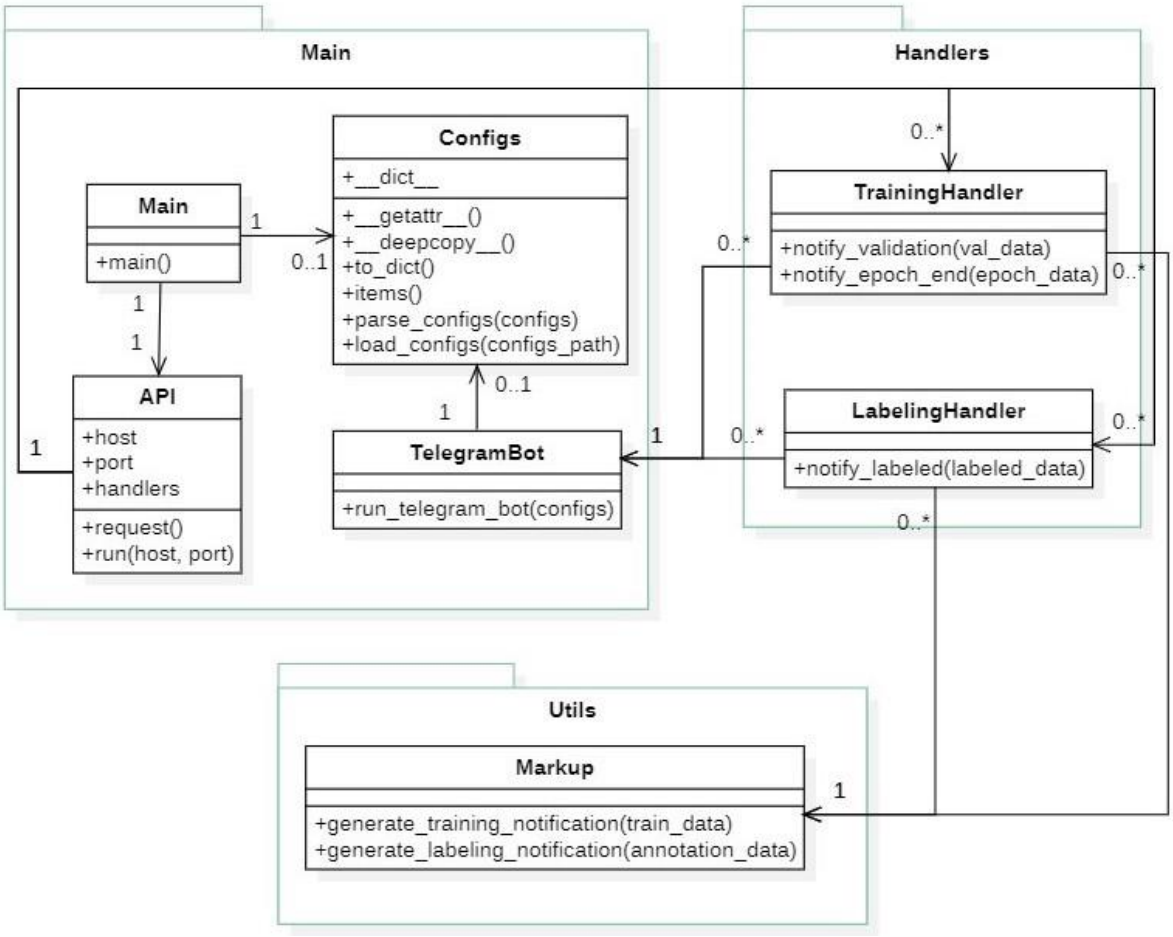

Рисунок 4.4 – Діаграма класів сервісу сповіщень

Main – головний клас сервісу, конфігурує усі необхідні компоненти для роботи та запускає їх.

- main() – головний метод системи, запускає усі інші компоненти.

Configs – клас, що формує ієрархічне представлення для конфігурації у форматі словника та організовує простий доступ до її параметрів.

- items() – повертає список пар ключ-значення

- parse_configs(configs) – розбирає конфігурацію у json-форматі та формує на її основі об’єкт
- load_configs(configs_path) – завантажує конфігураційний файл та трансформує його у json.

TelegramBot – клас, що конфігурує взаємодію з користувачами за допомогою Telegram Bot API.

- run_telegram_bot(configs) – запускає бота за параметрами.

API – реалізує точки доступу, на які надходять запити від сервісів моделювання і розмітки даних, а також відповідає за запуск сервера.

- request() – запит до сервера
- run(host) – запускає сервер на відповідному host.

TrainingHandler – клас-обробник, що реалізує логіку для сповіщення користувача щодо тренувальних подій.

- notify_validation(val_data) – обробляє дані завершеної події валідації під час тренувального процесу, формує зручну структуру повідомлення та надсилає його користувачу
- notify_epoch_end(epoch_data) – обробляє дані завершеної епохи тренування та надсилаї відформатоване повідомлення користувачу.

LabelingHandler – клас-обробник, що реалізує логіку для сповіщення користувача щодо перебігу процесу розмітки даних для конкретної задачі.

- notify_labeled(labeled_data) – обробляє дані завершеної розмітки задачі та надсилає відформатоване повідомлення користувачу.

Markup – клас, що містить методи форматування та генерації зручних для перегляду повідомлень.

- generate_training_notification(train_data) – генерує повідомлення про успішне завершення тренувальної події (валідації чи епохи)
- Generate_labeling_notification(annotation_data) - генерує повідомлення про успішне завершення події розмітки даних.

На діаграмі класів (рис. 4.5) продемонстровано статичну декомпозицію сервісу моделювання на пакети, їх компоненти та методи, додатково зазначено зв’язки між класами та їх кратність.

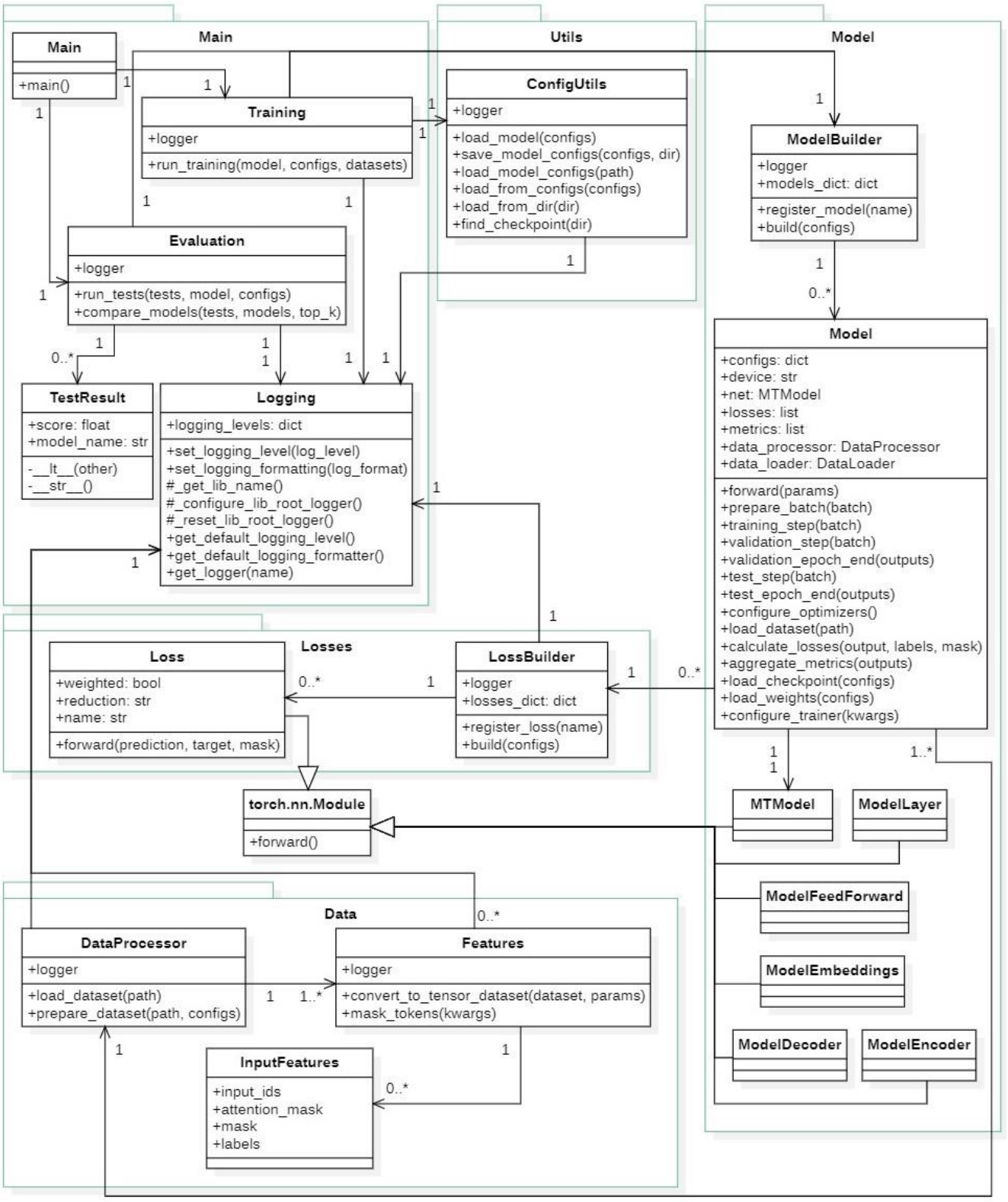

Рисунок 4.5 – Діаграма класів сервісу моделювання

Main – головний клас сервісу, конфігурує усі необхідні компоненти для роботи та запускає їх.

- main() – головний метод системи, запускає усі інші компоненти.

Training – клас для тренування моделей машинного навчання.

- run_training(model, configs, datasets) – виконує тренування моделі за заданою конфігурацією на обраних наборах даних.

Evaluation – клас для тестування моделей машинного навчання.

- run_tests(tests, model, configs) – проводить тестування навченої моделі за заданою конфігурацією на обраних тестових наборах даних
- compare_models(tests, models, top_k) – оцінює навчені моделі на заданих тестових наборах даних та повертає відсортовані результати, згруповані за кожним тестом.

TestResult – структура для збереження результату тестування моделі.

- __it__(other) – метод для порівняння результатів.

Logging – реалізує функціонал для логування з використанням шаблону проектування “Мультитон”: глобальна змінна містить словник з екземплярами логувальника для кожного простору імен.

- set_logging_level(log_level) – встановлює рівень логування
- set_logging_formatting(log_formatting) – встановлює формат повідомлень для логування
- _get_lib_name() – отримує назву програмного пакету
- configure_lib_root_logger() – налаштовує об’єкт логування за замовчуванням
- reset_lib_root_logger() – повторно налаштовує об’єкт логування
- get_default_logging_level() – отримати рівень логування за замовчуванням
- get_default_logging_formatter() – отримати формат повідомлень для логування за замовчуванням
- get_logger(name) – отримати об’єкт логування за ключем.

Loss – абстрактний клас, що використовується для реалізації функцій втрат.

- forward(prediction, target, mask) – метод для обчислення похибки.

LossBuilder – клас, що відповідає за створення об’єктів функцій втрат. Реалізує шаблон проектування “Мультитон”: створює обмежену кількість екземплярів функцій втрат та організовує зручний доступ до них за ключем-конфігурацією.

- register_loss(name) – метод-декоратор для внесення функцій втрат до списку доступних
- build(configs) – будує функцію втрат за вказаною конфігурацією.

DataProcessor – абстрактний клас, призначений для завантаження та попередньої обробки даних.

- load_dataset(path) – завантажує набір даних за вказаним шляхом
- prepare_dataset(path, configs) – виконує підготовку набору даних за вказаним шляхом та конфігурацією.

Features – абстрактний клас, що використовується для побудови ознак.

- convert_to_tensor_dataset(dataset, params) – трансформує набір даних у набір ознак
- mask_tokens(kwargs) – маскує токени послідовності.

InputFeatures – структура, що використовується для збереження даних записів з набору даних.

ConfigUtils – клас, що відповідає за роботу з конфігураціями.

- load_model(configs) – завантажує модель за вказаною конфігурацією
- save_model_configs(configs, path) – зберігає конфігурації моделі
- load_model_configs(path) – завантажити конфігурації моделі за шляхом
- load_from_configs(configs) – завантажити модель з конфігурації
- load_from_dir(dir) – завантажити модель з директорії
- find_checkpoint(dir) – знайти стан моделі за шляхом.

ModelBuilder – клас, що відповідає за створення об’єктів моделей. Реалізує шаблон проектування “Мультитон”: створює обмежену кількість об’єктів моделей та надає зручний доступ до них за ключем-конфігурацією.

- register_model(name) – метод-декоратор для внесення моделей до списку доступних
- build(configs) - будує модель за вказаною конфігурацією.

Model – абстрактний клас, що використовується для реалізації моделей.

- forward(params) – реалізує алгоритм прямого поширення
- prepare_batch(batch) – заготовляє тренувальний пакет для використання
- training_step(batch) – реалізує тренувальний крок
- validation_step(batch) – реалізує валідаційний крок
- validation_epoch_end(outputs) – обробляє дані, отримані під час валідації
- test_step(batch) – реалізує тестовий крок
- test_epoch_end(outputs) – обробляє дані, отримані під час тестування
- configure_optimizers() – налаштовує оптимізаційний алгоритм
- load_dataset(path) – завантажує набір даних за вказаним шляхом
- calculate_losses(output, labels, mask) – обчислює функцію втрат
- aggregate_metrics(outputs) – збирає дані та обчислює метрики
- load_checkpoint(configs) – завантажує стан моделі за конфігурацією
- load_weights(configs) – завантажує ваги моделі за конфігурацією
- configure_trainer(kwargs) – налаштовує тренувальний алгоритм.

MTModel – абстрактний клас, що використовується для реалізації багатозадачних моделей.

ModelLayer – абстрактний клас, призначений для реалізації довільних шарів моделей.

ModelFeedForward – абстрактний клас, призначений для реалізації шарів прямого поширення.

ModelEmbeddings – абстрактний клас, призначений для реалізації шарів ембеддінгів.

ModelEncoder – абстрактний клас, призначений для реалізації моделей-кодувальників.

ModelDecoder – абстрактний клас, призначений для реалізації моделей-декодерів.

Також варто зазначити, що кожний метод-декоратор фактично реалізує шаблони проектування “Спостерігач” та “Замісник”, так як методи реагують на виклик відповідних об’єктів, підставляються замість них під час їх виклику та дозволяють виконати необхідні дії до передачі виклику оригінальному об'єкту.

4.5 Проектування сховища даних

Високорівневе проектування сховища даних здійснено за допомогою ER-діаграми. Модель “сутність-зв’язок” дозволяє візуалізувати основні компоненти та зв’язки між ними. Побудована діаграма продемонстрована на рисунку 4.6.

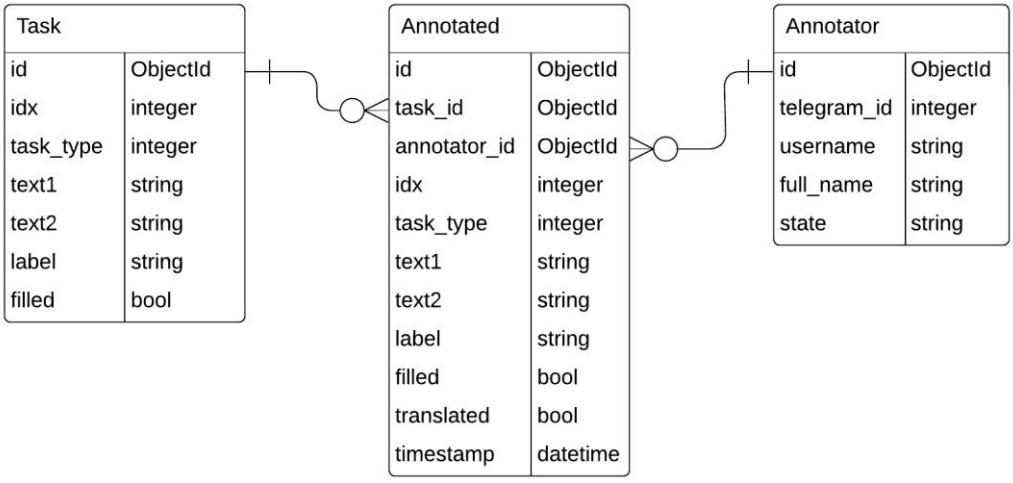

Рисунок 4.6 – ER-діаграма

Task – структура, що зберігає інформацію про завдання. Містить поля id і idx для ідентифікації у сховищі даних та у оригінальному наборі даних, цілочисельне task_type визначає тип завдання. Поля text1 та text2 містять текстовий опис завдання, а label – мітку (класифікації чи будь-якого іншого типу задачі). Логічне значення filled визначає: було завдання розмічено чи ні.

Annotated — структура даних, що містить інформацію щодо розміченого

завдання. Кожне завдання може мати декілька розміток, кожен користувач також може мати декілька розміток. Поле id ідентифікує запис у сховищі даних, а task_id та annotator_id використовуються для реалізації відносин «один-до-багатьох». Поля idx, task_type, text1, text2 та label відповідають полям з таблиці задач, при цьому текстові поля можуть бути відредаговані під час розмітки. Логічне значення filled позначає статус виконання завдання, поле translated – перекладено завдання чи ні, а timestamp зберігає часову мітку завершення виконання завдання.

Annotator – структура, що описує користувача сервісу розмітки. Поле id ідентифікує запис у сховищі даних, а telegram_id зберігає ідентифікатор користувача у месенджері Telegram. Username та full_name – текстові поля, що містять персональні відомості про користувача. Поле state використовується для збереження поточного стану користувача (стартове меню, меню виконання завдання чи його вибору).

4.6 Висновки до розділу

У розділі проектування було візуалізовано функціональні вимоги до продукту за допомогою діаграми варіантів використання, після цього вони були відсортовані за пріоритетом та складністю реалізації і детально описані, окрім того, були деталізовані і нефункціональні вимоги до програмної розробки. Також в результаті виконання цього етапу спроектовано архітектуру системи, продемонстровано ієрархію та взаємозв’язок компонентів з використанням діаграм класів та моделі “сутність-зв’язок”.

5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

5.1 Використані технології

Python – високорівнева інтерпретована мова програмування з широкими функціональними можливостями. До переваг можна віднести націленість інструменту на високу ефективність розробників та відносну простоту синтаксичних конструкцій. Вищезазначене, однак, призводить до слабкої продуктивності рішень у порівнянні з системними мовами програмування, такими як C/C++ та Golang. Важливим аспектом під час вибору Python у якості основного інструменту для реалізації програмної системи став той факт, що мова має найбільшу кількість загальнодоступних розробок за напрямом машинного навчання та аналізу даних [55].

PyTorch – низькорівневий фреймворк глибинного навчання для мови програмування Python від FAIR (Facebook AI Research). Складається з функцій, що реалізують математичні операції, і методів для побудови та обходу графу обчислень на усіх основних обчислювальних платформах, додатково PyTorch містить основні блоки для побудови нейромереж: шари, функції активації та втрат. В програмній системі фреймворк використовується для усіх задач, пов’язаних з прототипуванням, тренуванням та тестуванням нейронних мереж.

PyTorch Lightning – високорівневий фреймворк для глибинного навчання, що базується на PyTorch. Методи фреймворку реалізують основні сценарії тренування та тестування рішень, значно спрощуючи етап побудови та дослідження моделей глибинного навчання. Простоту роботи з PyTorch Lightning відзначив і організаційний комітет конференції NeurIPS, встановивши фреймворк у якості стандарту для подачі матеріалів на секцію Reproducibility Challenge [56]. В роботі фреймворк використовується для реалізації компонентів прототипування, тренування та тестування моделей.

Transformers – високорівневий фреймворк для вирішення широкого спектру задач у галузі обробки природної мови. При цьому для побудови рішень підготовлено більше 30 моделей на базі архітектури Transformer (BERT, T5, FNet,

Reformer, BigBird), а також доступні попередньо навчені мережі для більше ніж 100 мов [57]. У програмній системі використовується для інтеграції деяких архітектурних компонентів з метою розширення конфігураційних та функціональних можливостей.

Flask – мікрофреймворк для побудови веб-застосунків на мові програмування Python. Проста архітектура з прозорими функціональними конструкціями стала основним чинником його вибору у якості інструменту для реалізації REST-компонентів мікросервісів програмної системи.

MongoDB – документо-орієнтоване NoSQL сховище даних з AGPLv3 ліцензією, що поширюється з 2009 року. Дані у сховищі зберігаються у бінарному варіанті JSON (BSON) – це забезпечує легкість керування базою даних та взаємодії з іншими компонентами програмних систем. Окрім того, MongoDB підтримує вкладену структуру даних, що досить важливо для програмної розробки, адже деякі завдання з галузі обробки природної мови мають ієрархічну розмітку і потребується нетривіальна трансформація для їх зберігання у SQL базах даних.

Amazon S3 – сервіс провайдера хмарних послуг AWS (Amazon Web Services), призначений для збереження та організації доступу до великих об’ємів статичних даних. Використовується у проекті для збереження конфігураційних файлів імплементованих архітектур та попередньо навчених моделей, які користувач за потреби зможе завантажити через інтерфейс програмної системи.

Google Cloud Translation API – сервіс провайдера хмарних послуг GCP (Google Cloud Platform), надає доступ до високоточних рішень для текстового перекладу на базі технологій нейронного машинного перекладу від Google. Інструмент підтримує більше 100 мов, що стало важливим критерієм для його використання у функціоналі автоматичного перекладу текстових наборів даних сервісу розмітки.

Мікросервісна архітектура – поширене архітектурне рішення для проектування програмних розробок, в якому функціонал системи розділяється на декілька сервісів з мінімальним ступенем зв’язаності [58] та максимальною пов’язаністю [59]. Зазвичай сервіси реалізують REST-архітектуру та передають інформацію між собою за допомогою протоколу передачі даних HTTP з JSON або

XML для представлення даних, однак, може використовуватися і непрямий формат комунікації за допомогою проксі-серверів або брокерів повідомлень (Kafka, RabbitMQ, ActiveMQ) для забезпечення нефункціональних вимог, таких як балансування навантаження, логування та відновлюваність. На етапі проектування програмної системи особливості мікросервісної архітектури: простота оновлення окремих функціональних компонентів та можливість паралельної роботи над ними – стали основними перевагами цього підходу перед монолітною архітектурою. Необхідно також зазначити, що мікросервісна архітектура, як і будь-яка розподілена система, потребує додаткової реплікації у сховищах даних, що випливає з CAP-теореми [60].

5.2 Реалізація сховища даних

Структура NoSQL документо-орієнтованої СКБД MongoDB представлена набором колекцій документів у форматі BSON. Відносини один-до-багатьох реалізовані у вигляді включення до однієї сторони поля з ідентифікатором документу іншої сторони.

Для кожного сервіса, що взаємодіє зі сховищем, необхідно створити обліковий запис та позначити привілеї. Розглянемо процедуру на прикладі сервісу розмітки текстових даних:

```
db.createUser(  // функція створення нового користувача
... {
... user: "labelingService",  // логін користувача
... pwd: "password",  // пароль
... roles: [  // задання привілеїв для користувача
... {role: "readWrite", db: "LabelingDB" }   // можливість редагувати базу
messages
... ]})
```

Для того, щоб сервіс розмітки мав можливість встановлювати зв’язок зі сховищем даних, необхідно також занести авторизаційні дані в спеціальний конфігураційний файл зі сховищем перед початком роботи.

Розглянемо приклад bson-документу колекції Task:

```
{
    "_id": {"$oid": "6338d54b19a144204c6587ce"},
    "idx": 1,
```

```
    "task_type": 2,
    "text1": "The party guests were hiding behind the couch.",
    "text2": "Choice #0: It was a surprise party. Choice #1: It was a birthday.",
    "label": "0",
    "filled": false,
}
```


Колекцію реалізовано відповідно до спроектованої ER-діаграми. Далі наведено приклад запису з Annotated:


```
{
    "_id": {"$oid": "633b796163f893666354f82c"},
    "task_type": 2,
    "text1": "Гості вечірки ховалися за диваном.",
    "text2": "Choice #0: Це була вечірка-сюрприз. Choice #1: Це був день народження.",
    "label": "0",
    "filled": false,
    "idx": 1,
    "t_id": {
        "$oid": "6338d54b19a144204c6587ce"
    },
    "u_id": {
        "$oid": "6339dab80671b9101d31cb6e"
    },
    "translated": true
}
```


Елементи колекції Task пов’язані із Annotated за допомогою поля t_id останньої, що є посиланням на ідентифікатор елемента колекції завдань, а елементи колекції Annotators пов’язані із Annotated за допомогою поля u_id – ідентифікатора з колекції користувачів. Приклад документу колекції Annotators:


```
{
    "_id": {
        "$oid": "6339dab80671b9101d31cb6e"
    },
    "t_id": 311277887,
    "username": "niksyromyatnikov",
    "full_name": "Nikita Syromiatnikov",
    "state": "2.6.633b796163f893666354f82c"
}
```


5.3 Інструкція з встановлення сервісу моделювання

Сервіс моделювання розробленої системи встановлюється на пристрій користувача, для цього спочатку необхідно завантажити архівований файл з компонентами програмного пакету та сценарієм його встановлення у файлі setup.py (рис. 5.1).

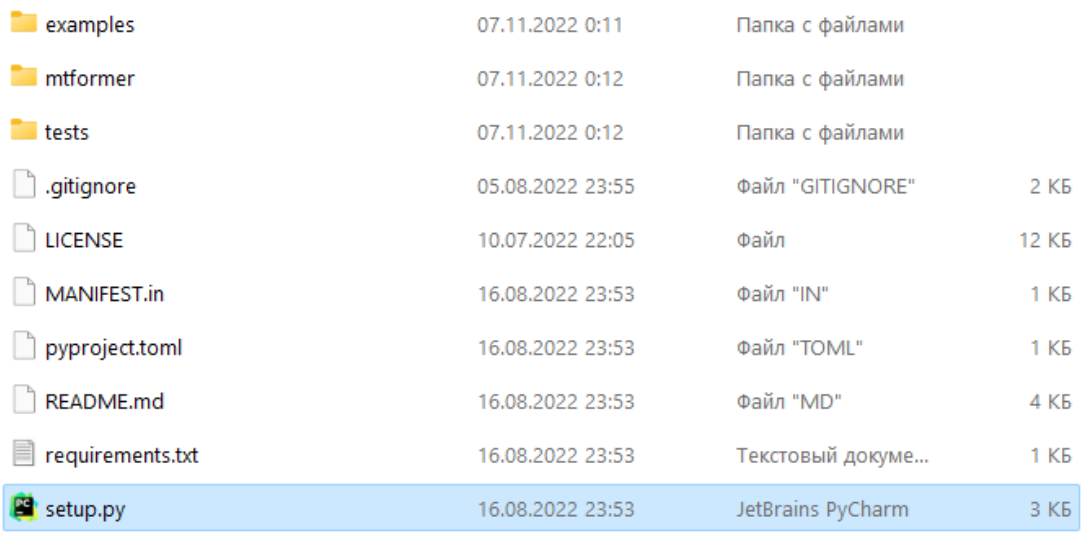

Рисунок 5.1 – Вміст програмного пакету моделювання

Запускати файл встановлення необхідно з терміналу операційної системи за допомогою команди “pip install .” менеджера пакетів середовища Python (потрібно встановити заздалегідь). Під час виконання операції користувач зможе побачити на екрані процес завантаження та встановлення усіх необхідних для функціонування програмного пакету залежностей (рис. 5.2).

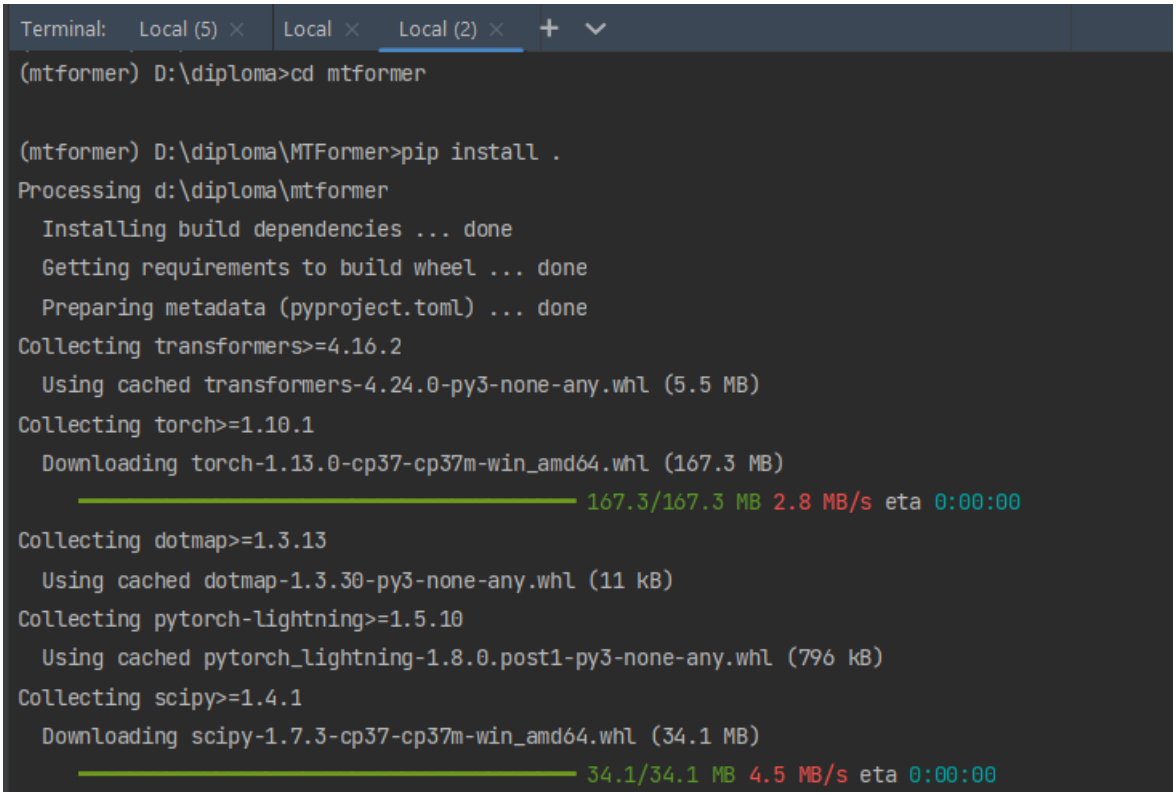

Рисунок 5.2 – Виконання процедури інсталяції програмного пакету

Після закінчення встановлення у терміналі буде зображено повідомлення про успішне завершення операції (рис. 5.3), а у сховищі пакетів Python буде створено директорію з компонентами завантаженого інструмента (рис. 5.4).

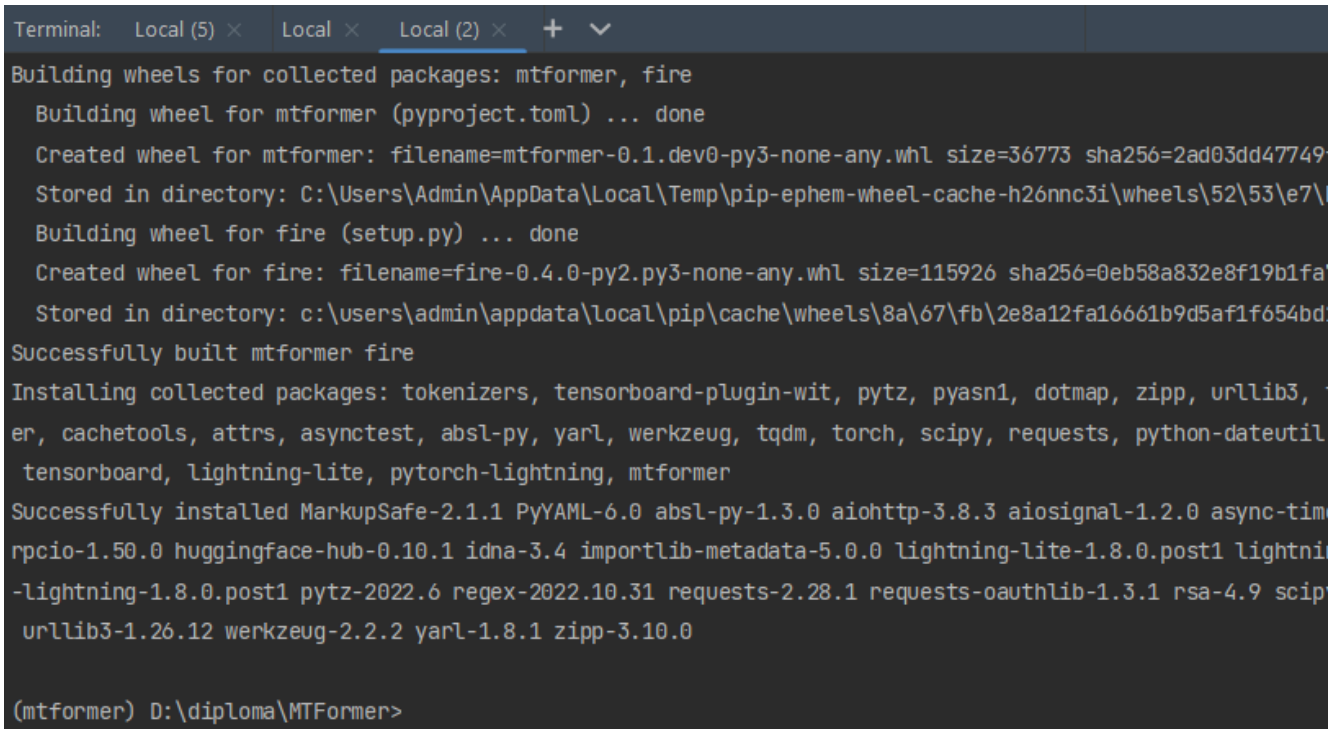

Рисунок 5.3 – Успішне завершення процедури інсталяції

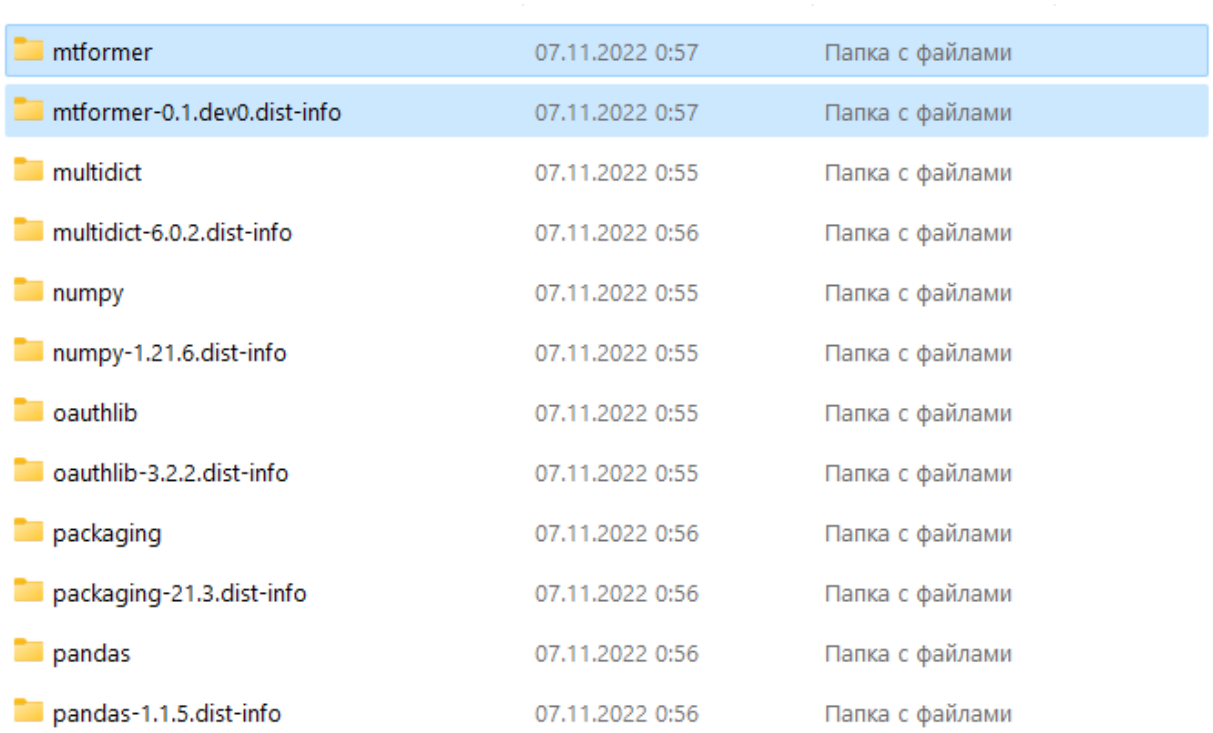

Рисунок 5.4 – Встановлений програмний пакет у локальному репозиторії Python

Опціонально користувач може встановити середовище інтерактивної розробки Jupyter для зручної взаємодії з програмним сервісом моделювання або скористатися будь-якою іншою утилітою, що підтримує розробку на Python (PyCharm, Visual Studio Code та ін.).

5.4 Приклад використання системи

Для тренування моделі необхідно спочатку сформувати набір даних, для цього скористаємося розробленим сервісом для розмітки. У якості тренувального набору було обрано датасет вибору вірогідних альтернатив (CoPA) [61], який треба перекласти з англійської на українську. Для цього необхідно запустити бота командою “/start” та обрати бажаний тип задачі (рисунок 5.5).

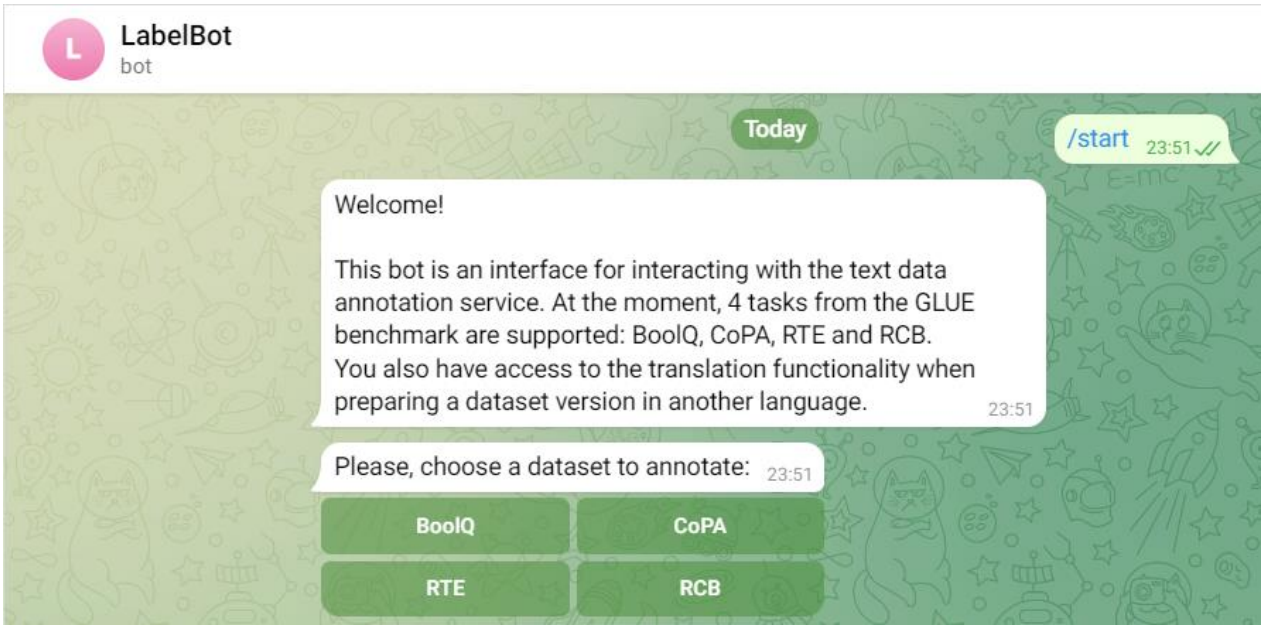

Рисунок 5.5 – Стартове меню сервісу розмітки текстових даних

Результат вибору “CoPA” зображено на рисунку 5.6. Сервіс змінив меню вибору задачі на завдання, що було отримано зі сховища даних. Задача вибору вірогідних альтернатив використовується для оцінки каузального міркування на основі здорового глузду та полягає у тому, що для текстової послідовності наведено дві альтернативні, одна з яких пов’язана з оригінальним текстом, а інша є

випадковою, і, таким чином, необхідно визначити яка з альтернатив має однаковий з оригіналом сенс. Позиції альтернатив перемішуються таким чином, що очікувана ймовірність випадкового вгадування становить 50%. Загалом же датасет вибору вірогідних альтернатив складається з 1000 питань, розділених порівну на тренувальний та тестовий набори.

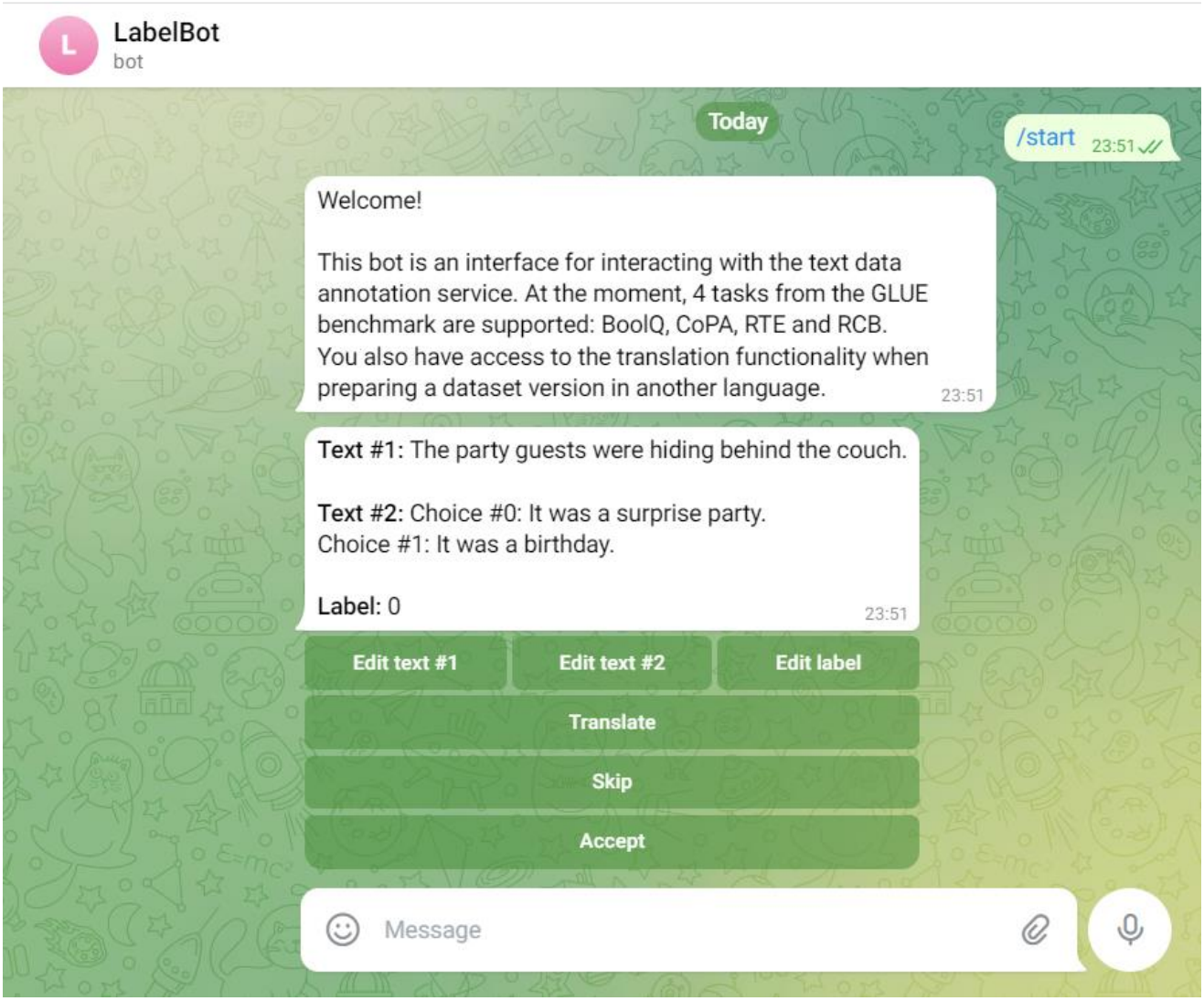

Рисунок 5.6 – Робота над задачею вибору вірогідних альтернатив

Так як ми формуємо українську версію датасету, то текстові компоненти необхідно перекласти з англійської мови, для цього скористаємося кнопкою перекладу. Після цього сервіс відправить перекладене за допомогою хмарного сервісу трансляції завдання (рисунок 5.7).

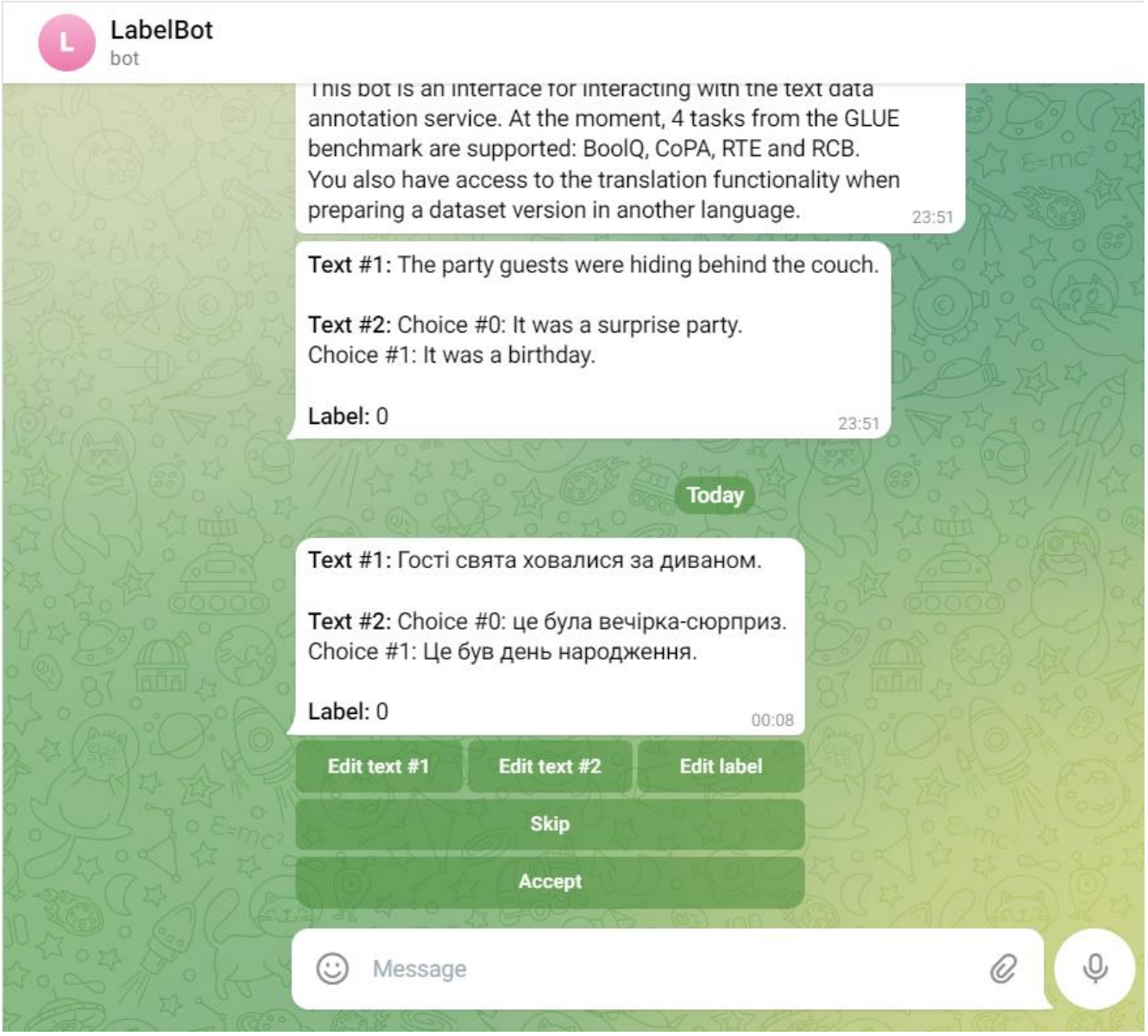

Рисунок 5.7 – Результат використання функції перекладу

З рисунку можна побачити, що в першій альтернативі (Choice #0) речення починається з маленької літери. Для виправлення цієї помилки необхідно скористатися меню редагування розмітки, а саме кнопкою “Edit text2”. Після натиску користувачу буде запропоновано ввести відредагований текст, а далі сервіс поверне відредаговану версію розмітки (рисунок 5.8). Аналогічно доступний функціонал зміни першого текстового поля, а також мітки, вона в даному випадку представлена у вигляді бінарного значення (0 – для першої альтернативи, 1 – для другої). Окрім того, меню редагування також містить кнопку для пропуску поточного завдання, а також його прийняття.

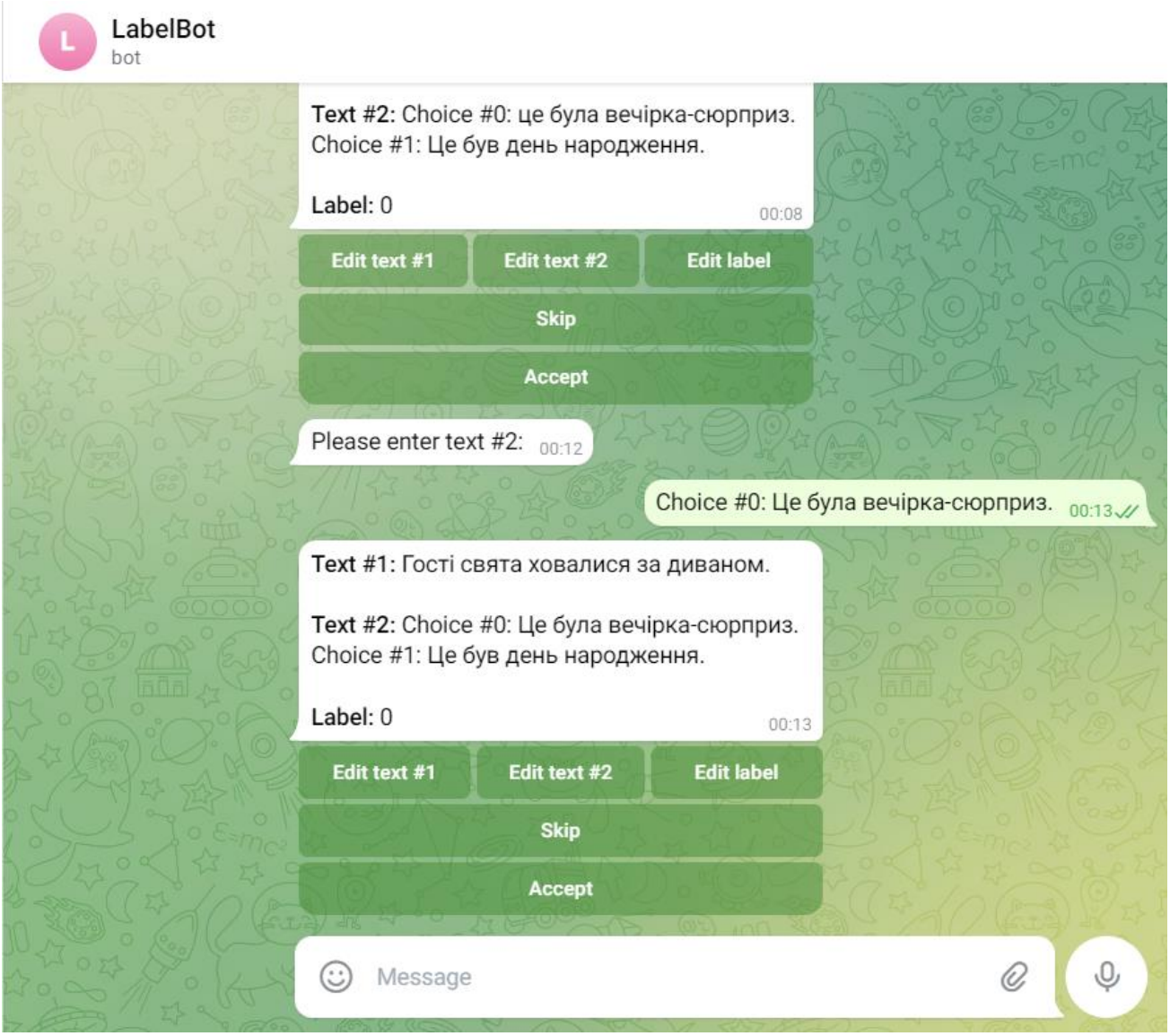

Рисунок 5.8 – Внесення змін у розмітку

Мітка завдання вже є коректною, а тому нам не залишається нічого окрім підтвердження розмітки кнопкою “Accept”. Система у свою чергу зафіксує розмітку у сховищі даних, змінить статус завдання у відповідній таблиці для гарантії неможливості появи дублікатів у майбутньому, а також відправить користувачу нове завдання цього ж типу у випадку його наявності (рисунок 5.9). Додатково у прийнятому завданні проставляється мітка з відповідним повідомленням, а меню редагування замінюється на одну кнопку – “Reject”, що відповідає за відхилення підтвердження у разі випадкового натискання або для зміни розмітки, в такому випадку у цього завдання знову з’явиться меню редагування і його буде встановлено у якості поточного для користувача.

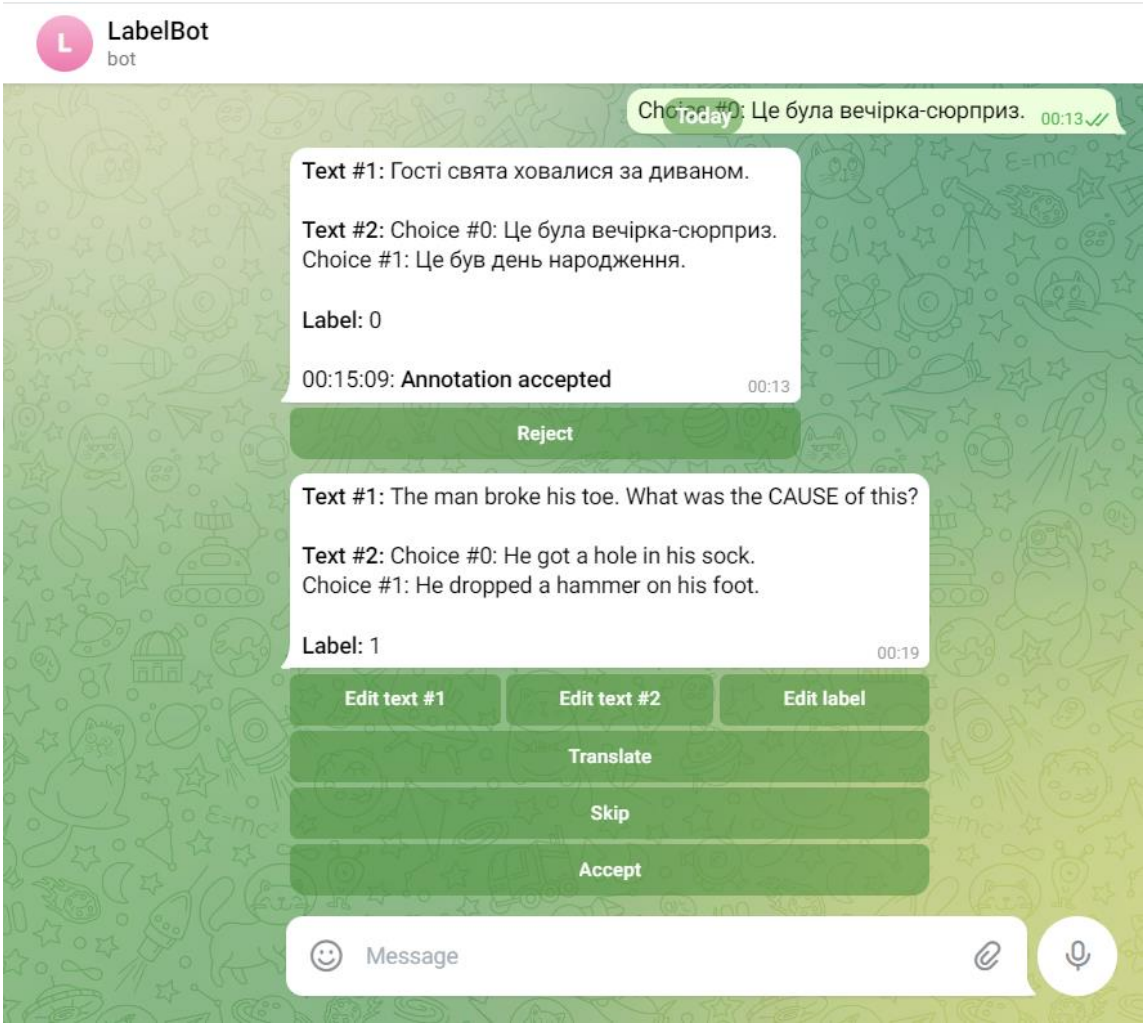

Рисунок 5.9 – Результат підтвердження розмітки

Якщо ж у сховищі даних не лишилося завдань обраного типу, то користувачу буде виведено відповідне повідомлення та буде запропоновано обрати іншу задачу або почекати надходження нових завдань у разі їх занесення у базу даних в ручному чи автоматичному режимі за допомогою API.

Після розмітки необхідної кількості тренувальних прикладів можна перейти до конфігурації моделі глибинного навчання. На рисунку 5.10 зображено вміст конфігураційного файлу TransformerForMT у форматі JSON. Для тренування було обрано архітектуру на базі моделі мови FNet з підтримкою багатозадачного навчання. Можна побачити, що налаштування поділені на 6 секцій, кожна з яких містить специфічні конфігураційні параметри. Секція налаштувань архітектури моделі містить такі параметри, як розмір вхідної послідовності та прихованих шарів, кількість голів самоуваги та шарів, а також різноманітні регуляційні методи.

TransformerForMT.json

```json
{
  "model_type": "FNetForMT",
  "model_tag": "v1-multilingual-cased",
  "model_configs": {
    "dropout_proba": 0.15,
    "max_seq_length": 512,
    "input_size": 768,
    "hidden_size": 64,
    "intermediate_size": 3072,
    "num_layers": 6,
    "num_attention_heads": 4,
    "spec_attention_layers": [4, 5]
  },
  "data": {
    "dataset_path": "dataset/dataset.pt",
    "lazy_preprocessing": true,
    "batch_size": 24,
    "train_set_prop": 0.85,
    "val_set_prop": 0.1
  },
  "optimizer": {"learning_rate": 2e-05, "epsilon": 1e-08},
  "trainer": {
    "accelerator": "gpu",
    "max_epochs": 2,
    "default_root_dir": "training_logs/"
  },
  "tasks": [
    {
      "task_type": "mlm",
      "mask_proba": 0.2,
      "loss": {"loss_type": "cross_entropy", "weighted": false, "reduction": "sum", "name": "mlm_loss"},
      "metrics": ["f1"]
    },
    {
      "task_type": "copa",
      "loss": {"loss_type": "binary_cross_entropy", "weighted": true, "reduction": "avg", "name": "copa_loss"},
      "metrics": ["f1", "precision", "recall"]
    },
    {
      "task_type": "cl",
      "loss": {"loss_type": "triplet_loss", "weighted": false, "reduction": "avg", "name": "cl_loss"},
      "metrics": ["f1"]
    }
  ],
  "notifier": {"notifier_type": "telegram", "user_id": "31125280"}
}
```

Рисунок 5.10 – Налаштування багатозадачної моделі мови

Для налаштування даних використовуються шлях до датасету, тип попередньої обробки, розмір пакету та параметри поділу набору даних на тренувальну та валідаційну вибірки. Процедура тренування також потребує конфігураційних сегментів оптимізаційного алгоритму, де вказується бажана швидкість навчання, і тренера, що налаштовує пристрій, кількість епох тренування,

а також папку для збереження результатів. Окрім того, необхідно визначити і завдання для тренування моделі. У прикладі обрано багатозадачний підхід: одночасне тренування на задачах маскованого мовного моделювання, контрастного навчання, а також вибору вірогідних альтернатив. Додатково для кожної задачі вказуються функції втрат та метрики для оцінювання якості.

Тренувальний набір підготовлено, конфігурацію моделі визначено і тепер можна перейти до процедури тренування з використанням розробленого сервісу моделювання. Для цього скористаємося середовищем інтерактивної розробки Jupyter Notebook (рисунок 5.11).

Training with advanced settings

[2]:
```
from pathlib import Path
from mtformer import set_logging_level, set_logging_formatting
from mtformer.training import run_training
```

[3]:
```
set_logging_level('debug')
```

[4]:
```
path = Path('.').absolute().parents[0]
data_path = path / 'example_data'
model_configs_path = path / 'mtformer' / 'examples' / 'model_configs' / 'TransformerForMT.json'
```

[5]:
```
datasets = {
    'train_mask': {
        'dataset_path': data_path / 'train.pt',
        'mask_proba': 0.2,
        'batch_size': 24,
        'trainer': {
            'max_epochs': 1,
            "default_root_dir": path / "training",
        }
    },
    'train_no_mask': {
        'dataset_path': data_path / 'train_no_mask.pt',
        'mask_proba': 0.0,
        'batch_size': 24,
        'lazy_preprocessing': True,
        'trainer': {
            'accelerator': "cpu",
            "default_root_dir": path / "training",
            'max_steps': 10,
        }
    },
}
```

[6]:
```
run_training(configs_path=model_configs_path, datasets=datasets, evaluate=True, verbose=False)
```

```
Epoch 1: 100%

5/5 [00:00<00:00, 7.65it/s, v_num=0, avg_val_loss=3.3194, avg_val_mlm_loss=7.401, avg_val_copa_loss=0.5192, avg_val_cl_loss=1.289,
avg_mlm_f1=0.482, avg_copa_f1=0.761, avg_copa_precision=0.831, avg_copa_recall=0.696, avg_cl_f1=0.638]
```

Рисунок 5.11 – Процедура тренування багатозадачної моделі

Можна побачити, що перший блок коду на рисунку відповідає за імпортування усіх необхідних для роботи компонентів, у тому числі і сервісу моделювання. Після цього встановлюється рівень логування та декларуються змінні шляхів до датасету та файлу конфігурації моделі. Для демонстрації функціональних можливостей сервісу було також визначено окремі налаштування наборів даних: перше тренування моделі відбуватиметься протягом однієї епохи з маскуванням токенів та на GPU, а друге - вже з вимкнутою задачею маскованого моделювання мови, лінивою обробкою даних та тренуванням протягом 10 ітерацій на CPU. Метод run_training, отримавши конфігурацію моделі, налаштування датасетів та додаткові параметри оцінювання і логування, запустить процедури тренування.

Після закінчення навчання з першою версією налаштувань буде проведено оцінювання на валідаційному наборі з виведенням усіх заданих метрик та функцій втрат (рисунки 5.11), а також буде проведено аналогічну послідовність дій для другої версії конфігурації (рисунок 5.12).

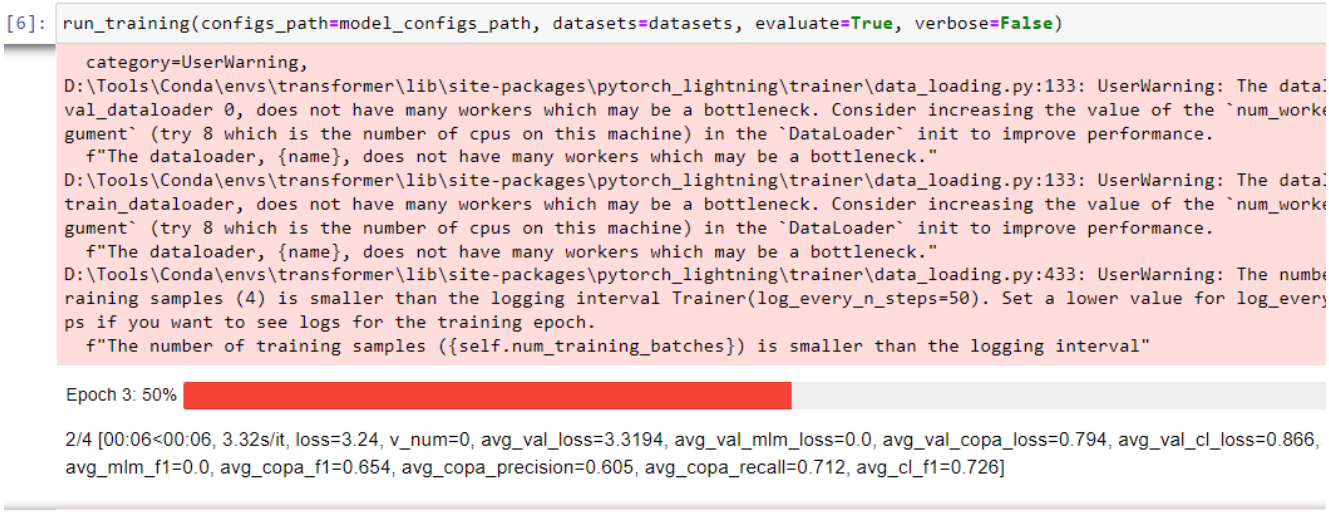

Рисунок 5.12 – Результат оцінювання другої версії налаштувань

На рисунку зображено, що тренування другої версії закінчилося на середині третьої епохи. Це стало результатом встановлення числа 10 у якості параметра максимальної кількості ітерацій: так як кожна епоха містить по чотири ітерації, то

було проведено дві повні епохи тренування (вісім ітерацій) та ще дві ітерації, що становить половину епохи.

У конфігураційному файлі моделі можна помітити поле “notifier”, воно відповідає за взаємодію з сервісом оповіщення користувача про хід процедур тренування та оцінювання моделей. В даному випадку було налаштовано відправку повідомлень в месенджер Telegram відповідному користувачу (вказано його ідентифікатор). В результаті сервіс моделювання надсилатиме інформацію про хід тренування моделі сервісу сповіщення, а той у свою чергу інформуватиме користувача через вказаний у налаштуваннях сервіс. На рисунку 5.13 продемонстровано приклад роботи сервісу сповіщення під час тренування з першою версією налаштувань.

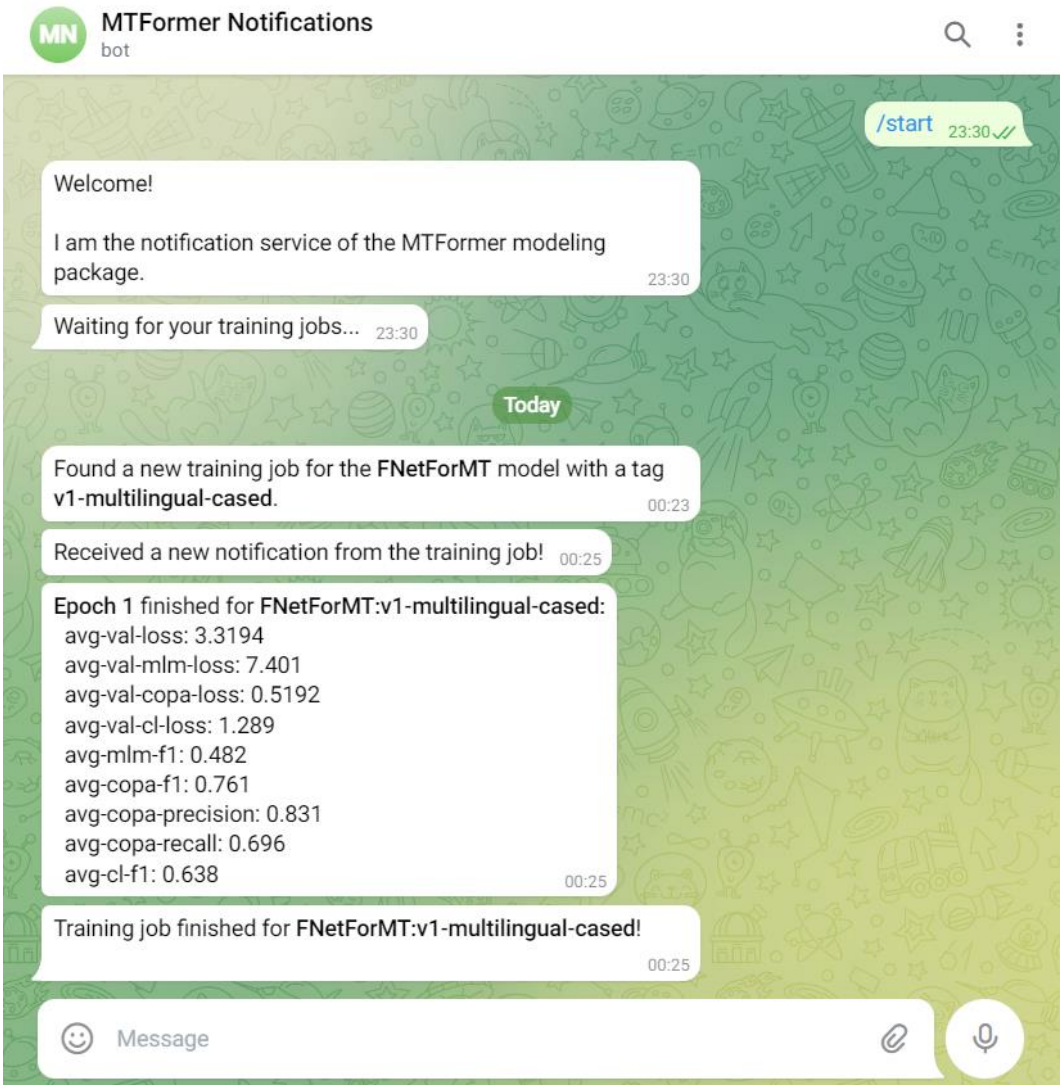

Рисунок 5.13 – Робота сервісу сповіщень під час тренування моделі

Отже, на прикладі тренування моделі у багатозадачному форматі було продемонстровано широкі можливості конфігурації розробленого сервісу моделювання та його взаємодію з сервісом сповіщень, а попередня процедура розмітки даних на практиці підтвердила простоту інтерфейсу користувача реалізованого сервісу текстової розмітки.

5.5 Тестові сценарії

З перевіркою функціональних аспектів програмної розробки допоможуть тест-кейси, для їх формування скористаємося деталізованими у розділі проектування варіантами використання системи.

Підготовлені тестові сценарії описано у таблицях 5.1-5.4. Кожний сценарій містить такі поля: ідентифікатор та назву сценарію, передумову – послідовність кроків, що ініціює тест-кейс, а також стислий опис послідовності дій і результати її виконання.

Таблиця 5.1 – Опис тестового сценарію №1

Ідентифікатор	ТС #1
Назва	Перевірка завантаження моделі сервісом моделювання
Передумова	1. Користувач переходить до середовища розробки. 2. Формує конфігурацію для побудови моделі.
Кроки	1. Користувач імпортує метод завантаження моделі. 2. Викликає метод завантаження моделі.
Очікуваний результат №1 для віддаленої конфігурації з доступною архітектурою моделі	Сервіс обробляє надану конфігурацію для отримання типу архітектури моделі, робить запит до хмарного сховища, завантажує конфігурацію моделі та її ваги, ініціалізує об’єкт моделі за налаштуваннями та повертає його користувачу.
Очікуваний результат №2 для локальної JSON конфігурації з доступною архітектурою моделі	Сервіс обробляє конфігурацію моделі, ініціалізує об’єкт вказаної моделі з використанням наданих налаштувань та повертає його користувачу.

Продовження таблиці 5.1

Очікуваний результат №3 для побудови невідомої архітектури моделі	Сервіс обробляє надану конфігурацію, проводить пошук у локальному репозиторії та хмарному сховищі, виводить повідомлення про відсутність даного типу моделі та демонструє список доступних архітектур.

Таблиця 5.2 – Опис тестового сценарію №2

Ідентифікатор	TC #2
Назва	Перевірка функціоналу тренування моделі
Передумова	1. Користувач зібрав тренувальний набір даних 2. Побудував модель або підготував її конфігурацію (або конфігурації).
Кроки	1. Користувач імпортує метод тренування моделі. 2. Викликає метод тренування моделі.
Очікуваний результат №1 для наданого об’єкта моделі з існуючим датасетом та вказаним шляхом для збереження результатів тренування	Сервіс моделювання завантажує тренувальні дані та проводить їх підготовку відповідно до типу моделі. Далі модель переводиться у тренувальний режим та ініціалізується тренер, якому надається посилання на об’єкт моделі. Сервіс проводить процедуру тренування, зберігає контрольну точку тренувального процесу та повертає об’єкт моделі користувачу.
Очікуваний результат №2 для наданого об’єкта моделі з неіснуючим датасетом	Сервіс моделювання намагається завантажити тренувальні дані, повідомляє користувачу про їх відсутність та повертає помилку.
Очікуваний результат №3 для наданого об’єкта моделі без вказаного шляху для збереження результатів тренування	Сервіс моделювання завантажує тренувальні дані та проводить їх підготовку відповідно до типу моделі. Далі модель переводиться у тренувальний режим та ініціалізується тренер, якому надається посилання на об’єкт моделі. Користувачу демонструється повідомлення-попередження про використання сервісом поточної директорії у якості головної для збереження даних тренування (контрольні точки та файли логування). Сервіс проводить процедуру тренування, зберігає контрольну точку та повертає об’єкт моделі користувачу.

Продовження таблиці 5.2

Очікуваний результат №4 для наданої JSON конфігурації моделі	Сервіс моделювання виконує побудову моделі за JSON конфігурацією одразу у тренувальному режимі, завантажує необхідні тренувальні дані та проводить їх попередню обробку відповідно до вказаних налаштувань. За конфігурацією ініціюється об’єкт тренера та проводиться процедура тренування. Після цього збережена контрольна точка використовується для ініціалізації нового об’єкта моделі, який повертається користувачу одразу після успішного завершення процедури тренування.
Очікуваний результат №5 для наданого списку JSON конфігурацій моделей	Сервіс моделювання будує моделі за відповідними конфігураціями. Якщо у всіх конфігурацій спільний тренувальний набір, то проводиться його підготовка та кешування, інакше кожний набір завантажується перед тренуванням відповідної моделі. Сервіс формує чергу з конфігурацій та у циклі проводить побудову моделі та її тренування. Після кожної операції зберігається контрольна точка, а після завершення усіх процедур сервіс моделювання повертає користувачу агреговані результати.

Таблиця 5.3 – Опис тестового сценарію №3

Ідентифікатор	TC #3
Назва	Перевірка функціоналу завантаження даних
Передумова	1. Користувач формує набір даних. 2. Переходить до середовища розробки.
Кроки	1. Користувач імпортує метод завантаження даних. 2. Викликає метод завантаження даних.
Очікуваний результат №1 для наявного набору даних	Сервіс моделювання завантажує набір даних та повертає його користувачу у вигляді словника
Очікуваний результат №2 для наявного набору даних з вказаною конфігурацією трансформації	Сервіс моделювання завантажує набір даних, виконує зазначену операцію трансформацію та повертає підготовлений об’єкт датасету користувачу
Очікуваний результат №3 для неіснуючого набору даних	Сервіс моделювання намагається завантажити тренувальні дані, повідомляє користувачу про їх відсутність та повертає помилку.

Таблиця 5.4 – Опис тестового сценарію №4

Ідентифікатор	TC #4
Назва	Перевірка функції перекладу контенту завдання з розмітки даних
Передумова	1. Користувач запустив месенджер Telegram. 2. Перейшов до діалогу з ботом розмітки даних.
Кроки	1. Користувач обирає задачу зі списку доступних. 2. Переглядає отримане від сервісу розмітки завдання та натискає кнопку перекладу.
Очікуваний результат №1	Сервіс розмітки отримує встановлену у налаштуваннях мову користувача та робить запит до API сервісу перекладу. Перекладене завдання надсилається користувачу новим повідомленням, а у поточному видаляється меню редагування.
Очікуваний результат №2 для перекладу без встановленої цільової мови	Сервіс розмітки намагається отримати встановлену у налаштуваннях мову користувача та повертає користувачу повідомлення про необхідність вибору цільової мови перекладу і меню налаштувань.

В результаті було сформовано чотири тестові сценарії з високим рівнем деталізації, що допоможе проводити якісне тестування компонентів системи не тільки під час реалізації програмного продукту, але також і на етапі розширення функціональних можливостей готової розробки.

Важливим етапом процесу тестування є побудова матриці відповідності вимог. Зазвичай даний артефакт представляється у вигляді двовимірної таблиці (табл. 5.5). Один вимір матриці займає список варіантів використання системи, інший – тестові сценарії, а використання позначки “+” у якості елемента матриці демонструє покриття відповідним тест-кейсом функціоналу системи.

Таблиця 5.5 – Матриця відповідності вимог

Функціональні вимоги/тести	Тренування моделі	Завантаження моделі	Завантаження даних	Розмітка даних
TC #1	+	+		
TC #2	+	+	+	
TC #3			+	
TC #4				+

З матриці відповідності вимог можна побачити, що здебільшого кожний тестовий сценарій покриває дві функціональні вимоги. Такий розподіл допоможе проводити не тільки тестування методом “чорного ящика”, але й частково полегшить інтеграційне тестування, так як зазначені функціональні вимоги реалізуються на рівні окремих компонентів системи, а сформовані тестові сценарії допомагають їх поєднати.

5.6 Висновки до розділу

У поданому розділі розглянуто усі інструменти та технології, що були використані для реалізації програмної системи, кожний вибір при цьому було аргументовано. Для спроектованого сховища даних було описано процедуру його побудови, а також наведено приклади представлення даних у ньому. Для користувачів деталізовано інструкції з інсталяції компонентів системи та їх подальшого використання. Також було проведено функціональне тестування розробленої системи з подальшою побудовою матриці відповідності вимог.

6 ВИПРОБУВАННЯ СИСТЕМИ ТА ОЦІНКА РЕЗУЛЬТАТІВ

6.1 Підготовка даних для попереднього навчання

Для тренування багатомовних моделей мови було побудовано датасет на базі Tatoeba Translation Challenge, що містить 3708 паралельних текстових корпусів для 557 мов [62], з яких було обрано переклади текстів з англійської, російської, німецької, чеської, польської та білоруської на українську та російську. Розподіл текстів-оригіналів за отриманим набором даних наведено на діаграмі (рис. 6.1).

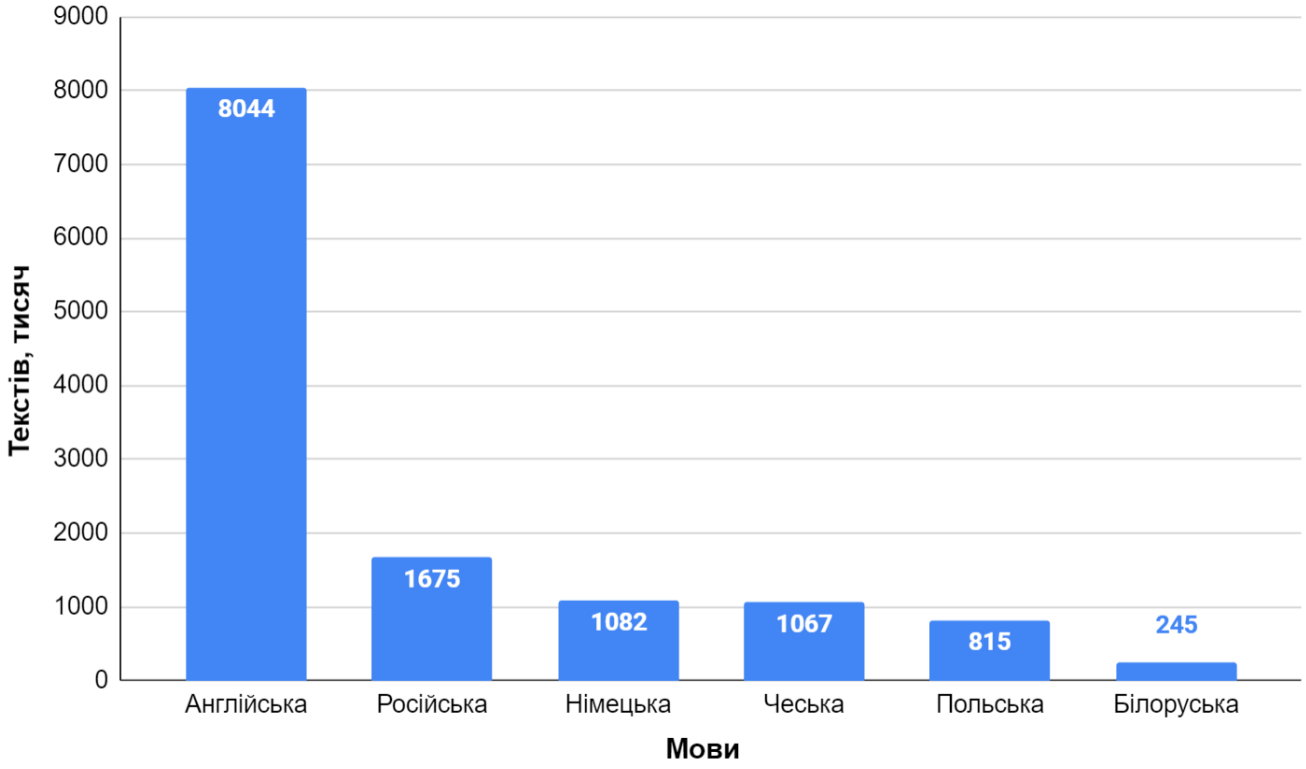

Рисунок 6.1 – Діаграма розподілу текстів-оригіналів датасету за мовами

У свою чергу, текстам-оригіналам, що будуть використані для задач маскованого мовного моделювання та передбачення наступної послідовності, відповідають 7 мільйонів текстів-перекладів російською та 5,9 мільйонів – українською. Переклади будуть використані для задачі багатомовного контрастного навчання, що потребує двох однакових контекстів на різних мовах.

З діаграми розподілу можна побачити, що тексти українською мовою відсутні серед оригіналів. Для вирішення цієї проблеми було проведено заміну частини

іншомовних текстів-оригіналів на відповідні переклади. Результати нормалізації продемонстровано на рисунках 6.2-6.3.

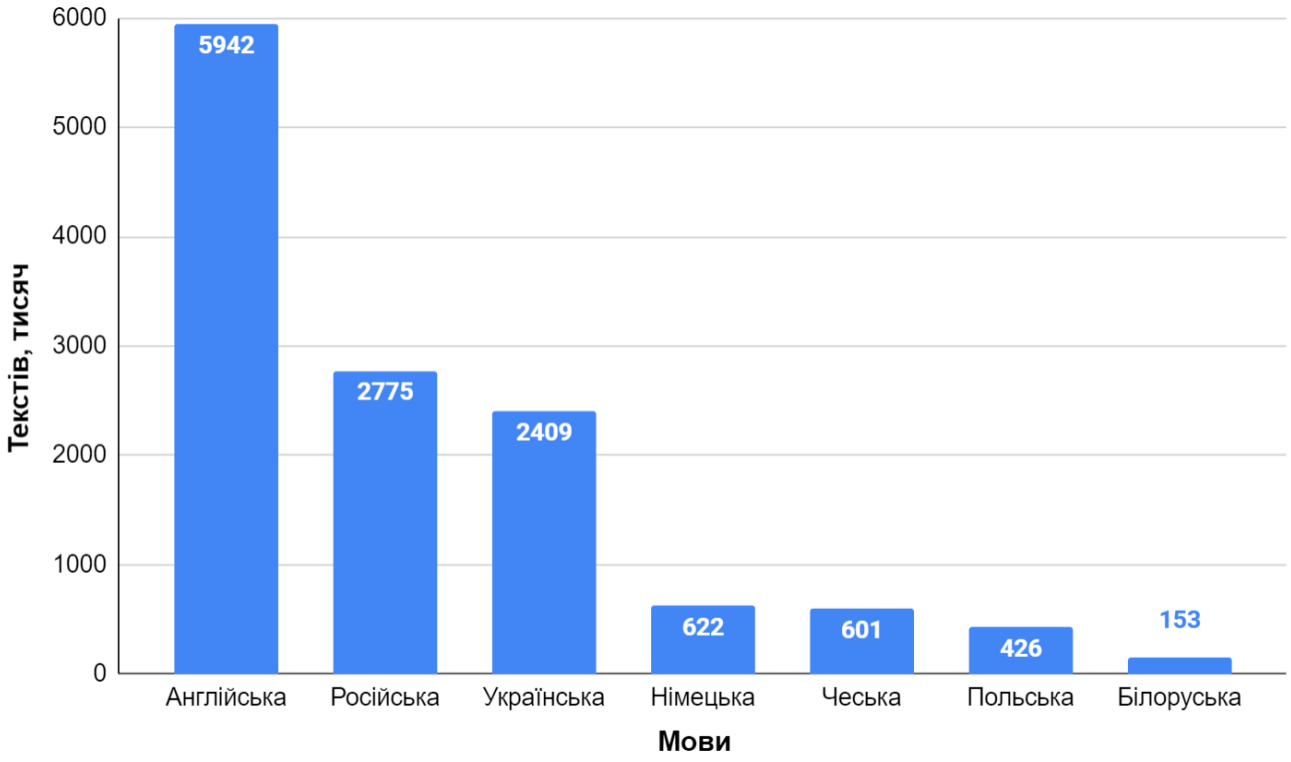

Рисунок 6.2 – Діаграма нормалізованого розподілу текстів-оригіналів за мовами

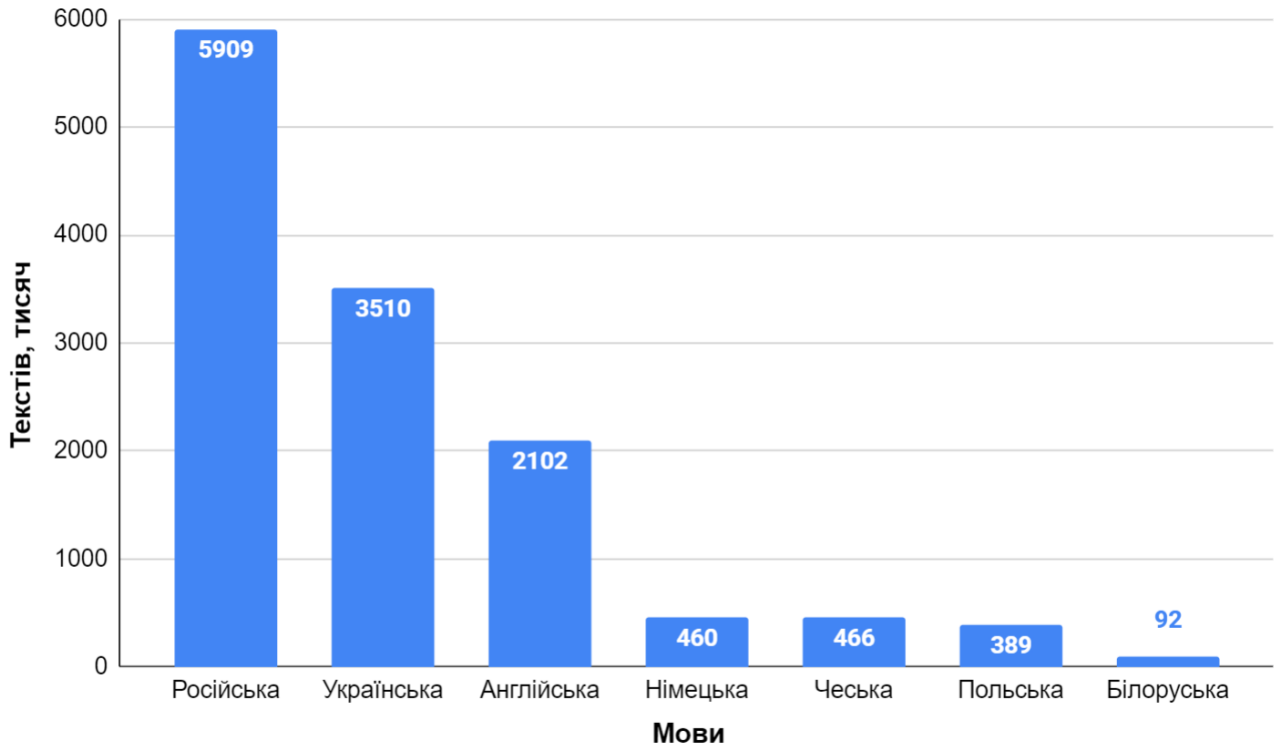

Рисунок 6.3 – Діаграма нормалізованого розподілу текстів-перекладів за мовами

В результаті виконання заміни вдалося розширити тексти-оригінали двома мільйонами прикладів українською та одним мільйоном – російською. Додатковим результатом стало розширення списку мов текстів-перекладів, що може покращити узагальнюючу здібність шляхом багатомовного контрастного навчання.

6.2 Підготовка моделей для попереднього навчання

Для порівняння вдосконаленого методу багатозадачного тренування з існуючими було обрано три варіанти попереднього тренування мережі на базі архітектури FNet: тільки масковане мовне моделювання (MLM), масковане мовне моделювання з передбаченням наступної послідовності (MLM+NSP) та запропонований метод (MLM+NSP+CL), що полягає у додаванні контрастного навчання. В усіх варіантах архітектура мережі складається з 12 шарів кодування (L) розмірністю прихованого шару (D) - 768, а також два останні шари кодування містять по 12 голів самоуваги (H), що становить 24 одиниці загалом. Для усіх трьох моделей було обрано алгоритм токенізації на базі підслів BPE (Byte Pair Encoding) [63] для побудови токенізатора, що трансформує вхідну текстову послідовність у числову з розміром словника (V) - 72000. Окрім того, для порівняння результатів тренування моделей з існуючими рішеннями було обрано багатомовні версії мереж BERT та DistilBERT. Гіперпараметри усіх використаних у роботі моделей наведено у таблиці 6.1.

Таблиця 6.1 – Гіперпараметри моделей

№	Модель	Задачі	V	D	L	H	Параметри, млн.
1	BERT [39]	MLM+NSP	119547	768	12	144	270,4
2	DistilBERT [42]	MLM	119547	768	6	72	227,3
3	FNet-Hybrid	MLM	72000	768	12	24	173,2
4	FNet-Hybrid	MLM+NSP	72000	768	12	24	173,79
5	FNet-Hybrid	MLM+NSP+CL	72000	768	12	24	174

З таблиці можна побачити, що моделі на базі FNet містять набагато менше параметрів для тренування, що пов’язано зі зменшеним розміром словника та використанням механізму самоуваги тільки у останніх двох шарах. Отже, зменшена кількість параметрів дозволить проводити тренування моделей значно швидше, а підготовлені рішення будуть більш універсальними завдяки зменшеним вимогам до оперативної пам’яті та дискового простору.

6.3 Оцінка попереднього навчання моделей

Процедура попереднього навчання обраних моделей глибинного навчання проводиться для кожної з них на відповідних тренувальних задачах одночасно. У якості багатомовних рішень на базі архітектур BERT та DistilBERT були обрані попередньо натреновані їх авторами моделі для 134 мов, а для моделей на базі FNet були проведені процедури навчання після визначення тренувальних гіперпараметрів: кількості ітерацій навчання (I), розміру пакету тренувальних прикладів (BS) та максимальної довжини вхідної послідовності (SL).

Для оцінки попередньо навчених моделей мови скористаємося значеннями функцій втрат задач маскованого мовного моделювання, передбачення наступної послідовності та контрастного навчання на тестовому наборі, а також спеціальною метрикою оцінки якості мовних моделей - перплексивністю.

Перплексивність - це міра того, наскільки добре розподіл імовірності або статистична модель прогнозує вибірку, у випадку моделей мови цей показник можна інтерпретувати як обернену та нормалізовану за кількістю слів ймовірність тестового набору даних [64]. Враховуючи те, що тестовий набір формує коректне представлення мови, а задача моделі мови якраз і полягає у її моделюванні, то мета процедури тренування полягає у зменшені дистанції між реальним та відтвореним розподілами і відповідно у зниженні перплексивності. Загалом же чим більше перплексивність, тим більш заплутаною є мовна модель. Для обчислення даної метрики необхідно піднести експоненту до ступеня перехресної ентропії моделі мови:

$$PP = e^{L_{CE}} = e^{MLM} \quad (6.1)$$

де L_{CE} – перехресна ентропія,

MLM – значення функції втрат маскованого мовного моделювання.

Тренувальні гіперпараметри та результати попереднього навчання багатомовних моделей мови представлені у таблиці 6.2.

Таблиця 6.2 – Параметри та результати попереднього навчання мовних моделей

№	Модель	I, млн.	BS	SL	Значення функцій втрат			PP
					MLM	NSP	CL	
1	BERT	1,1	256	128-512	1,8948	0,36	-	6,6512
2	DistilBERT	-	4000	128-512	3,1977	-	-	24,4762
3	FNet-Hybrid	1,25	24	128-512	3,2382	-	-	25,4878
4	FNet-Hybrid	1,25	24	128-512	3,2227	0,28	-	25,0958
5	FNet-Hybrid	0,75	24	128	3,1760	0,1751	0,2093	23,9508

Результати попереднього навчання демонструють, що версія FNet з додатковою задачею контрастного навчання змогла за значно меншу кількість ітерацій (750 тисяч проти 1,25 мільйони) досягти кращих результатів ніж моделі, що були натреновані з використанням тільки маскованого мовного моделювання або разом із задачею передбачення наступної послідовності протягом 1,25 мільйонів ітерацій. Також модель, натренована з використанням запропонованого методу, виявилася дещо кращою за більшу модель DistilBERT у задачі MLM, окрім того, застосування контрастного навчання покращило результати для задачі передбачення наступної послідовності та навіть дозволило продемонструвати набагато кращий результат за оригінальну версію BERT зі значно більшою кількістю параметрів та довшою процедурою тренування. Проте варто зазначити, що жодна модель не змогла наблизитися до результатів BERT у задачі маскованого мовного моделювання, основною причиною цього стало зменшення кількості голів самоуваги (24 проти 144), що впливають на формування представлення мови на рівні слів. Водночас, контрастне навчання допомогло нівелювати різницю у

кількості голів самоуваги між FNet та DistilBERT (24 проти 72), що підтверджує гіпотезу про відсутність критичного впливу значної кількості голів самоуваги на формування результатів [65].

6.4 Оцінка тонкої настройки моделей

Після оцінювання попереднього тренування мовних моделей необхідно також оцінити їх якість на реальних задачах зі сфери обробки природної мови. Для цього було обрано два діагностичних набори даних: CoPA та SQuAD.

Діагностичний датасет Choice of Plausible Alternatives (CoPA) уже розглядався у рамках даної роботи, його метою є перевірка якості сформованих представлень мови на рівні речень, що на практиці реалізується у форматі задачі бінарної класифікації, де для кожного речення є дві “альтернативи”: перефразоване речення та речення з іншим контекстом. Діагностичний набір є збалансованим та ймовірність випадкового вгадування правильної відповіді становить 50%. Набір CoPA доступний лише для англійської мови, але з використанням розробленого сервісу розмітки його було перекладено і на українську. Підготовлені моделі тестувалися і на англійській і на українській версіях датасету з метою порівняння ефективності рішень для високоресурсних та низькоресурсних мов. Для оцінки результатів передбачення моделей на цій задачі після 10 епох тренування використовувалась метрика точності:

$$Accuracy = \frac{\text{число коректних передбачень}}{\text{загальне число передбачень}} \tag{6.2}$$

Діагностичний набір даних Stanford Question Answering Dataset (SQuAD) використовується для оцінювання здатності моделі машинного навчання розуміти прочитане на рівні слів та словосполучень. Сам датасет складається з текстових параграфів та запитань до них з наявною у контексті відповіддю. Модель при цьому за поданими контекстом та запитанням має надати індекси початку та кінця

текстового фрагменту з контексту, що є найближчою за сенсом відповіддю. Тренування моделей проводилося на наборах SQUAD v2.0 [66] з частиною питань без відповіді та SberQuAD [67] протягом 5 епох, для оцінювання побудованих рішень використовувався тестовий набір SQUAD v1.1 для англійської мови та датасет ручної збірки для української мови [68]. Точність передбачень підготовлених моделей оцінювалася двома метриками: F1 та EM. F1, або середнє гармонійне точності та повноти [69] обчислюється наступним чином:

$$F_1 = 2 * \frac{\text{точність} * \text{повнота}}{\text{точність} + \text{повнота}} \tag{6.2}$$

де $\text{точність} = \frac{\text{істинно-позитивні}}{\text{істинно-позитивні+хибно позитивні}}$,

$\text{повнота} = \frac{\text{істинно-позитивні}}{\text{істинно-позитивні+істинно-негативні}}$.

У задачах відповіді на запитання точність визначається відношенням кількості коректно передбачених слів відповіді до загальної передбаченої кількості слів відповіді, а повнота – відношенням коректно передбачених слів відповіді до загальної кількості слів у істинній відповіді [70]. Метрика EM (exact match) означає точний збіг: при точному збігу передбаченого фрагменту з дійсною відповіддю EM = 1, в усіх інших випадках EM = 0. Результати тонкої настройки моделей представлено у таблиці 6.3.

Таблиця 6.3 – Порівняння результатів тонкої настройки моделей

№	Модель	CoPA		SQuAD			
		$Accuracy_{uk}$	$Accuracy_{en}$	$F1_{uk}$	EM_{uk}	$F1_{en}$	EM_{en}
1	BERT	0,572	0,706	0,831	0,644	0,891	0,818
2	DistilBERT	0,542	0,572	0,753	0,581	0,867	0,795
3	$FNet\text{-}Hybrid_{MLM}$	0,513	0,541	0,746	0,573	0,776	0,684
4	$FNet\text{-}Hybrid_{MLM+NSP}$	0,536	0,566	0,762	0,601	0,783	0,698
5	$FNet\text{-}Hybrid_{MLM+NSP+CL}$	0,556	0,618	0,795	0,668	0,804	0,726

Результати тонкої настройки моделей для задачі CoPA демонструють, що застосування запропонованого методу багатозадачного тренування (модель №5) дозволяє покращити показник точності на 3,73% для української мови та на 9,19% для англійської. При цьому отримана модель виявилася кращою і за рішення на базі DistilBERT, що використовує на 50 мільйонів параметрів більше для процедури попереднього тренування.

Після тонкого налаштування для задачі SQuAD найкращі показники демонструє модель на базі BERT завдяки найбільшій кількості тренувальних параметрів. При цьому рішення на базі FNet, натреновані з використанням задач передбачення наступної послідовності та контрастного навчання, виявилися дещо кращими за DistilBERT для української мови. Це свідчить про те, що формування представлень на рівні речень під час попереднього тренування мовних моделей також відіграє важливу роль у подальшому налаштуванні під такі задачі, як відповідь на запитання за контекстом. Також варто зазначити, що запропонований метод дозволив покращити метрику F1 на 4,5% для української та на 2.7% для англійської мови, показники точного влучання (EM) при цьому збільшилися на 11,2% та 4% відповідно.

Отже, можна зробити висновок, що додавання контрастного навчання до задач маскованого мовного моделювання та передбачення наступної послідовності під час попереднього тренування моделей мови допомагає покращити результати на етапі тонкої настройки і у задачах, що оперують на рівні речень (бінарна класифікація), і у задачах, що фокусуються на рівні слів та словосполучень (відповідь на запитання за контекстом).

6.5 Оцінка продуктивності моделей

Окрім точності моделі, важливою метрикою є і показник її продуктивності – наскільки швидко вона здатна опрацювати вхідну текстову послідовності та сформувати передбачення. Для production-середовища цей показник є чи не найважливішим, адже він напряму впливає на кількість серверного обладнання, що

потребуватиметься для використання підготовленої моделі, а отже і на операційні витрати бізнесу. Окрім того, швидкодія моделі впливає і на процедуру тренування: більша швидкість обробки даних – менше часу та обчислювальних ресурсів витрачається на тренування моделі. Для оцінки швидкості обробки даних розглянутими моделями мови було використано тестовий набір даних для моделювання мови на 10 тисяч текстових прикладів, при цьому використовувалася пакетна обробка даних з розміром пакету 24. Тестування швидкодії проводилося на двох обчислювальних пристроях: графічному прискорювачі NVIDIA RTX 2080TI та центральному процесорі Intel Core i7-9700K 3.6 GHz. Результати тестів наведено у таблиці 6.4.

Таблиця 6.4 – Порівняння продуктивності моделей

№	Модель	Середня швидкість обробки текстових послідовностей, одиниць/с	
		NVIDIA RTX 2080TI	Intel Core i7-9700K 3.6 GHz
1	BERT	184,5869	6,7797
2	DistilBERT	233,0097	9,4862
3	FNet-Hybrid$_{MLM}$	255,0712	8,9261
4	FNet-Hybrid$_{MLM+NSP}$	254,7771	8,9219
5	FNet-Hybrid$_{MLM+NSP+CL}$	252,5263	8,9188

З таблиці порівняння продуктивності моделей можна побачити, що найкращі показники на GPU демонструють рішення на базі архітектури FNet, що цілком очікувано, адже вони мають найменшу кількість тренувальних параметрів та усього два шари самоуваги з обчислювальною складністю $O(N^2)$, де N – довжина послідовності, усі інші шари при цьому завдяки застосуванню алгоритму швидкого перетворення Фур'є не перевищують O(NlogN). На CPU модель на базі архітектури DistilBERT дещо продуктивніша за FNet через значно меншу кількість шарів кодувальника (6 проти 12), проте на цій обчислювальній платформі усі рішення виявилися у десятки разів повільнішими за GPU через те, що у глибинних нейронних мережах використовується значна кількість операцій матричного множення, під яку CPU пристосовані гірше. Також варто зазначити, що

запропонований метод багатозадачного тренування зменшує продуктивність усього на 0,89% для GPU та на 0,04% для CPU.

6.6 Висновки до розділу

У поданому розділі було проведено випробування для оцінки відповідності розробленого продукту поставленій меті, основними критеріями при цьому стали точність побудованих моделей мови після попереднього тренування і після тонкого налаштування під конкретні задачі, необхідна кількість тренувальних ітерацій, а також швидкість обробки текстових даних підготовленими рішеннями.

Результати випробування продемонстрували, що застосування вдосконаленого методу багатозадачного тренування моделей мови дозволяє зменшити необхідну кількість тренувальних ітерацій на 40% без погіршення у якості розуміння природної мови. Також запропоноване рішення після тонкого налаштування покращує точність у задачах бінарної класифікації на 3,73% для української мови та 9,19% для англійської, а у задачі відповіді на запитання за контекстом – на 4,5% та 2,7% відповідно. При цьому продуктивність моделі зменшується лише на 0,89% для GPU та на 0,04% для CPU, що відповідає поставленим вимогам до системи.

За результатами випробування постановлено, що розроблена у рамках виконання кваліфікаційної роботи система відповідає описаним вимогам, а мету у вигляді зменшення необхідної кількості тренувальних кроків та збільшення точності рішень для низькоресурних мов шляхом створення удосконаленого методу багатозадачного навчання моделей мови досягнуто.

ВИСНОВКИ

У рамках виконання даної кваліфікаційної роботи проведено удосконалення методу багатозадачного навчання на базі задач маскованого мовного моделювання, передбачення наступної послідовності і контрастного навчання, на цій основі розроблено повноцінну програмну екосистему, що містить сервіси прототипування, тренування та оцінювання моделей глибинного навчання, розмітки текстових даних, а також оповіщення щодо прогресу у тренуванні і розмітці.

Для реалізації кваліфікаційного проекту було поставлено та вирішено наступні задачі:

1. Проведено аналіз існуючих загальнодоступних рішень для роботи з моделями глибинного навчання у галузі обробки природної мови.

2. Розглянуто наявні інструменти для багатозадачного навчання мовних моделей.

3. Розроблено деталізоване технічне завдання на побудову програмної системи для прототипування, тренування та оцінювання якості мовних моделей, додатково сформовано план роботи та проведено його декомпозицію на стадії з описом очікуваних результатів.

4. Проведено дослідження існуючих підходів до мовного моделювання.

5. Для обраного підходу проведено аналіз архітектур моделей мови, а також задач для їх тренування, включаючи багатозадачні підходи.

6. На базі найпридатніших архітектури та тренувальних задач спроектовано та реалізовано вдосконалений метод багатозадачного навчання багатомовних моделей мови з покращенням точності та зменшенням часу тренування.

7. Візуалізовано та детально описано функціональні та нефункціональні вимоги до продукту.

8. Спроектовано високорівневу архітектуру системи та проведено її декомпозицію на ключові компоненти з описом специфіки їх функціонування.

9. Виконано програмну реалізацію усіх компонентів програмної системи відповідно до сформованих вимог та проведено функціональне тестування.

10. Проведено випробування для оцінки відповідності запропонованого методу багатозадачного навчання моделей мови та розробленого продукту в цілому поставленій меті.

У результаті випробувань було виявлено, що запропонований метод багатозадачного тренування моделей мови на базі задач маскованого мовного моделювання, передбачення наступної послідовності і контрастного навчання дозволяє отримати аналогічний рівень якості розуміння природної мови при зменшенні кількості тренувальних ітерацій на 40% та погіршенню продуктивності моделі лише на 0,89% для GPU та на 0,04% для CPU. Застосування вдосконаленого методу під час попереднього тренування моделей мови також дозволяє покращити їх точність при тонкому налаштуванню під задачі, що оперують представленнями як на рівні окремих слів та словосполучень, так і на рівні речень. Тестування рішення на діагностичному наборі даних CoPA продемонструвало покращення точності на 3,73% для української мови та 9,19% для англійської, а на датасеті SQuAD – зростання на 4,5% та 2,7% відповідно.

Подальший розвиток реалізованої програмної системи включає у себе розширення доступного набору архітектур нейромереж та тренувальних задач з метою покращення точності процедур тонкого налаштування під конкретні задачі з галузі обробки природної мови. Заплановані напрямки роботи над вдосконаленим методом багатозадачного тренування включають у себе проведення процедур тренування моделей зі збільшеним розміром пакету тренувальних прикладів, тестування підготовлених рішень на повному діагностичному наборі GLUE та проведення експериментів з авторегресивними архітектурами нейромереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Louis A. A Brief History of Natural Language Processing — Part 2 [Електронний ресурс] – Режим доступу: https://medium.com/@antoine.louis/a-brief-history-of-natural-language-processing-part-2-f5e575e8e37 (дата звернення: 21.10.2022).

2. Rajpurkar P. The Stanford Question Answering Dataset. [Електронний ресурс] – Режим доступу: https://rajpurkar.github.io/SQuAD-explorer/ (дата звернення: 21.10.2022).

3. Alexandre Magueresse, Vincent Carles, Evan Heetderks. Low-resource Languages: A Review of Past Work and Future Challenges. arXiv:2006.07264.

4. Or Sharir, Barak Peleg, Yoav Shoham. The Cost of Training NLP Models: A Concise Overview. arXiv: 2004.08900.

5. Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew. Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus. arXiv: 2104.08758.

6. L. Bahl, P. Brown, P. de Souza, R. Mercer. A tree-based statistical language model for natural language speech recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 7, pp. 1001-1008, July 1989. DOI: 10.1109/29.32278.

7. Joshua Goodman. A Bit of Progress in Language Modeling. arXiv:cs/0108005.

8. J. Howard, S. Ruder. Universal Language Model Fine-tuning for Text Classification. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 328–339 Melbourne, Australia, July 15 - 20, 2018.

9. Yuri Kuratov, Mikhail Arkhipov. Adaption of Deep Bidirectional Multilingual Transformers for Russian Language. arXiv:1905.07213.

10. Mykyta Syromiatnikov, Victoria Ruvinskaya. Increasing the Performance of the Multilingual Language Model with FNet Architecture / Modern Information Technology 2022 (MIT-2022): Proceedings of the twelfth international conference of young scientists and students, May 19-20, 2022 - pp. 23-25.

11. Mykyta Syromiatnikov, Victoria Ruvinskaya. Natural Language Processing for Intelligent Virtual Assistant System / Information Control Systems and Technologies (ICST-ODESSA-2021), September 23–25, 2021.

12. AI Modeling: Driving Intelligence in Analytics [Електронний ресурс] – Режим доступу: https://www.intel.com/content/www/us/en/analytics/data-modeling.html (дата звернення: 26.11.2022).

13. Transfer learning and fine-tuning [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/tutorials/images/transfer_learning (дата звернення: 26.11.2022).

14. Generalization | Machine Learning | Google Developers [Електронний ресурс] – Режим доступу: https://developers.google.com/machine-learning/crash-course/generalization/video-lecture (дата звернення: 26.11.2022).

15. Fine-Tune a Model – Amazon SageMaker [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-fine-tune.html (дата звернення: 26.11.2022).

16. Karatas G. A Data Annotation: What it is, Why it matters, and Implementations [Електронний ресурс] – Режим доступу: https://research.aimultiple.com/data-annotation/ (дата звернення: 26.11.2022).

17. Dan C. Marinescu, Chapter 11 - Cloud Application Development, Cloud Computing, 2013, Pages 317-359, ISBN 9780124046276, https://doi.org/10.1016/B978-0-12-404627-6.00011-7.

18. Owen-Hill A. Multilingual Software Development: The Key to a Successful Global Product [Електронний ресурс] – Режим доступу: https://rubric.com/en-US/multilingual-software-development/ (дата звернення: 26.11.2022).

19. Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Jamie Brew. HuggingFace's Transformers: State-of-the art Natural Language. Processing (2019). arXiv:1910.03771v4.

20. Flair - A very simple framework for state-of-the-art Natural Language Processing (NLP) [Електронний ресурс] – Режим доступу: https://github.com/flairNLP/flair (дата звернення: 21.10.2022).

21. AllenNLP - An open-source NLP research library, built on PyTorch. [Електронний ресурс] – Режим доступу: https://github.com/allenai/allennlp (дата звернення: 21.10.2022).

22. spaCy: Industrial-strength NLP. [Електронний ресурс] – Режим доступу: https://github.com/explosion/spaCy (дата звернення: 21.10.2022).

23. Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. arXiv: 2009.09796

24. Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. arXiv: 1804.07461

25. M3TL: BERT for Multitask Learning. [Електронний ресурс] – Режим доступу: https://github.com/JayYip/m3tl (дата звернення: 21.10.2022).

26. PaddlePALM. [Електронний ресурс] – Режим доступу: https://github.com/PaddlePaddle/PALM (дата звернення: 21.10.2022).

27. Multi-task-NLP: an utility toolkit enabling NLP developers to easily train and infer a single model for multiple tasks. [Електронний ресурс] – Режим доступу: https://github.com/hellohaptik/multi-task-NLP (дата звернення: 21.10.2022).

28. Daniel Jurafsky, James Martin. Speech and Language Processing. Chapter 3. [Електронний ресурс] – Режим доступу: web.stanford.edu/~jurafsky/slp3/3.pdf (дата звернення: 21.10.2022).

29. Timo Schick, Hinrich Schütze. Few-Shot Text Generation with Natural Language Instructions. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 390–402. November 7–11, 2021.

30. Pandu Nayak. Understanding searches better than ever before. [Електронний ресурс] – Режим доступу: https://blog.google/products/search/search-language-understanding-bert/ (дата звернення: 21.10.2022).

31. Github Copilot - Your AI pair programmer. Режим доступу: https://github.com/features/copilot (дата звернення: 21.10.2022).

32. Markov A. Theory of Algorithms. Imprint Moscow, Academy of Sciences of the USSR, 1954.

33. Yoshua Bengio, Rejean Ducharme, Pascal Vincent, Christian Jauvin. A Neural Probabilistic Language Model. Journal of Machine Learning Research 3 (2003) 1137–1155.

34. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep contextualized word representations. arXiv: 1802.05365.

35. Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215.

36. Carsten Schnober, Steffen Eger, Erik-Lân Do Dinh, Iryna Gurevych. Still not there? Comparing Traditional Sequence-to-Sequence Models to Encoder-Decoder Neural Networks on Monotone String Translation Tasks. arXiv:1610.07796.

37. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473.

38. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. Attention Is All You Need. Proceedings of the NIPS 2017. arXiv:1706.03762.

39. Devlin Jacob, Chang Ming-Wei, Lee Kenton, Toutanova Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (11 October 2018). arXiv:1810.04805v2.

40. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah. Language Models are Few-Shot Learners. arXiv:2005.14165.

41. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683.

42. Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108.

43. Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer. MLP-Mixer: An all-MLP Architecture for Vision. arXiv:2105.01601.

44. J. Lee-Thorp, J. Ainslie, I. Eckstein, S. Ontanon. FNet: Mixing Tokens with Fourier Transforms. arXiv:2105.03824.

45. Nikita Kitaev, Lukasz Kaiser, Anselm Levskaya. Reformer: The Efficient Transformer. arXiv:2001.04451.

46. Iz Beltagy, Matthew Peters, Arman Cohan. Longformer: The Long-Document Transformer. arXiv:2004.05150.

47. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929.

48. Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084.

49. B. Wang, L. Shang, C. Lioma, X. Jiang, H. Yang, Q. Liu, J. Simonsen, On Position Embeddings in BERT. ICLR (2021).

50. Jimmy Ba, Jamie Ryan Kiros, Geoffrey Hinton. Layer Normalization. arXiv:1607.06450.

51. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958.

52. CrossEntropyLoss – Pytorch 1.12 docs. [Електронний ресурс] – Режим доступу: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html (дата звернення: 21.10.2022).

53. Florian Schroff, Dmitry Kalenichenko, James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. arXiv:1503.03832.

54. Alfirna Lahitani, Adhistya Permanasari, Noor Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. 4th International Conference on Cyber and IT Service Management (2016).

55. KDnuggets – Python leads the 11 top Data Science, Machine Learning platforms: Trends and Analysis. [Електронний ресурс] – Режим доступу:

https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html.

56. NeurIPS 2019 – Reproducibility Challenge [Електронний ресурс] – Режим доступу: https://reproducibility-challenge.github.io/neurips2019/resources/ (дата звернення: 21.10.2022).

57. Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX. [Електронний ресурс] – Режим доступу: https://github.com/huggingface/transformers (дата звернення: 21.10.2022).

58. Larman, Craig. Applying UML and Patterns — Third Edition.

59. Martin Fowler. Microservices. [Електронний ресурс] – Режим доступу: martinfowler.com (дата звернення: 21.10.2022).

60. Eric A. Brewer. Towards robust distributed systems (2000). DOI:10.1145/343477.343502.

61. Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning, 2011. Association for the Advancement of Artificial Intelligence (www.aaai.org). [Електронний ресурс] – Режим доступу: http://commonsensereasoning.org/2011/papers/Roemmele.pdf (дата звернення: 21.10.2022).

62. The Tatoeba Translation Challenge. [Електронний ресурс] – Режим доступу: https://github.com/Helsinki-NLP/Tatoeba-Challenge (дата звернення: 21.10.2022).

63. Rico Sennrich, Barry Haddow, Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. arXiv:1508.07909.

64. Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, Yonghui Wu. Exploring the Limits of Language Modeling. arXiv:1602.02410.

65. Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, Ivan Titov. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5797-5808. July 2019.

66. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250.

67. Pavel Efimov, Andrey Chertok, Leonid Boytsov, Pavel Braslavski. SberQuAD -- Russian Reading Comprehension Dataset: Description and Analysis. arXiv: 1912.09723.

68. Dev-human-v2.0. [Електронний ресурс]. – Режим доступу: https://github.com/s-e-r-g-y/context-based-qa-for-uk (дата звернення: 20.10.2022).

69. Scikit-learn – f1_score [Електронний ресурс] – Режим доступу: https://scikitlearn.org/stable/modules/generated/sklearn.metrics.f1_score.html (дата звернення: 20.10.2022).

70. Do-Hyoung Park and Vihan Lakshman. Question Answering on the SQuAD Dataset. [Електронний ресурс] – Режим доступу: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2761899.pdf (дата звернення: 20.10.2022).

ДОДАТОК А. ЛІСТИНГИ ПРОГРАМНОЇ СИСТЕМИ

setup.py

```
import os
import setuptools
_PATH_ROOT = os.path.dirname(__file__)

def parse_requirements() -> dict:
    """parses requirements from requirements.txt"""
    reqs_path = os.path.join(_PATH_ROOT, 'requirements.txt')
    with open(reqs_path, encoding='utf-8') as f:
        reqs = [line.strip() for line in f if not
line.strip().startswith('#')]
    names = []
    links = []
    for req in reqs:
        if '://' in req:
            links.append(req)
        else:
            names.append(req)
    return {'install_requires': names, 'dependency_links': links}

with open('README.md', 'r', encoding='utf-8') as fh:
    long_description = fh.read()

setuptools.setup(
    name="mtformer",
    version="0.1.dev0",
    author="Nikita Syromiatnikov",
    author_email="nik@serafima.ai",
    description="Neural networks training and evaluation tool for natural
language processing (NLP).",
    long_description=long_description,
    long_description_content_type="text/markdown",
    keywords="machine learning deep learning transformer language model nlp
pytorch",
    license="Apache",
    url="https://github.com/niksyromyatnikov/MTFormer",
    project_urls={
        "Bug Tracker":
"https://github.com/niksyromyatnikov/MTFormer/issues",
        "Source Code": "https://github.com/niksyromyatnikov/MTFormer",
    },
    classifiers=[
        "Development Status :: 2 - Pre-Alpha",
        "Intended Audience :: Developers",
        "Intended Audience :: Education",
        "Intended Audience :: Science/Research",
        "Programming Language :: Python :: 3",
        "Programming Language :: Python :: 3.6",
        "License :: OSI Approved :: Apache Software License",
        "Operating System :: OS Independent",
        "Topic :: Scientific/Engineering :: Artificial Intelligence",
    ],
    include_package_data=True,
```

```
    packages=setuptools.find_packages(exclude=('tests', 'utils')),
    python_requires=">=3.6",
    **parse_requirements()
)
```

__init__.py

```
# flake8: noqa
# There's no way to ignore "F401 '...' imported but unused" warnings in
this
# module, but to preserve other warnings. So, don't check this module at
all.

from mtformer.logging import (
    get_logger,
    set_logging_level,
    set_logging_formatting
)
```

logging.py

```
import logging
import os
import sys
import threading
from typing import Optional

_lock = threading.Lock()
_default_handler: Optional[logging.Handler] = None
_default_logging_level = logging.WARNING
_default_logging_formatter = logging.Formatter('%(pathname)s:%(lineno)s:
%(levelname)s: %(message)s')

logging_levels = {
    "critical": logging.CRITICAL,
    "error": logging.ERROR,
    "warning": logging.WARNING,
    "info": logging.INFO,
    "debug": logging.DEBUG,
}

def set_logging_level(log_level: Optional[str] = None):
    global _default_logging_level

    log_level = os.getenv("OHLCFORMER_VERBOSITY", None) if log_level is
None else log_level

    if log_level and log_level in logging_levels:
        _default_logging_level = logging_levels[log_level]
        _reset_lib_root_logger()
        _configure_lib_root_logger()
    else:
        logging.getLogger().warning(
            f"Unknown logging level = {log_level}, expected one of: {',
'.join(logging_levels.keys())}."
        )
```

```
def set_logging_formatting(log_formatting: Optional[str] = None):
    global _default_logging_formatter

    log_formatting = os.getenv("OHLCFORMER_LOG_FORMATTING", None) if
log_formatting is None else log_formatting

    try:
        if isinstance(log_formatting, str):
            _default_logging_formatter = logging.Formatter(log_formatting)
            _reset_lib_root_logger()
            _configure_lib_root_logger()
        elif log_formatting is not None:
            raise TypeError("Incorrect log formatting type {} - expected
string.".format(type(log_formatting)))
    except (ValueError, TypeError) as e:
        logging.getLogger().error(e, exc_info=True)
        raise e

def _get_lib_name() -> str:
    return __name__.split(".")[0]

def _configure_lib_root_logger() -> None:
    global _default_handler

    with _lock:
        if _default_handler:
            return

        _default_handler = logging.StreamHandler()
        _default_handler.flush = sys.stderr.flush
        _default_handler.setFormatter(_default_logging_formatter)

        library_root_logger = logging.getLogger(_get_lib_name())
        library_root_logger.addHandler(_default_handler)
        library_root_logger.setLevel(_default_logging_level)
        library_root_logger.propagate = False

def _reset_lib_root_logger() -> None:

    global _default_handler

    with _lock:
        if not _default_handler:
            return

        library_root_logger = logging.getLogger(_get_lib_name())
        library_root_logger.removeHandler(_default_handler)
        library_root_logger.setLevel(logging.NOTSET)
        _default_handler = None

def get_default_logging_level() -> int:
    return _default_logging_level

def get_default_logging_formatter() -> logging.Formatter:
    return _default_logging_formatter

def get_logger(name: Optional[str] = None) -> logging.Logger:
```

```
    if name is None:
        name = _get_lib_name()

    if not isinstance(name, str):
        raise TypeError("Incorrect logger name type {} - expected
string.".format(type(name)))

    _configure_lib_root_logger()

    return logging.getLogger(name)
```

training.py

```
from pathlib import Path
from typing import Union
from dotmap import DotMap
from mtformer import logging
from mtformer.models import ModelForMT
from mtformer.utils import load_model

logger = logging.get_logger(__name__)

def run_training(model: ModelForMT = None,
                 configs: DotMap = None,
                 configs_path: Union[str, Path] = None,
                 model_dir: Union[str, Path] = None,
                 datasets: dict = None,
                 evaluate: bool = False,
                 verbose: bool = True,
                 ) -> dict:
    eval_result = {}

    logger.info('Running model training.')

    for name, dataset in datasets.items():
        train_configs = {}

        model = model if model is not None else load_model(configs,
configs_path, model_dir)

        if model is None:
            raise ValueError('Expected one of [model object, configs,
configs_path, model_dir] to be provided')

        configs = model.get_configs()

        if isinstance(dataset, str) or isinstance(dataset, Path):
            train_configs['dataset_path'] = dataset
            dataset = train_configs

        if isinstance(dataset, dict):
            for config_name, config_val in dataset.items():
                try:
                    model.set_config(config_name, config_val)
                except TypeError as e:
                    logger.error(f'Failed to set config {config_name} to
{config_val}', exc_info=True)
```

```
                    logger.error(e, exc_info=True)

        model.load_dataset()

        trainer_configs = model.get_configs().get('trainer', {})

        if model_dir is not None:
            trainer_configs['default_root_dir'] = model_dir
        elif trainer_configs.get('default_root_dir', None) is None:
            raise ValueError('Expected model_dir or
trainer.default_root_dir to be provided')

        trainer_configs['default_root_dir'] =
Path(trainer_configs['default_root_dir']) / name

        trainer = model.configure_trainer(**trainer_configs)
        trainer.fit(model)

        if evaluate:
            logger.info(f'Running model evaluation on {name} dataset.')
            eval_result[name] = trainer.test(model, verbose=verbose)
            logger.debug(f'{name} evaluation result: {eval_result[name]}')

        model = None

    if evaluate:
        return eval_result
```

evaluation.py

```
from pathlib import Path
from typing import Union
from dotmap import DotMap
from heapq import nsmallest, nlargest
from mtformer import logging
from mtformer.utils import load_model
from mtformer.losses import get_metric_direction
from mtformer.models import ModelForMT

logger = logging.get_logger(__name__)

def run_tests(tests: dict,
              model: ModelForMT = None,
              configs: DotMap = None,
              configs_path: Union[str, Path] = None,
              model_dir: Union[str, Path] = None
              ) -> dict:
    logger.info(f'Running model evaluation on {len(tests)} tests.')

    model = model if model is not None else load_model(configs,
configs_path, model_dir)

    if model is None:
        raise ValueError('Expected one of [model object, configs,
configs_path, model_dir] to be provided')

    result = {}
```

```
    default_model_configs = model.get_configs()

    trainer = model.configure_trainer()

    for name, test in tests.items():
        test_configs = {}

        if isinstance(test, str) or isinstance(test, Path):
            test_configs['dataset_path'] = test
        elif isinstance(test, dict):
            for config_name, config_val in test.items():
                if config_name == 'dataset_path':
                    test_configs['dataset_path'] = config_val
                    continue
                try:
                    model.set_config(config_name, config_val)
                except TypeError as e:
                    logger.error(f'Failed to set config {config_name} to
{config_val}', exc_info=True)
                    logger.error(e, exc_info=True)

        model.load_dataset({'test_dataset': test_configs['dataset_path']})
        result[name] = trainer.test(model, verbose=False)
        model.set_configs(default_model_configs)

    return result

class TestResult(object):
    def __init__(self, score: float, model_name: str):
        self.score = score
        self.model_name = model_name

    def __lt__(self, other):
        return self.score < other.score

    def __str__(self):
        return f'{self.model_name}: {self.score}'

    def __repr__(self):
        return f'{self.model_name}: {self.score}'

def compare_models(tests: dict, models: dict, top_k=0) -> (dict, dict):
    if not tests or not models:
        raise ValueError('Expected at least one test and one model!')

    top_k = len(models) if top_k <= 0 else top_k

    logger.info(f'Running {len(tests)} tests on {len(models)} models
outputting top-{top_k} results for each test.')

    results = {name: {} for name in tests.keys()}
    top = {name: {} for name in tests.keys()}

    for model_name, model in models.items():
        tests_result = run_tests(tests, **model)
        logger.debug(f'{model_name} evaluation results: {tests_result}.')
```

```
        for test_name, test_result in tests_result.items():
            if not test_result:
                continue

            result = test_result[0]

            for metric_name, score in result.items():
                if results[test_name].get(metric_name, None) is None:
                    results[test_name][metric_name] = []
                results[test_name][metric_name].append(TestResult(score,
model_name))

    for test_name in top.keys():
        top[test_name] = {
            metric_name: nsmallest(top_k, results[test_name][metric_name])
if not get_metric_direction(metric_name)
            else nlargest(top_k, results[test_name][metric_name]) for
metric_name in results[test_name].keys()
        }

    return (top, results) if top_k != len(models) else top
```

configuration_utils.py

```
import os
import json
from pathlib import Path
from typing import Union
from dotmap import DotMap
from mtformer import logging
from mtformer.models import Model
from mtformer.models.builder import ModelBuilder

logger = logging.get_logger(__name__)

def load_model(configs: DotMap = None,
               configs_path: Union[str, Path] = None,
               model_dir: Union[str, Path] = None
               ) -> Model:
    logger.info('Loading model.')

    configs = load_model_configs(configs_path) if configs is None and
configs_path is not None else configs

    if configs is not None:
        model = load_from_configs(configs)
    elif model_dir is not None:
        model = load_from_dir(model_dir)
    else:
        raise ValueError('Expected one of [model configs, configs_path,
model_dir] to be provided')

    return model

def save_model_configs(configs, model_dir: Union[str, Path]):
    if configs is None:
        raise TypeError('Expected configs object but got None instead')
```

```
    if not isinstance(model_dir, Path):
        model_dir = Path(model_dir)

    configs_path = model_dir / 'configs.json'

    logger.info(f'Saving model configs to {configs_path}.')

    with open(configs_path, 'w', encoding='utf8') as json_file:
        json.dump(configs, json_file, indent=2)

def load_model_configs(path: Union[str, Path]):
    configs = None
    default_configs_file = 'configs.json'

    if not isinstance(path, Path):
        path = Path(path)

    try:
        if path.is_dir() and 'configs.json' in os.listdir(path):
            logger.warning(f'Model configs directory specified instead of
file. Looking for the default configs file '
                           f'{default_configs_file}.')
            path /= default_configs_file
        if not path.is_file():
            raise FileNotFoundError(f'Model configs file {path.as_posix()}
not found.')

        logger.info(f'Loading model configs from {path}.')

        with open(path, 'r', encoding='utf8') as json_file:
            configs = DotMap(json.load(json_file))
    except Exception as e:
        logger.error(f'Error occurred while loading model configs from
{path}.', exc_info=True)
        logger.error(e, exc_info=True)

    return configs

def load_from_configs(configs):
    logger.info('Loading model from configs.')

    if configs is None:
        raise TypeError('Expected configs object but got None instead')

    model = ModelBuilder.build(configs)

    return model

def load_from_dir(model_dir: Union[str, Path]):
    if not isinstance(model_dir, Path):
        model_dir = Path(model_dir)

    logger.info(f'Loading model from {model_dir}.')

    configs = load_model_configs(model_dir)
```

```
    if configs is None:
        raise FileNotFoundError('No configs found in ' +
model_dir.as_posix())

    checkpoint = find_checkpoint(model_dir)
    if checkpoint is not None:
        configs.checkpoint = checkpoint
        logger.info(f'Loading model from checkpoint {configs.checkpoint}.')

    model = ModelBuilder.build(configs)

    return model

def find_checkpoint(model_dir: Union[str, Path]):
    if not isinstance(model_dir, Path):
        model_dir = Path(model_dir)
    checkpoint_dir = model_dir / 'checkpoints/'

    logger.info(f'Looking for model checkpoint in {checkpoint_dir}.')

    checkpoint = None

    try:
        for file in os.listdir(checkpoint_dir):
            file = str(file)
            if file.endswith('.ckpt'):
                checkpoint = checkpoint_dir / file
    except Exception as e:
        logger.error(f'Error occurred while looking for model checkpoint in
{checkpoint_dir}.', exc_info=True)
        logger.error(e, exc_info=True)

    return checkpoint
```

losses/builder.py

```
import copy
from dotmap import DotMap
from mtformer import logging

logger = logging.get_logger(__name__)
losses_implementations_dict = {}

def register_loss(name: str = None) -> type:
    """
    This decorator function is used to register (store in a dictionary)
classes of implemented loss architectures.
    Args:
        name (str): The key of the loss class in a dictionary. Can be
derived from the class name if no name is
            specified.
    Returns:
        type: Type of the registered class.
    """
    def decorate(cls: type, loss_name: str = None) -> type:
        global losses_implementations_dict
```

```
        if not loss_name:
            loss_name = cls.__module__ + '.' + cls.__name__
        if loss_name in losses_implementations_dict:
            logger.warning(f"Loss class {loss_name} is already registered
and will be overwritten!")

        losses_implementations_dict[loss_name] = cls
        return cls

    return lambda cls: decorate(cls, name)

class LossBuilder(object):
    """
    This class is a factory for losses creation.
    Attributes:
        losses_dict (dict): Dictionary with loss name as a key and class as
a value.
    """
    losses_dict = losses_implementations_dict

    @classmethod
    def build(cls, configs: DotMap):
        """
        Creates loss object using loss type specified in configs.
        Args:
            configs (DotMap): Object contains parameters required for loss
usage.
        Raises:
            KeyError: No architecture found for the specified loss type.
        Returns:
            Loss: Created object of loss
        """
        conf = copy.deepcopy(configs)
        loss_type = conf.pop('loss_type')
        try:
            return cls.losses_dict[loss_type](**conf)
        except KeyError:
            logger.error(f'No implementation found for the specified loss
type {loss_type}.')
            raise KeyError("{} loss not implemented!".format(loss_type))
        except TypeError:
            logger.error(f'Incorrect loss type {type(loss_type)}.')
            raise TypeError("Loss type is not specified!")
```

losses/loss.py

```
import torch
from torch import nn

class Loss(nn.Module):
    def __init__(self, weighted=False, reduction='sum', name=None):
        super(Loss, self).__init__()
        self.weighted = weighted
        self.reduction = reduction
        self.name = name if name is not None else self.__class__.__name__
```

```
    def forward(self, prediction: torch.Tensor, target: torch.Tensor, mask,
ignored_index=0, reduction=None):
        pass


def get_metric_direction(metric_name) -> bool:
    if metric_name == "f1":
        return True
    return False
```

models/builder.py

```
from dotmap import DotMap
from mtformer import logging

logger = logging.get_logger(__name__)
models_implementations_dict = {}

def register_model(name: str = None) -> type:
    """
    This decorator function is used to register (store in a dictionary)
classes of implemented model architectures.
    Args:
        name (str): The key of the model class in a dictionary. Can be
derived from the class name if no name is
            specified.
    Returns:
        type: Type of the registered class.
    """
    def decorate(cls: type, model_name: str = None) -> type:
        global models_implementations_dict

        if not model_name:
            model_name = cls.__module__ + '.' + cls.__name__
        if model_name in models_implementations_dict:
            logger.warning(f'Model class {model_name} is already registered
and will be overwritten!')

        models_implementations_dict[model_name] = cls
        return cls

    return lambda cls: decorate(cls, name)

class ModelBuilder(object):
    """
    This class is a factory for models creation.
    Attributes:
        models_dict (dict): Dictionary with model name as a key and class
as a value.
    """
    models_dict = models_implementations_dict

    @classmethod
    def build(cls, configs: DotMap):
        """
        Creates model object using model type specified in configs.
        Args:
```

```
            configs (DotMap): Object contains parameters required for model
usage.
        Raises:
            KeyError: No architecture found for the specified model type.
        Returns:
            Model: Created object of model
        """
        try:
            return cls.models_dict[configs.model_type](configs)
        except AttributeError as e:
            logger.error(e, exc_info=True)
            raise AttributeError(f'Unsupported configuration object type
{type(configs)}.')
        except KeyError:
            logger.error(f'No implementation found for the specified model
type {configs.model_type}.')
            raise KeyError("{} architecture not
implemented!".format(configs.model_type))
        except TypeError:
            logger.error(f'Incorrect model type
{type(configs.model_type)}.')
            raise TypeError("Model type is not specified!")
```

models/model.py

```
import torch
import pytorch_lightning as pl
from dotmap import DotMap
from collections import defaultdict
from copy import deepcopy
from torch.utils.data import DataLoader
from mtformer.data import DataProcessor, mask_tokens
from mtformer.losses import LossBuilder, default_loss
from . import Model

class ModelForLM(pl.LightningModule):

    def __init__(self, configs):
        super(ModelForLM, self).__init__()
        self.configs = configs

        self.model_device = None
        self.device_name = None
        self.set_device(trainer=configs.get('trainer', None))

        self.net = Model(configs, self.model_device)
        self.losses = []
        for loss_config in configs.get('losses', default_loss):
            self.losses.append(LossBuilder.build(loss_config))
        self.data_processor = DataProcessor(configs)
        self.metrics = configs.get('metrics', [])

        self.load_checkpoint(configs)
        self.load_weights(configs)

        self.train_loader = None
        self.val_loader = None
        self.test_loader = None
```

```
    def forward(self, **params):
        device_params = {k: v.to(self.model_device) if v is not None else v
for k, v in params.items()}
        outputs = self.net(**device_params)
        return outputs

    def prepare_batch(self, batch) -> tuple:
        if not self.configs.get('lazy_preprocessing', False):
            return batch

        input_ids, attention_mask = batch[:2]

        args = {'input_ids': input_ids, 'attention_mask': attention_mask}
        if self.configs.get('max_seq_length', None) is not None:
            args['seq_len'] = self.configs['max_seq_length']
        if self.configs.get('mask_proba', None) is not None:
            args['mask_proba'] = self.configs['mask_proba']
        if self.configs.get('prediction_len', None) is not None:
            args['prediction_len'] = self.configs['prediction_len']

        input_ids, labels, mask = mask_tokens(**args)
        return [input_ids, attention_mask, mask, labels] + batch[2:]

    def training_step(self, batch, batch_nb) -> dict:
        input_ids, attention_mask, mask, labels =
self.prepare_batch(batch)[:4]
        outputs = self.forward(input_ids=input_ids,
attention_mask=attention_mask)
        pooler_output = outputs[1]
        # print(pooler_output, pooler_output.shape)
        return self.calculate_losses(pooler_output, labels, mask)

    def validation_step(self, batch, batch_nb) -> dict:
        input_ids, attention_mask, mask, labels =
self.prepare_batch(batch)[:4]
        outputs = self.forward(input_ids=input_ids,
attention_mask=attention_mask)
        pooler_output = outputs[1]
        return self.calculate_losses(pooler_output, labels, mask,
stage='val')

    def validation_epoch_end(self, outputs):
        avg_loss, avg_metrics = self.aggregate_metrics(outputs,
stage='val')
        tensorboard_logs = {**{'avg_val_loss': avg_loss}, **avg_metrics}
        self.log_dict(tensorboard_logs, prog_bar=True)
        # return {'avg_val_loss': avg_loss, 'progress_bar':
tensorboard_logs}

    def test_step(self, batch, batch_nb) -> dict:
        input_ids, attention_mask, mask, labels =
self.prepare_batch(batch)[:4]
        outputs = self.forward(input_ids=input_ids,
attention_mask=attention_mask)
        pooler_output = outputs[1]
```

```
        return self.calculate_losses(pooler_output, labels, mask,
stage='test')

    def test_epoch_end(self, outputs):
        avg_loss, avg_metrics = self.aggregate_metrics(outputs,
stage='test')
        tensorboard_logs = {**{'avg_test_loss': avg_loss}, **avg_metrics}
        self.log_dict(tensorboard_logs, prog_bar=True)

        # return {'avg_test_loss': avg_loss, 'progress_bar':
tensorboard_logs}

    def configure_optimizers(self):
        return torch.optim.Adam([p for p in self.parameters() if
p.requires_grad],
                                lr=self.configs.optimizer.learning_rate,
                                eps=self.configs.optimizer.epsilon)

    def train_dataloader(self) -> DataLoader:
        return self.train_loader

    def val_dataloader(self) -> DataLoader:
        return self.val_loader

    def test_dataloader(self) -> DataLoader:
        return self.test_loader

    def load_dataset(self, dataset_path=None) -> tuple:
        dataset_path = dataset_path if dataset_path is not None else
self.configs.dataset_path
        dataset = self.data_processor.prepare_dataset(dataset_path,
self.configs.train_set_prop, self.configs.val_set_prop,
self.configs.test_set_prop, self.configs.batch_size)
        self.train_loader = dataset[0] if dataset[0] is not None else
self.train_loader
        self.val_loader = dataset[1] if dataset[1] is not None else
self.val_loader
        self.test_loader = dataset[2] if dataset[2] is not None else
self.test_loader
        return self.train_loader is not None, self.val_loader is not None,
self.test_loader is not None

    def calculate_losses(self, output: torch.Tensor, labels: torch.Tensor,
mask: torch.Tensor, stage: str = '') -> dict:
        calculated_losses = {}
        for loss in self.losses:
            name = (stage + '_' if stage is not None and len(stage) > 0
else '') + loss.name
            if isinstance(loss, MaskedDirectionLoss):
                loss_val, f1 = loss(output.detach(),
labels.to(self.model_device), mask.to(self.model_device),
                                    return_f1=True)
                calculated_losses[name] = loss_val
                calculated_losses['_'.join(loss.name.split('_')[:-1] +
['f1'])] = f1
            else:
```

```
                calculated_losses[name] = loss(output,
labels.to(self.model_device), mask.to(self.model_device))
                calculated_losses[name + '_mean'] = loss(output,
labels.to(self.model_device),
mask.to(self.model_device), reduction='mean')
        return calculated_losses

    def aggregate_metrics(self, outputs: list, stage='val'):
        avg_loss = torch.stack([x[stage + '_loss'] for x in
outputs]).mean()
        avg_metrics = defaultdict(list)
        for x in outputs:
            for k, v in x.items():
                key = 'avg_' + k
                if isinstance(v, torch.Tensor):
                    avg_metrics[key].append(v)
                else:
                    avg_metrics[key].append(torch.tensor(v))
        for k, v in avg_metrics.items():
            avg_metrics[k] = torch.stack(v).mean()
        return avg_loss, avg_metrics

    def load_checkpoint(self, configs):
        if configs.get('checkpoint', None) is not None:
            try:
self.load_state_dict(torch.load(configs.checkpoint)['state_dict'])
            except Exception as e:
                print(e)
                print('Failed to load checkpoint from
{}'.format(configs.checkpoint))

    def load_weights(self, configs):
        if configs.get('weights_path', None) is not None:
            try:
                self.load_state_dict(torch.load(configs.weights_path))
            except Exception as e:
                print(e)
                print('Failed to load weights from
{}'.format(configs.weights_path))

    def get_configs(self):
        return deepcopy(self.configs)

    def set_config(self, configs_name, configs_value):
        self.configs[configs_name] = configs_value

    def set_configs(self, configs):
        self.configs = deepcopy(configs)

    def get_device(self, return_str=True, return_pytorch=False):
        if return_str:
            return self.device_name
        elif return_pytorch:
            return self.model_device

    def set_device(self, device: str = None, trainer: DotMap = None):
        self.device_name = get_device_type(device, trainer)
```

```
        self.model_device = torch.device(self.device_name)
        self.to(self.model_device)

    def configure_trainer(self, **kwargs):
        trainer_configs = deepcopy(self.configs.trainer)
        for k, v in kwargs.items():
            trainer_configs[k] = v
        self.set_device(trainer=trainer_configs)
        trainer_configs['accelerator'] =
get_accelerator_type(get_device_type(trainer=trainer_configs))

        if trainer_configs.get('devices', None) is None:
            trainer_configs['devices'] = "auto"
        trainer = pl.Trainer(**trainer_configs)
        return trainer

def get_device_type(device: str = None, trainer: DotMap = None):
    device = trainer.get('accelerator', None) if trainer else device
    return device if device in ['cpu', 'cuda', 'xla'] else 'cuda' if
torch.cuda.is_available() else 'cpu'

def get_accelerator_type(device):
    replacements = {'xla': 'tpu', 'cuda': 'gpu'}
    for x, y in replacements.items():
        device = device.replace(x, y)
    return device if device in ['cpu', 'gpu', 'tpu'] else 'gpu' if
torch.cuda.is_available() else 'cpu'
```

data/processor.py

```
import torch
from pathlib import Path
from typing import Union
from torch.utils.data import random_split, RandomSampler, DataLoader
from mtformer import logging

logger = logging.get_logger(__name__)

class DataProcessor:
    def __init__(self, configs=None):
        pass

    def load_dataset(self, dataset_path: Union[str, Path, dict]) -> tuple:
        dataset, val_dataset, test_dataset = None, None, None

        if isinstance(dataset_path, str) or isinstance(dataset_path, Path):
            logger.info(f'Loading dataset from {dataset_path}')
            dataset = torch.load(dataset_path)

        elif isinstance(dataset_path, dict):
            if dataset_path.get('train_dataset', None) is not None:
                logger.info(f'Loading train dataset from
{dataset_path["train_dataset"]}')
                dataset = torch.load(dataset_path['train_dataset'])

            if dataset_path.get('val_dataset', None) is not None:
                logger.info(f'Loading validation dataset from
{dataset_path["val_dataset"]}')
```

```
                val_dataset = torch.load(dataset_path['val_dataset'])

            if dataset_path.get('test_dataset', None) is not None:
                logger.info(f'Loading test dataset from
{dataset_path["test_dataset"]}')
                test_dataset = torch.load(dataset_path['test_dataset'])

        return dataset, val_dataset, test_dataset

    def prepare_dataset(self, dataset_path: str, train_set_split_prop:
float = 0.87, val_set_split_prop: float = 0.13,
                        test_set_split_prop: float = 0.0, batch_size: int =
32):
        train_dataloader, val_dataloader, test_dataloader = None, None,
None
        train_subset, val_subset, test_subset = None, None, None
        logger.info(f'Preparing dataset from {dataset_path}.')
        dataset, val_dataset, test_dataset =
self.load_dataset(dataset_path)
        nb_train_samples = int(train_set_split_prop * len(dataset)) if
dataset is not None else 0
        nb_val_samples = int(val_set_split_prop * len(dataset)) if dataset
and val_dataset is None else 0
        nb_test_samples = len(dataset) - nb_train_samples - nb_val_samples
if dataset and test_dataset is None else 0

        if dataset is not None:
            dl = len(dataset)
            nb_train_samples += dl - nb_train_samples - nb_val_samples -
nb_test_samples if nb_train_samples > 0 else 0
            nb_val_samples += dl - nb_train_samples - nb_val_samples -
nb_test_samples if nb_val_samples > 0 else 0
            nb_test_samples += dl - nb_train_samples - nb_val_samples -
nb_test_samples if nb_test_samples > 0 else 0

            logger.debug('Samples distribution:')
            logger.debug(f'{nb_train_samples} training samples.')
            logger.debug(f'{nb_val_samples} validation samples.')
            logger.debug(f'{nb_test_samples} test samples.')

            train_subset, val_subset, test_subset = random_split(dataset,
[nb_train_samples, nb_val_samples, nb_test_samples])
            train_sampler = RandomSampler(train_subset)
            train_dataloader = DataLoader(train_subset,
sampler=train_sampler, batch_size=batch_size)

        if nb_val_samples and val_subset:
            val_sampler = RandomSampler(val_subset)
            val_dataloader = DataLoader(val_subset, sampler=val_sampler,
batch_size=batch_size)
        elif val_dataset is not None:
            val_dataloader = DataLoader(val_dataset, shuffle=True,
batch_size=batch_size)

        if nb_test_samples and test_subset:
            test_sampler = RandomSampler(test_subset)
```

```
            test_dataloader = DataLoader(test_subset, sampler=test_sampler,
batch_size=batch_size)
        elif test_dataset is not None:
            test_dataloader = DataLoader(test_dataset,
batch_size=batch_size)
        return train_dataloader, val_dataloader, test_dataloader
```

data/features.py

```
import random
import torch
from torch.utils.data import TensorDataset
from mtformer import logging

logger = logging.get_logger(__name__)

class InputFeatures:
    """
    A single set of features of data.
    Property names are the same names as the corresponding inputs to a
model.
    """

    def __init__(self, input_ids, attention_mask, mask=None, labels=None):
        self.input_ids = input_ids
        self.attention_mask = attention_mask
        self.mask = mask
        self.labels = labels

def convert_to_tensor_dataset(dataset: list,
                              max_seq_len: int = 2000,
                              mask_proba: float = 0.2,
                              prediction_len: int = 5,
                              perform_masking: bool = True,
                              min_seq_len_coeff: float = 0.3,
                              seed: int = None
                              ) -> TensorDataset:
    logger.info(f'Converting dataset with {len(dataset)} rows to tensors
{"with masking" if perform_masking else ""}.')

    if seed is not None:
        logger.debug(f'Setting custom seed={seed}.')
        rand = random.Random(seed)
    else:
        rand = random

    features = []
    labels_pad_len = int(max_seq_len * (mask_proba + 0.05)) if mask_proba >
0 else 0

    for row in dataset:
        input_ids = row.copy()
        seq_len = len(input_ids)
        pad_len = max_seq_len - seq_len

        if seq_len < prediction_len + min_seq_len_coeff * seq_len:
            continue
        attention_mask = [1] * max_seq_len
```

```
        attention_mask[seq_len:] = [0] * pad_len

        if perform_masking:
            mask = [0] * max_seq_len
            labels = []
            vals = [True, False]
            weights = [1 - mask_proba, mask_proba]
            for i, elem in enumerate(input_ids):
                if i >= seq_len - prediction_len or not rand.choices(vals,
weights)[0]:
                    labels.append(elem)
                    input_ids[i] = [0] * len(elem)
                    mask[i] = 1

            labels = labels + [[0] * 4] * (labels_pad_len - len(labels) if
labels_pad_len - len(labels) > 0 else 0)
            input_ids += [[0] * len(input_ids[0])] * pad_len
            features.append(InputFeatures(input_ids, attention_mask, mask,
labels))
        else:
            input_ids += [[0] * len(input_ids[0])] * pad_len
            features.append(InputFeatures(input_ids, attention_mask))

    if perform_masking:
        return TensorDataset(torch.tensor([f.input_ids for f in features],
dtype=torch.float), torch.tensor([f.attention_mask for f in features],
dtype=torch.long), torch.tensor([f.mask for f in features],
dtype=torch.long), torch.tensor([f.labels for f in features],
dtype=torch.float))
    else:
        return TensorDataset(torch.tensor([f.input_ids for f in features],
dtype=torch.float), torch.tensor([f.attention_mask for f in features],
dtype=torch.long))

def mask_tokens(input_ids: torch.Tensor,
                attention_mask: torch.Tensor,
                seq_len: int = 2000,
                mask_proba: float = 0.2,
                prediction_len: int = 5
                ) -> tuple:
    batch_size = input_ids.shape[0]
    element_len = input_ids.shape[-1]

    masked_input_ids = input_ids.clone()
    mask = torch.rand(batch_size, seq_len, device=input_ids.device) <
mask_proba
    mask[attention_mask == 0] = 0

    labels_pad_len = torch.max(torch.sum(mask, 1)).item() + prediction_len
    labels_batch = torch.zeros(batch_size, labels_pad_len, element_len,
device=input_ids.device)
    end_of_seq = torch.argmin(attention_mask, 1)
    end_of_seq[(attention_mask[:, 0] == 1) & (end_of_seq == 0)] = seq_len

    for i, row in enumerate(input_ids):
        mask[i][end_of_seq[i]-prediction_len:end_of_seq[i]] = 1
        labels = row[mask[i]]
```

```
        labels_batch[i][:labels.shape[0]] = labels
    masked_input_ids[mask] = torch.zeros(1, element_len,
device=input_ids.device)
    return masked_input_ids, labels_batch, mask.long()
```