Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів глибокого навчання у вирішенні задач граматичної корекції текстів
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-23-1
Максим Харченко
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник проф. Наталія Рябова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системи штучного інтелекту
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Харченку Максиму Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів глибокого навчання у вирішенні задач граматичної корекції текстів

затверджена наказом університету від 21 квітня 20 25 р. № 295Ст

2. Термін подання студентом роботи до екзаменаційної комісії 4 червня 20 25 р.

3. Вихідні дані до роботи Науково-технгічні публікації, документація мови Python, модель ModernBERT, набір даних W&I+Locness, набір даних CoEdIT, документація до бібліотек Pytorch, Transformers

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та існуючих рішень
2) Теоретичні дослідження
3) Експериментальні дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	21.04.2025	виконано
2	Дослідження предметної галузі	25.04.2025	виконано
3	Постановка задачі	26.04.2025	виконано
4	Теоретичні дослідження існуючих методів	01.05.2025	виконано
	вирішення задачі		
5	Розробка архітектури системи	04.05.2025	виконано
6	Навчання моделі	10.05.2025	виконано
7	Написання пояснювальної записки	19.05.2025	виконано
8	Нормоконтроль	28.05.2025	виконано
9	Перевірка на академічний плагіат	29.05.2025	виконано
10	Підготовка презентації та доповіді	02.06.2025	виконано
11	Захист перед ЕК	04.06.2025	

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Наталія Рябова
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 71 с., 31 рис., 4 табл., 1 дод., 23 джерела.

ГРАМАТИЧНА КОРЕКЦІЯ ТЕКСТІВ, ОБРОБКА ПРИРОДНОЇ МОВИ, ПОСЛІДОВНІСТЬ ДО ВИПРАВЛЕНЬ, СИНТЕТИЧНІ ДАНІ, ТРАНСФОРМЕРИ.

Об’єкт дослідження – процес граматичної корекції текстів.

Предмет дослідження – методи глибокого навчання у вирішенні задач граматичної корекції текстів.

Мета роботи – розробка гібридної моделі граматичної корекції текстів.

Методи дослідження – сучасні методи машинного та глибокого навчання, дослідження, порівняльний аналіз та вдосконалення можливостей архітектур трансформерних моделей глибоких нейромереж щодо вирішення поставлених задач.

Розроблено архітектуру нейронної мережі «послідовність до виправлень» для вирішення задачі граматичної корекції текстів, що дозволяє більш ефективно використовувати ресурси для роботи моделі. Розроблена нейронна мережа поєднана з алгоритмічною словниковою системою виявлення орфографічних помилок для підвищення якості та передбачуваності роботи системи.

В результаті проведених досліджень навчено нейронну мережу, яка вирішує задачу граматичної корекції текстів. Навчена нейромережа поєднана з алгоритмічною системою, що дає змогу отримувати більш стабільні та передбачувані результати для користувача.

Розроблену архітектуру доцільно використовувати й для інших задач обробки природної мови, де необхідна зміна вхідної послідовності, таких як, наприклад, перефразування текстів.

ABSTRACT

Master’s thesis contains: 71 pp., 31 fig., 4 tabl., 1 ann., 23 references.

GRAMMATICAL ERRORS CORRECTION, NATURAL LANGUAGE PROCESSING, SEQUENCE TO EDITS, SYNTHETIC DATA, TRANSFORMERS.

The object of the research is the process of grammatical errors correction.

The subject of the research is deep learning methods for solving problems of grammatical correction of texts.

The purpose of the research is development hybrid model for grammatical errors correction.

The methods of the research are modern methods of machine and deep learning, research, comparative analysis, and improvement of the capabilities of deep neural network transformer model architectures in addressing the stated tasks.

The sequence-to-edits architecture of neural network for solving grammatical errors correction task, which allows to use resources more efficiently, was developed. Developed neural network is combined with algorithmic dictionary-based spell-checking system to increase quality and explainability of the system.

Based on the developed architecture the neural network is trained to solve grammatical errors correction task. Network is combined with an algorithmic approach into a hybrid system which allows to gain more stable and robust results for the user.

Developed architecture can be used for other natural language processing tasks where it is necessary to modify the input sequence, such as text rephrasing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BERT – Bidirectional Encoder Representations from Transformers – двонаправленні представлення енкодера від трансформерів;

BPE – Byte Pair Encoding – кодування пар байтів;

GEC – Grammatical Errors Correction – граматична корекція текстів;

LSTM – Long Short-Term Memory – довга короткочасна пам’ять;

NLP – Natural Language Processing – обробка природної мови;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

Seq2Edits – Sequence to Edits – послідовність до виправлень;

Seq2Seq – Sequence to Sequence – послідовність до послідовності.

ВСТУП

Один із основних способів комунікації між людьми в мережі інтернет є текстові повідомлення. Люди використовують текст для висловлювання, як під час особистого переписування, так і для написання статей, наукових робіт тощо. Особливістю текстової комунікації є те, що вона містить безліч правил, у тому числі граматичних, що відрізняються від однієї мови до іншої. Зважаючи на це, люди часто допускають помилки при написанні текстів, що може псувати враження читача, або, навіть, змінювати увесь сенс написаного. Для запобіганню цього, почали створюватись системи граматичної корекції текстів. Перші такі системи не використовували технології штучного інтелекту, а базувались на алгоритмічних методах обробки тексту. Якість таких рішень є не дуже високою, тому такі системи завжди покращують. З розвитком технологій штучного інтелекту, зокрема й технологій обробки природної мови, ці технології почали використовувати для вирішення задачі граматичної корекції текстів. Використання технологій штучного інтелекту значно покращило якість роботи таких систем.

Хоча, останні розробки у галузі обробки природної мови й надали значне покращення якості роботи алгоритмів граматичної корекції текстів, такі системи все одно мають ряд проблем, над вирішенням яких досі працюють. Це є проблеми швидкості роботи таких систем, проблема непередбачуваності результатів роботи системи та інші. В цій роботі досліджено створення системи, яка більш ефективно використовує обчислювальні ресурси при вирішенні задач граматичної корекції текстів. Також буде досліджено поєднання такої системи з алгоритмічною системою знаходження помилок для підвищення якості роботи системи та підвищення передбачуваності роботи системи для користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз актуальності проблеми

Ще до розвитку глобальних інтернет технологій одним із основних способів комунікації були текстові дані. З розвитком технологій, текстова комунікація залишається одним із головних способів людей висловлювати свої думки під час комунікації, шляхом написання статей, наукових робіт, тощо. Однією із перешкод у цьому способі є те, що люди є схильні до створення помилок у своїх текстах, які можуть бути створені через складність правил у мові, або через неуважність, особливо, якщо людина пише не своєю рідною мовою, тобто знання цієї мови недосконалі. Такі помилки можуть спотворювати написаний текст залишаючи погане враження, або зовсім змінювати сенс написаного, саме тому, однією із задач обробки природної мови (Natural Language Processing) є граматична корекція текстів.

Перші способи вирішення цієї проблеми не включали в себе машинне навчання, а покладались лише на алгоритми. Наприклад, системи знаходження та виправлення орфографічних помилок за допомогою словників, або системи граматичної корекції текстів використовуючи правила, описані експертами. Але, через складність опису природної мови за допомогою алгоритмів, такі способи не давали змоги знаходити складні помилки, які можливо виправити лише аналізуючи контекст написаного [1]. Тому, з розвитком машинного навчання, його методи почали використовуватися для вирішення задачі граматичної корекції текстів. Створення моделей, типу послідовність до послідовності (seq2seq) [2], а потім і великих мовних моделей (Large Language Models) надало поштовх до вирішення цієї задачі.

Але, хоча методи глибокого навчання й демонструють високу точність у розпізнаванні та виправленні помилок, а також здатність

знаходження неочевидних залежностей, вони мають свої недоліки. Однією із проблем є можлива зміна сенсу під час роботи системи автоматичної корекції помилок. Це особливо критично там, де від формулювань дуже залежить контекст, що передається, наприклад, у наукових статтях, медицині, документах тощо.

Іншою проблемою є непередбачуваність результатів роботи автоматичної системи виправлення помилок [3]. Нелінійність роботи систем глибокого навчання має свій недолік у тому, що одна й та сама помилка може бути виправлена та не виправлена, або виправлена в інший спосіб у різних контекстах, що є неочевидним для користувача системи. Неможливість впливати на результат системи іншими способами, окрім як покращення на більш ширшому наборі даних, також є проблемою.

Ще однією проблемою є обчислювальні потужності, що витрачаються на корекцію текстів. Так, наприклад, хоча великі мовні моделі й показують дуже хороші результати у вирішення задачі граматичної корекції текстів, ціна їх використання досі залишається досить високою, що робить недоцільним їх використання у системах виправлення помилок у реальному часі. Тому, одним із завдань також є вдосконалення архітектур моделей таким чином, щоб використовувати менше обчислювальних потужностей, зберігаючи якість роботи системи.

Беручи до уваги описані вище проблеми, можна сказати, що проблема граматичної корекції текстів є досі актуальною та потребує знаходження нових методів її вирішення. Напрямками досліджень має бути розробка нових архітектур систем, що дозволяють знайти баланс між якістю, стійкістю та кількістю необхідних ресурсів для використання системи. Ще одним напрямком є вдосконалення підходів глибокого навчання, таким чином, щоб зробити системи більш передбачуваними, для того, щоб надавати користувачу пояснення разом із самими виправленнями. Також напрямком вдосконалення є додавання можливості контролю над результатами роботи системи шляхом персоналізації, тощо.

1.2 Аналіз предметної галузі

1.2.1 Аналіз типів можливих помилок у текстах

Різноманіття можливих помилок, що можуть бути вчинені у тексті є дуже великим. Хоча задача граматичної корекції текстів (Grammatical Error Correction) і включає в свою назву лише граматику, насправді проблематика цієї задачі є більш ширшою та включає в себе виправлення орфографічних та інших видів помилок. Якщо подивитись на те, як сервіси для виправлення помилок в тексті розділяють виправлені помилки на типи, то можна побачити, що на найвищому рівні виділяють три типи помилок:

– граматичні;

– орфографічні;

– стилістичні.

Зазвичай, ці типи помилок виділяються різними кольорами в таких системах, як «Word» або інші. Орфографічні помилки виділяються на рівні слів, та відповідають за неправильне написання слова. Слово може бути неправильно написане по різним причинам, через те, що автор не знає як правильно його написати, або просто випадково, так як набирання тексту на клавіатурі призводить до того, що певне слово написано неправильно. Стилістичні помилки є більш широким та складним класом помилок та стосуються невідповідності тексту певним мовним стандартам. На відміну від інших типів помилок, ці помилки є більш суб’єктивними, оскільки в різних сталях написання допускають різні формулювання. До стилістичних помилок відносяться зокрема:

– тавтології;

– надмірно складні фрази;

– невідповідність стилю.

Граматика, зазвичай є найширшою категорією помилок, до якої відносять усі помилки, що не підходять під інші категорії. До граматики

можуть відноситись як правила узгодження слів, відмінювання, пунктуаційні помилки, використання великих літер, тощо. Виправлення таких помилок ускладняється тим, що для їх знаходження потребується глибоке розуміння контексту. Далі, кожен з описаних типів може ділитись на безліч підтипів, зокрема для того, щоб надати можливість створити опис для кожного підтипу помилки, завдяки якому можливо пояснити користувачу, чому саме це є помилкою та потребує виправлення.

Хоча задача граматичної корекції текстів і зосереджується на граматичних помилках, як на самій складній частині, але також охоплює виправлення й інших помилок. Поділення помилок на групи дозволяє виправляти їх різними способами та краще аналізувати.

1.2.2 Методи граматичної корекції текстів

Перші методи корекції текстів були засновані на програмних алгоритмах. Для виправлення орфографічних помилок використовуються системи, в основі яких лежить словник. Так як орфографічні помилки стосуються саме слів, то першим елементом такої системи повинно бути розбиття вхідного тексту на окремі слова, та окрема обробка кожного слова. Словникові системи дуже просто можуть знаходити помилки, якщо певного слова не існує у заданому словнику, то таке слово є неправильно написаним. Авжеж, словники не є досконалими, тому можливі хибні спрацювання системи. Знаходження виправлень для неправильно написаних слів є більш складною задачею. Для цього потрібно знайти подібні слова у наявному словнику, для чого використовуються такі алгоритми, як відстань Левенштейна, методи фонетичної подібності, тощо. Перевагами таких методів є їх передбачуваність, швидкість та стійкість, неправильне слово буде знайдено у будь-якому контексті, а якщо слово не потребує виправлень, воно легко може бути додано до словнику користувачем тим самим розвиваючи систему під час її роботи. Недоліками таких систем є

низька якість виправлень, так як для створення виправлень не аналізується контекст.

Виправлення граматичних та стилістичних помилок є більш складним завданням, але і це завдання намагаються вирішити використовуючи алгоритмічні методи. Для цього були створені системи, що працюють на правилах (rule-based systems), що задаються експертами. Такі системи зазвичай містять в собі такі елементи:

– лінгвістичний аналізатор;

– база правил;

– модуль генерації виправлень.

Лінгвістичний аналізатор виконує аналіз вхідного речення, розпізнавання частин мови, відмінків, часів, тощо. Правила описуються лінгвістичним експертом використовуючи семантичну інформацію, як частини мови. Таким чином, знання лінгвістичного експерта вдається генералізувати у вигляді правил, які можливо застосовувати для різних вхідних текстів. Модуль генерації виправлень застосовує правила із бази правил до тексту, проаналізованого лінгвістичним аналізатором, що дозволяє знайти помилки та отримати виправлення до них. Повна схема роботи системи граматичної корекції текстів, заснованої на правилах, можна побачити на рисунку 1.1. Перевагою таких систем є:

– передбачуваність;

– можливість контролю.

Недоліками ж є низька якість запропонованих виправлень, адже деякі помилки потребують глибокого аналізу контексту речення та їх складно записати у вигляді правила. Деякі правила мають велику кількість виключень, що також ускладнює розробку такої системи. Також недоліком є складність розробки, так як кожне окреме правило потрібує детального аналізу експертом, який повинен розроблювати нові правила та підтримувати вже існуючу базу правил.

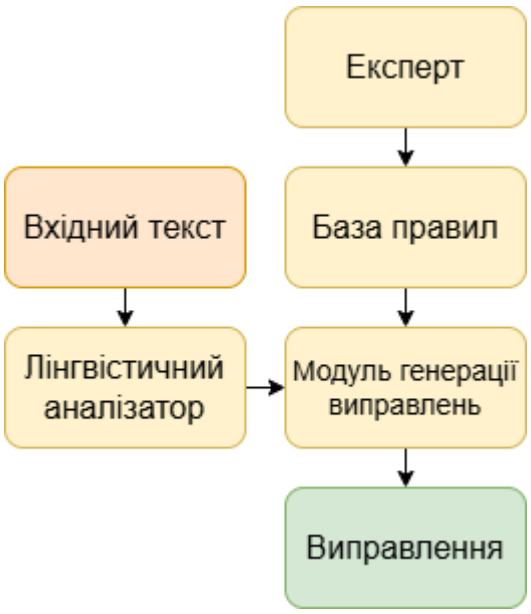

Рисунок 1.1 – Схема роботи системи на правилах

До розвитку глибокого навчання широко використовувались статистичні методи. Одним із таких методів є статистичний машинний переклад (Statistical Machine Translation) [4]. Хоча цей метод був розроблений для вирішення задачі машинного перекладу текстів, він добре підходить для задачі граматичної корекції текстів. У машинному перекладі, вхідна послідовність на одній мові повинна бути перетворена у вихідну послідовність на іншій мові, в задачі граматичної корекції текстів необхідно вхідну, неправильну послідовність перетворити на правильну, що робить ці задачі спорідненими. Такий спосіб дозволяє натренувати модель на великій кількості пар неправильних речень та їх виправленнях. Використання таких методів штовхнуло розвиток задачі граматичної корекції текстів вперед, так як при цьому не потрібно витрачати багато часу лінгвістичних експертів для генерації правил. Але такий спосіб все одно мав свої обмеження, так як контекст не був глибоко аналізований, що не давало можливості знаходити складні помилки.

З розвитком глибокого навчання його методи почали застосовувати і для вирішення задачі граматичної корекції текстів. Значний поштовх дало

створення моделей послідовність до послідовності (seq2seq), такі моделі приймають на вхід певну послідовність, та результатом їх роботи є також послідовність. Вхідною послідовністю в такому випадку є текст з помилками, а вихідною послідовністю є виправлений текст. Першими такими архітектурами стали рекурентні нейронні мережі (RNN), які здатні оброблювати елементи у послідовності один за одним, як зображено на рисунку 1.2. Потім архітектура рекурентних нейронних мереж була замінена їх покращеним варіантом – довготривалою короткочасною пам’яттю (LSTM) [5]. Якість таких систем переважає якість попередніх систем, так як в цьому випадку аналізується вся вхідна послідовність, що дозволяти знаходити більш неочевидні помилки. Але, все одно такі системи мають свої недоліки, такі як неможливість виконувати обчислення паралельно, так як наступний стан залежить від минулого, а також забування попередньої інформації на довгих текстах.

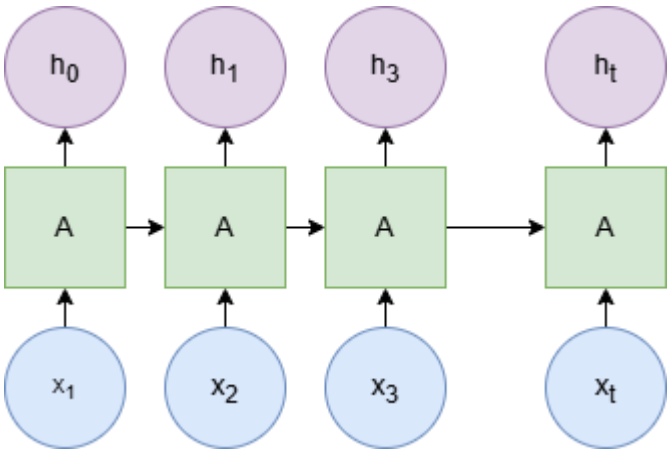

Рисунок 1.2 – Рекурентна нейронна мережа (RNN)

Вирішення цих проблем було знайдено у архітектурі трансформерів (Transformers) [6]. Механізм послідовної обробки був замінений механізмом самоуваги (self-attention), що дозволяє оброблювати усі елементи вхідної послідовності одночасно, незалежно від їх розташування, що дозволяє виконувати обчислення паралельно, а значить

швидше виконувати процес навчання та використовувати значно більше даних для проведення навчання. Здатність оброблювати довгі послідовності також була покращена, що покращило якість обробки текстів у різних задачах, зокрема граматичної корекції текстів.

1.3 Огляд існуючих рішень

Моделі класу послідовність до послідовності (seq2seq) є ефективними у вирішені таких задач обробки природніх мов (NLP), як машинний переклад та граматична корекція текстів. Хоча, задачі машинного перекладу та граматичної корекції текстів є спорідненими, вони мають важливу розбіжність. Задача машинного перекладу текстів потребує повної переробки вхідної послідовності, адже перекладений текст сильно відрізняється від вхідного тексту. У задачі граматичної корекції текстів, більшість частин вхідної послідовності не потребують змін, та навіть значна кількість вхідних послідовностей взагалі не потребують змін. При використанні моделей класу послідовність до послідовності, необхідно перегенерувати всю послідовність, навіть якщо вона не потребує втручань. Генерація послідовності виконується авторегресивно, тобто генерація наступної частки залежить від минулого стану, що унеможливлює її виконування паралельно. Це створює проблему перегенерації, тобто велика частка обчислювальних ресурсів витрачаються на перегенерацію вхідної послідовності, замість виправлення помилок.

Для вирішення проблеми перегенерації були створені моделі класу послідовність до виправлень (seq2edits) [7]. Такі моделі також приймають послідовність на вхід, але генерують лише потрібні виправлення, що можуть бути представлені у різних формах. Таким чином, якщо послідовність не потребує змін, то й не будуть витрачені обчислювальні ресурси у великій кількості.

Прикладом такого рішення є модель «GECToR» [8]. Основу моделі складає трансформер енкодер (Transformer Encoder), що виконує обробку вхідної послідовності. Зазвичай для цього використовуються попередньо натреновані моделі, зокрема такі як:

– «BERT» [9];

– «RoBERTa» [10];

– «XLNet» [11].

Використання попередньо натренованої моделі в цьому випадку спрощує навчання системи, та потребує значно меншої кількості тренувальних даних, так як така модель вже навчена оброблювати природню мову, а тому може бути навчена на іншу задачу за допомогою навчанню з перенесенням. Для створення виправлень, використовується класифікатор, що навчається разом з основною моделлю, та класифікує кожен токен вхідної послідовності на такі класи:

– «keep»;

– «delete»;

– «append»;

– «replace».

Таким чином, над вхідною послідовністю можливо зробити усі необхідні операції перетворення. Але, так як такі операції, як вставка та заміна потребують також інформації, яка саме частка вставляється, або на яку частку відбувається заміна. Для цього, створюється спеціальний словник, який комбінується с операціями вставки та заміни, що означає, що класифікатор має значно більше ніж 4 класи, та утворює такі класи, як наприклад, «REPLACE_is». Повну схему роботи моделі можна побачити на рисунку 1.3. Так як за один раз, можливо, не всі помилки будуть виправлені, утворена нова послідовність ще раз проходить через всю модель, доки не будуть виправлені усі помилки.

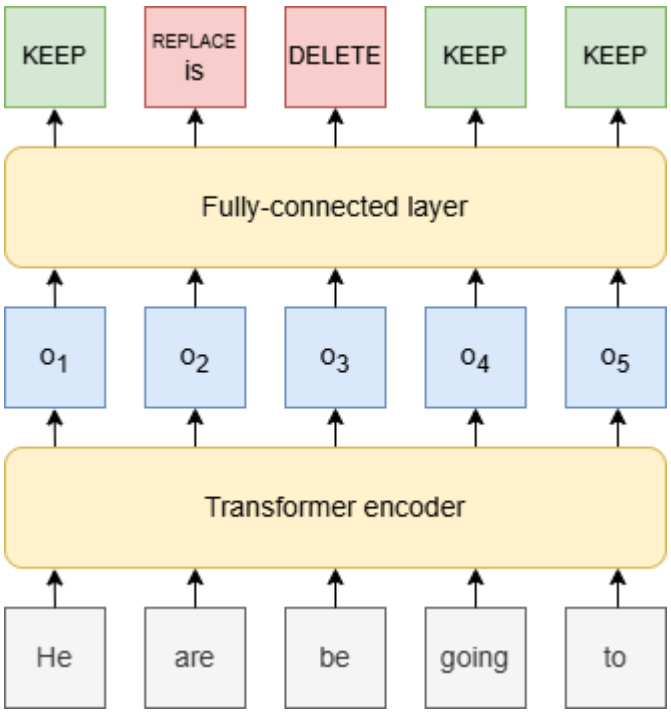

Рисунок 1.3 – Схема роботи моделі «GECToR»

Іншим способом отримання виправлень для вхідної послідовності є вказівні мережі (pointer networks). Цей тип мережі дозволяє отримувати координати у вхідній послідовності. Система «RedPenNet» [12] є прикладом такої мережі, в її основі також як і в моделі «GECToR» лежить трансформер енкодер, але, на відміну від «GECToR» в цій мережі використовується також трансформер декодер (Transformer Decoder) для створення виправлень. Кожне виправлення складається з інтервалу координат, тобто координата початку та кінця виправлення, а також самого виправлення. Координати виправлення отримуються за допомогою застосування класифікатора та механізму уваги у декодера. Текст виправлення отримується зі звичайного виходу декодера. Повна схема роботи моделі зображена на рисунку 1.4. Таким чином, якщо вхідна послідовність не потребує виправлень, то вони не будуть згенеровані, а отже не будуть витрачені обчислювальні ресурси для цього, що надає можливість використовувати таку модель для систем виправлення граматичних помилок у реальному часі.

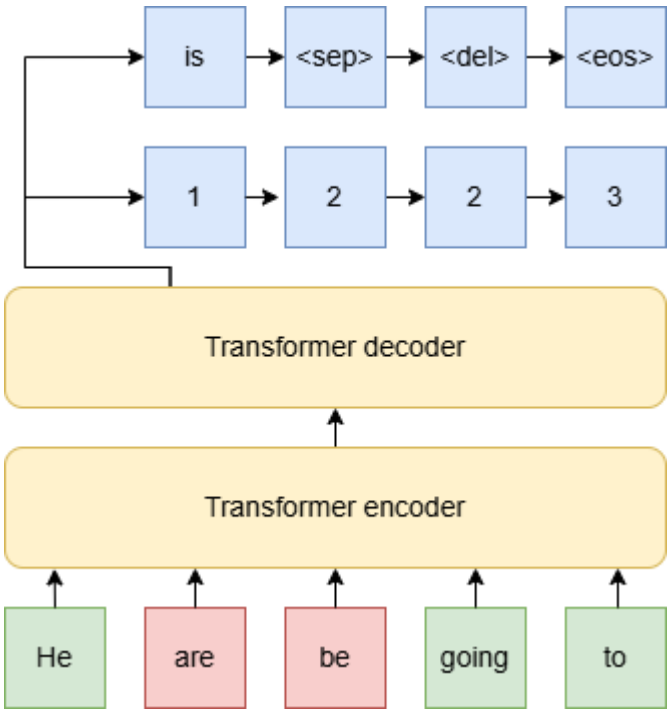

Рисунок 1.4 – Схема роботи моделі «RedPenNet»

Методи послідовність до виправлень надають перевагу для продуктивності рішень, так як вони використовують значно менше обчислювальних ресурсів ніж системи послідовність до послідовності. Але, ці методи мають недолік, те що їх результати є непередбачуваними. Так, наприклад, одна й та сама помилка може бути виправлена в одному контексті, але не виправлена в іншому.

1.4 Оцінка якості роботи систем граматичної корекції текстів

На відміну від інших задач, таких як, класифікація, оцінка роботи систем граматичної корекції текстів є складнішою [13]. Одним із способів автоматичної оцінки є F-міра. Ця метрика має дві складові:

– влучність;

– повнота.

Ця метрика оцінює системи граматичної корекції текстів за виправленнями, що були зроблені. Отже, спочатку необхідно розмітити

тестові дані, отримавши бажані виправлення, що зазвичай складаються з координати, виправлення та типу виправлення. Маючи множину очікуваних виправлень та множину виправлень системою, можливо обчислити влучність, що описує як багато із зроблених виправлень є релевантними за формулою 1.1, та повноту, що описує як багато із очікуваних виправлень дійсно зроблені системою, за формулою 1.2.

$$p = \frac{|s \cap t|}{|s|}, \tag{1.1}$$

де p – влучність;

s – множина виправлень системи;

t – множина очікуваних виправлень.

$$r = \frac{|s \cap t|}{|t|}, \tag{1.2}$$

де r – повнота.

Для обчислення балансу між влучністю та повнотою використовується F-міра, яку можна обчислити за формулою 1.3. Це значення має коефіцієнт, який відповідає за те, яка частина має більшу вагу на фінальне значення. В задачі граматичної корекції текстів, зазвичай, віддають перевагу влучності, так як неправильне виправлення значно погіршує враження користувача.

$$F_\beta = (1 + \beta^2) * \frac{p*r}{(p* \beta^2)+r}. \tag{1.3}$$

Недоліком такого способу оцінки є те, що не враховується складність виправлень, та ступінь критичності зроблених помилок. Для оцінки якості кожного окремого виправлення можливо використовувати окремо навчений

класифікатор, тоді більш критичні помилки будуть використовуватись з більшим коефіцієнтом під час оцінки.

Зважаючи на недоліки автоматичних способів оцінки якості роботи систем граматичної корекції текстів, системи зазвичай додатково перевіряються за допомогою лінгвістичних експертів. Так як критеріїв для оцінки є досить багато, спочатку домовляються про конкретні критерії, що будуть оцінюватись. Для цього можна використовувати як розмічені дані, так і випадкові, включаючи дані, що вже були зібрані під час роботи системи. Експерти здатні об’єктивно оцінити критичність виправлених помилок, або критичність створених проблем та на основі цих даних дати свою оцінку роботи системи.

1.5 Постановка задачі дослідження

Архітектура моделей трансформерів надала поштовх до розвитку моделей для вирішення задач обробки природної мови, зокрема задачі граматичної корекції текстів. Для вирішення проблеми перегенерації у моделях «послідовність до послідовності», були запропоновані архітектури «послідовність до виправлень», що значно зменшує кількість необхідних обчислень для роботи системи граматичної корекції текстів. Але, одним із важливих недоліків залишається непередбачуваність роботи таких систем, так одна й та сама помилка може бути виправлена та не виправлена у різних, схожих контекстах, що псує враження користувача від користування системою. Неможливість контролювати та змінювати роботу системи, окрім як проводити додаткове навчання є слабким місцем систем глибокого навчання.

Деякі традиційні, алгоритмічні методи, такі як системи виправлення орфографічних помилок на базі словників, мають високу точність у знаходженні помилок у вхідних текстах, але не мають можливості надати правильне виправлення, так як при цьому не оброблюється контекст. Такі

системи можна легко контролювати, шляхом додаванням або видаленням певних слів із словника, або наданню користувачу такої можливості. Поєднання таких алгоритмічних методів разом із методами глибокого навчання дасть можливість використати сильні сторони обох способів, що призведе до покращення якості роботи фінальної системи та допоможе вирішити проблему прогнозованості результатів роботи такої системи.

Тому, метою даної кваліфікаційної роботи є розробка архітектури моделі «послідовність до виправлень», що може включати в себе координати помилок, які були отримані традиційними способами корекції текстів. Для досягнення поставленої мети необхідно виконати наступні задачі:

– збір навчальних даних;

– розробка архітектури моделі;

– навчання моделі;

– дослідження отриманих результатів.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

2.1 Розробка архітектури моделі

2.1.1 Виявлення помилок у вхідній послідовності

Одним із етапів граматичної корекції текстів є виявлення місць у вхідній послідовності, що потребують виправлень. Так як нейронна мережа не може напряму оброблювати текст, для обробки природної мови у глибокому навчанні використовується перетворення вхідного тексту до вектору, який може бути оброблений нейронною мережею.

В архітектурі трансформерів таке перетворення вхідного тексту до вектору називається токенізація. Токенізація відбувається за допомогою спеціально натренованого словника, який містить цілі слова, частини слів або окремі символи, на які може бути поділений текст. Кожному токену у словнику належить певний номер, тобто токенізація перетворює вхідний текст до вектору чисел.

Далі, маючи такий вектор, задача знаходження помилок може бути зведена до задачі бінарної класифікації, де кожен токен вхідної послідовності класифікується як правильний або такий, що потребує виправлення. Трансформери добре зарекомендували себе у різних задачах обробки природної мови зокрема й задачі класифікації.

У моделі «GECToR» [7] використовується трансформер енкодер як базова модель для знаходження виправлень за допомогою класифікації. Спростивши класифікатор, що використовується у моделі «GECToR» до бінарної класифікації може бути отриманий класифікатор, результатом роботи якого є бінарний вектор, такого самого розміру, як і вхідна послідовність, в якому нульове значення означає, що токен є правильним, а значення одиниці, означає, що такий токен потребує заміни. Такий вектор може бути отриманий шляхом використання класифікатору з кожним

вектором, що представляє окремий токен в останньому прихованому стані енкодера, як зазначено у формулі 2.1.

$$c_i = sigmoid(f(h_i)). \tag{2.1}$$

Тобто, класифікатор містить в собі модель трансформера енкодера, лінійний шар перетворення, та сигмоїдальну функцію, яка добре підходить для задачі бінарної класифікації. Загалом схему розробленого класифікатора можна побачити на рисунку 2.1.

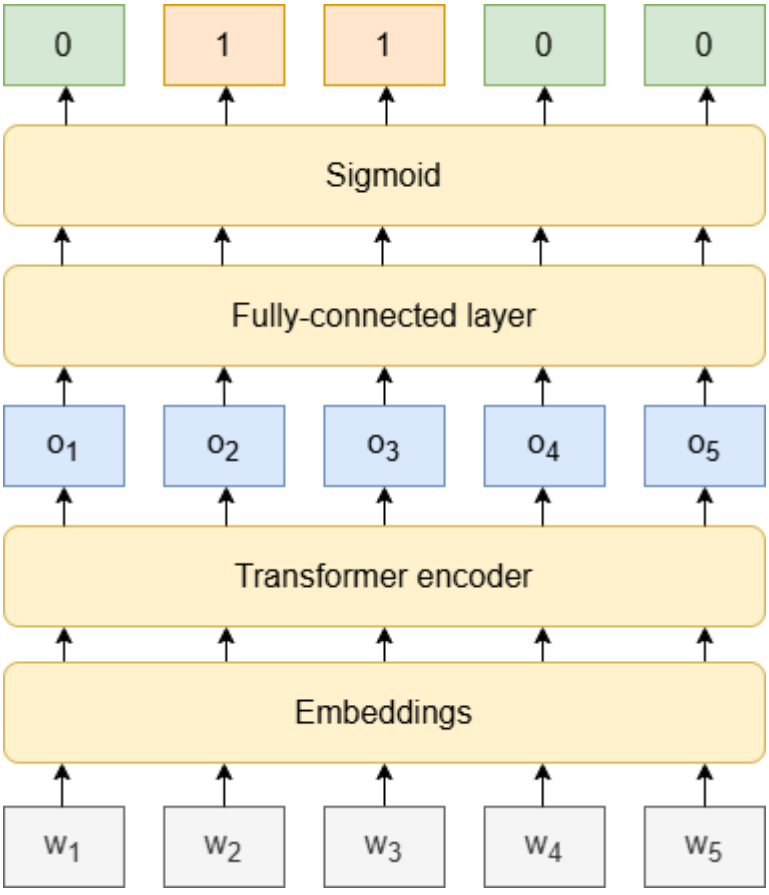

Рисунок 2.1 – Схема виявлення помилок

Після отримання вектору, що містить позиції токенів, які мають бути замінені, наступним кроком необхідно розбити такий вектор на декілька векторів, де кожен вектор містить в собі одне виправлення. Виправлення в

цьому контексті – це послідовно розташовані одиниці. Це необхідно для того, щоб обробляти кожне виправлення окремо. Для того, щоб виконати таке розбиття, необхідно розширити вхідний вектор, додавши до нього нульовий та кінцевий елемент. Далі, від кожного елемента вектору віднімається попередній елемент, як показано у формулі 2.2.

$$d_i = c_i - c_{i-1}, c \in \{0,1\}^n, \tag{2.2}$$

де с – вхідний вектор коректності.

Таким чином, отриманий вектор містить три можливих значення:

– нуль;

– один;

– мінус один.

Позиції, де елемент в отриманому векторі дорівнює одиниці вважаються початком виправлення, а позиції, де значення дорівнює мінус одиниці вважаються закінченням виправлень, що показано у формулах 2.3 та 2.4.

$$S = \{s_k | d_{k_i} = 1\}, \tag{2.3}$$

$$E = \{e_k | d_{k_i} = -1\}. \tag{2.4}$$

Маючи координати початку та кінця кожного виправлення, можливо відновити вектори коректності, де кожен окремий вектор відповідає одному виправленню. Отримання такого вектору описано у формулі 2.5.

$$v_k(i) = \begin{cases} 1, & s_k \le i < e_k \\ 0, & \text{інше} \end{cases}. \tag{2.5}$$

Маючи вектори одного розміру, кожен з яких містить координати окремого виправлення, їх необхідно об’єднати у матрицю, кожен рядок якої містить рівно одне виправлення, що показано у формулі 2.6. Кількість рядків у такій матриці – кількість необхідних виправлень, що означає, що матриця може бути пустою.

$$V = \begin{matrix} v_0 \\ v_1 \\ v_m \end{matrix}, \tag{2.6}$$

де m – кількість необхідних виправлень.

2.1.2 Створення виправлень для знайдених координат

Після отримання матриці із координатами помилок, необхідно створити текстові заміни для кожної знайденої помилки. Для цього в даній архітектурі було використано техніку «Highlight and Decode» яка була описана у «Multi-headed Architecture Based on BERT for Grammatical Errors Correction» [14]. В цій техніці використовується трансформер декодер для створення текстових замін для заданих координат.

Декодер у архітектурі трансформерів є ключовим компонентом, який використовується для задачі генерації тексту. Він використовує результат роботи енкодера, як контексте представлення вхідного тексту та генерує авторегресивно текст токен за токеном. Як і енкодер, декодер складається з декількох однакових шарів. Ключовим компонентом кожного шару декодера є механізм само-уваги. За допомогою цього механізму декодер може добре моделювати залежність між токенами на великій відстані, що перевершує традиційні рекурентні підходи. Також перевагою є можливість паралельної обробки вхідної послідовності, що значно оптимізує його роботу. Загальну схему декодера можна побачити на рисунку 2.2.

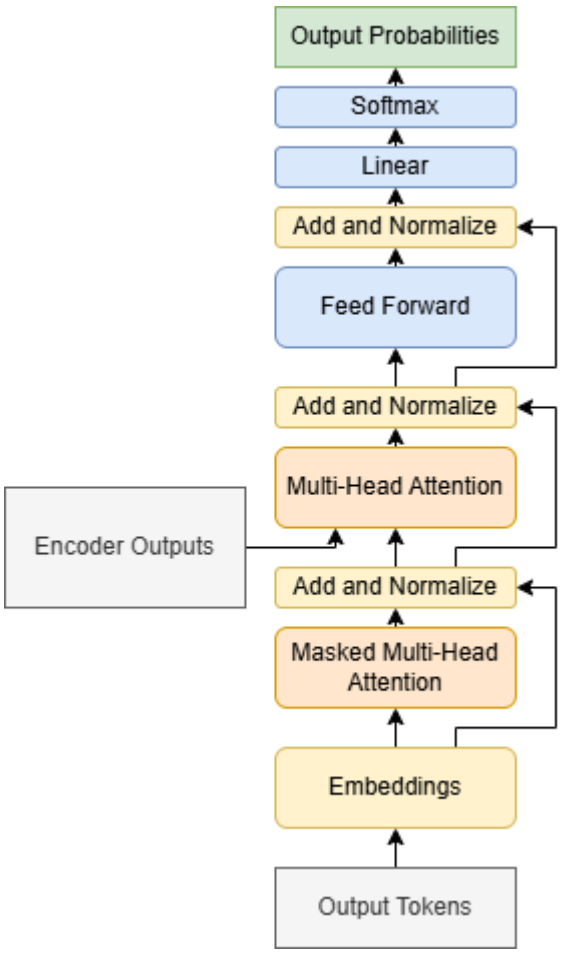

Рисунок 2.2 – Схема трансформер декодера

Але, у звичайній реалізації декодера неможливо вказувати конкретні позиції, на які необхідно звернути увагу. Для вирішення цієї проблеми у техніці «Highlight and Decode» використовується спеціальний вектор, розміром з прихований розмір моделі, що додається до значень, що відповідають необхідним токенам за формулою 2.7.

$$H' = 1_m H + V \odot E, \tag{2.7}$$

де H – останній прихований стан енкодера;

V – матриця коректності;

E – спеціальний вектор підсвічування;

m – кількість необхідних виправлень.

В якості координат використовується матриця коректності, отримана на минулому кроці. Таким чином, для однієї вхідної послідовності формується пакет, де кожен елемент містить одне виправлення, завдяки цьому підвищується ефективність використання ресурсів, так як кожне виправлення оброблюється паралельно, що надає перевагу для обчислень на відеочіпах та зменшує час обробки запиту. Кожен елемент пакету містить модифіковану копію пам’яті енкодера, далі цей пакет подається на вхід до декодера, який генерує заміни для виділених частин тексту, приклад цього процесу зображений на рисунку 2.3.

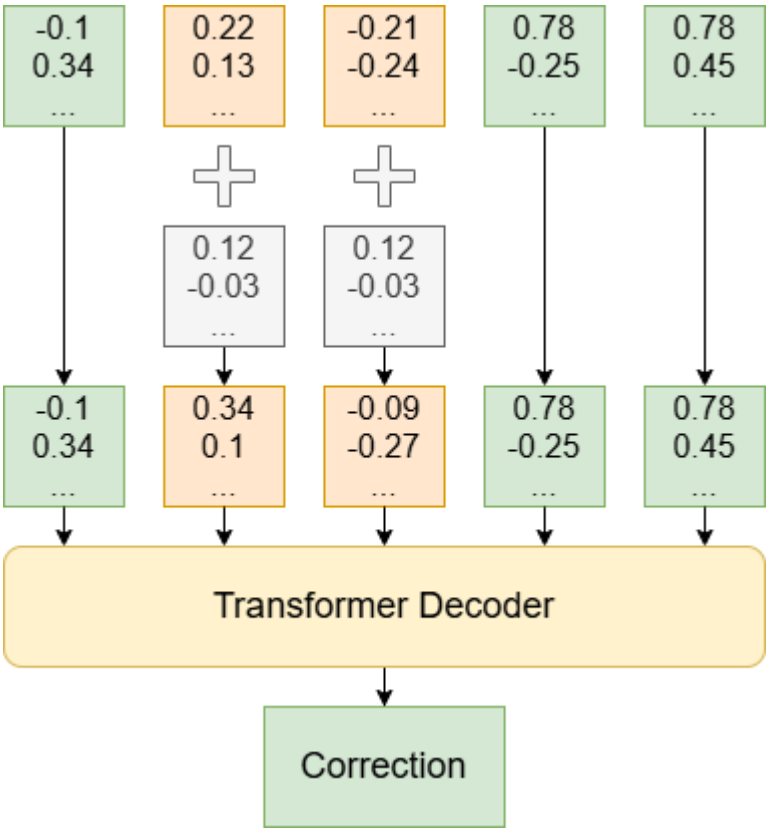

Рисунок 2.3 – Процес створення виправлень

Спеціальний вектор, який використовується для виділення необхідних частин вхідної послідовності, ініціалізується випадковим чином та надалі навчається разом із всією моделлю.

2.1.3 Поєднання з алгоритмічними системами

Алгоритмічними системами граматичної корекції текстів є словникові системи для виправлення орфографічних помилок, системи засновані на правилах для виправлення граматичних та стилістичних помилок, тощо. Їх головним недоліком є неможливість надання якісних замін, так як для цього необхідний глибокий аналіз контексту. Але, сильними сторонами таких систем є контрольованість результатів, адже правила, за якими працює система задає людина, що робить її роботу передбачуваною.

Наприклад, словникові системи працюють таким чином, що правильними вважаються лише слова, які є наявними у словнику, а усі інші слова вважаються неправильними. Користувач може легко контролювати роботу такої системи шляхом додавання або видалення слів із словника. Результат роботи такої системи є стабільним, одне й те саме неправильно написане слово буде виявлено у будь яких прикладах. Така система добре показує себе у виявлені позицій помилок, але її недоліком є низька якість виправлень, так як для надання якісного виправлення необхідно проаналізувати контекст в якому зроблена помилка.

Використання такої системи для пошуку координат помилок підвищить якість роботи фінальної системи, а також зробить її роботу більш передбачуваною для користувача. В цій роботі в якості такої системи була обрана саме словникова система. Поєднання словникової системи із нейронною мережею відбувається так, що вхідна послідовність перевіряється окремо словниковою системою, утворюючи вектор коректності такого самого розміру, як з нейронної мережі. Далі обидва вектори коректності поєднуються в один за допомогою логічного оператору «АБО», утворюючи один єдиний вектор, що містить в собі поєднані координати помилок двох систем. Поєднання результатів двох систем показано у формулі 2.8.

$$c_i = c_a(i) \vee c_m(i), \quad (2.8)$$

де c_a – вектор коректності алгоритмічної системи;

c_m – вектор коректності нейронної мережі.

Далі, використовуючи координати помилок з об’єднаного вектору повинні бути згенеровані виправлення використовуючи техніку «Highlight and Decode» описану раніше. Приклад поєднання результатів роботи двох систем зображено на рисунку 2.4.

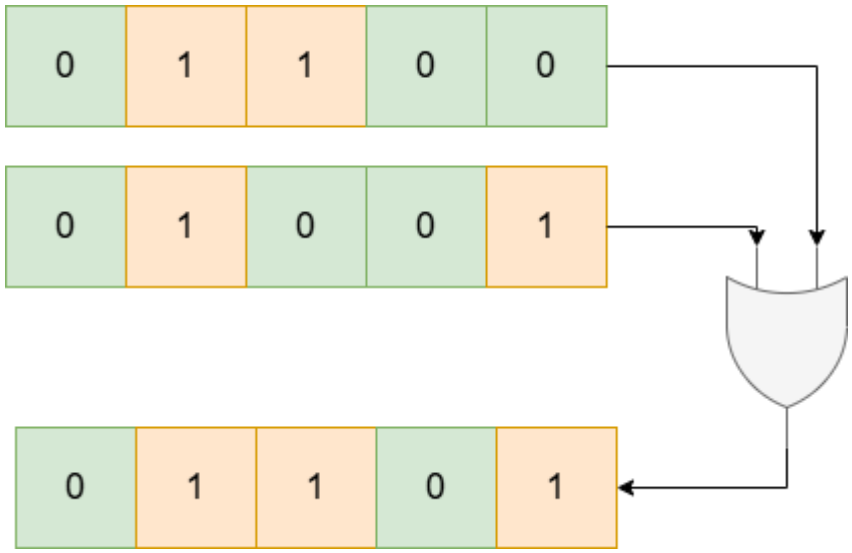

Рисунок 2.4 – Поєднання результатів двох систем

2.1.4 Повна архітектура розробленої системи граматичної корекції текстів

Отже, основою розробленої системи є попередньо натренована модель трансформер енкодер. Обирати модель можна зважаючи на мову, для якої розробляється система та на домен, для якого розробляється система. В цій роботі модель обрана за допомогою практичного експерименту, який буде описано далі. Енкодер використовується для аналізу вхідної послідовності. Результат такого аналізу використовується

для знаходження токенів, що потребують виправлення та як вхідна пам’ять на вхід для модуля генератора виправлень, що використовує трансформер декодер. Для створення пам’яті використовується техніка «Highlight and Decode», для чого кожна помилка оброблюється окремо. Для поєднання з алгоритмічною системою знаходження орфографічних помилок вхідна послідовність оброблюється окремо такою системою, а результати об’єднуються за допомогою логічного оператору «АБО». Загальна схема розробленої системи представлена на рисунку 2.5.

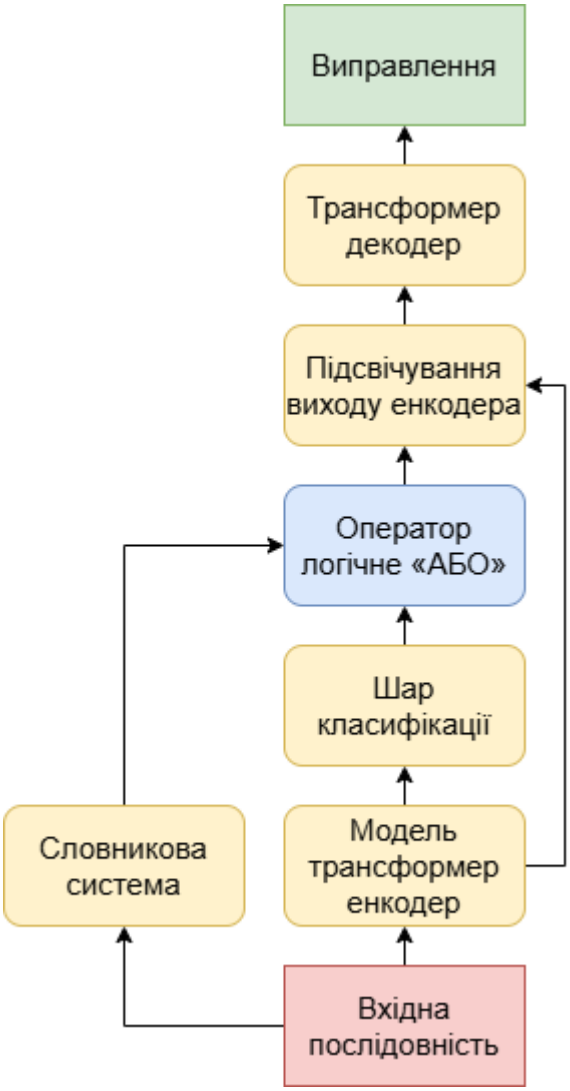

Рисунок 2.5 – Повна схема розробленої архітектури системи граматичної корекції текстів

Таким чином, розроблена система ефективно використовує ресурси. Модель трансформер енкодер відпрацьовує лише один раз, результат її роботи використовується як для виявлення помилок у послідовності так і для генерації виправлень. Словникова система працює таким чином, що слово вважається правильним, якщо воно наявне в словнику, та неправильним, якщо такого слова немає в словнику. Алгоритмічна складність такої перевірки є константною «O(1)». Трансформер декодер є авторегресивним, тобто генерує токени один за одним, але у описаній системі генеруються лише виправлені токени, а також усі виправлення оброблюються паралельно, що підвищує ефективність роботи системи. У випадку ж, коли виправлення не потрібні, декодер не буде виконувати жодну роботу.

Ще одним способом оптимізації є розмір декодера. Так як вхідна послідовність вже оброблена, для задачі виправлення граматичних помилок непотрібно використовувати велику кількість шарів декодера. Також, декодер може мати словник, що відрізняється від словника базової моделі. В задачі граматичної корекції текстів, зазвичай, помилки є типовими, тому для їх виправлення немає необхідності використовувати словник великого розміру. Зменшення словнику збільшує ефективність роботи моделі, так як значно зменшується остання матрична операції у декодері, яка обчислює розподіл ймовірностей наступних токенів.

2.2 Навчання моделі

2.2.1 Функція втрат при навчанні

Розроблена система складається з наступних елементів:

– модель трансформер енкодер;

– класифікатор коректності;

– спеціальний вектор;

– трансформер декодер.

Усі перелічені компоненти системи повинні навчатися разом у складі однієї системи. Але, неможливо оцінити помилку усіх компонентів системи використовуючи одну функцію втрат. Для ефективного навчання використовується функція втрат складена з декількох частин.

Задача виявлення токенів, що потребують заміни є задачею бінарною класифікацію. Для обчислення помилки в таких задачах часто використовують функцію бінарної перехресної ентропії, яка заміряє відстань між двома розподілами ймовірностей, істинним класом та передбачуваною ймовірністю, що показано у формулі 2.9.

$$l_c = -\frac{1}{N}\sum_{i=1}^{N}(y_i * \log x_i + (1 - y_i) * \log(1 - x_i)), \quad (2.9)$$

де N – розмір пакету;

x – передбачена вірогідність;

y – істинний клас.

Іншою частиною навчання моделі є генерація виправлень. На кожному авторегресивному кроці мережа передбачає наступний токен з переліку доступних, що зводиться до задачі багатокласової класифікації. Для знаходження помилки у багатокласовій класифікації використовується функція перехресної ентропії, що представлена у формулі 2.10.

$$l_d = -\frac{1}{N}\sum_{i=1}^{N}\sum_{d=1}^{D} y_{i,d} * \log x_{i,d}, \quad (2.10)$$

де N – розмір пакету;

D – кількість токенів у словнику;

x – передбачена вірогідність;

y – істинний клас.

Обидві функції втрат оцінюють роботу різних частин моделі, бінарна перехресна ентропія оцінює класифікацію вхідних токенів як правильні та неправильні. В цій класифікації приймає участь модель трансформер енкодер та класифікатор. Перехресна ентропія оцінює помилку при генерації виправлень, в яких приймають участь наступні компоненти системи:

– модель трансформер енкодер;

– спеціальний вектор підсвічування;

– трансформер декодер.

Усі компоненти системи навчаються одночасно, але на кожному виклику моделі обчислюються дві функції втрат, які необхідно об’єднати з однаковою вагою, що показано у формулі 2.11.

$$l = l_c + l_d, \tag{2.11}$$

де l_c – помилка класифікатора;

l_d – помилка декодера.

2.2.2 Алгоритм навчання моделі

Хоча різні частини моделі й оцінюються по-різному, результати їх роботи не є незалежними. Наприклад, декодер в якості входу отримує підсвічену пам’ять використовуючи координати отримані під час класифікації. Так як під час навчання моделі, класифікатор може видавати неточні координати помилок, вони будуть надані декодеру, який також зробить помилки, що призведе до накопичення помилку. Для уникнення цієї проблеми на кожному етапі навчання моделі дають правильні дані на вхід, це означає, що незважаючи на результат класифікатору, декодер отримає на вхід правильно підсвічену пам’ять. Ця техніка називається примушення до навчання (teacher forcing).

Ця техніка також використовується і при навчанні самого трансформер декодера. Так як цей компонент є авторегресивним, тобто кожен наступний токен передбачається беручи до уваги минулу послідовність, для уникнення накопичення помилки при передбаченні кожного наступного токена на вхід подається правильна послідовність. Таким чином, одна хибна відповідь не призводить до подальших хибних передбачень, що дозволяє моделі навчатись більш стабільно.

Але ця техніка має й недоліки. Один із недоліків, це те що модель звикає до ідеального сценарію, коли під час навчання вона отримує правильні дані на вхід. Але під час реальної роботи модель покладається на власні передбачення, які бувають помилковими. Через це може відбуватись падіння якості під час реальної роботи моделі.

Для подолання цієї проблеми використовують різноманітні техніки, такі як запланований семплінг (scheduled sampling), що дозволяє моделі інколи використовувати власні передбачення при генерації. Така техніка поступово зменшує залежність моделі від примушення до навчання. Таким чином досягається кращий баланс між стабільністю навчання та реальною поведінкою моделі.

Однією із проблем що може виникнути під час навчання моделей, що складаються з декількох частин є те, що одна із частин моделі може навчатись швидше за інші, що призведе до її перенавчання та погіршення результату роботи всієї системи. Для запобігання цьому можливо використовувати техніку відкладеного навчання, коли одна із частин моделі починає вчитись пізніше за інші. В цьому рішенні такою частиною є класифікатор коректності вхідних токенів, адже ця задача набагато простіша ніж задача генерації самих виправлень. Тому, є сенс починати навчання моделі лише з генерації виправлень, ігноруючи результати роботи класифікатора, та вже на пізнішому етапі починати навчання класифікатора разом із іншими частинами моделі.

Основою системи є попередньо навчена модель трансформер енкодер, яка складається з багатьох шарів. Така модель вже є навченою та може розпізнавати природню мову, але інші компоненти системи, такі як класифікатор або трансформер декодер ініціалізовані випадковим чином. Щоб дати іншим компонентам системи адаптуватися для роботи з енкодером можливо заморозити перші декілька шарів енкодера з метою запобігання їх деградації на ранніх етапах навчання. Перші шари енкодера відповідають за базове розуміння природної мови, тому на ранніх етапах навчання їх зміна може погіршити якість навчання.

2.3 Вибір навчальної вибірки

2.3.1 Розмічені дані

Навчальні вибірки для задачі граматичної корекції – це зазвичай розмічені дені, які складаються з пар речень, одне з яких є неправильним, а інше його виправленою версією. Таким чином, система навчається перетворювати неправильну послідовність у виправлену. У випадку із системами «послідовність до виправлень», що розглядаються у цій роботі, система вчиться робити такі виправлення, які перетворюють вхідну послідовність на виправлену.

Першим таким набором даних є «W&I+Locness» [15], що є одним із провідних наборів даних на англійській для задачі граматичної корекції текстів. Цей набір даних був створений для спільного завдання «BEA-2019 Shared Task», та складається з двох наборів даних:

– «Write and Improve»;

– «Locness».

Перший набір даних складається з есе, що написані іноземними студентами на платформі «Write and Improve», які поділені на рівні володіння англійською мовою за шкалою «CEFR». Ці есе були ретельно

перевірені експертами викладачами, які виправили знайдені помилки. Другий набір даних включає в себе есе написані носіями англійської мови, та використовується для контрасту в якості правильних речень що не потребують виправлень.

Іншим використаним набором даних є «CoEdIT» [16]. Цей набір даних був створений командою «Grammarly» та складається з декількох наборів даних для перетворення текстів. В цьому наборі даних є приклади для наступних задача редагування тексту:

– граматична корекція текстів;

– перефразування;

– спрощення тексту;

– підвищення зв’язності;

– нейтралізація тексту;

– підвищення чіткості.

Для цієї роботи необхідна лише частина набору даних, яка відповідає за граматичну корекцію текстів, так як інші задачі включать необов’язкові виправлення, або повне переписування вхідного тексту. Незважаючи, що для цієї задачі підходить лише одна частина набору даних із шести, набір усе одно містить досить велику кількість даних, тому більше десяти тисяч пар речень підходять для задачі граматичної корекції текстів. Як і інші набори даних для граматичної корекції текстів, ця вибірка складається з пар речень, одне з яких містить помилки, а інше повністю виправлене речення.

Особливістю цього набору даних є наявність додаткових інструкцій, в яких людською мовою описано дію, яку необхідно виконати над заданою послідовністю. Така форма дуже добре підходить для чат-бот систем, які можуть виконувати широкий спектр задач над текстом. В таких інструкціях може знаходитись додаткова інформація про текст або деталі задачі, яку необхідно виконати. Для системи яка розглядається в цій роботі така інформація є зайвою, тому має бути прибрана під час попередньої обробки даних. Приклад пар речень із набору даних можна побачити на рисунку 2.6.

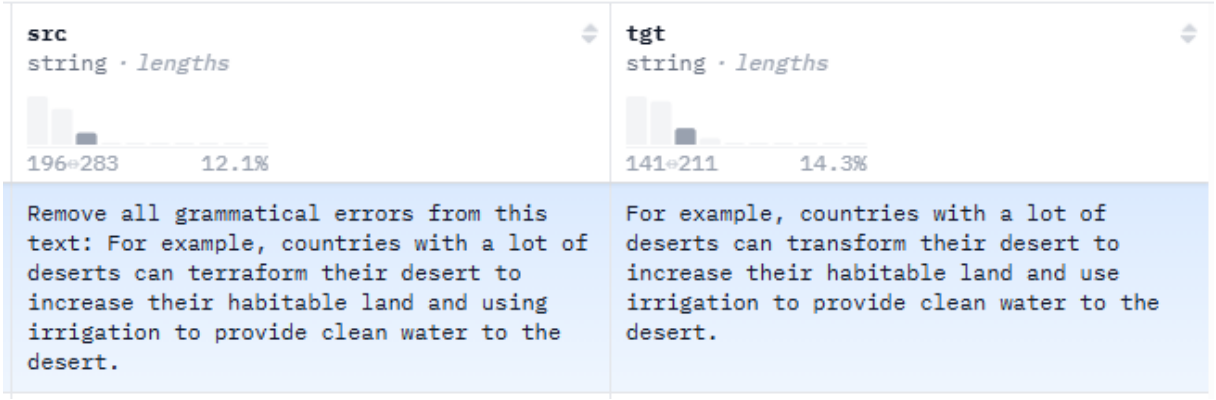

Рисунок 2.6 – Приклад даних у «CoEdIT»

Так як деякі приклади з набору даних «CoEdIT» збігаються з даними «W&I+Locness», він має бути відфільтрований від повторюваних даних. Загальну кількість даних можна побачити у таблиці 2.1.

Таблиця 2.1 – Кількість навчальних розмічених даних

Назва набору даних	Кількість навчальних даних	Кількість валідаційних даних
«W&I+Locness»	34308	4384
«CoEdIT»	12522	485
Всього	46830	4869

2.3.2 Синтетичні дані

Як можна побачити, даних, що розмічені людьми є не дуже багато. Це пов’язано з тим, що для створення таких наборів даних необхідно витрачати велику кількість часу експертів, що створюють такі дані. В задачі граматичної корекції текстів перевагою є те, що варіативність помилок, що може зробити людина у текстах хоча і є дуже широкою, але скінченною. Це дозволяє розроблювати синтетичні дані. Такі дані, на відміну від звичайних даних створені програмно, що означає, що їх можна зробити дуже велику

кількість. Більша кількість даних відкриває можливість навчання моделей великого масштабу та дозволяє моделям краще орієнтуватись у тексті та виправляти помилки. Також, синтетична генерація дозволяє керувати типами та частотою помилок у даних, що надає більше можливостей для налаштування навчання.

Але, використання синтетичних даних також налічує недоліки. Головною проблемою є розходження між штучно створеними помилками та реальними помилками, які люди допускають при написанні текстів. Якщо модель навчається на синтетичних даних, вона може добре справлятись з подібними текстами, але слабко узагальнюватись на реальних ситуаціях. Для того, щоб уникнути цієї проблеми синтетичні дані використовують як старт для навчання моделі, а реальні дані розмічені людьми використовують для донавчання рішень.

В цій роботі в якості синтетичних даних для навчання був обраний набір даних «c4 200m» [17]. На відміну від інших способів створення синтетичних даних, в цьому наборі даних були використані спеціальні моделі, які можуть створювати певний тип помилки у будь-якому реченні. На відміну від систем, що використовують правила для створення помилок, така система створює дані, які більше схожі на реальні, а також такі дані є більш варіативними.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

У цьому розділі описані експериментальні дослідження, що були виконані у цій роботі. Показано процес обирання базової моделі системи, попередня обробка даних та навчання системи. Продемонстровано результати роботи моделі після навчання.

3.1 Опис програмних засобів реалізації системи

Створення систем штучного інтелекту важко уявити без використання спеціалізованих інструментів, адже це включає виконання складних операцій, які легше використовувати вже реалізовані з різних бібліотек. Найбільш поширеною мовою програмування для створення систем штучного інтелекту є мова «Python». Ця мова має велику кількість різних спеціалізованих бібліотек для досліджень в гулузі систем штучного інтелекту, а також дуже проста у використанні, що робить її зручною для проведення досліджень. Тому, саме мова програмування «Python» була обрана в цій роботі.

Для роботи з машинним навчанням можливо використовувати спеціалізовані бібліотеки, які автоматизують процес роботи та навчання нейронних мереж. В мові «Python» існують дві великі такі бібліотеки:

– «TensorFlow»;

– «PyTorch».

Бібліотека «TensorFlow» розрахована на продуктивність та масштабованість рішень штучного інтелекту. Але, головним недоліком є складність використання, так як код має бути написаний у вигляді обчислювальних графів, що ускладнює налагоджування системи та експерименти.

Бібліотека «PyTorch» ж надає можливість використовувати динамічний обчислювальний граф, що дозволяє виконувати операції у

режимі реального часу. Це спрощує налагоджування системи та пошук помилок у рішенні, також значно простіше стає виконувати експерименти. Ще однією перевагою «PyTorch» є те що, в основному нові, сучасні моделі виходять з реалізацією саме на «PyTorch», що полегшує їх інтеграцію та використання. Зважаючи на це, саме ця бібліотека була обрана в цій роботі.

Усі набори даних, що використані у цій роботі складаються з пар речень, де перше речення містить помилки, а друге речення є його виправленою версією. Такі дані підходять для навчання моделі «послідовність до послідовності», але для навчання моделі «послідовність до виправлень» необхідно отримати список виправлень, які необхідно зробити для того, щоб виправити вхідну послідовність, та навчати модель робити такі виправлення. Для перетворення пар речень на список виправлень використовується бібліотека «errant» [18]. Приклад анотованого речення можна побачити на рисунку 3.1.

```
S In India we have various types of Public transport , like Cycle , Bike , Car , Train & Flight .
A 2 2|||M:PUNCT|||,|||REQUIRED|||-NONE-|||0
A 7 8|||R:ORTH|||public|||REQUIRED|||-NONE-|||0
A 11 12|||U:NOUN||||||REQUIRED|||-NONE-|||0
A 13 14|||R:NOUN:NUM|||bikes|||REQUIRED|||-NONE-|||0
A 15 16|||R:NOUN:NUM|||cars|||REQUIRED|||-NONE-|||0
A 17 18|||R:NOUN:NUM|||trains|||REQUIRED|||-NONE-|||0
A 19 20|||R:NOUN|||planes|||REQUIRED|||-NONE-|||0
```

Рисунок 3.1 – Приклад виправлень

Але дану бібліотеку можна використовувати не лише для попередньої обробки даних, а й для оцінки роботи готової системи. Для цього, необхідно за допомогою готової системи зробити виправлення у тестовому наборі речень. Після цього, за допомогою бібліотеки «errant» необхідно порівняти вхідні та виправлені форми. Усі виправлення розбиваються за їхніми категоріями, які залежать від дії, частини мови тощо. Після чого, отримані виправлення порівнюються із набором цільових виправлень, що надає змогу отримати такі метрики, як точність, повнота та «F0.5».

3.2 Попередня обробка даних

Як вже було сказано, більшість наборів даних для задачі граматичної корекції текстів складаються з пар речень, де перше речення є вхідним, а друге речення є його виправленою формою. Так як тексти у наборах даних є реальними текстами, що були написані людьми, в них містяться символи різної форми, такі як різні форми тире, ком, апострофу тощо. Для ефективної обробки таких текстів нейронною мережею вони повинні бути приведені до нормальної форми, що включає в себе:

– нормалізацію букв та цифр;

– приведення усієї пунктуації до єдиної форми;

– видалення невидимих символів;

– нормалізація пробілів.

Код для попередньої обробки текстів можна побачити на рисунку 3.2.

```
def preprocess_sentence(text):
  text = text.strip()
  text = unicodedata.normalize('NFKC', text)
  text = text.replace("\n", ' ')

  text = re.sub(r"[‘’`′]", "'", text)
  text = re.sub(r"[“”«»]", '"', text)
  text = re.sub(r"[-------–•]", "-", text)
  text = re.sub(r"[：]", ':', text)
  text = re.sub(r"[、]", ',', text)

  text = "".join(c for c in text if unicodedata.category(c) != "Cc")
  text = "".join(" " if unicodedata.category(c) == "Zs" else c for c in text)
  text = "".join("-" if unicodedata.category(c) == "Pd" else c for c in text)

  text = text.replace("\x03", "")
  text = re.sub(r'\s+', ' ', text)

  return text
```

Рисунок 3.2 – Код для попередньої обробки текстів

Важливим етапом обробки текстів є токенізація. В граматичній корекції текстів скорочення та пунктуація відіграють важливу роль, так як

часто необхідно вставити, видалити або замінити саме ці частини. Тому, під час попередньої обробки важливо відділити такі скорочення та пунктуацію, що дозволяє більш точно оброблювати текст. Приклад такої токенізації показаний на рисунку 3.3.

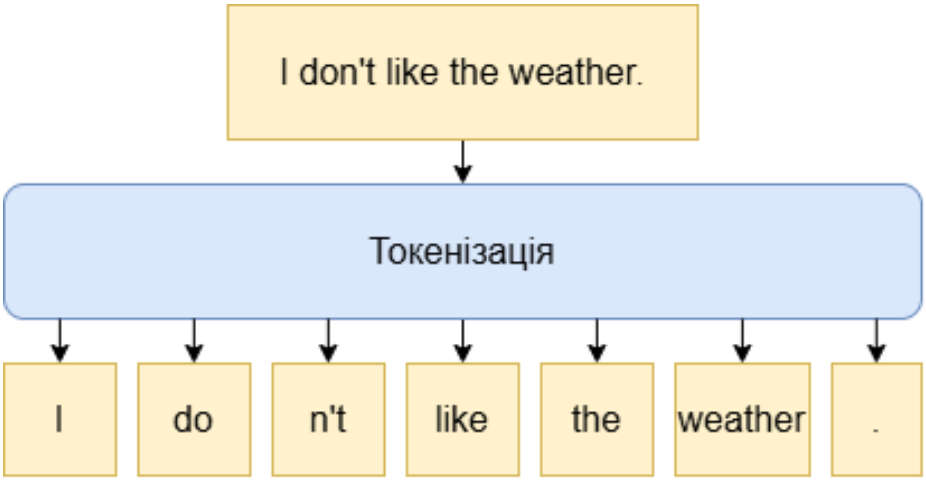

Рисунок 3.3 – Приклад токенізації речення

Як було зазначено під час розробки архітектури системи, система має оброблювати кожну помилку окремо. Для цього, вектор коректності розбивається на окремі виправлення, що є послідовність токенів, які потребують виправлень. Одне таке виправлення може містити декілька логічних виправлень, якщо вони розташовані поряд. Таким чином, одним із етапів є поєднання виправлень в одне, якщо вони перетинаються або розташовані поряд, для того щоб система оброблювала їх як одне ціле. Кожне виправлення має наступні характеристики:

– координата початку;

– координата кінця;

– текстова заміна.

Деякі виправлення мають бути об’єднанні в одне виправлення, для цього їх текстові заміни поєднуються в одне, а координати розширяються. Код для поєднання виправлень зображений на рисунку 3.4.

```
def merge_edits(edits):
    merged_edits = []
    current_edit = edits[0]

    for i in range(1, len(edits)):
        if current_edit.end == edits[i].start:
            current_edit.end = edits[i].end
            if len(edits[i].sug) > 0:
                current_edit.sug += ' ' + edits[i].sug

            continue

        merged_edits.append(current_edit)
        current_edit = edits[i]

    merged_edits.append(current_edit)

    return merged_edits

def widen_edits(edits, words):
    words_n = len(words)

    for edit in edits:
        if edit.start != edit.end:
            continue

        if edit.end < words_n:
            edit.sug = edit.sug + ' ' + str(words[edit.end])
            edit.end = edit.end + 1

    return edits
```

Рисунок 3.4 – Код поєднання виправлень

Для навчання моделі необхідно поділити навчальні дані на пакети. Для цього необхідно токенізувати вхідні тексти за допомогою токенізатора, який відповідає моделі енкодера який використовується. Так як в залежності від обраної моделі токенізатор відрізняється, необхідно розрахувати позиції токенів, які повинні бути замінені, та за необхідності відповідно розширити виправлення. Декодер має свій окремий токенізатор, тому текстові заміни токенізуються окремо. Код підготовки пакету зображений на рисунку 3.5.

```
def prepare_batch(batch_texts, batch_highlights, tokenizer, decoder_tokenizer):
    encoded = tokenizer(batch_texts, return_tensors="pt", padding=True,
        truncation=True, max_length=256, return_offsets_mapping=True)
    batch_size = len(batch_texts)
    input_ids = encoded["input_ids"]
    attention_mask = encoded["attention_mask"]
    offset_mappings = encoded["offset_mapping"]

    highlight_masks = []
    rewritten_texts = []
    chunk_counts = []

    for idx in range(batch_size):
        text = batch_texts[idx]
        highlights = batch_highlights[idx]
        highlight_mask = torch.zeros(input_ids.shape[1])

        highlights.sort(key=lambda x: x[0])

        for edit in highlights:
            start_pos, end_pos, expected_suggestion = edit

            for i, (start_token_pos, end_token_pos) in enumerate(offset_mappings[idx].tolist()):
                if start_token_pos != end_token_pos and start_token_pos + 1 >= start_pos\
                    and end_token_pos > start_pos and end_token_pos <= end_pos:
                    highlight_mask[i] = 1

            if len(expected_suggestion.strip()) == 0:
                expected_suggestion = '[DEL]'

            rewritten_texts.append(expected_suggestion)

        highlight_masks.append(highlight_mask)
        chunk_counts.append(len(highlights))

    highlight_masks = torch.stack(highlight_masks) if highlight_masks else torch.empty(0)

    if rewritten_texts:
        decoder_inputs = decoder_tokenizer(rewritten_texts, return_tensors="pt",
            padding=True, truncation=True, max_length=64)
    else:
        decoder_inputs = {"input_ids": torch.empty(0)}

    return batch_texts, offset_mappings, input_ids, attention_mask, highlight_masks,
        decoder_inputs["input_ids"], chunk_counts
```

Рисунок 3.5 – Код підготовки пакету для навчання

3.3 Створення словника токенізатора

Як було зазначено, в архітектурі, що досліджується у цій роботі трансформер декодер використовує окремий токенізатор, який не зв’язаний з токенізатором базової моделі трансформера енкодера. Це зроблено, так як в задачі граматичної корекції текстів більшість виправлень є типовими, тому словник токенізатора, який використовується для генерації текстових виправлень можна зробити значно меншим за словник вхідної моделі. Менший розмір словника дозволяє витрачати менше ресурсів на генерацію кожного токена [19].

Існує багато видів токенізаторів, що використовуються у нейронних мережах для обробки природної мови. Одним із таких методів токенізації є кодування пар байтів (byte pair encoding). Головною метою цього методу є ефективне перетворювання тексту у послідовність токенів зберігаючи баланс між надто дрібною та надто грубою сегментацією. Принцип його роботи полягає у ітеративному злитті найчастіших пар символів у нові токени. Процес такого злиття виконується багаторазово, доки розмір словника не буде дорівнювати бажаному. Ключовою перевагою такого метода є можливість роботи з рідковживаними або навіть неіснуючими словами, що необхідно для роботи з текстом, який містить помилки. Процес створення словнику зображений на рисунку 3.6.

```
tokenizer = Tokenizer(models.BPE(unk_token="[UNK]"))
tokenizer.normalizer = normalizers.Sequence([normalizers.NFKC()])
tokenizer.pre_tokenizer = pre_tokenizers.Metaspace()
tokenizer.decoder = decoders.Metaspace()
special_tokens = ["[UNK]", "[PAD]", "[BOS]", "[EOS]", "[DEL]"]

trainer = trainers.BpeTrainer(vocab_size=8192, special_tokens=special_tokens)

tokenizer_texts = []

for edit in edits:
    for start, end, suggestion in edit:
        if suggestion and len(suggestion) > 0:
            tokenizer_texts.append(suggestion)

def get_training_corpus():
    for i in range(0, len(corrected_texts), 1000):
        yield corrected_texts[i : i + 1000]

tokenizer.train_from_iterator(get_training_corpus(), trainer=trainer)

bos_token_id = tokenizer.token_to_id("[BOS]")
eos_token_id = tokenizer.token_to_id("[EOS]")
tokenizer.post_processor = processors.TemplateProcessing(
    single=f"[BOS]:0 $A:0 [EOS]:0",
    special_tokens=[("[BOS]", bos_token_id), ("[EOS]", eos_token_id)],
)

tokenizer.save("tokenizer.json")
```

Рисунок 3.6 – Код створення словника токенізатора

Особливістю токенізації тексту для моделей обробки природної мови є використання спеціальних токенів, які використовуються для розставлення спеціальних міток в тексті. В цій задачі окрім стандартних спеціальних символів також необхідний спеціальний токен, який означає видалення, для того, щоб надати системі змогу виконувати явну операцію видалення певних частин тексту. Тобто, створений словник повинен містити наступні спеціальні символи:

– початок тексту;

– кінець тексту;

– невідомий символ;

– пустий символ;

– видалення.

Символ початку тексту використовується як початковий стан для генерації виправлення. Символ кінця тексту сигналізує про те, що процес генерації завершено. Невідомий символ потрібний для таких випадків, для кодування символів, яких не було у навчальній вибірці. Пустий (pad) символ необхідний для того, щоб сформувати пакет даних з текстів різної довжини. Для цього їх необхідно розширити, щоб кожен вектор мав однакову довжину, розширення відбувається за рахунок додавання пустих символів в кінець послідовності.

3.4 Вибір попередньо натренованої моделі

3.4.1 Теоретичне порівняння моделей

Основою розробленої системи є попередньо натренована модель архітектури трансформер енкодера. Її основна задача полягає у обробці вхідній послідовності таким чином, щоб результати цієї обробки могли бути використані для знаходження помилок та створення виправлень для знайдених помилок. Як вже було зазначено, можна обирати будь-яку

модель, що була натренована на різні мови та домени. Але від якості обраної моделі залежить якість роботи усієї створеної системи. В цій роботі в якості експерименту розробляється модель для виправлення помилок на англійській мові в загальному домені. Для того, щоб обрати найбільш підходящу модель для вирішення задачі граматичної корекції текстів, були порівняні наступні моделі:

– «BERT»;

– «RoBERTa»;

– «ModernBERT».

Перша модель, що була обрана – це «Bidirectional Encoder Representations from Transformers», або «BERT». Ця модель була розроблена компанією «Google» та є однією з найперших моделей, що використовують архітектуру трансформер енкодера. Модель одразу показала хороші результати у вирішенні більшості задач обробки природної мови. Модель була навчена на великій кількості даних. Під час навчання модель вчилася вирішувати дві задачі:

– передбачення схованого токена;

– передбачення послідовності речень.

Задача передбачення послідовності речень (next sentence prediction) полягає у тому, щоб навчити модель передбачувати, чи можуть два наданні речення бути написані один за одним. Приклад такої задачі зображений на рисунку 3.7.

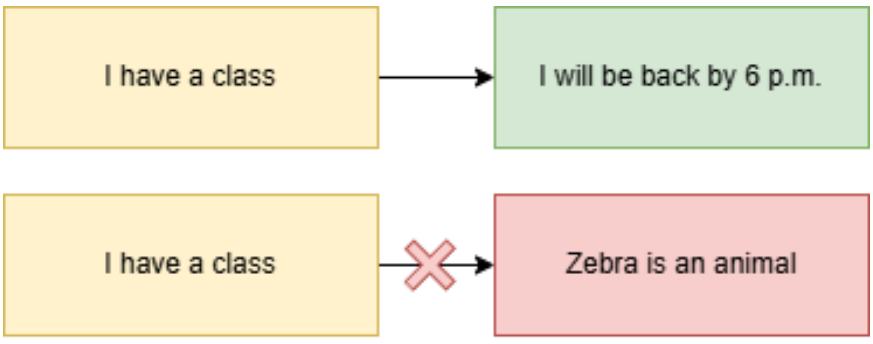

Рисунок 3.7 – Приклад задачі передбачення послідовності речень

Задача передбачення схованого токена (masked language modeling) є поширеною задачею обробки природної мови, та часто використовується для навчання енкодер моделей. У вхідній послідовності замінюється випадковий токен на маску. Задача моделі полягає у передбаченні значення закритого токену використовуючи контекст. Таким чином модель здатна опановувати природню мову. Приклад такої задачі зображений на рисунку 3.8.

Рисунок 3.8 – Приклад задачі передбачення схованого токена

Недоліком моделі для задачі виправлення граматичних помилок є те, що модель використовує токенізатор «WordPiece» досить невеликого розміру, що не дуже добре підходить для задачі граматичної корекції текстів, так як такий тип токенізатора складніше кодує слова, які раніше не зустрічались.

Наступною моделлю є «RoBERTa». Ця модель була розроблена компанією «Facebook» та використовує таку саму архітектуру, як і «BERT», але має ряд відмінностей у реалізації та навчанні:

– інший токенізатор з більшим словником;

– більша кількість даних для навчання;

– тренування на більш довгих послідовностях.

Такі зміни у підході до тренування моделі дало можливість моделі показувати кращі результати на різних задачах обробки природної мови. Зміна токенізатора на «BPE» також надає перевагу у використанні цієї моделі для задачі граматичної корекції текстів, так як даний тип токенізатора дозволяє більш ефективно кодувати послідовності, що містять неправильно написані слова.

Третьою моделлю, що була обрана для порівняльного аналізу є «ModernBERT» [20]. Ця модель є найновішою з проаналізованих та вона була розроблена з урахуванням недоліків минулих моделей та з використанням нових технологій навчання для покращення результатів обробки природної мови. Однією із переваг моделі є збільшений максимальний розмір вхідної послідовності, що дозволяє використання моделі для аналізу більш великих текстів. Також, особливістю моделі є використання «Flash Attention 2» [21] – сучасної реалізації механізму уваги, який дозволяє більш ефективно використовувати ресурси для обчислень. Модель була навчена без використання задачі передбачення послідовності речень, так як вона вважається застарілою. Таким чином, ця модель загалом краще показує себе на більшості задач обробки природної мови.

Також усі порівнянні моделі відрізняються розміром, кількістю шарів енкодера та максимальною довжиною послідовності. Порівняння моделей за цими параметрами вказано у таблиці 3.1.

Таблиця 3.1 – Порівняння параметрів моделей

Назва моделі	Кількість параметрів (мільйонів)	Кількість шарів енкодера	Максимальна довжина послідовності
BERT-base-cased	110	12	512
RoBERTa-base	125	12	512
ModernBERT-base	149	22	8192

3.4.2 Експериментальне порівняння моделей

Від обраної моделі залежить якість роботи всієї системи. Тому, для більш детального аналізу був проведений експеримент. Системи з різними моделями були натреновані на малій вибірці даних та їх результати були порівняні. Такий експеримент показує, яка модель здатна швидше та ефективніше навчатись на рішення задачі граматичної корекції текстів. В якості вибірки для навчання була взята навчальна вибірка «W&I+Locness».

Після навчання кожної з трьох систем з однаковими параметрами у продовж трьох епох, кожна система була оцінена за допомогою тестовій частині вибірки використовуючи інструмент «errant» та метрик точності, повноти та F-міри. Результат всіх трьох моделей можна побачити на рисунку 3.9.

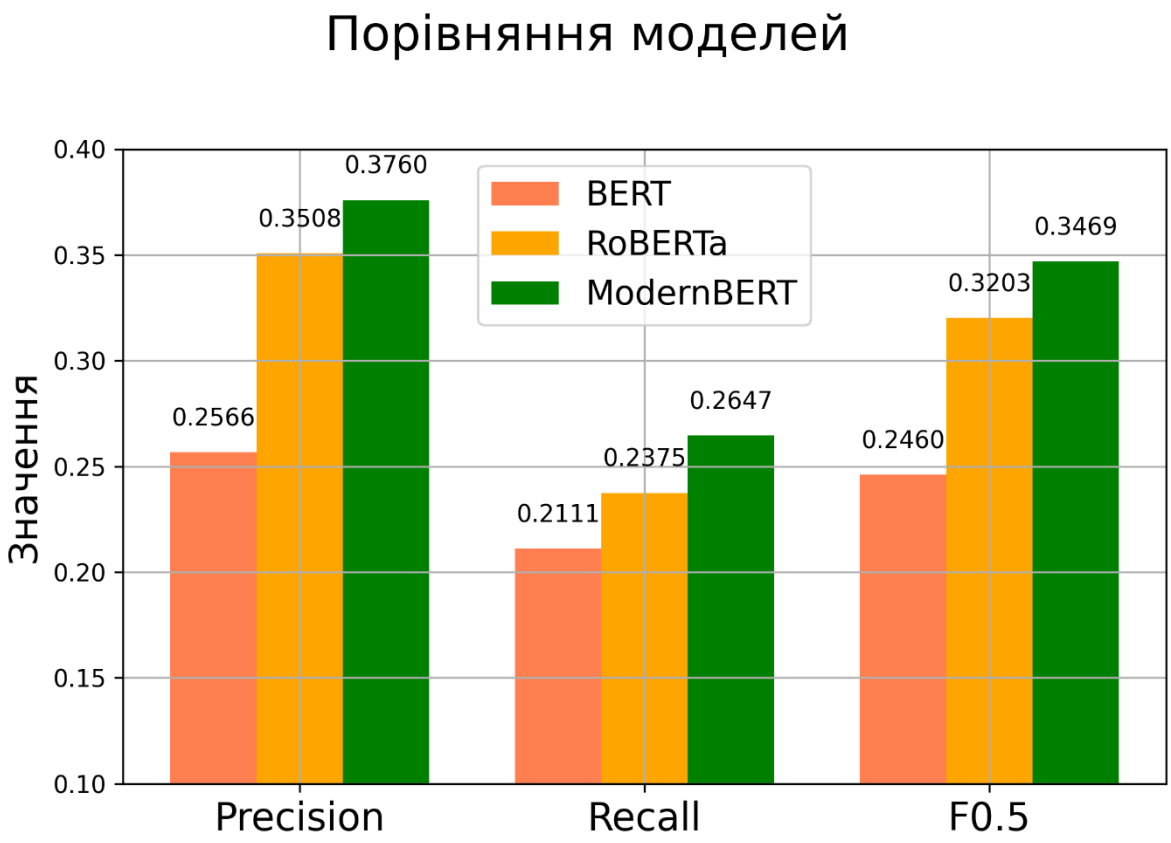

Рисунок 3.9 – Графік порівняння результатів моделей

Як можна побачити, найкращий результат показала саме модель «ModernBERT». Більша кількість навчальних даних, більший розмір та більш сучасні техніки навчання моделі надають змогу моделі більш швидко та ефективно адаптуватись для задачі граматичної корекції текстів. Модель хоч і є більшою за інші, але завдяки більш сучасному механізму уваги різниця по швидкості не є дуже великою.

3.5 Реалізація навчання моделі

3.5.1 Програмна реалізація моделі

Першим кроком до навчання моделі є реалізація структури моделі та логіки її роботи. В цій роботі використано бібліотеку «PyTorch» для роботи з моделлю. Спочатку необхідно описати структуру моделі, створивши необхідні компоненти, їх створення зображено на рисунку 3.10. Отримана модель має наступні компоненти:

– модель трансформер енкодер;

– класифікатор неправильних токенів;

– спеціальний вектор підсвічування;

– трансформер декодер.

```
class HighlightRewriteBERT(nn.Module):
    def __init__(self, bert_model_name='answerdotai/ModernBERT-base', decoder_layers=2, decoder_heads=8,
                 decoder_dropout_prob=0.15, dropout_prob=0.25, decoder_tokenizer_vocab_size=8192, device='cpu'):
        super(HighlightRewriteBERT, self).__init__()

        self.device = device
        self.bert = ModernBertModel.from_pretrained(bert_model_name)
        self.hidden_size = self.bert.config.hidden_size

        self.highlight_embedding = nn.Embedding(1, self.hidden_size)
        self.dropout = nn.Dropout(p=dropout_prob)
        self.highlight_predictor = nn.Linear(self.hidden_size, 1)

        decoder_layer = nn.TransformerDecoderLayer(d_model=self.hidden_size, nhead=decoder_heads, dropout=decoder_dropout_prob)
        self.decoder = nn.TransformerDecoder(decoder_layer, num_layers=decoder_layers)
        self.decoder_embeddings = nn.Embedding(decoder_tokenizer_vocab_size, self.hidden_size)

        self.output_layer = nn.Linear(self.hidden_size, decoder_tokenizer_vocab_size)
```

Рисунок 3.10 – Створення структури моделі

Як було описано, одним із етапів роботи системи є розбиття вектору коректності на групи та підсвічування останнього прихованого стану моделі енкодера. Під час тренування використовується правильний вектор коректності, який вже розбитий на відповідні зміни, тому під час навчання необхідно лише розмножити останній прихований стан енкодера та підсвітити його використовуючи техніку «Highlight and Decode». Під час реальної роботи моделі, необхідно ділити вектор коректності таким чином, щоб кожен вектор мав лише одне виправлення, та лише після цього підсвічувати останній прихований стан енкодера. Код такого розбиття, розширення та підсвічування зображений на рисунку 3.11.

```
def _expand_encoder_hidden_state(self, encoder_hidden_states, attention_mask, chunk_counts):
    expanded_hidden_state = []
    expanded_attention_mask = []

    for i, count in enumerate(chunk_counts):
        hidden_state = encoder_hidden_states[i]

        for _ in range(count):
            expanded_hidden_state.append(hidden_state.unsqueeze(0).clone())
            expanded_attention_mask.append(attention_mask[i].unsqueeze(0).clone())

    expanded_hidden_state = torch.cat(expanded_hidden_state, dim=0)
    expanded_attention_mask = torch.cat(expanded_attention_mask, dim=0)

    return expanded_hidden_state, expanded_attention_mask

def _divide_highlight_mask_to_chunks(self, highlight_masks):
    chunks = []
    for batch_idx in range(highlight_masks.shape[0]):
        tensor_flat = highlight_masks[batch_idx]

        diffs = torch.diff(torch.cat([torch.tensor([0]).to(tensor_flat.device), tensor_flat,
                                      torch.tensor([0]).to(tensor_flat.device)]))

        start_indices = torch.where(diffs == 1)[0]
        end_indices = torch.where(diffs == -1)[0]

        for start, end in zip(start_indices, end_indices):
            chunk = torch.zeros_like(tensor_flat)
            chunk[start:end] = tensor_flat[start:end]
            chunks.append(chunk)

        chunk_tensors = torch.stack(chunks) if chunks else torch.empty(0)

    return chunk_tensors
```

Рисунок 3.11 – Код розширення на підсвічування

Маючи усі компоненти, необхідні для побудови моделі та логіку для розбиття вектору коректності та підсвічування останнього прихованого стану моделі трансформер енкодера, можна розробити логіку роботи моделі під час навчання. Схему роботи моделі було описано у минулому розділі, але під час навчання не використовується гібридна модель, тому результати роботи алгоритмічної системи корекції текстів непотрібні. Результатом одного кроку навчального виклику моделі має бути:

– вихід декодера;

– результат класифікації токенів.

Використовуючи ці результати роботи моделі можна буде порахувати відповідні значення функцій втрат для оновлення ваг моделі та подальшого навчання. Код одного кроку навчання моделі можна побачити на рисунку 3.11.

```
def forward(self, input_ids, attention_mask, highlight_masks, tgt_input, chunk_counts, train_mode=True):
    bert_outputs = self.bert(input_ids, attention_mask=attention_mask)
    encoder_hidden_states = bert_outputs.last_hidden_state

    highlight_logits = torch.sigmoid(self.highlight_predictor(self.dropout(encoder_hidden_states))).squeeze(-1)
    highlight_logits = highlight_logits * attention_mask

    if tgt_input.shape[0] == 0:
        return tgt_input, highlight_logits

    chunk_tensors = self._divide_highlight_mask_to_chunks(highlight_masks)
    expanded_hidden_state, expanded_attention_mask = self._expand_encoder_hidden_state(encoder_hidden_states,
                                                                                       attention_mask, chunk_counts)

    num_chunks, seq_len = chunk_tensors.shape

    highlight_emb = self.highlight_embedding(torch.tensor([0]).to(self.device))

    highlight_emb_expanded = highlight_emb.unsqueeze(0).expand(num_chunks, seq_len, self.hidden_size)
    chunk_mask = chunk_tensors.unsqueeze(-1).expand(-1, -1, self.hidden_size)
    modified_hidden_state = expanded_hidden_state + chunk_mask * highlight_emb_expanded

    decoder_embedding = self.decoder_embeddings(tgt_input).permute(1, 0, 2)
    tgt_seq_len = decoder_embedding.shape[0]
    tgt_mask = torch.triu(torch.ones((tgt_seq_len, tgt_seq_len)), diagonal=1).bool().to(input_ids.device)
    memory_mask = expanded_attention_mask == 0

    decoder_output = self.decoder(
            tgt=decoder_embedding,
            memory=modified_hidden_state.permute(1, 0, 2),
            tgt_mask=tgt_mask,
            memory_key_padding_mask=memory_mask
        )

    logits = self.output_layer(decoder_output)
    logits = logits.permute(1, 2, 0)

    return logits, highlight_logits
```

Рисунок 3.12 – Код одного кроку навчання моделі

Після навчання моделі її необхідно буде використовувати для роботи, зокрема для оцінки якості роботи такої моделі. Так як кроки тренування мають свої особливості, такі як використання примушення до навчання, необхідно розробити логіку роботи моделі без навчання. Відмінність такої логіки полягає у тому, що модель використовує тільки власні результати для створення виправлень. Також, важливо додати можливість об'єднання з результатами алгоритмічної системи знаходження помилок. Результатом роботи такого методу повинно стати список згенерованих виправлень разом з їх координатами. У випадку, коли вхідна послідовність не потребує виправлень цей список має бути пустий. Код такого методу можна побачити на рисунку 3.13.

```
def generate_suggestions(self, input_ids, attention_mask, known_highlights, max_length):
    bert_outputs = self.bert(input_ids, attention_mask=attention_mask)
    encoder_hidden_states = bert_outputs.last_hidden_state

    highlight_logits = torch.sigmoid(self.highlight_predictor(encoder_hidden_states)).squeeze(-1)
    predicted_mask = (highlight_logits > 0.5).int()
    predicted_mask = torch.bitwise_or(predicted_mask, known_highlights)

    if torch.all(predicted_mask == 0):
        return [], predicted_mask

    chunk_tensors = self._divide_highlight_mask_to_chunks(predicted_mask)
    num_chunks, seq_len = chunk_tensors.shape
    expanded_hidden_state, expanded_attention_mask = self._expand_encoder_hidden_state(encoder_hidden_states,
                                                                                       attention_mask, [num_chunks])

    highlight_emb = self.highlight_embedding(torch.tensor([0]).to(self.device))

    highlight_emb_expanded = highlight_emb.unsqueeze(0).expand(num_chunks, seq_len, self.hidden_size)
    chunk_mask = chunk_tensors.unsqueeze(-1).expand(-1, -1, self.hidden_size)
    modified_hidden_state = expanded_hidden_state + chunk_mask * highlight_emb_expanded

    memory_mask = expanded_attention_mask == 0
    memory = modified_hidden_state.permute(1, 0, 2)

    batch_size = memory.shape[1]

    generated = torch.full((batch_size, 1), bos_token_id, dtype=torch.long, device=self.device)

    for _ in range(max_length):
        tgt_emb = self.decoder_embeddings(generated).permute(1, 0, 2)
        tgt_mask = torch.nn.Transformer.generate_square_subsequent_mask(tgt_emb.shape[0]).to(device)
        decoder_output = self.decoder(tgt_emb, memory, tgt_mask=tgt_mask, memory_key_padding_mask=memory_mask)

        logits = self.output_layer(decoder_output)
        logits = logits[-1, :, :]
        next_token = torch.argmax(logits, dim=-1, keepdim=True)
        generated = torch.cat([generated, next_token], dim=1)

        if (next_token == eos_token_id).all():
            break

    return generated, predicted_mask
```

Рисунок 3.13 – Код генерації виправлень

3.5.2 Попереднє навчання моделі

Навчання моделі складається з двох етапів:

– попереднє навчання;

– тонке налаштування.

Етап попереднього навчання відбувається використовуючи синтетичні дані для навчання. Використання великої кількості синтетично розмічених даних дозволяє моделі пристосуватись для вирішення задачі граматичної корекції текстів, так як даних розмічених людьми є недостатньо для навчання великих систем.

В цьому експерименті використовується п’ять мільйонів пар синтетично згенерованих речень. Параметри попереднього навчання системи перелічені у таблиці 3.2.

Таблиця 3.2 – Гіперпараметри попереднього навчання

Назва параметру	Значення
Розмір пакету	16
Коефіцієнт навчання	3e-5
Оптимізатор	«Adam»

Розмір пакету навчання обумовлений лише кількістю доступної оперативної пам’яті графічного процесору, що використовується для навчання, маючи більше пам’яті можливо проводити навчання при більш великому розмірі пакету. При зазначеному розмірі пакету, кількість ітерацій навчання на п’ять мільйонів пар речень складає близько трьох ста тисяч ітерацій.

Для того, щоб надати іншим компонентам системи змогу адаптуватись до задачі граматичної корекції текстів, перші сто тисяч ітерацій половина шарів моделі енкодера були заморожені, тобто їх ваги не оновлювались. Після ста тисяч ітерацій навчання, коли інші компоненти

системи були достатньо навчені, уся модель трансформер енкодер була навчена разом з усією системою.

Основною задачею попереднього навчання системи є її адаптація до задачі граматичної корекції текстів на великій та різноманітній вибірці даних. Тому, максимальна мінімізація функції втрат не є пріоритетом. Графіки функцій втрат декодера та класифікатора зображені на рисунку 3.14.

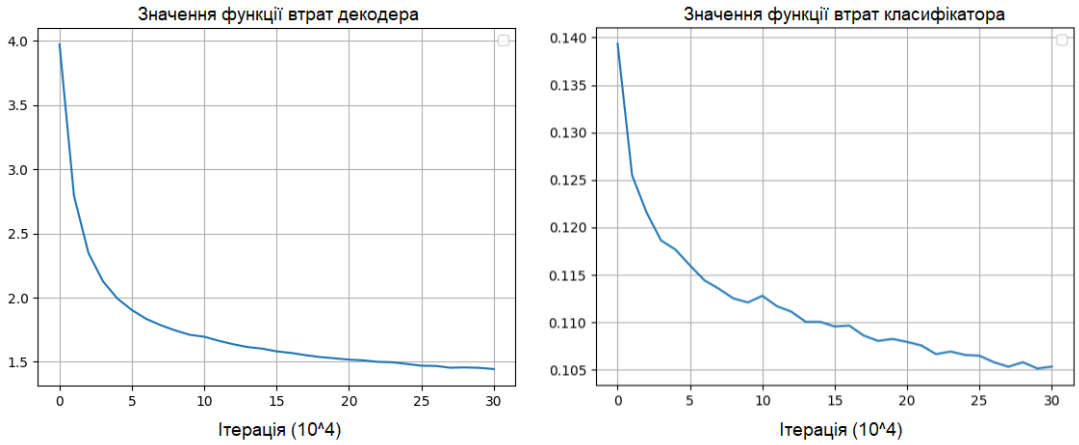

Рисунок 3.14 – Графіки функцій втрат декодера та класифікатора

Як можна побачити, модель стає гірше навчатись після проходження двохсот тисяч ітерацій. Так як кожний приклад у даних подається на модель під час навчання лише один раз, це запобігає перенавчанню на цих даних. Таке навчання дозволяє моделі засвоїти загальні шаблони помилок, які можуть бути допущені у тексті.

Модель, що була отримана в результаті попереднього навчання вже має змогу вирішувати задачу граматичної корекції текстів. Для того, щоб оцінити якість її роботи було використано валідаційні вибірки «W&I+Locness» та «CoEdIT». Результати роботи моделі на цих вибірках можна побачити на рисунку 3.15. Проаналізувавши результати, можна дійти висновку, що попередньо навчена модель вже може добре справлятись з вирішенням задачі граматичної корекції текстів.

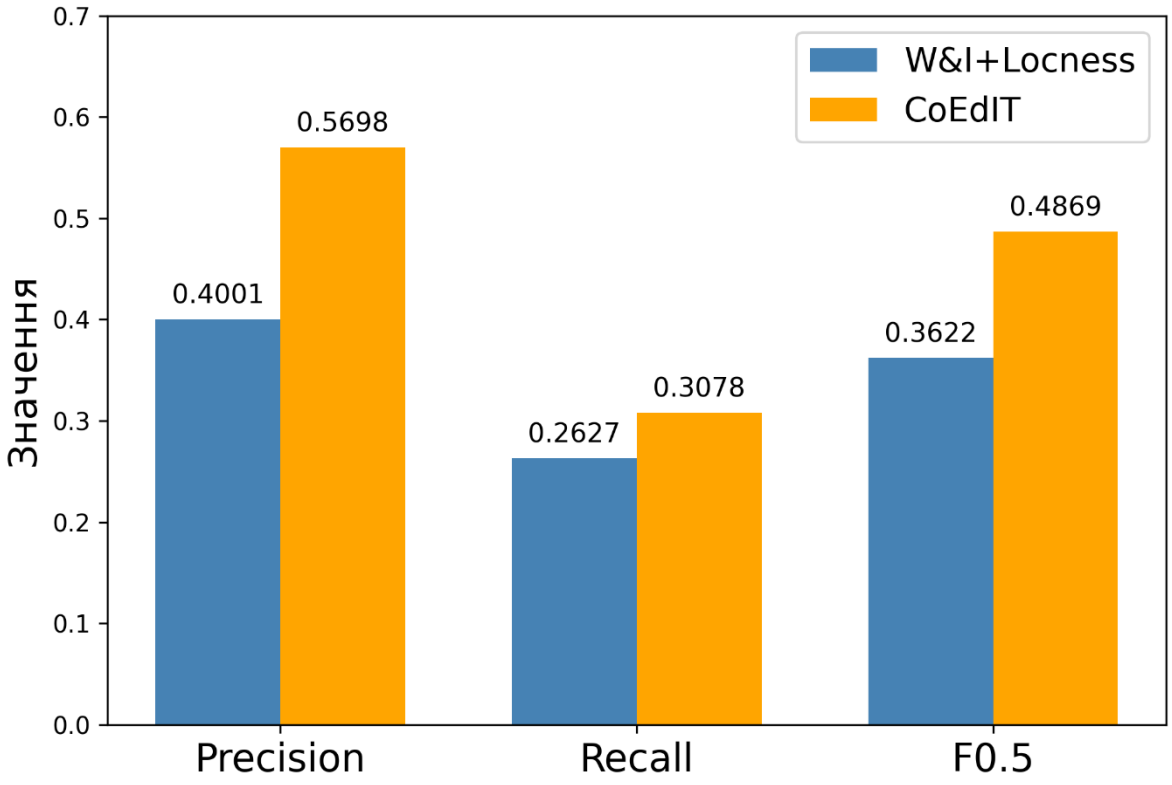

Рисунок 3.15 – Результати роботи попередньо навченої моделі

3.5.3 Тонке налаштування моделі

Наступним етапом у навчанні системи є тонке налаштування моделі. Тонке налаштування – це навчання моделі використовуючи дані високої якості, що були розмічені експертами. В якості таких даних у цьому експерименті були використані такі набори даних:

– «W&I+Locness»;

– «CoEdIT».

Метою такого навчання є адаптація попередньо натренованої моделі до конкретного домену та підвищення якості її роботи завдяки більш точно розміченим даним. Цей процес є більш нестійким, ніж попереднє навчання, тому більше залежить від вибору гіперпараметрів навчання. Гіперпараметри, що були використані під час тонкого налаштування моделі у цьому експерименті перелічені у таблиці 3.3.

Таблиця 3.3 – Гіперпараметри тонкого налаштування моделі

Назва параметру	Значення
Розмір пакету	16
Коефіцієнт навчання	5e-6
Кількість епох	8
Оптимізатор	«AdamW»

Задачею цієї частини навчання є вже мінімізація функцій втрат та максимізація якості роботи моделі на валідаційних даних. Але, необхідно також брати до увагу проблему перенавчання (overfitting), так як при занадто інтенсивному навчанні модель може добре показувати себе на навчальних прикладах, але погано працювати в реальних умовах. Графіки функцій втрат декодера та класифікатора можна побачити на рисунку 3.16.

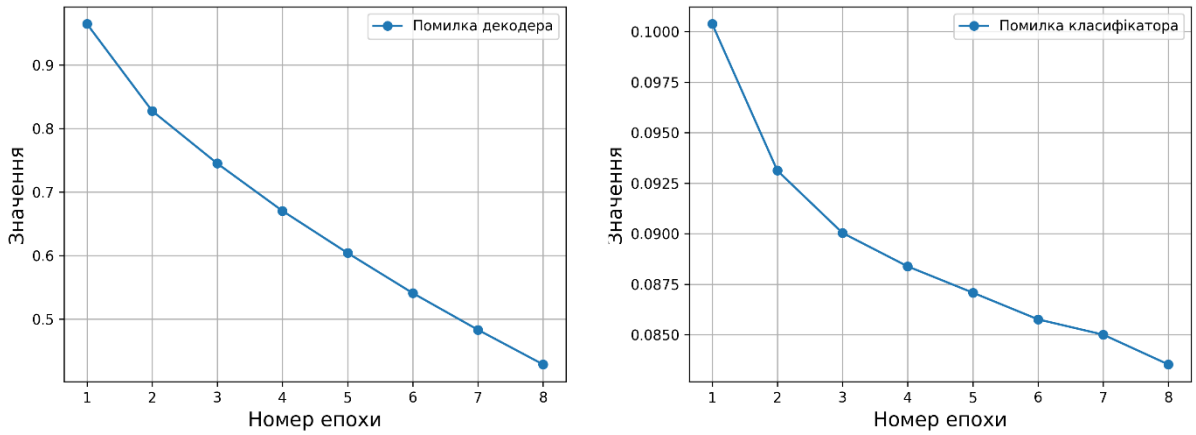

Рисунок 3.16 – Графіки функцій втрат класифікатора та декодера

Для запобіганню перенавчанню, після кожної епохи навчання проводиться оцінка моделі на валідаційній частині набору даних «W&I+Locness», оцінюючи метрики точності, повноти та F-міри. Графік зміни результатів моделі показаний на рисунку 3.17. Як можна побачити, після п’ятої епохи навчання точність моделі починає погіршуватись. Хоча повнота й продовжує зростати до шостої епохи, в

цьому досліджені надається перевага саме точності, адже менша точність значно погіршує досвід користувача, пропонуючи необов'язкові виправлення. Тому, в якості фінальної моделі була взята модель з п'ятої епохи.

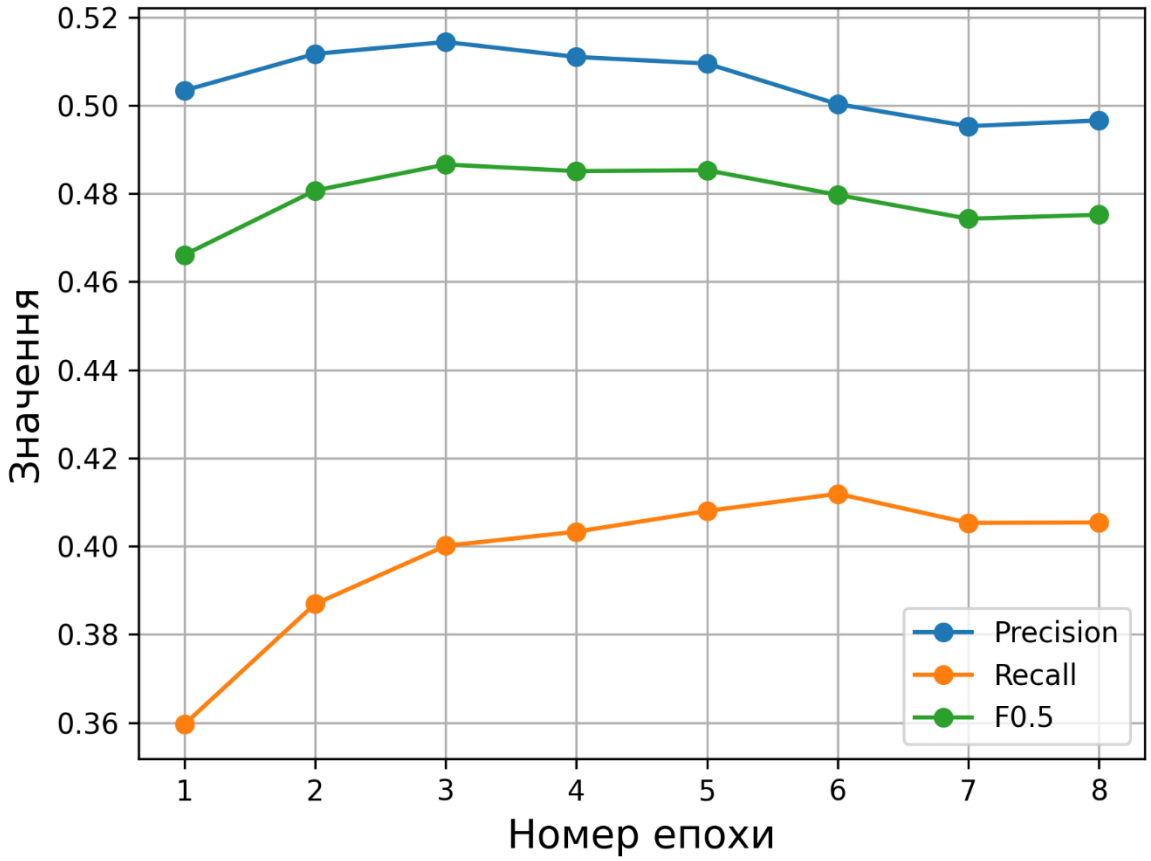

Рисунок 3.17 – Графік якості роботи моделі за епохами

3.6 Аналіз впливу поєднання з словниковою системою на якість роботи усієї системи

Метою цього дослідження є створення гібридної системи, що поєднує нейронну мережу та алгоритмічну систему виправлення орфографічних помилок в одну систему. В цьому підрозділі буде оцінено вплив поєднання з алгоритмічною системою на якість роботи всієї системи.

Для аналізу цього впливу, система була порівняна у чистому та гібридному варіантах. У чистому варіанті, модель працює без поєднання з алгоритмічною системою. В гібридному варіанті, кожне речення також перевіряється системою виявлення орфографічних помилок, та координати помилок, що були виявлені такою системою також надаються моделі для генерації виправлень. Результати роботи чистої та гібридної моделей на вибірках «W&I+Locness» та «CoEdIT» зображено на рисунку 3.18.

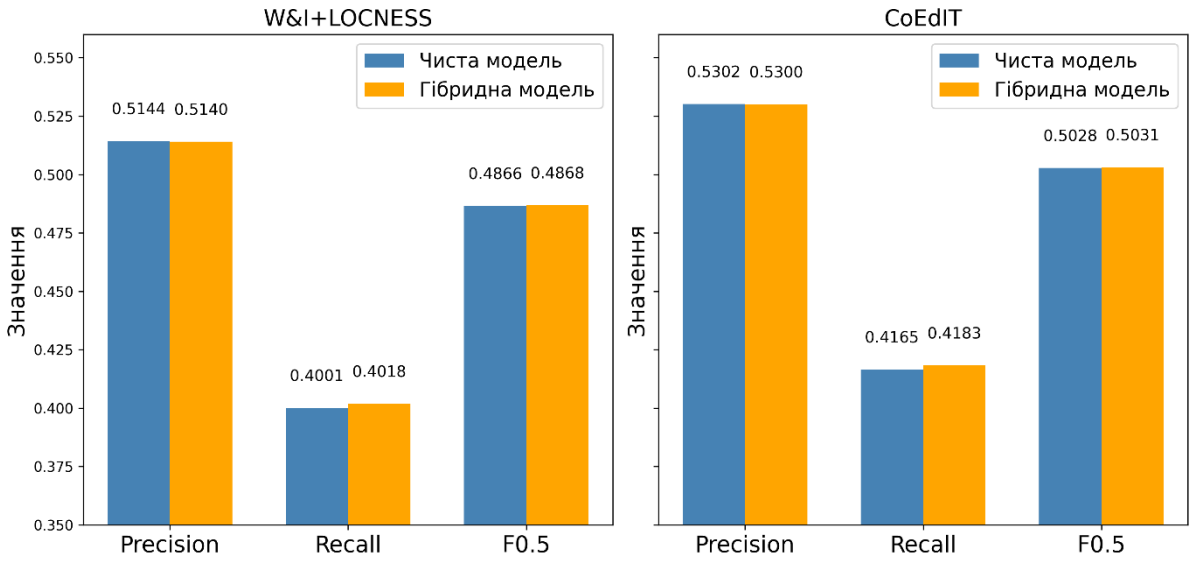

Рисунок 3.18 – Результати роботи чистої та гібридної моделі

Як можна побачити, різниця між чистою та гібридною моделлю не є досить високою. Хоча, результати гібридної моделі і є кращими, зокрема стає вище показник повноти, але різниця досить мала та не перевищує половину відсотка. Це можна пояснити тим, що набори даних «W&I+Locness» та «CoEdIT» зосереджені саме на граматичних помилках, та мають відносно невелику кількість орфографічних помилок у даних. Так як алгоритмічна система в цьому експерименті є саме орфографічна система виправлення помилок, різниця невелика. Загальна кількість виправлений та невиправлених помилок в обох вибірках у чистій та гібридній моделях зображена на рисунку 3.19.

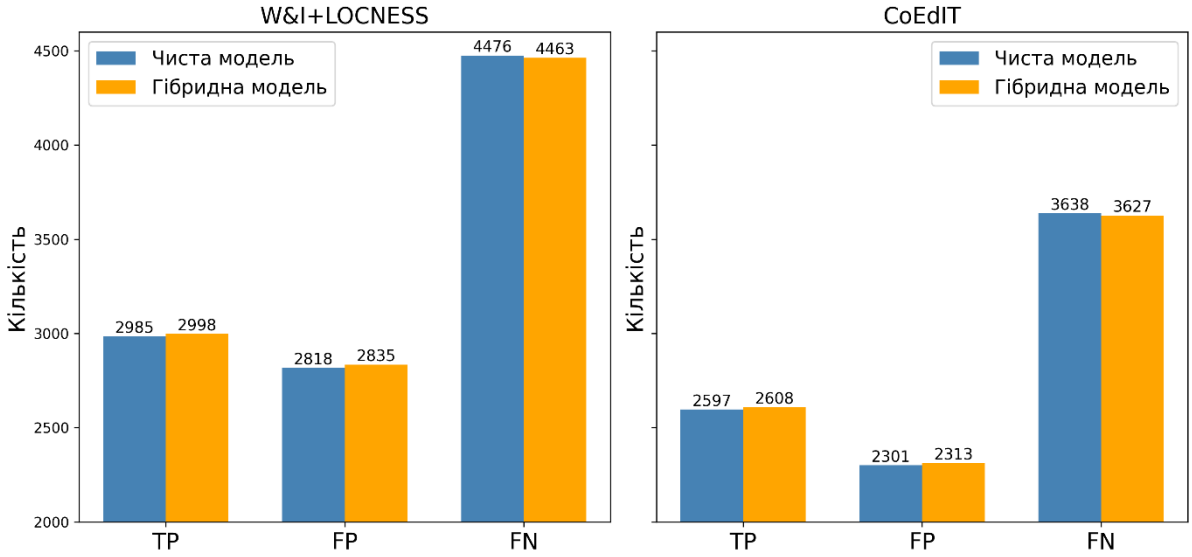

Рисунок 3.19 – Загальна кількість виправлених та невиправлених помилок

Для більш точної оцінки впливу алгоритмічної системи виявлення орфографічних помилок в цьому дослідженні було створено синтетичний набір даних, який містить лише орфографічні помилки на основі набору даних «BookCorpus» [22]. Цей набір даних містить велику кількість речень з книг написаних англійською мовою.

Вважаючи, що усі речення є граматично коректні, в них були створені орфографічні помилки роблячи такі операції у випадкових словах:

– видалення випадкової літери;

– додавання випадкової літери;

– переміщення випадкової літери.

Виконуючи такі операції від одного до трьох разів у випадковому слові із речення було зроблено десять тисяч речень, що містять синтетичні орфографічні помилки. Так як набір даних містить лише орфографічні помилки, повинно бути краще видно різницю між чистою та гібридною моделями. Тому, використовуючи отриманий набір даних було проведено експеримент з оцінки якості роботи чистої та гібридної моделей використовуючи метрики точності, повноти на F-міри. Результати цього експерименту можна побачити на рисунку 3.20.

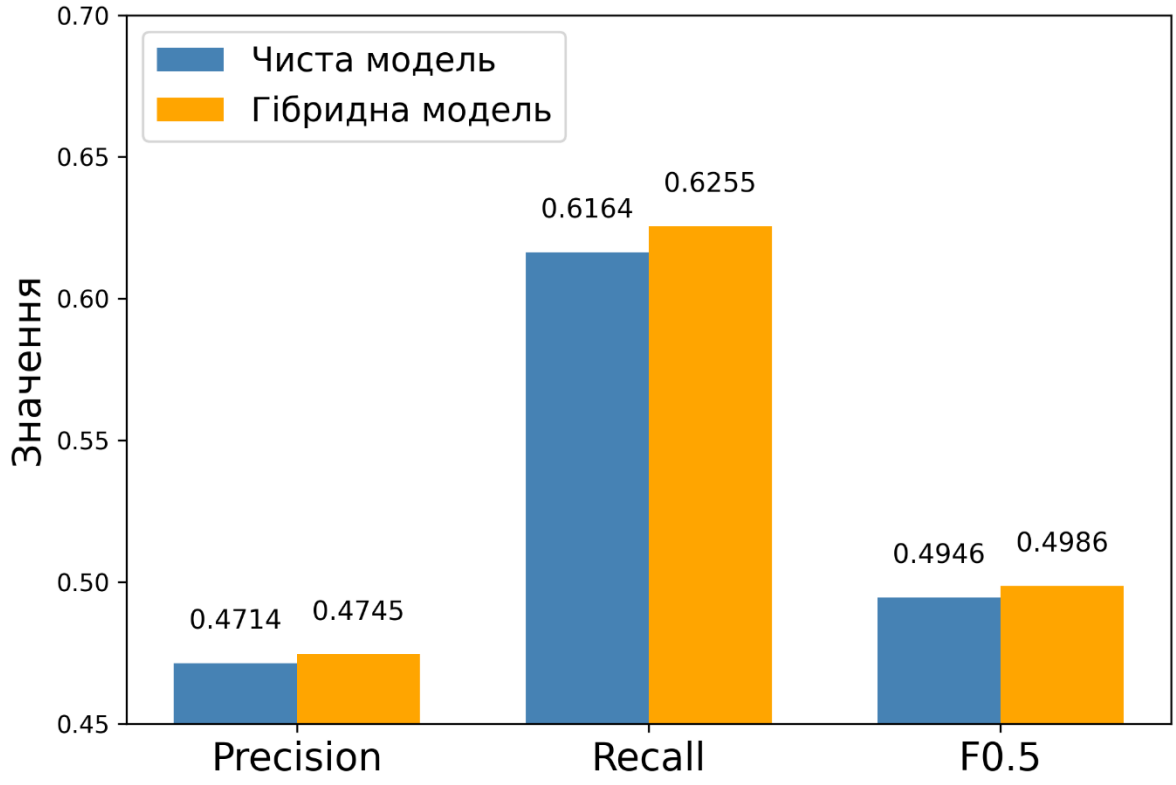

Рисунок 3.20 – Порівняння моделей на синтетичних даних

Як можна побачити, хоча точність роботи системи й покращилась, але ще більший розрив видно у повноті. Так як таким чином система має змогу виправляти більше помилок, повнота виправлення помилок збільшується. За точність виправлення помилок відповідає декодер, так як навіть маючи точні координати помилки, він може згенерувати неправильне виправлення.

Хоча гібридна система й дає підвищення загальної якості роботи всієї системи, головною перевагою такої системи є її контрольованість. Користувачу зручніше працювати із системою, результати роботи якої можна контролювати. В даному випадку, при використанні словникової системи знаходження орфографічних помилок користувач має змогу додавати або видаляти слова із словника, контролюючи роботу гібридної системи.

Розроблена система має й недоліки. Хоча алгоритмічна система знаходження орфографічних помилок й добре справляється з виявленням

помилок, така система не здатна розпізнавати сутності, які непотрібно виправляти, такі як імена людей, власні назви тощо. Координати таких сутностей попадають в модель, яка генерує для них виправлення. Рішенням цієї проблеми може бути використання системи розпізнавання власних назв, для того щоб їх пропускати, або ж окреме навчання моделі для розпізнавання елементів, які непотрібно замінювати.

Іншою відомою проблемою розробленої системи є те, що усі виправлення генеруються одночасно, без інформації про інші виправлення. Хоча це дає перевагу у використані обчислювальних ресурсів, але створює неузгодженість отриманих виправлень [23]. Іноді, виправлення можуть бути залежними один від одного. Так, наприклад, для переміщення слова в тексті необхідно видалити його в одному місці та вставити в іншому. Приклад такої операції можна побачити на рисунку 3.21.

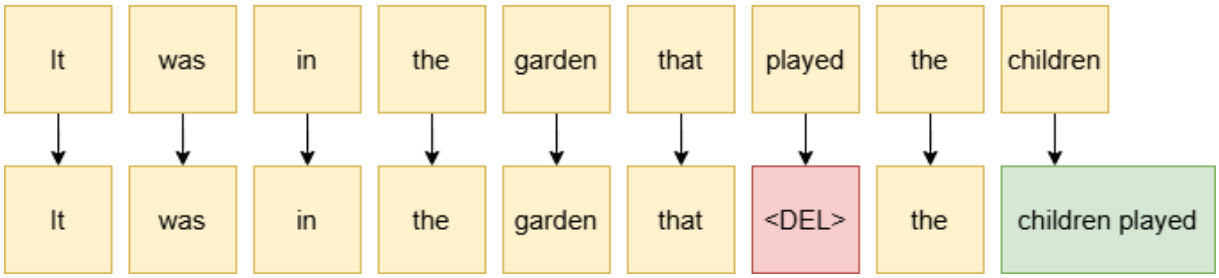

Рисунок 3.21 – Приклад операції переміщення

Як можна побачити, операція переміщення розділена на дві незалежні операції, що означає, що одна із операцій може бути пропущена. Але, так як дані операції є залежними, пропуск однієї з них зіпсує усе речення.

ВИСНОВКИ

В цій кваліфікаційній роботі була досліджена предметна галузь граматичної корекції текстів, що відноситься до більш широкої галузі – обробка природної мови. Під час дослідження предметної галузі було виявлено проблеми, які є досі невирішеними. Такими проблемами є швидкість роботи систем граматичної корекції текстів, що базуються на глибокому навчанні. Також проблемою є непередбачуваність результатів роботи таких систем та неможливість їх контролювати.

Були досліджені сучасні рішення, що використовуються для вирішення задачі граматичної корекції текстів, проаналізовані їх сильні та слабкі сторони. Основним підходом глибокого навчання у цій задачі є моделі «послідовність до послідовності», але, було виявлено, що такі моделі мають проблему перегенерації, що значно збільшує кількість обчислювальних ресурсів, що необхідні для роботи таких систем. Як вирішення цієї проблеми, досліджені рішення «послідовність до виправлень», проаналізовані сучасні рішення, що базуються на такому підході.

Було виявлено, що досліджені підходи мають проблему непередбачуваності результатів роботи, та неможливість їх контролювати. Для вирішення цієї проблеми, було запропоновано поєднання традиційних, алгоритмічних підходів та підходів глибокого навчання. Деякі алгоритмічні підходи добре показують себе у знаходженні позицій помилок, наприклад, системи корекції орфографічних помилок на базі словників. Такі підходи передбачувані та контрольовані, але мають проблему у пропонуванні правильних виправлень. Поєднання цих підходів, може допомогти вирішити проблему непередбачуваності у підходах глибокого навчання.

Результатом роботи є розроблена та навчена гібридна система виправлення граматичних помилок, що поєднує в собі нейронну мережу та алгоритмічну систему знаходження орфографічних помилок. Було

перевірено якість роботи системи за допомогою автоматичних оцінок та порівняно чисту нейронну мережу із гібридною системою, що показало, що гібридна система підвищує якість роботи усієї системи, а також робить роботу такої системи більш контрольованою та передбачуваною.

Перевагами розробленої системи є більш ефективне використання обчислювальних ресурсів, адже система витрачає ресурси лише не генерацію потрібних виправлень, та не витрачає ресурси не перегенерацію всієї послідовності, що вирішує проблему перегенерації. Ще одною перевагою системи є універсальність, так як в розробленій системи можливо використовувати будь-яку попередньо натреновану модель трансформер енкодера, яка може підтримувати будь-яку мову та домен. В цій роботі була використана модель «ModernBERT» для англійської мови. Також перевагою системи є можливість контролювати її роботу, контролюючи словник у словниковій системі знаходження орфографічних помилок.

Розроблена система також має і обмеження. Одним із таких обмежень є паралельна генерація виправлень, що з одного боку надає перевагу у більш ефективному використанні ресурсів, але з іншого боку, створює проблему неузгодженості виправлень, так як система не має інформації про інші виправлення під час генерації конкретного виправлення.

Також варто зазначити, що розроблена архітектура, завдяки використаній техніці «Highlight and Decode» може бути використана не лише у задачі граматичної корекції текстів, але й у будь якій іншій задачі обробки природної мови, де необхідна зміна вхідної послідовності. Наприклад, архітектуру можна використати для розробки системи перефразування текстів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Харченко М., Рябова Н. Застосування методів глибого навчання в задачах nlp: граматична корекція текстів. Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квіт. 2025 р. Харків, 2025. Т. 6 С. 76–78.

2. Improving seq2seq grammatical error correction via decoding interventions / H. Zhou та ін. *Association for computational linguistics*. 2023. С. 7393–7405. URL: https://doi.org/10.18653/v1/2023.findings-emnlp.495 (дата звернення: 18.05.2025).

3. RobustGEC: robust grammatical error correction against subtle context perturbation / Y. Zhang та ін. *Association for computational linguistics*. 2023. С. 16780–16793. URL: https://doi.org/10.18653/v1/2023.emnlp-main.1043 (дата звернення: 18.05.2025).

4. Factored statistical machine translation for grammatical error correction / Y. Wang та ін. *Association for computational linguistics*. 2014. С. 83–90. URL: https://doi.org/10.3115/v1/W14-1711 (дата звернення: 18.05.2025).

5. Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1409.3215 (дата звернення: 18.05.2025).

6. Attention is all you need / A. Vaswani та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1706.03762 (дата звернення: 18.05.2025).

7. Stahlberg F., Kumar S. Seq2Edits: sequence transduction using span-level edit operations. *Association for computational linguistics*. 2020. С. 5147–5159. URL: https://doi.org/10.18653/v1/2020.emnlp-main.418 (дата звернення: 18.05.2025).

8. GECToR – grammatical error correction: tag, not rewrite / K. Omelianchuk та ін. *Association for computational linguistics*. 2020. С. 163–170. URL: https://doi.org/10.18653/v1/2020.bea-1.16 (дата звернення: 18.05.2025).

9. BERT: pre-training of deep bidirectional transformers for language understanding / J. Devlin та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1810.04805 (дата звернення: 18.05.2025).

10. RoBERTa: A robustly optimized BERT pretraining approach / Y. Liu та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1907.11692 (дата звернення: 18.05.2025).

11. XLNet: generalized autoregressive pretraining for language understanding / Z. Yang та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1906.08237 (дата звернення: 18.05.2025).

12. Didenko B., Sameliuk A. RedPenNet for grammatical error correction: outputs to tokens, attentions to spans. *Association for computational linguistics*. 2023. С. 121–131. URL: https://doi.org/10.18653/v1/2023.unlp-1.15 (дата звернення: 18.05.2025).

13. Islam M. A., Magnani E. Is this the end of the gold standard? A straightforward reference-less grammatical error correction metric. *Association for computational linguistics*. 2021. С. 3009–3015. URL: https://doi.org/10.18653/v1/2021.emnlp-main.239 (дата звернення: 18.05.2025).

14. Didenko B., Shaptala J. Multi-headed architecture based on BERT for grammatical errors correction. *Association for computational linguistics*. 2019. С. 246–251. URL: https://doi.org/10.18653/v1/W19-4426 (дата звернення: 18.05.2025).

15. The BEA-2019 shared task on grammatical error correction / C. Bryant та ін. *Association for computational linguistics*. 2019. С. 52–75. URL: https://doi.org/10.18653/v1/W19-4406 (дата звернення: 18.05.2025).

16. CoEdIT: text editing by task-specific instruction tuning / V. Raheja та ін. *Association for computational linguistics*. 2023. С. 5274–5291. URL: https://doi.org/10.18653/v1/2023.findings-emnlp.350 (дата звернення: 18.05.2025).

17. Stahlberg F., Kumar S. Synthetic data generation for grammatical error correction with tagged corruption models. *Association for computational linguistics*. 2021. С. 37–47. URL: https://aclanthology.org/2021.bea-1.4/ (дата звернення: 18.05.2025).

18. Bryant C., Felice M., Briscoe T. Automatic annotation and evaluation of error types for grammatical error correction. *Association for computational linguistics*. 2017. С. 793–805. URL: https://doi.org/10.18653/v1/P17-1074 (дата звернення: 18.05.2025).

19. Dagan G., Synnaeve G., Rozière B. Getting the most out of your tokenizer for pre-training and domain adaptation. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.2402.01035 (дата звернення: 18.05.2025).

20. Smarter, better, faster, longer: a modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference / B. Warner та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.2412.13663 (дата звернення: 18.05.2025).

21. Dao T. FlashAttention-2: faster attention with better parallelism and work partitioning. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.2307.08691 (дата звернення: 18.05.2025).

22. Aligning books and movies: towards story-like visual explanations by watching movies and reading books / Y. Zhu та ін. *arXiv.org*. URL: https://doi.org/10.48550/arXiv.1506.06724 (дата звернення: 18.05.2025).

23. Bodyanskiy Y., Kharchenko M., Ryabova N. Improving GEC based on combining algorithmic approach and sequence-to-edit model. CoLInS : Proc. of the 9th Int. Conf. On Computational Linguistics and Intelligent Systems. Computational Linguistics Workshop, м. Харків, 15–16 трав. 2025 р.