НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА АНАЛІЗУ ДАНИХ

«До захисту допущено»
В.о. завідувача кафедри
_____ І.М. Терещенко
«__» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності: 113 Прикладна математика
на тему: **«Порівняння методів токенізації WordPiece, та SentencePiece на прикладі задачі автоматизованого реферування»**

Виконав: студент 4 курсу, групи ФІ-91
Шморгун Данило Олександрович

Керівник: асистент кафедри ММАД Яворський О. А. _____

Рецензент: доцент кафедри ММЗІ, к.т.н. Яковлєв С. В._____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____

Київ — 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти — перший (бакалаврський)
Спеціальність (освітня програма) — 113 Прикладна математика,
ОПП «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ І.М. Терещенко
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Шморгун Данило Олександрович

1. Тема роботи: «*Порівняння методів токенізації WordPiece, та SentencePiece на прикладі задачі автоматизованого реферування*»,

керівник: асистент кафедри ММАД Яворський О. А.,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Вихідні дані до роботи:

4. Зміст роботи: *Порівняльний аналіз методів токенізації (англ. tokenization) тексту: частина слова (англ. WordPiece), частина речення (англ. SentencePiece), для архітектури глибинної нейронної мережі — трансформер (англ. Transformer), на прикладі задачі автоматичного реферування (англ. summarization). Огляд інших існуючих методів токенізації та моделей глибинного навчання (англ. DL, deep learning), для даної задачі.*

5. Перелік ілюстративного матеріалу: *«Презентація доповіді»*

6. Дата видачі завдання: 11 грудня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	грудень 2022 р. - січень 2023 р.	Виконано
2	Пошук джерел за тематикою дослідження	січень-лютий 2023 р.	Виконано
3	Опрацювання джерел, підготовка даних для проведення експеременту	лютий-березень 2023 р.	Виконано
4	Написання програмного забезпечення та проведення дослідження	березень-квітень 2023 р.	Виконано
5	Написання програмного забезпечення та проведення дослідження	березень-квітень 2023 р.	Виконано
6	Оформлення та опис результатів	травень 2023 р.	Виконано
7	Написання та оформлення дипломної роботи	травень-червень 2023 р.	Виконано
8	Отримання рекомендації до захисту	08.06.2023	Виконано

Студент _____ Шморгун Д. О.

Керівник _____ Яворський О. А.

РЕФЕРАТ

Кваліфікаційна робота містить: 58 сторінок, 11 рисунків, 6 таблиць, 38 джерел.

У даній роботі розглядаються методи обробки даних для моделей глибинного навчання, а саме: частина речення, частина слова. Для порівняння даних методів, було вибрано модель трансформер, а задача – автоматизоване реферування тексту, або підсумовування тексту.

В ході дослідження, було показано що метод частина речення є кращим методом за метрикою Rouge для поданих в даній роботі даних та конфігурації моделі.

ОБРОБКА ПРИРОДНЬОЇ МОВИ, МАШИННЕ НАВЧАННЯ, ТРАНСФОРМЕР, ОПТИМІЗАЦІЯ, МЕТОДИ ТОКЕНІЗАЦІЇ, SENTENCEPIECE, WORDPIECE

ABSTRACT

This paper examines data processing methods for deep learning models, such as text tokenization methods: part-of-sentence, part-of-word. To compare these methods, a transformer model was chosen, and the task was automated abstracting or summarization of the text.

During the research, it was shown that the SentencePiece method is the best method according to the Rouge metric with respect to the data and model configuration presented in this paper.

NATURAL LANGUAGE PROCESSING, MACHINE LEARNING, TRANSFORMER, OPTIMIZATION, TOKENIZATION METHODS, SENTENCEPIECE, WORDPIECE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP — обробка природної мови (англ. natural language processing)

WordPiece — частина слова. Метод токенізації тексту

SentcencePiece — частина речення. Метод токенізації тексту

BPE — кодування парою байтів (англ. Byte pair encoding). Метод токенізації тексту

LexRank — лексичне ранжування. Екстрактивний метод автоматичного реферування тексту

ROUGE — Орієнтоване на пригадування піддослідження для оцінки значення (англ. (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE-L — Орієнтоване на пригадування піддослідження для оцінки значення, найдовша спільна підпослідовність (англ. Recall-Oriented Understudy for Gisting Evaluation, Longest Common Subsequence)

ROUGE-N — Орієнтоване на пригадування піддослідження для оцінки значення, n-грами (англ. Recall-Oriented Understudy for Gisting Evaluation, n-grams)

LSA — латентно-семантичний аналіз (англ. Latent semantic analysis)

LDA — латентне розміщення Діріхле (англ. Latent Dirichlet Allocation)

TextRank — екстрактивний метод автоматичного реферування тексту — ранжування тексту

Transformer — архітектура моделей глибинного навчання.

Attention, attention head — увага, голова уваги. Метод для побудови моделей глибинного навчання

Multi-Head Attention — багатоголова увага. Метод для побудови моделей глибинного навчання

NER — розпізнавання іменованих сутностей. Задача іменування та

класифікації сутностей тексту (англ. named entity recognition)

Seq2seq — підходи машинного навчання, які використовуються для обробки природної мови, перетворення послідовності в послідовність (англ. sequence to sequence).

BERT — Модель глибинного навчання. Двоспрямовані енкодерні представлення з трансформерів (англ. Bidirectional Encoder Representations from Transformers)

DistilBERT — Модель глибинного навчання. Спрощена модель BERT. Дистильований BERT (англ. Distilled BERT)

ALBERT — Модель глибинного навчання. Спрощена модель BERT (англ. a light BERT)

ELECTRA — Модель глибинного навчання. Ефективне навчання енкодеру, який точно класифікує заміни токенів (англ. Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

RoBERTa — Модель глибинного навчання. Надійно оптимізований підхід до переднавчання BERT (англ. A Robustly Optimized BERT Pretraining Approach)

GPT — Модель глибинного навчання. Трансформер навчаний для генерації тексту (англ. Generative pre-trained transformer)

CTRL — Модель глибинного навчання. Модель мови умовного трансформера для керованої генерації (англ. a Conditional Transformer Language Model for Controllable Generation)

Transformer-XL — модель глибинного навчання. Модель мови з увагою, за межами контексту фіксованої довжини тексту (англ Attentive Language Models Beyond a Fixed-Length Context)

BART — модель глибинного навчання. Двонаправлені і авторегресивні трансформери (англ. Bidirectional and Auto-Regressive Transformers)

mBART — модель глибинного навчання. Багатомовний автоенкодер, з застосуванням усунення шуму, навчаний на моделі BART.

Marian — модель глибинного навчання з архітектурою

енкодер-декодер. Названа на честь Мар'яна Раєвського, польського математика і криптолога.

T5 — модель глибинного навчання з архітектурою енкодера-декодера, навчена на багатозадачному поєднанні неконтрольованих і контрольованих задач.

n-грама — послідовність з n елементів з певної вибірки тексту або мовлення.

ВСТУП

Актуальність дослідження. Оскільки попит на ефективні моделі глибинного навчання NLP зростає, науковці та комерційні компанії все більше зосереджуються на пошуку шляхів покращення продуктивності цих моделей за допомогою методів попередньої обробки даних. Методи токенізації WordPiece та SentencePiece широко використовуються в різних алгоритмах машинного навчання та показали свою ефективність у задачах NLP, як-от: автоматизоване реферування (англ. summarization), запитання-відповідь (англ. QA, question-answering), класифікація (англ. classification), генерація тексту (англ. text generation), покращенні продуктивності моделі та зменшенні шуму. У контексті дослідження моделей генерації тексту, оптимальний підбір методу токенізації може забезпечити більшу точність моделі, та економію виробничих-комп'ютерних ресурсів. Дані методи були застосовані на корпусі медичних статей, що дозволяє дізнатись результати медичних досліджень в короткому підсумковому тексті з-за допомогою даних методів машинного навчання.

Метою дослідження є порівняльний аналіз методів попередньої обробки — токенізації тексту, на прикладі методів SentencePiece та WordPiece для алгоритму глибинного навчання Transformer, для підвищення продуктивності у розв'язанні задачі автоматизованого реферування медичних результатів.

Об'єктом дослідження є методи токенізації текстових даних для глибинного навчання в застосуванні в NLP на прикладі автоматизованого реферування.

Предметом дослідження є особливості використання WordPiece та SentencePiece, для моделі глибинного навчання Transformer на прикладі застосування підсумовування тексту медичних результатів.

Наукова новизна полягає в дослідженні роботи токенізаторів

SequencePiece, WordPiece для seq2seq моделей на прикладі завдання автоматизованого реферування для результатів медичних статей, на тему ознаки симптомів хвороб.

Практичне значення результатів полягає в використанні запропонованих методів, задля таких задач, як: запитання-відповідь (англ. QA, question-answering), класифікація (англ. classification), генерація тексту (англ. text generation), покращенні продуктивності моделей архітектури енкодер-декодер.

1 МЕТОДИ ТА ПІДХОДИ ВИРІШЕННЯ ЗАДАЧІ АВТОМАТИЗОВАНОГО РЕФЕРУВАННЯ

В даному розділі будуть основні теоретичні відомості про об'єкт дослідження. Коротко оглянуто суміжні розділи в даній сфері.

1.1 Застосування задачі автоматизованого реферування

Автоматизоване реферування тексту це процес вибірки великої текстової інформації у стислі та зв'язні підсумовування-резюме, цей підхід став важливим інструментом у різних сферах. Сфери застосування задачі автоматизованого реферування: новини, журналістика, аналіз наукових статей, бізнес та інші сфери. Автоматизоване реферування змінило підхід до розуміння великих обсягів текстових даних.

В цифрову епоху, яка характеризується великим обсягом даних, можливість ефективно та швидко відокремлювати головні ідеї та сенси з великого обсягу інформації набуває все більшого значення. Автоматизоване реферування вирішує цю потребу, з-за допомоги алгоритмів, що аналізують текст, та виділяють основну інформацію, дозволяючи користувачам даних алгоритмів, зрозуміти основні концепції без вичерпного читання тексту.

Дані методи використовуються в журналістиці, де потрібно узагальнити численні джерела інформації, та відокремити найважливіші факти про події, скорочуючи об'ємні сттаті, репортажі та трансляції новин.

В академічній та дослідницькій сферах дані методи потрібні для швидкого розуміння великих обсягів наукових статей та дослідницьких робіт, що значно підвищує швидкість пошуку та обробки інформації дослідниками. Також дані алгоритми дозволяють розуміти основну суть

складних текстів, визначити релевантні джерела та ефективно синтезувати інформацію.

Також автоматизоване реферування тексту допомагає фахівцям зі сфери бізнесу та економіки аналізувати ринкові тенденції, стратегії конкурентів, аналізуючи ключову інформацію з об'ємних галузевих статей та звітів, збираючи основні текстові дані в невеликий за обсягом звіт.

Задача порівняння методів кодовування (токенізації) текстів, для застосування в моделях глибинного навчання розглянуто в статті [1], де розглядалися алгоритми SentencePiece [2] та Mecab-Ko [3] на задачі класифікування використовуючи різноманітні алгоритми, такі як: KNN, SVM, ANN, LSTM-RNN. В даному дослідженні, алгоритм SentencePiece [2] показав вищу продуктивність порівняно з токенізатором на основі морфемного аналізу в задачі класифікації в онлайн-оглядах товарів [1].

Також дана задача розглядалась в статті [4], в даній статі розглядалось 13 токенайзерів на прикладі медичних статей з вебресурсу MEDLINE, це токенайзери з вільного доступу. Методи розглянуті в різних сценаріях, як приклад: токенізація слів зі зворотною рискою, зі складених слів, та інші. Дана статя не застосовує методи глибинного навчання, а надаються рекомендації, де краще застосовувати той, або інший токенайзер.

Існує два типи розв'язання задачі автоматичного реферування тексту: екстрактивна та абстрактнивна. Екстрактивні методи автоматичного реферування працюють шляхом вибору найбільш значущих речень та (або) слів для формування висновку. Замість того, щоб генерувати нові слова або фрази, цей підхід спирається на наявний текст. Абстрактивне автоматичне реферування навчається формулювати той текст, якого не було в початковому (на якому навчалась ця модель), та сформувати свій висновок в короткому рефераті. Воно передбачає створення нових слів і фраз, їх поєднання та передання важливої інформації з оригінального тексту. Абстрактивні методи реферування є

складнішими, ніж екстрактивні, і вимагають більше обчислювальних ресурсів.

Прикладом екстрактивних моделей є лексичне ранжування (англ. LexRank) [5], латентно-семантичний аналіз (англ. Latent semantic analysis, LSA) [6], латентне розміщення Діріхле (англ. Latent Dirichlet Allocation, LDA) [7], а також ранжування тексту (англ. TextRank) [8].

Однією з відомих моделей екстрактивного реферування є TextRank. TextRank — це алгоритм автоматизованого реферування тексту, який використовує підхід на основі графів. Він був запропонований у 2004 році в роботі «TextRank: Bringing Order into Texts» [8]. TextRank — модель ранжування для обробки тексту. Дана модель використовується для обробки природної мови. Зокрема, ця модель є методом для отримання ключових слів і речень з тексту, тобто виконує задачу автоматизованого реферування, та є прикладом екстрактивного методу видобутку ключового тексту. TextRank визначає найважливіші речення з тексту, покладаючись лише на сам текст, на відмінну від абстрактивних моделей, що покладаються на побудову підсумовування, намагаючись вивчати інші еталонні реферування. [8, ст. 8]

Існує кілька моделей, які можуть бути використані для абстрактного реферування тексту, такі як:

1) Pointer-Generator Network [9] — поєднує в собі механізми екстрактивного та абстрактивного генерування тексту. Модель може вибирати слова з оригінального тексту або генерувати нові слова, які краще підходять для стислого опису тексту.

2) BART (Bidirectional and Auto-Regressive Transformer) [10] — ця модель базується на трансформерній архітектурі і була натренована для абстрактного реферування. Вона може виконувати як генерацію тексту, так і виділення ключових фраз з оригінального тексту.

3) PreSumm [11] — Це модель, яка використовує енкодерний підхід для абстрактного реферування. Вона використовує кодувальну модель для отримання складових тексту, а потім генерує компактне реферативне

представлення.

1.2 Машинне навчання та нейронні мережі

Машинне навчання (англ. Machine Learning, ML) — це розділ прикладної математики, що зосереджений на розвитку алгоритмів, здатних імітувати певні типи поведінки людини, що надалі використовується в розв'язанні різних задач.

Розглянему цей процес на прикладі прогнозування опадів. Використовуючи традиційний підхід — створити фізичне представлення атмосфери та поверхні Землі, обчислюючи велику кількість рівнянь динаміки рідини. Це є складною задачею. Для ML-моделі надають великі обсяги погодних даних, поки ML-модель не пристосувалась до зв'язків між параметрами погодних умов, що спричиняють різну кількість дощу. Потім ми надавали б моделі нові погодні дані, і вона прогнозувала кількість дощу.

Нейрон є основною одиницею обробки інформації в нейронних мережах. Він функціонує як модель біологічного нейрона і має здатність обробляти та передавати сигнали. Нейрон отримує вхідні дані, обробляє їх і генерує вихідний сигнал. Кожен нейрон має вхідні зв'язки, ваги і функцію активації. Вхідні зв'язки представляють собою сигнали, які надходять до нейрона з інших нейронів або зовнішніх джерел. Кожному вхідному зв'язку присвоюється вага, яка визначає важливість цього зв'язку для нейрона. Вага множиться на вхідний сигнал і використовується для вирішення, наскільки великим буде вплив цього сигналу на активацію нейрона. Функція активації визначає, коли нейрон буде активований і який буде його вихідний сигнал на основі обчислень, які він виконує. Ця функція може бути сигмоїдною, гіперболічним тангенсом, або іншою. В результаті обчислень нейрон генерує вихідний сигнал, який може бути поданий на вхід іншим нейронам у мережі або використовуватися для прийняття рішень. Чим більша кількість нейронів

у прихованому шарі, тим більше нелінійностей може модель враховувати. Оптимальний розмір та кількість прихованих шарів зазвичай визначаються шляхом експерименту та налаштування моделі для конкретного завдання, оскільки завелика кількість шарів, може призвести що модель буде заскладною, для виконання певного завдання, та відбуватиметься перенвачання моделі (англ. overfitting) — проблема у машинному навчанні, коли модель набуває високої точності на навчальних даних, але показує погані результати на нових даних.

Нейронні мережі — це багатошарові мережі нейронів, що використовуються для задач класифікації, прогнозування тощо. Нижче наведено схему простої нейронної мережі, з 5 входами та виходами (1.1). Функція прихованого шару полягає в обчисленні складних внутрішніх представлень та виконанні необхідних обчислень між вхідними та вихідними шарами.

Глибинне навчання — це частина машинного навчання, прикладом моделі глибинного навчання є нейронна мережа з трьома або більше шарами. Ці нейронні мережі вчаться на великих обсягах даних. Хоча нейронна мережа з одним шаром все ще може робити приблизні прогнози, додаткові приховані шари можуть допомогти оптимізувати та вдосконалити точність.

1) Помаранчевим — вхідний шар нашої моделі.

2) Прихований шар нейронів синього кольору.

3) Вихідний шар нейронів пурпуровим.

4) Вихідні дані зеленим кольором.

Абстрактивна модель генерації тексту — модель машинного навчання, що може створювати новий текст, напротивагу екстрактивним методам. Також вони не дістають слова, або частини слів зі вхідного тексту, а навчаються генерувати нові словосполучення, в залежності від вхідних даних. Деякі моделі з сімейства Transformer [12], використовують для абстрактивного автоматизованого реферування тексту.

Енкодер — це архітектура, що використовується в глибинному

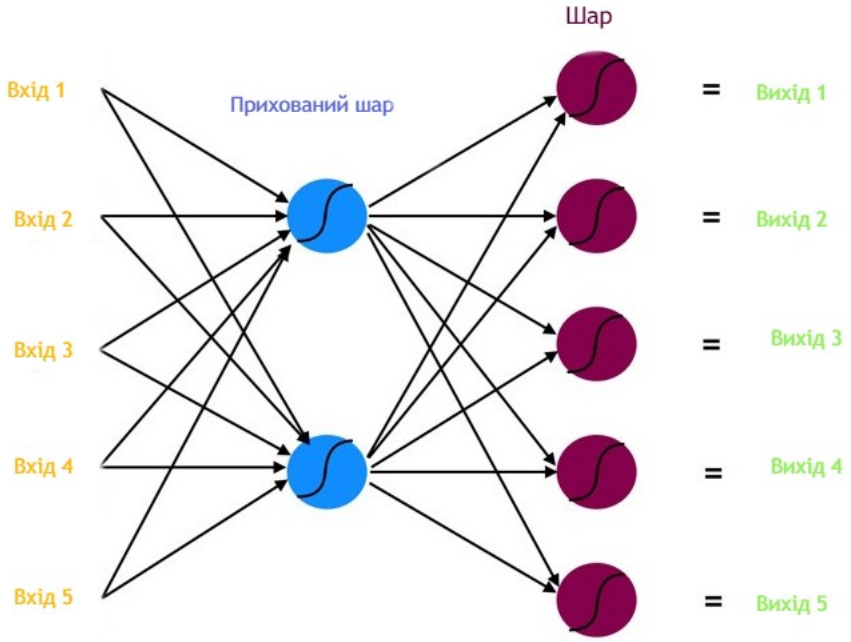

Рисунок 1.1 – Приклад архітектури нейронної мережі

навчані, що кодує вхідні дані, в коротше представлення, потім ці дані передаються декодеру, що переводить закодовану інформацію у вихідну форму.

Декодер — це тип архітектури нейронної мережі, що теж використовується в глибинному навчанні. Його функція полягає в тому, щоб генерувати вихідні дані з закодованих представлень.

У структурі Transformer енкодер приймає на вхід послідовність закодованих символів $(x_1, \ldots, x_n)$ і перетворює її в послідовність неперервних репрезентацій $(z_1, \ldots, z_n)$. Декодер, використовуючи закодовані репрезентації (z), генерує вихідну послідовність $(y_1, \ldots, y_m)$, видаючи по одному символу за раз. Модель працює в авторегресійному режимі, тобто використовує попередньо згенеровані символи як нові вхідні дані при генерації наступного символу на кожному кроці. [12, ст. 2-3]. Рисунок моделі продемонстрований нижче на рисунку (1.2).

Трансформери часто використовуються для задач NLP (обробка природної мови), вони можуть виконувати ряд задач, таких як: генерація

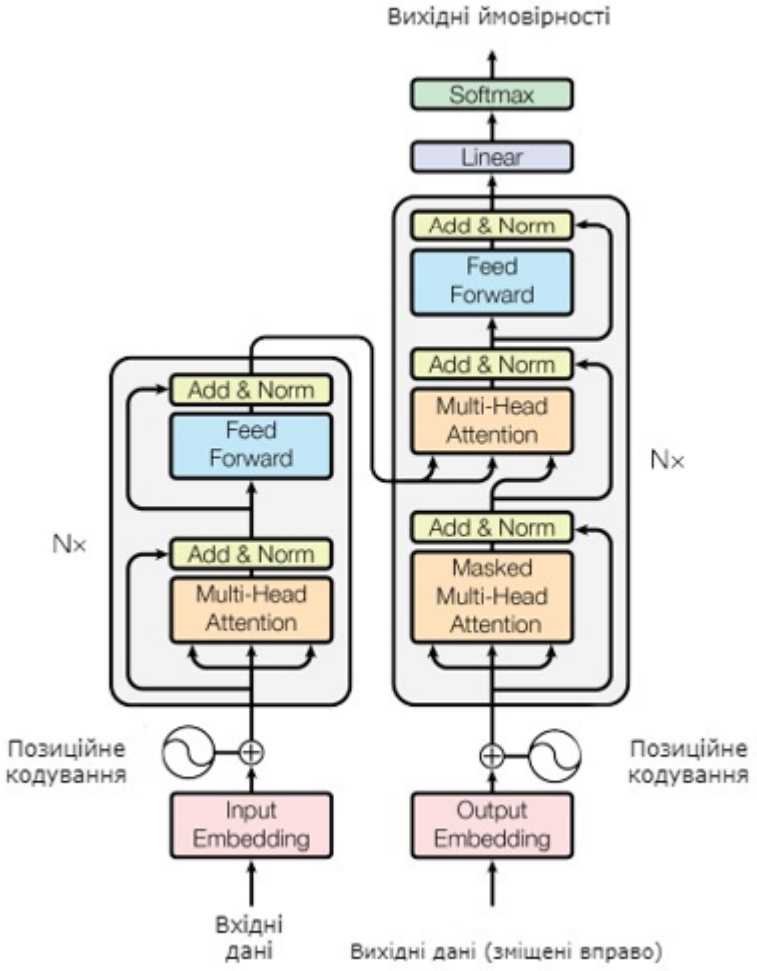

Рисунок 1.2 – Архітектура моделі трансформер

тексту, підставляння слова на позицію (маску), переклад, питання-відповідь, та інші задачі NLP. якщо масштабувати архітектуру трансформера, тоді його можливо звести До енкодера-декодера, як на рисунку (1.3).

Увага (англ. attention) — частина архітектури трансформер. Увага зберігає контекстне відношення слова (або частини слова, токена) в реченні. Розглянемо такий приклад: є речення «Я хочу приїхати сьогодні додому», для того, щоб точно інтерпретувати слова, модель повинна враховувати контекст і залежність між ними. Щоб інтерпретувати слово «приїхати», модель повинна врахувати попередні й наступні слова, такі як «сьогодні» і «додому», бо слово «додому» в семантичному значенні

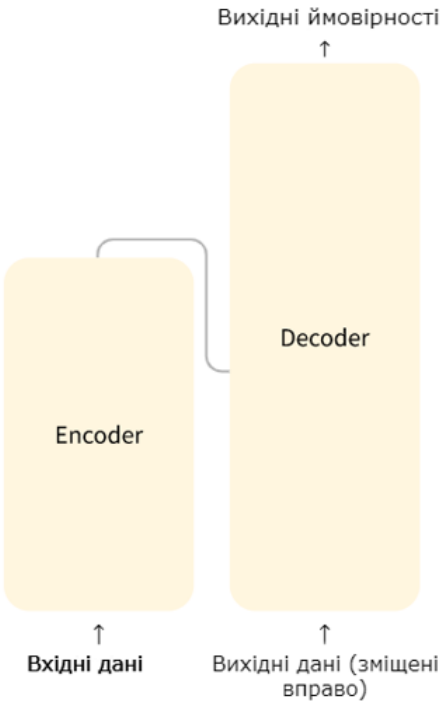

Рисунок 1.3 – Спрощена архітектура моделі трансформер

позначає ціль, або напрямок, тому якщо модель не буде це враховувати, вона зовсім не буде розуміти позиційний контекст частини речення, що може призвести до неточної моделі. Модель повинна точно розуміти позиційне значення частини речення, або слова, навіть якщо два слова не є суміжними між собою.

Трансформер використовує багатоголову увагу (англ. Multi-Head Attention), як було зазначено в статті [12, ст. 5]. На малюнку (1.4) V, K, Q — це вхідні дані: значення (англ. value, V), ключ (англ. key, K), запит (англ. query Q). Кожне обчислень називається головою уваги. Модуль уваги розбиває свої вищезазначені параметри Запит, Ключ, Значення на N частин і обчислює кожну незалежну частину через окрему голову. Модуль уваги, повторює свої обчислення декілька разів паралельно. Далі обчислення об'єднуються разом, щоб отримати кінцеву оцінку уваги.

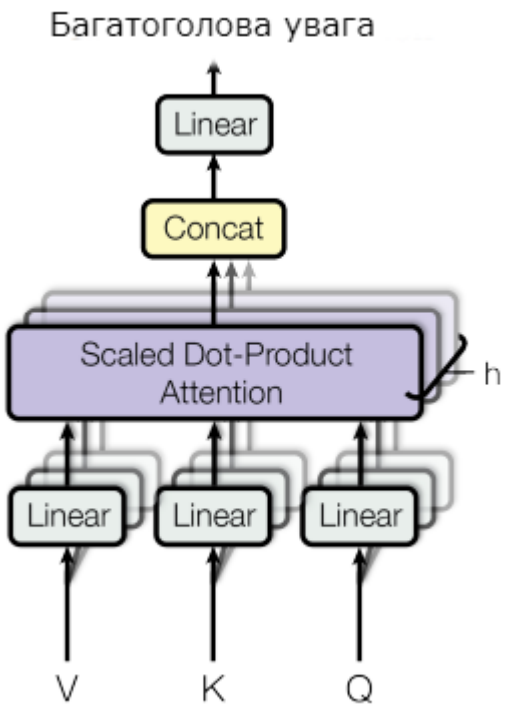

Рисунок 1.4 – Архітектура моделі трансформер

1.3 Відомі приклади трансформерів

Наразі існує така класифікація трансформерів:

1) Моделі з використанням лише енкодерів — ці моделі підходять для задач, де потрібно класифікувати текст, або розпізнавання іменованої сутності (англ. NER, named entity recognition), що означає знайти і класифікувати певні частини тексту, наприклад; Дмитро$_{[\text{Людина}]}$ влаштувався працювати в Гугл$_{[\text{Компанія}]}$.

2) Моделі з використанням лише декодерів — ці моделі підходять для задач генерації тексту

3) Енкодер-декодер моделі — підходять для задач генерації тексту, де на вхід йде певна послідовність тексту, наприклад переклад, або автореферальне підсумовування.

Відомими прикладами енкодерів є такі моделі, як: ALBERT [13], BERT (Bidirectional Encoder Representations from Transformers,

двоспрямовані кодувальні представлення з трансформерів) [14], ELECTRA [15], RoBERTa [16]. Основним представником в цьому списку є BERT [14], публікація була здійснена незадовго після публікації першої моделі GPT [17]. Модель BERT [14] складається з енкодерів з моделі трансформер, в статті пропонується декілька різних варіацій моделі, важливим є те, що використання тільки енкодерів, як і в GPT [17], може призвести до втрачання поставленої задачі.

По-перше, вони навчають модель передбачати замасковані слова в реченні, де певні слова випадковим чином маскуються, а модель має передбачити їхні початкові значення.

По-друге, вони навчають модель визначати, чи два вхідні речення з'являються послідовно, чи ні, допомагаючи їй вивчити зв'язки між реченнями. Ці дві схеми попереднього навчання дозволяють BERT [14] досягти сильного розуміння мови та покращити її продуктивність при виконанні широкого спектра завдань. Такі моделі як ALBERT [13], ELECTRA [15], RoBERTa [16], є покращеннями моделі BERT [14], та пристосуванням до певних задач.

Відомими прикладами моделі трансформер, з використанням лише декодера, є такі моделі як: CTRL [18], GPT [17], GPT-2 [19], Transformer XL [20]. Найвідомішим представником таких моделей є GPT-2 [19], який використовує багато шарів декодерів, що дозволяє їй виконувати задачі генерації тексту.

Відомими прикладами моделі трансформер з використанням енкодер-декодер, або під назвою «послідовність в послідовність» (англ. seq-2-seq) є такі представники як: BART [10], mBART [21], Marian [22], T5 [23].

Першопрохідцем в даному списку є BART [10], він використовує архітектуру seq2seq, зі статті [12], але, так само як і в GPT [17], змінює функцію активації. Модель використовує 6 шарів енкодерів-декодерів, а для більшої реалізації моделі використовує 12 шарів, архітектура схожа з моделлю BERT [14], лише з різницею в тому, що декодер виконує

додаткову, перехресну увагу, для кожного прихованого шару енкодера, як в seq2seq, також використовує мережу прямого поширення, перед передбачуванням [10, ст. 2], також в статті вказано, що модель має найкращі показники в метриці ROUGE-L [24], в порівнянні з попередниками, на задачі питання-відповідь [10, ст. 7].

В даній роботі буде використовуватись архітектура seq2seq, через те, що тут розглядається задача автоматизованого реферування, також буде використовуватись метрика ROUGE-L [24].

1.4 Методи токенізації

Токенізація в обробці природної мови, це важливий процес, що перетворює дані (текст), в структурований формат, потрібний для моделювання, початковий крок, це збір даних (тексту), його розподіл на навчальний та тестовий, обробка тексту, така як: видалення незначущих слів, приведення слів до одного формату (до малого регістру), видалення незначущих знаків (числа, часто повторювані незначущі слова). Далі проходить токенізація —- приведення обробленого тексту до формату, прийнятного для моделі, яким є числа, матриці, або тензори.

Існує багато моделей токенізації, але основні, такі як байт-парне кодування [25, 26] (англ. byte-pair encoding, BPE), WordPiece [27], SentencePiece [2] будуть розглядатися в даній публікації.

1) BPE — цей метод токенізації був запропонований в статтях [25, 26]. Метод його роботи доволі простий, по-перше, створюється словник символів, за допомогою певного словника, після цього кожне слово представляється як послідовність символів, з додавання спеціального символу кінця слова «-», це допоможе відтворити слово, після його закодовування з-за допомоги токенайзера. Далі ітеративно підраховуються всі пари символів, і замінюється кожне входження найбільш частих пар символів, наприклад («Б» «Г») на «БГ». Кожна операція злиття створює новий символ, є певною символьною енграмою,

такі n-грами, що часто зустрічаються об'єднуються в один символ. [25, ст. 3]. Даний алгоритм токенізації показав свою ефективність, та був застосований такими моделями як: GPT [17], GPT-2 [19], RoBERTa [16], BART [10].

2) WordPiece — цей метод був запропонований в статті [27]. Алгоритм токенізації був розроблений компанією Google, для попереднього навчання моделі BERT [14]. З часу винайдення цього методу, він неодноразово був використаний і в інших моделях, які покладались на модель BERT [14], одна з яких є DistilBERT [28]. Як і BPE [25, 26], WordPiece [27] починає свій алгоритм з певного створеного словника, який містить частини речення, тексту, та заданий алфавіт. Він позначає початок підслова певним символом, наприклад як в моделі BERT [14] «##». Кожне слово розбивається на частини, разом з додаванням префіксу – позначення початку підслова, далі слово розбивається на частини, візьмемо за приклад слово «небо»: «н, ##е,##б, ##о». Потім так само, як і в BPE [25, 26], вивчає правила конкатенації знаків, основна різниця в тому, що вибір пари для об'єднання виконується іншим чином, WordPiece [27] розраховує оцінку для кожної пари за формулою: с

$$\text{Оцінка} = \frac{\text{частота\_пари}}{\text{частота\_першого\_елемента} \times \text{частота\_другого\_елемента}} \quad (1.1)$$

Далі відбувається сам процес токенізації, він відрізняється від BPE [25, 26], тим що WordPiece [27] не зберігає кінцевий словник. Коли подається слово для токенізації, він шукає найдовше підслово, що є в словнику, потім розбиває слово на основі цього підслова. Наприклад якщо є натренований словник слова «руки», то буде розбиття буде проведеено таким чином: «рук», «##и». Якщо токенайзер не може знайти слово в словнику, тоді слово позначається спеціальним знаком «UNK».

3) SentencePiece — вперше цей токенайзер був представлений в статті [2]. Цей токенайзер зарекомендував себе в таких моделях як

ALBERT [13], T5 [23], mBart [21] та інші. Це токенізатор тексту, створений в для генерації тексту, для моделей глибинного навчання, або моделей машинного навчання, він застосовується для таких мереж, де розмір словника задається перед навчанням нейронної моделі, він реалізує підслова, як за прикладом BPE [25, 26], об'єднуючись з моделлю Unigram [29]. SentencePiece [2] – це ефективний спосіб розв'язання проблеми проблеми нестійкого словника, як вже сказано вище, ця модель є поєднання двох токенізаторів зазначених вище, з такими відмінностями:

– Кількість унікальних лексем (частин речення) є наперед визначеною, ця модель передбачає не передбачає нескінченний розмір словника, вона тренується таким чином, щоб словник був фіксованого розміру (наприклад 8, 16, 32 тисячі символів).

– Токенайзер може навчатись на основі необроблених попередньо речень.

– Пробіли — це базовий символ, він не опускається під час токенізації, ця модель обробляє пробіли як символ з юнікоду «_», тому пропадає двозначність під час детокенізації.

1.5 Метрики оцінки точності моделей, для задачі автоматизованого реферування

Існують різноманітні способи та методи оцінки якості моделей, наприклад, експертне оцінювання: людина вручну оглядає результати моделі, та оцінює якість автоматичного реферування. Проте така оцінка не завжди є надійною, також ця оцінка не ефективна для оцінки моделі на великих даних, коли модель може дати збій. Тому дослідники запропонували такі метрики оцінки, як Rouge [24], Blue [30]. Розпочнемо огляд метрик з метрики Blue, тому, що вона є попередником сімейства моделей Rouge [24].

1) Bleu — вперше ця метрика була описана в статті [30]. Ця метрика

працює за принципом обчислення точності (англ. precision) - частина токенів, символів, що покриваються в реферальному реченні. Значення оцінки даної метрики - це числа від 0 до 1. Нехай в нас є речення «Він любить гладити котів», якщо в нашому реферальному реченні не зустрічається жодного слова з даного речення, тоді оцінка точності буде 0/4, де 4 - це кількість слів у вихідному реченні. Нехай наше реферальне речення задане таким чином: «Я гладити гладити котів», тоді точність даного реферального, або підсумкового речення буде 0.50. Проте не важко побачити, що слово «гладити» зустрічається зайвий раз, та надає шуму нашому реферальному реченню, що означає меншу якість моделі. Bleu [30] враховує такі моменти, та він штрафує слова, що входять в реферальне речення більшу кількість разів, ніж в початковому реченні, або статті. Тобто основним завданням Bleu є порівняння n-грам кандидата з n-грамами реферального речення та підрахувати кількість збігів. В даній метриці, береться середнє геометричне значення модифікованих оцінок точності, що множиться на коефіцієнт (під експонентою) штрафу стислості. Також, перед обчисленням виконується нормалізація тексту, а саме приведення до єдиного регістру. Нехай тоді геометричне середньої точності n-грам буде p_n, а c - довжина претендента на реферування, або переклад, а число r - довжина вхідного тексту, w_n - це ваги n-грам, тоді штраф, за недостатність тексту в реферальному тексту буде [30, ст. 5]:

$$\mathrm{BP} = \begin{cases} 1 & \text{if } \ c > r \\ e^{(1-r/c)} & \text{if } \ c \leq r \end{cases} \tag{1.2}$$

А сама метрика набуватиме значення [30, ст. 5]:

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{1.3}$$

Але ця оцінка має краще поведінку при логарифмуванні [30, ст. 5]:

$$\log \text{BLEU} = \min\left(1 - \frac{r}{c}, 0\right) + \sum_{n=1}^{N} w_n \log p_n \tag{1.4}$$

Таким чином існують різні метрики Bleu [30]:

– BLEU-1 використовує оцінку точності уніграми.

– BLEU-2 використовує середнє геометричне значення точності уніграми та біграми.

– BLEU-3 використовує середнє геометричне значення точності уніграми, біграми та триграми.

– І так далі.

Перевагами даної метрики можна назвати:

– Ця метрика проста для розуміння

– Метрика схожа на оцінку людини

– Метрика може бути використана для великих даних

Недоліками є:

– Вона не може оцінювати значення слів, лише їх частоту (відсутнє поняття синонімів).

– Метрика містить різні варіації слова, наприклад слова «писала», «писати», «написаний», це все різні слова.

– Метрика не зважає на значущість слова в реченні.

– Метрика не зважає на перестановки слів, що може змінити значення речення.

2) Rouge — вперше сімейство цих метрик та програмний пакет було описано в статті [24], він розроблений для оцінювання автоматизованого реферування, машинного перекладу, та аналогічних задач. Метрики зрівнюють створене моделлю (наприклад, моделлю глибинного навчання) створене підсумовування з певним еталоном, перекладом, вхідним реченням. Розпочнемо огляд даного сімейства з Rouge-N:

Rouge-N [24] — вираховує, аналогічно з метрикою Bleu [30], кількість збігів n-грам між текстом, що згенерувала модель, та певним текстом,

створеним людиною. Проте окрім поняття точності, в Rouge [24] ще є поняття відгук (англ. recall) та Ф1-значення (англ. F1-score). Розглянемо приклад: реферальне речення - «Цей стіл зроблений майстром», та машинне підсумовування - «Мій стіл та та майстром». Тоді точність Rouge-1 [24] метрики буде 1/2. Вираховується за формулою з формули нижче (1.6). Де $Count_{match}$ — це кількість збігів в n-грамах, $Count$ — це кількість n-грам, $ReferenceSummaries$ — еталонні речення.

$$\frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)} \tag{1.5}$$

Значення показника відгуку, в прикладі вище буде 2/5 - це кількість слів (уніграм, 1-грам), що збігаються з реферальним реченням, до загальної кількості слів у висновку. Таким чином, маючи показники точності та відгуку, можна вирахувати F1-score, що є середньо гармонічним між цими показниками, так вираховується за формулою нижче (1.6):

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1.6}$$

Вирахуємо F1-значення для нашого прикладу: $F_1 = 2 \cdot \frac{\frac{1}{2} \cdot \frac{2}{5}}{\frac{1}{2} + \frac{2}{5}} = \frac{4}{9} = 0.44$. Обрахунки для Rouge-2 [24] проводяться аналогічним чином, лише обраховується покривання входжень біграм в речення.

Rouge-L [24] — полягається на найдовшу спільну підпослідовність (англ. longest common subsequence, LCS) в реферальному та змодельованому реченні. LCS - це не обов'язково послідовна послідовність, але впорядкована. Найдовша спільна підпослідовність повинна означати більшу схожість між двома реченнями, аналогічно до Rouge-N [24], ця метрика використовує відгук, точність та F1-score. Наведемо приклад: «я танцюю з вечірньою зіркою» — наше еталонне речення, а змодельоване — «я перу з зіркою». В цьому випадку LCS буде «я з зіркою» та прийматиме числове значення — 3. Тоді точність даного випадку — $\frac{3}{4}$. Обрахуємо точність, в такому випадку випадку відгук буде $\frac{3}{5}$ — це

співвідношення LCS, до числа уніграм в еталонному реченні. Тоді F1-score буде $F_1 = 2 \cdot \frac{\frac{3}{5} \cdot \frac{3}{4}}{\frac{3}{5} + \frac{3}{4}} = \frac{2}{3} = 0.66$.
Порівняємо сімейство метрик Rouge [24] та BLEU [30]: BLEU [30] більше звертає увагу на значення точності, тобто наскільки n-грами зі змодельованого речення потрапляються в вхідне, напроти метрика Rouge [24] сконцентрована на показнику відгуку, тобто скільки слів, чи n-грам з еталонного речення потрапляються в модель.

Висновки до розділу 1

В цьому розділі було оглянуто основні теоретичні відомості про об'єкт та предмет дослідження, а саме просто автоматизоване реферування тексту, наявні моделі машинного навчання, та моделі для обробки природної мови. Було здійснено короткий огляд суміжних розділів, таких як машинне навчання, екстрактивні моделі автоматизованого реферування. Були оглянуті основні методи токенізації, метрики, що оцінюють якість роботи алгоритмів автоматизованого реферування. З наявних моделей, для виконання дослідження обрано архітектуру seq2seq та метрики сімейства Rouge [24].

Ми також вказали на цінність токенізації для глибинного навчання в обробці природньої мови і важливість цієї теми, підтверджуючи значимість цього дослідження.

2 ПІДГОТУВАННЯ ДО ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

В даному розділі знаходиться огляд основних інструментів і підходів для аналізу даних, також буде зазначено використані інструменти та ресурси для моделювання. Проаналізована текстова статистика, на основі якої оброблено текстові дані.

2.1 Використані інструменти та ресурси

Мовою програмування було вибрано Python v3.8 [31], це досить ефективна на час програмування проблеми та гнучка мова, для розв'язання задач машинного навчання, в ній існує велика кількість готових бібліотек та ресурсів, які дозволяють доцільно розв'язувати задачі, такі як задача підсумовування тексту. Основною бібліотекою для створення моделі була Tensorflow v. 2.12.0 [32], ця бібліотека надає навчальні посібники, приклади, та інші ресурси, для прискорення побудови моделей і створення масштабованих рішень з машинного навчання, особливо цікавою для досліджень були дві підбібліотеки, такі як:

– Keras v. 2.12.0 [33] — це інтерфейс прикладного програмування (англ. Application Programming Interface, API), для глибокого навчання на основі Python v3.8 [31], який працює поверх TensorFlow v. 2.12.0 [32], платформи машинного навчання. Його основна мета - полегшити швидке експериментування, пропонуючи простий інтерфейс.

– keras_nlp v. 0.5.2 [34] — спеціалізована бібліотека, з готовими рішенням для задач NLP (обробки природної мови), вона має заготовлені рішення для створення енкодерів-декодерів, тюнінгу гіперпараметрів (пошуку найкращих поєднань гіперпараметрів моделі), метрики, та інші рішення. В даному дослідженні буде використовуватись енкодер-декодер

шари (англ. encoder-decoder) з даної бібліотеки, в них можна задати велику кількість параметрів, наприклад, такі: проміжна розмірність, кількість голів самоуваги, відкидання (англ. dropout) - це параметр, що визначає кількість випадкових нейронів, що при будь-яких вхідних даних, повинні повертати 0, що знаходяться як в прихованих, так і в явних шарах моделі, цей параметр допомагає знизити залежність моделі від шуму; шарова норма епсілон - також дозволяє нормалізувати зашумлені дані. Ці всі параметри надаються як для енкодера, так і для декодера. Також використані функції WordPiece [27] та SentencePiece [2] токенайзерів, та підготовки до створення словника WordPiece [27] та протофункції SentencePiece [2] для створення кодувальника речень.

– Для аналізу та препроцесингу були використані такі пакети, як: scikit-learn v. 0.22 [35], nltk v. 3.4.5 [36]. Метрикою, для оцінки рішення була Rouge-L та Rouge-1 [24].

Проаналізувавши можливі сервіси розташування даних, вебресурсом для даних став вебсайт https://pubmed.ncbi.nlm.nih.gov/ — цей вебресурс також є базою даних, яка має доступ до великої кількості науково-медичних статей, що є частиною Національної медичної бібліотеки США. Цей вебресурс використовують дослідники медичної сфери. Ця база даних цінна ще тим, що вона дозволяє фільтрувати за різними показниками статті, під час дослідження були використані дані зі статей про ознаки симптомів, які побудовані на випадкових контрольованих дослідженнях, за 2005-2023 роки. З відібраних статей, було взято результати та висновки досліджень. Задача – за вхідним результатами дослідження згенерувати висновок, було відібрано сто тисяч статей, що є мінімальним, проте достатнім для обчислювальних ресурсів, для перевірки ефективності методів токенізації, через те, що ці дані потребували відсортовування та обробки.

2.2 Попередня обробка даних

Після отримання даних, їх було використано в текстовому аналізі. Текстовий аналіз містить аналіз частоти слів, довжини речень, та середньої довжини слова — ці показники дозволяють дослідити фундаментальні характеристики видобутих даних. Всього не очищених даних — 62,947 пари текст-підсумовування. Спочатку було проведено первинний огляд даних. Первинний огляд дає загальне уявлення про характеристики та властивості тексту, що допоможе в подальшому опрацюванню даних.

Після первинного огляду, було відокремлено ті результати, що більші за висновки в 3 рази, це було зроблено, для спрощеного навчання моделі, через те, що зменшується ентропія можливих значень, та для моделі простіше буде зробити висновки з меншою кількістю фінальних символів. Після даного методу, залишилось 23,151 пари значень. Далі оглянуто кількість знаків, кількість слів, речень, та середня довжина слова і речення в реферальному та основному текстах. В таблиці (2.1) зображено огляд даних в текстах-результатах. В таблиці (2.2) зображено огляд даних в текстах-результатах.

	Min	**Mean**	**Max**
Кількість символів	61.0000	644.9777	3615.0000
Кількість слів	10.0000	155.0797	1034.0000
Кількість речень	1.0000	4.8931	26.0000
Середня довжина слова	2.2500	4.2218	6.8682
Середня довжина речення	10.0000	33.8532	319.0000

Таблиця 2.1 – Статистичний огляд даних - медичних результатів

Як видно з таблиці 2.2, у висновках існують речення що складаються з 1 слова — такі пари результат-підсумок погано повпливають на модель, тому, що з великої кількості визначити лише

	Min	Mean	Max
Кількість символів	1.0000	165.4340	793.0000
Кількість слів	1.0000	29.8695	162.0000
Кількість речень	1.0000	1.6485	6.0000
Середня довжина слова	1.0000	5.5856	18.0000
Середня довжина речення	1.0000	20.1100	80.0000

Таблиця 2.2 – Статистичний огляд даних - медичних висновків

одне слово — це не доцільна методика для визначення підсумовування, подальші дії будуть віднайти, та видалити такі пари. Також було визначено додатковий параметр для аналізу тексту — співвідношення кількості слів в парі результат-висновок (2.1) Як бачимо з рисунка (2.1),

Рисунок 2.1 – Початкова пропорція слів в текстах

дані зміщені, існує великий перепад між співвідношенням довжин. Вирахуване відхилення - 6.68. Надалі, відбираємо дані, пропорції довжин пар текстів яких не перевищують значення 3, щоб довжина тексту-результату була більшою, та ми напевне мали відображення з більшої множини в меншу. Далі відсортовуємо базу даних, за пропорцією, що відхиляється не менша за середнє значення плюс 5 відхилень, це

значення було емпірично встановлене, під час тестування моделі. Тепер графік пропорції виглядає таким чином (2.2).

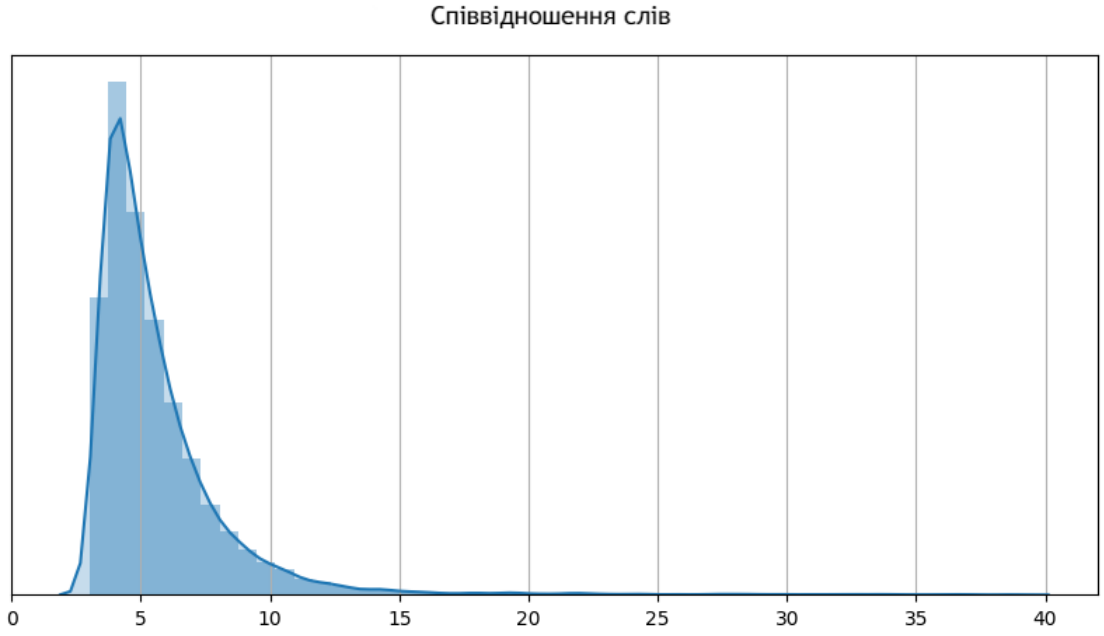

Рисунок 2.2 – Пропорція слів в текстах, після обробки

Як бачимо, що в цих текстах є викиди, співвідношення яких більше за значення 40, були відкинуті, це збільшить якість дослідження токенайзерів, та зменшить відхилення під час прогнозування. Також були відкинуті пропорції в 1-3 слова, що запобігає викидам в даних, коли підсумок - це лише назва дослідження, або певне посилання. Після видалення залишилося 22,664 пари текстів. Після оброблення тексту, кількість слів в підсумках виглядає таким чином — (2.3). Як бачимо з графіку, правий край викидів був частково обрізаний, але немає сенсу відкидати всі дані, що більше, наприклад, числа 40, тому, що тексти від 40 до, 80 (число 80 вибране експериментальним шляхом) можуть мати змістовні тексти, та не сильно впливати на зашумленість даних, видаливши їх, буде втрачена цінна інформація. Також, на рисунку (2.4), видно як змінився розподіл кількості слів в текстах-результатах досліджень.

Далі було знову оглянуто текст, та проведено попередньо обробку

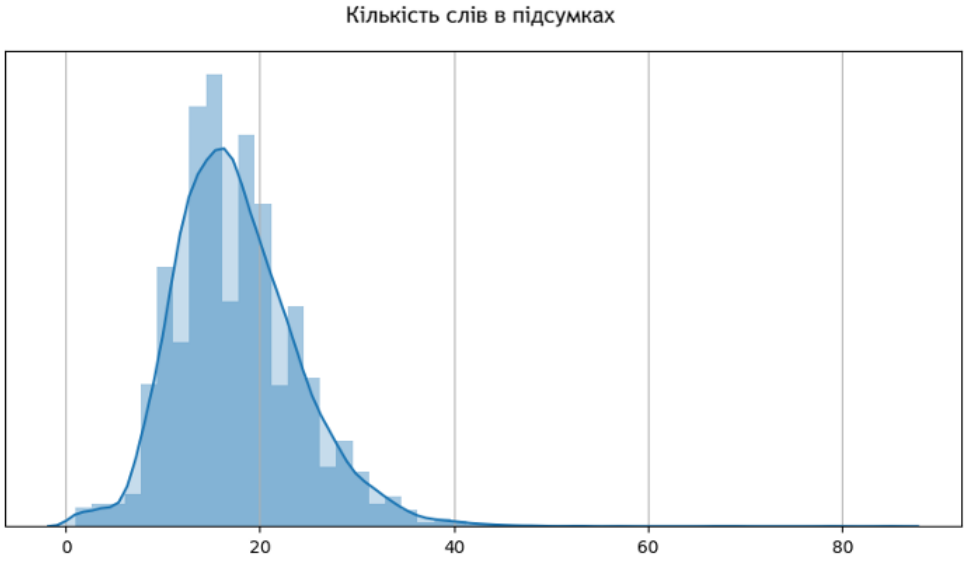

Рисунок 2.3 – Кількість слів в підсумках після обробки

вже самого тексту (англ. preprocessing), видалено пунктуацію, приведено до нижнього регістру, видалено слова, що занадто часто зустрічаються в текстах, такі як: «say», «said», «new», «p», «ci», «v», «n», «vs», та інші, як бачимо, оскільки текст — це медичний дослід, то в ньому зустрічаються часто такі словосполучення, як «p-value», довірчий інтервал (англ. confidence interval, ci), ці слова збільшують об'єм даних, та не є виправданими для моделювання на невеликій моделі, що застосовується в даному дослідження. Малий вплив на сенс змісту підсумку мають такі частини речення, як: пунктуація, числа, посилання на вебресурс — вони всі були видалені.

Після цих процедур, було проведено аналіз та порівняння підходів стемінгу, та лематизації, та вибрано найкраще поєднання цих методик, для даного набору даних. В наступній секції, розглянуто алгоритми стемінгу та лематизації.

2.3 Стемінг та Лематизація

Стеммінг і лематизація — це методи нормалізації тексту, для задач обробки природної мови, це методики підготовки або обробки текстових

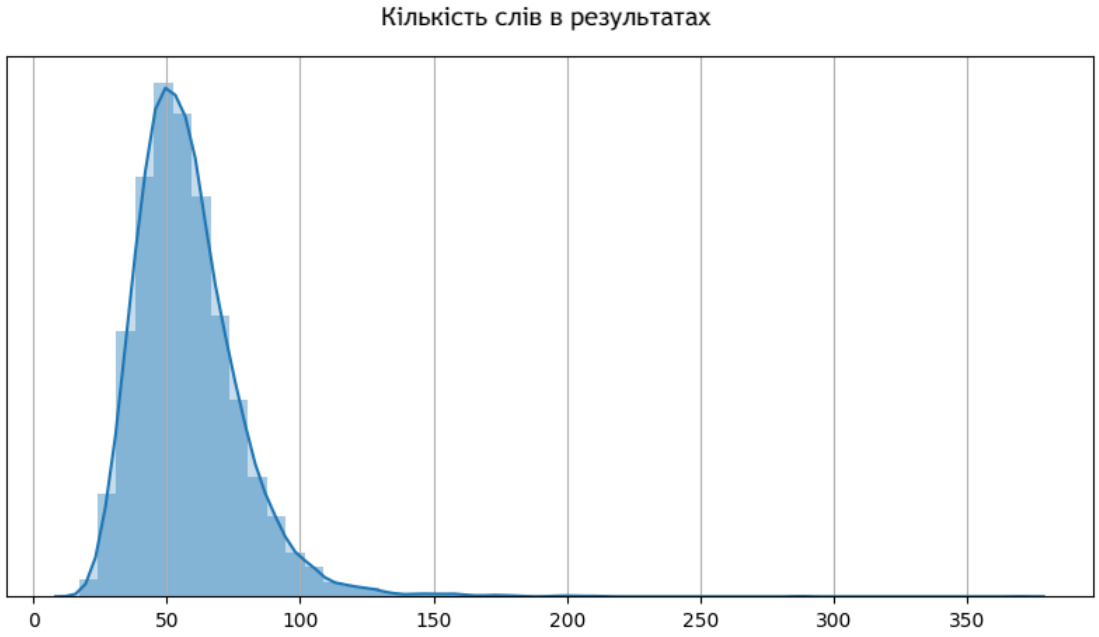

Рисунок 2.4 – Кількість слів в результатах після обробки

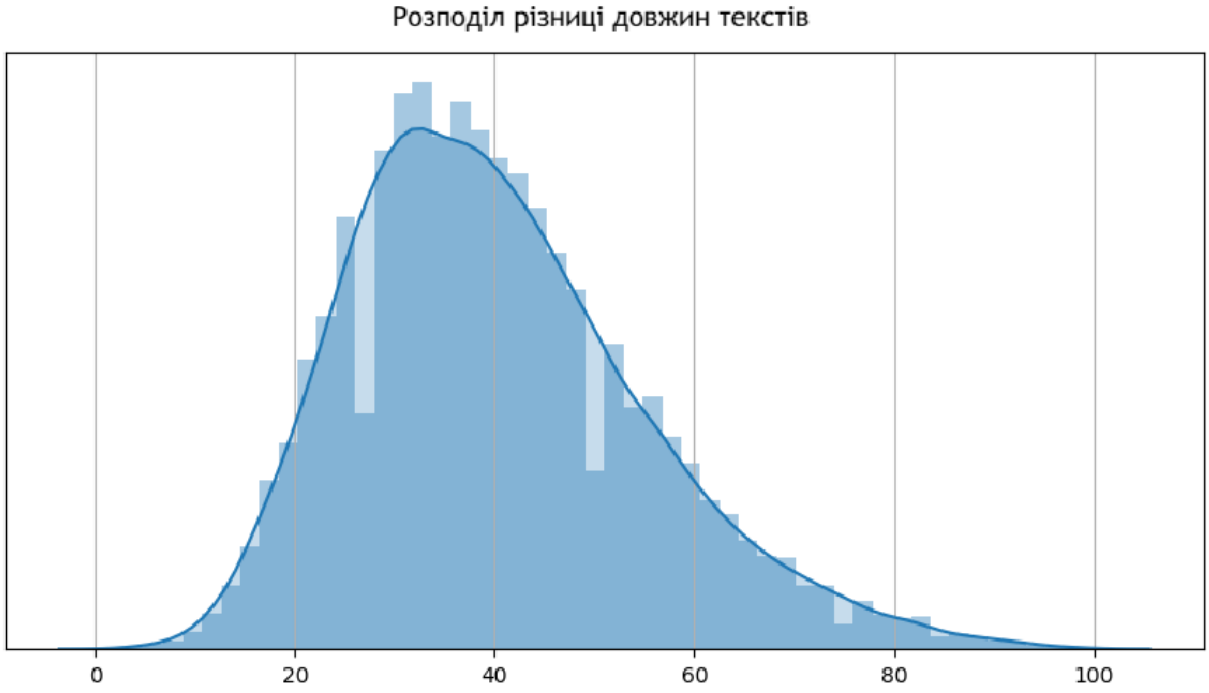

Рисунок 2.5 – Різниця довжин текстів

даних для подальшого використання в різноманітних алгоритмах, наприклад, алгоритмах глибинного машинного навчання, існують різні пакети в мові Python v. 3.8 [31] для виконання даних алгоритмів, в даному дослідженні використаний пакет nltk v.3.4.5 [36]. Іноді потрібно, задля ефективності, в сенсі точності моделі, та економії обчислювальних

ресурсів потрібно, щоб слово «look», «looking», «looked» - це просто різні форми одного дієслова — дані методи є зведення слів, до їх коренів, в цьому випадку до слова «look».

– Відсікання (англ. stemming) — це процес створення морфологічних різновидностей кореня слова, програми що проводять дану операцію — стемери. Наприклад, коли відбувається пошук, в пошукових системах за ключовим словом, доцільно, щоб система сприймала різні форми слова, наприклад на слово «дитина» система відповідала схожими словами, за потреби, такими як «діти», «дитячий» та інші. Стемінг основи слова — дещо евристичний метод класифікації споріднених слів, в його основі лежить відсікання закінчення слова, допоки не буде досягнуто його основи. Часто, це працює добре, проте в англійські та українській мовах бувають випадки, коли дана методика може призвести до зайвого відсікання, та зіпсувати слово, додавши ще більше шуму в текст.

– Лематизація — цей метод відрізняється від стемінгу, лематизація не просто обрізає, або скорочує слова, цей метод застосовує морфологічний аналіз, наприклад цей метод має випадки, такі як «was» перетворюється на «be», та слова «universe» та «univarsity», мають зовсім різне значення, але якщо в них обрізати закінчення, вони стануть одним, у випадку стемінгу. Вважається, що лематизація залишає більше інформації, чим просто розбиття на частини. Лематизація також звертає увагу на значення слова в залежності від позиції в реченні, аби дізнатись до якої частини речення є дане слово.

Даним чином, розробка лематизаторів для якоїсь певної мови, я складнішою задачею, аніж розробка алгоритмів стемінгу, оскільки для розробки ефективних лематизаторів потрібні глибокі лінгвістичні знання про будову мови, що потребує не лише залучення професійних програмістів в розробку, а й лінгвістів. Як і стемінг, так і лематизація мають на меті отримати базову або початкову форму відмінювання слів. Проте, головна відмінність полягає в тому, що стемінг не завжди може

отримати повнозначне слово, тоді як леми, в алгоритмі лематизації є повнозначними словами в заданій мові.

Стемінг дотримується заздалегідь визначеного алгоритмічного підходу, що сприяє його швидкому виконанню. З іншого боку, лематизація покладається на вокабуляр мови, або словник, що надає леми, що робить її повільнішою, ніж стемінг. Крім того, лематизація часто вимагає вказівки-мітки на частини мови для отримання правильної леми.

Враховуючи вищезазначене, якщо швидкість є вирішальним фактором, перевага надається стемінгу, оскільки лематизаторам потрібен додатковий час на сканування та обробку корпусу. Провівши декілька тестових запусків стемера та лематизатора, в даному дослідженні було прийнято рішення використати лематизатор.

Також було проведено видалення слів, частота входження в всі тексти має менше ніж 5 входжень. Це запобігає від входжень низькочастотних слів, які можуть вносити шум в дані та знижувати якість моделювання. Видалення слів, які зустрічаються менше ніж 5 разів, допомагає моделі навчатись на більш семантично значущих та репрезентативних словах у текстах.

Цей процес видалення низькочастотних слів є важливим кроком в попередній обробці тексту, оскільки допомагає зосередитись на більш інформативних та релевантних словах, що можуть мати велике значення для аналізу та моделювання даних. Після видалення цих низькочастотних слів, вислідний набір даних стає чистішим та зменшується кількість шуму, що допомагає покращити якість моделі та точність прогнозів.

Після очищення даних, було проведено розподіл даних на тренінговий, тестовий, та валідаційний. Дані для навчання моделі вибрано 70% від всього обсягу даних, валадіційні та тестові — інші 30%, з яких валідаційні, це 24%, а тестові 6%. Такий розподіл даних на тренувальний, тестовий і валідаційний набори був обраний з метою забезпечити надійну оцінку та валідацію моделі. Використання 70% даних для тренування дозволяє моделі отримати достатньо інформації для

навчання та створення вірного представлення. Це дозволяє моделі виявляти загальні закономірності і залежності в даних. Тестовий набір, що становить 6% від загального обсягу даних, залишається необробленим протягом усього процесу навчання та налаштування моделі. Він використовується в кінцевій стадії для оцінки кінцевої продуктивності моделі та порівняння методів токенізації.

Висновки до розділу 2

У даному розділі проведено огляд основних інструментів і підходів для аналізу даних. Для моделювання використаний вебресурс Pubmed, який є базою даних науково-медичних статей. Було проведено аналіз текстових даних з цього ресурсу шляхом обробки текстової статистики. Після первинного огляду було відібрано результати, що були більші за висновки в 3 рази. Це було зроблено для спрощення навчання моделі та зменшення ентропії можливих значень. Після цього було залишено 23,151 пару значень. Далі було проведено аналіз довжини текстів, кількості знаків, слів, речень, а також середньої довжини слів і речень в реферальних та основних текстах. Було встановлено та усунуто відхилення між співвідношенням довжин текстів. Після цього були відібрані пари текстів, чия довжина не перевищувала значення 3, щоб мати краще представлення для підсумків результатів. Далі база даних була відсортована за відхиленням від середньої довжини тексту, що було більше за середнє значення в 5 відхилень. Після цього залишилося 22,664 пари текстів.

Далі проведена попередня обробка тексту, включаючи видалення пунктуації, перетворення до нижнього регістру та видалення найрідших за частотою входження в текст слів, що не несуть сильного семантичного змісту. Було встановлено, що лематизація має кращу якість моделювання, і тому вона була обрана для подальшої обробки тексту.

3 ПОБУДОВА МОДЕЛІ ТА ОЦІНКА МЕТОДІВ

В даному розділі описано основні та додаткові параметри побудованої моделі дослідження. Також описано процес навчання моделі та результати порівняння методів токенізації. Вказана перевірка методів на тестових даних, з-за допомогою метрик Rouge [24].

3.1 Підготовка токенайзерів

Перед тим як будувати модель, потрібно підготувати токенайзери, спочатку потрібно створити протословники, для застосування методів токенізування. Було визначено параметри довжин вокабуляру текстів, це 9,207 для результатів та 5,113 для підсумків — це максимально можливі значення для протофункції SentencePiece [2], оскільки дані тексти не мають більше частин, щоб використати більший словник.

Маскування — це спосіб позначити для шарів, що обробляють послідовність, що певні дані у вхідному векторі відсутні, отже потрібно їх пропустити, під час обчислювання.

Наповнення (англ. padding) — це особлива форма маскування, коли маскування відбувається на початку, або кінці послідовності. Таке розбиття пов'язане з необхідністю кодування даних в суміжні пакети (англ. batches), для того, щоб всі пакети відповідали заданій довжині, відбувається розбиття послідовностей на чистини.

Після обчислення протофункцій, розглянемо з яких символів складається словник SentencePiece [2], переглянемо перші 20 значень. Як бачимо, в словнику є такі символи, як: «pad», «unk», «s», «/s», «_group», «_patient», «_ed», «_difference», «_significantly». Символ «pad» — символ початку наповнення. Наповнення додає спеціальний маркер «pad», щоб коротші послідовності мали таку саму довжину, як і найдовша

послідовність у пакеті або максимальну довжину, прийнятна для моделі. Також в SentencePiece наявний символ «unk» — що означає невідомий символ, тобто це ті символи, які модель кодування не знайшла в словнику, та не змогла закодувати наявними в ній числовими відповідниками. Символи «s» та «/s» маркують початок та кінець послідовностей. Розмір словника для результатів склав 9,207 закодованих символів, а для висновків – 5,113 символів. Також зауважимо, що цей метод закодовування залишає знак пропуску, або пробіл, та позначає його символом «_».

Таким чином, виглядає словник WordPiece [27]: «##ed», «##g», «##ing», «##ly», «##o», «[PAD]», «[UNK]», «[START]», «[END]», «significant», «month», «mean», «treatment». Аналогічно до SentencePiece, WordPiece має символи наповнення, невідомого символу, початку та кінця послідовності — «[PAD]», «[UNK]», «[START]», «[END]». Проте цей метод не позначає початок слова символом «_», як це зроблено в SentencePiece, напротивагу, даний метод позначає закінчення, що від'єднані від основної частини слова, такі як «##g», «##ing», «##ly», символами «##».

3.2 Підготовка гіперпараметрів моделі

Надалі, після проведення перевірки найкращих параметрів, для навчання моделі, з-за допомоги гіперпараметричного тюнінгу під назвою Баєсівська оптимізація (англ. BayesianOptimization) [37], отримано дані параметри моделі (3.1):

Кількість шарів визначає кількість послідовних енкодерів, та кількість послідовних декодерів. Занадто велика кількість шарів, може призвести до перенавчання моделі (англ. overfitting), коли модель занадто добре навчається на тренувальних даних, проте дає погані показники, при тестових. На основі даних гіперпараметрів, побудовано модель (3.3).

Параметер	**Значення**
Розмір партії (англ. batch size)	32
Довжина послідовності	64
Відсоток відсіву (англ. dropout rate)	0.1000
Кількість голів уваги	6
Проміжна розмірність	64
Розмірність наповнення	64
Кількість шарів	3

Таблиця 3.1 – Гіперпараметри моделі

3.3 Навчання та оцінка моделі. Перевірка SentencePiece та WordPiece

Оцінювальним параметром якості моделі стала заплутаність (англ. preplexity). Розглянемо з чого складається perplexity. Нас цікавить ймовірність, з якою наша модель приймає речення W, що складається з послідовності слів (w_1,w_2,...,w_N) (3.1)

$$P(W) = P(w_1,w_2,\ldots,w_N) \tag{3.1}$$

Наприклад, якщо потрібно, щоб модель надавала більшу ймовірність реченням, які є схожими з еталонним реченням. Уніграмна модель працює лише на рівні окремих слів. Якщо задано послідовність слів W, де кожна w_i – це одне слово, або токен, всього в реченні N слів. Уніграмна модель виводить ймовірність: (3.2)

$$P(W) = P(w_1)P(w_2)\cdots P(w_N) \tag{3.2}$$

Тоді n-грамна модель розглядає попередні (n-1) слова, щоб оцінити наступне. Наприклад, триграмна модель розглядає попередні 2 слова, так що: (3.3)

$$P(W) = P(w_1)P(w_2|w_1)P(w_3|w_2,w_1)\cdots P(w_N|w_{N-1},w_{N-2})$$

Де w_i: і-те слово

N: довжина речення (3.3)

Perplexity обраховується таким чином:

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1,w_2,...,w_N)}}$$

Де w_i: і-те слово

N: довжина речення (3.4)

Perplexity нормалізує ймовірність входження даного речення W, беручи логарифм від значення, та ділить на кількість слів в реченні. Логарифмування та ділення, та подальше, піднесення в експоненту (3.5).

$$\ln(P(W)) = \ln\left(\prod_{i=1}^{N} P(w_i)\right) = \sum_{i=1}^{N} \ln P(w_i)$$

$$\frac{\ln(P(W))}{N} = \frac{\sum_{i=1}^{N} \ln P(w_i)}{N}$$

Де w_i: і-те слово

$$e^{\frac{\ln(P(W))}{N}} = e^{\sum_{i=1}^{N} \frac{\ln P(w_i)}{N}} \tag{3.5}$$

$$\left(e^{\ln(P(W))}\right)^{\frac{1}{N}} = \left(e^{\sum_{i=1}^{N} \ln P(w_i)}\right)^{1/N}$$

$$P(W)^{\frac{1}{N}} = \left(\prod_{i=1}^{N} P(w_i)\right)^{\frac{1}{N}}$$

N: довжина речення

Далі з рівняння (3.5) отримуємо рівняння (3.4). Дана метрика інтерпретується як обернена ймовірність тестової множини, нормалізована за кількістю слів в множині. Оскільки це обернена

ймовірність, то нижче значення perplexity означає кращу модель.

Також використовувалась функція втрат (англ. loss function), а саме розріджена категоріальна кросентропія.

Тут приведений графік навчання для WordPiece [27] (3.1):

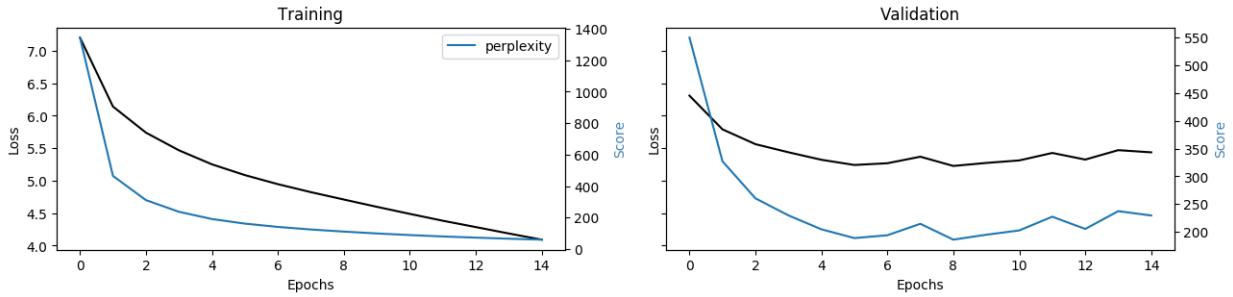

Рисунок 3.1 – Графік навчання моделі, з використанням WordPiece

Тут приведений графік навчання для SentencePiece [2] (3.2):

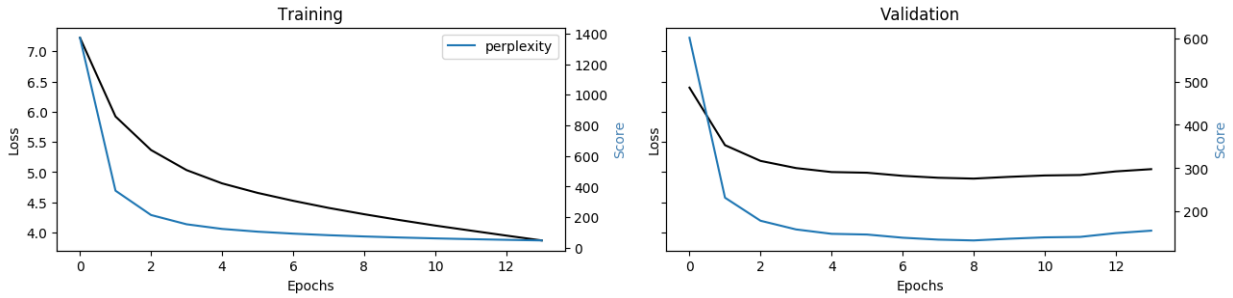

Рисунок 3.2 – Графік навчання моделі, з використанням SentencePiece

Зобразимо результати за метриками Rouge-l та Rouge-1 [24], для методу WordPiece [27] (3.2), було проведено 10 тестів з використанням вищезазначених метрик з генерацією речень.

Жадібний пошук (англ. Greedy Search) — спосіб генерації речень. Даний алгоритм працює наступним чином: відбирається слово, яке має найбільшу ймовірність на кожній позиції — воно і є наступним згенерованим словом. Він швидкий в обчисленні, простий у розумінні та часто дає правильний результат.

Променевий пошук (англ. BeamSearch) — алгоритм генерації речень.

Має два покращення порівняно з жадібним пошуком.

1) У жадібному пошуку приймається лише одне найкраще слово на кожній позиції. На противагу цьому, променевий пошук розширює цей діапазон і вибирає N найкращих слів.

2) У жадібному пошуку розглядається кожна позиція окремо. Визначивши найкраще слово для цієї позиції, не розглядається, що було попередній позиції або після нього. На противагу цьому, променевий пошук обирає «N» найкращих послідовностей і враховує ймовірність комбінації всіх попередніх слів разом зі словом у поточній позиції.

З-за допомоги методу пошук променем [38]. В даному дослідженні вибрано гіперпараметр кількості променів = 5.

Номер тесту	**Rouge-l**	**Rouge-1**
1	0.1193	0.1307
2	0.0891	0.0998
3	0.0891	0.0998
4	0.0891	0.0998
5	0.0891	0.0998
6	0.0889	0.0958
7	0.0877	0.0995
8	0.0809	0.0902
9	0.0745	0.0845
10-	0.0661	0.0804

Таблиця 3.2 – Результати за метриками Rouge-1 та Rouge-l для WordPiece

Додатково, в таблиці результатів (3.3) можна побачити порівняльні показники метрик сімейства Rouge [24] для методу SentencePiece [2]. Як видно, середні значення метрик Rouge-l [24] та Rouge-1 [24] для методу WordPiece [27] перевищують показники методу SentencePiece [2].

Це підкреслює перевагу методу WordPiece [27] при генерації підсумків

медичних текстів. Він забезпечує більш точні та зрозумілі результати.

Результати підтверджують, що використання WordPiece [27] для токенізації тексту дає кращу якість автоматизованого реферування та абстрактного генерування тексту для медичних результатів.

Номер тесту	**Rouge-l**	**Rouge-1**
0	0.0822	0.0881
1	0.0754	0.0838
2	0.0743	0.0798
3	0.0731	0.0829
4	0.0727	0.0826
5	0.0705	0.0742
6	0.0694	0.0782
7	0.0693	0.0746
8	0.0691	0.0779
9	0.0624	0.0651

Таблиця 3.3 – Результати для SentencePiece

Порівняємо, в середньому, на скільки відсотків метод WordPiece [27] краще за SentencePiece [2]. З цієї таблиці можна стверджувати, що метод WordPiece [27] є оптимальний для задачі, з заданими параметрами, в порівнянні з методом SentencePiece [2]

Metric	**Value**
Rouge-l	21.0508 %
Rouge-1	24.2816 %

Таблиця 3.4 – Середнє покращення методу WordPiece, відносно SentencePiece, в %

Висновки до розділу 3

На підставі отриманих результатів можна зробити висновок, що метод WordPiece [27], в порівнянні з методом SentencePiece [2], продемонстрував кращі результати за обома метриками Rouge-l та Rouge-1, з таблиці (3.4). Це свідчить про те, що WordPiece [27] має більшу точність при генерації текстів, медичного характеру. Враховуючи це, можна рекомендувати використання методу WordPiece [27] для подальших досліджень та застосувань в аналогічних архітектурах, що використовуються для вирішення проблеми NLP, для задачі автоматизованого реферування тексту.

ВИСНОВКИ

У ході даної роботи, було проведено порівняльний аналіз методів токенізації тексту WordPiece [27] та SentencePiece [2] для задачі автоматизованого реферування тексту, на сімействі моделей Transformer, а саме архітектурі seq2seq. Проведено огляд підходу до вирішення задач, за допомогою машинного навчання, а саме глибинного машинного навчання. Розглянуто два підходи до вирішення задачі автоматизованого реферування тексту: екстрактивний та абстрактивний, було надано короткий підсумок екстрактивних методів, включно з TextRank [8], як приклад сімейства даних методів. Також розглянуті наявні моделі абстрактивного підходу до створення автоматизованого реферування, такі як: BERT (Bidirectional Encoder Representations from Transformers, двоспрямовані кодувальні представлення з трансформерів) [14], ELECTRA [15], RoBERTa [16]. Також оглянуто попередні роботи, за тематикою порівняння методів токенізації, що було використано для розуміння підходів до дослідження, та актуальності даної теми.

В даному дослідженні розглянуто основні підходи в архітектурі Transformer, такі як, голови самоуваги та енкодер-декодери, що дозволяють моделі розуміти залежності між словами та використовувати цю інформацію для задачі генерування тексту, включно з задачою автоматизованого реферування тексту.

Також в даному дослідженні зображено кожен крок попередньої обробки даних, та вказано інструментарій для проєктування використаної в дослідженні архітектури моделі, з заданими версіями програмного забезпечення, задля розуміння умов під час проведення дослідження.

За результатами дослідження, виявлено, що метод WordPiece [27] показує кращі результати порівняно з методом SentencePiece [2] за сімейством метрик Rouge [24]. Зокрема, використання методу WordPiece [27] має покращення на 21.05% за метрикою Rouge-l [24], та покращення

на 24.28% за метрикою Rouge-1 [24], що свідчить про його вищу якість у підсумовуванні тексту на прикладі результатів та висновків медичних статей. Також варто вказати, що під час дослідження було використано невелика кількість даних, що залишає простір для дослідників, для розгляду даної тематики на більших даних, з більшими обчислювальними потужностями. Дані результати дають підставу для розгляду методу WordPiece [27] як потенційного варіанту для вдосконалення автоматичного підсумовування тексту в медичній та інших галузях.

ПЕРЕЛІК ПОСИЛАНЬ

1. S. Choo and W. Kim, "A study on the evaluation of tokenizer performance in natural language processing," *Applied Artificial Intelligence*, vol. 37, no. 1, p. 2175112, 2023.

2. T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," 2018.

3. Y. J. Lee, "eKoNLPy: A Korean NLP Python Library for Economic Analysis," 2018. https://github.com/entelecheia/eKoNLPy.

4. Y. He and M. Kayaalp, "A comparison of 13 tokenizers on medline," 12 2006.

5. G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization.," *J. Artif. Intell. Res. (JAIR)*, vol. 22, pp. 457–479, 2004.

6. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

7. D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

8. R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, (Barcelona, Spain), pp. 404–411, Association for Computational Linguistics, July 2004.

9. A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," 2017.

10. M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019.

11. Y. Liu and M. Lapata, "Text summarization with pretrained encoders," 2019.

12. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

13. Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," *CoRR*, vol. abs/1909.11942, 2019.

14. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

15. K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," 2020.

16. Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

17. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

18. N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "Ctrl: A conditional transformer language model for controllable generation," 2019.

19. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2018.

20. Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context,"

in *ACL (1)* (A. Korhonen, D. R. Traum, and L. Màrquez, eds.), pp. 2978–2988, Association for Computational Linguistics, 2019.

21. Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer, "Multilingual denoising pre-training for neural machine translation," 2020.

22. M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. Fikri Aji, N. Bogoychev, A. F. T. Martins, and A. Birch, "Marian: Fast neural machine translation in C++," in *Proceedings of ACL 2018, System Demonstrations*, (Melbourne, Australia), pp. 116–121, Association for Computational Linguistics, July 2018.

23. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

24. C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, (Barcelona, Spain), pp. 74–81, Association for Computational Linguistics, July 2004.

25. R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 1715–1725, Association for Computational Linguistics, Aug. 2016.

26. P. Gage, "A new algorithm for data compression," *C Users J.*, vol. 12, p. 23–38, feb 1994.

27. M. Schuster and K. Nakajima, "Japanese and korean voice search.," in *ICASSP*, pp. 5149–5152, IEEE, 2012.

28. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version

of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019.

29. T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Melbourne, Australia), pp. 66–75, Association for Computational Linguistics, July 2018.

30. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, (Philadelphia, Pennsylvania, USA), pp. 311–318, Association for Computational Linguistics, July 2002.

31. G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

32. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

33. F. Chollet *et al.*, "Keras." https://keras.io, 2015.

34. W. Matthew, Q. Chen, B. Jonathan, C. François, *et al.*, "Kerasnlp." https://github.com/keras-team/keras-nlp, 2022.

35. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay,

"Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

36. S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* "O'Reilly Media, Inc. 2009.

37. J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

38. M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," in *Proceedings of the First Workshop on Neural Machine Translation*, Association for Computational Linguistics, 2017.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 Модель seq2seq, на мові Python

```
class MyHyperModel(HyperModel):

    def build(self, hp):
        sequence_length = 64
        dropout_rate = 0.1
        num_heads = 6
        intermediate_dim = 64
        embed_dim = 64
        number_of_layers = 3

        encoder_inputs = keras.Input(
        shape=(None,),
        dtype="int64",
        name="encoder_inputs")

        x = keras_nlp.layers.TokenAndPositionEmbedding
            (
              vocabulary_size=ART_VOCAB_SIZE,
              sequence_length=sequence_length,
              embedding_dim=embed_dim,
              mask_zero=True,
        )(encoder_inputs)


        for _ in range(number_of_layers):
            encoder_outputs = keras_nlp.layers.
                TransformerEncoder(
                  intermediate_dim=intermediate_dim,
```

```
27                  num_heads=num_heads,
28                  )(inputs=x)
29
30          x = keras.layers.BatchNormalization()(x)
31          encoder = keras.Model(encoder_inputs,
                encoder_outputs)
32
33
34          # Decoder
35          decoder_inputs = keras.Input(
36              shape=(None,),
37              dtype="int64",
38              name="decoder_inputs"
39          )
40
41          encoded_seq_inputs = keras.Input(
42              shape=(None, embed_dim),
43              name="decoder_state_inputs"
44          )
45          x = keras_nlp.layers.TokenAndPositionEmbedding
                (
46              vocabulary_size=HIGH_VOCAB_SIZE,
47              sequence_length=sequence_length,
48              embedding_dim=embed_dim,
49              mask_zero=True,
50
51          )(decoder_inputs)
52          for _ in range(number_of_layers):
53              x = keras_nlp.layers.TransformerDecoder(
54                  intermediate_dim=intermediate_dim,
55                  num_heads=num_heads,
```

```
56                  # dropout=dropout_rate
57              )(decoder_sequence=x, encoder_sequence=
                    encoded_seq_inputs)
58          x = keras.layers.Dropout(dropout_rate)(x)
59          x = keras.layers.BatchNormalization()(x)
60          decoder_outputs = keras.layers.Dense(
61              HIGH_VOCAB_SIZE,
62              activation="softmax",
63              )(x)
64          decoder = keras.Model(
65              [
66                  decoder_inputs,
67                  encoded_seq_inputs,
68              ],
69              decoder_outputs,
70          )
71          decoder_outputs = decoder([decoder_inputs,
                encoder_outputs])
72
73          transformer = keras.Model(
74              [encoder_inputs, decoder_inputs],
75              decoder_outputs,
76              name="transformer",
77          )
78          optimizer = tf.keras.optimizers.Adafactor(
79              learning_rate=0.01,
80              beta_2_decay=-1,
81              epsilon_1=1e-30,
82              epsilon_2=0.001,
83              clip_threshold=1.0,
84              relative_step=True,
```

```
85                 weight_decay=None,
86                 # clipnorm=1,
87                 clipvalue=1,
88                 global_clipnorm=1,
89                 use_ema=False,
90                 ema_momentum=0.99,
91                 ema_overwrite_frequency=None,
92                 jit_compile=True,
93                 name="Adafactor",
94             )
95             transformer.compile(
96                 loss="sparse_categorical_crossentropy",
97                 optimizer=optimizer,
98                 metrics=[keras_nlp.metrics.Perplexity(name
                       ="perplexity")]
99             )
100            return transformer
101
102        def fit(self, hp, model, train_pairs,
               validation_pairs, callbacks=None, **kwargs):
103            batch_size = 32
104            train_dataset = make_dataset(train_pairs,
                   preprocess_batch_wordpiece, batch_size)
105
106            val_dataset = make_dataset(validation_pairs,
                   preprocess_batch_wordpiece, batch_size)
107
108            my_callback = keras.callbacks.EarlyStopping(
                   monitor='val_perplexity', patience=4, mode='
                   auto', min_delta=1)
109            with tf.device('/GPU:0'):
```

```
110             res = model.fit(
111                 train_dataset,
112                 shuffle=True,
113                 validation_data=val_dataset,
114                 callbacks=[my_callback],
115                 **kwargs,
116             )
117         return res
```