Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СИСТЕМА АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ТА ПЕРЕКЛАДУ СУБТИТРІВ НА БАЗІ НЕЙРОМЕРЕЖЕВИХ МОДЕЛЕЙ

A SYSTEM FOR AUTOMATIC SUBTITLE GENERATION AND TRANSLATION BASED ON NEURAL NETWORK MODELS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Омельчук Дмитро Юрійович

(підпис)

Керівник:
к.т.н., доцент
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 04 » червня 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т. Терлецький
« 10 » 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Омельчуку Дмитру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Система автоматичної генерації та перекладу субтитрів на базі нейромережевих моделей*

Керівник роботи *к.т.н., доцент Бортник Катерина Яківна*

затверджені наказом закладу вищої освіти від «04» січня 2025 року № 11/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 10.06.2025р.

3. Вихідні дані до роботи *джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Теоретичні основи системи автоматичної генерації та перекладу субтитрів.
Огляд сучасних нейромережевих моделей для розпізнавання мовлення та машинного перекладу.
Вибір та обгрунтування нейромережевих моделей для системи.
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Схематичне зображенняя архітектур нейромереж
Інтерфейс розробленої програми
Скріншоти з програм MSI Afterbuner та ThrottleStot

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Теоретичні відомості автоматичного субтитрування відео	*Бортник К.Я., доцент*		
Огляд сучасних нейромережевих моделей та метрик оцінки	*Бортник К.Я., доцент*		
Розробка та аналіз продуктивності на різних комп'ютерних системах	*Бортник К.Я., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,ст.викладач*		

7. Дата видачі завдання 10.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми, аналіз предметної області та наявних рішень*	до 10.02.2025 р.	Виконано
2.	*Теоретичні основи системи автоматичної субтитрування відео та огляд нейромережевих моделей та метрик оцінки*	до 02.03.2025 р.	Виконано
3.	*Розробка та аналіз продуктивності на різних комп'ютерних системах*	до 02.04.2025 р.	Виконано
4.	*Висновки та пропозиції*	до 10.04.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 15.04.2025 р.	Виконано
6.	*Формування додатків*	до 02.05.2025 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 10.05.2025 р.	Виконано
8.	*Представлення остаточного варіанту кваліфікаційної роботи бакалавра керівникові*	до 15.05.2025 р.	Виконано
9.	*Нормоконтроль*	до 30.05.2025 р.	Виконано
10.	*Інструментальна перевірка на академічний плагіат*	до 03.06.2025 р	Виконано
11.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Омельчук Д.Ю.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Бортник К.Я.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Омельчук Д.Ю. Система автоматичної генерації та перекладу субтитрів на базі нейромережевих моделей. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатку.

Перший розділ присвячено огляду предметної області, тут обґрунтовано актуальність обраної теми, розглянуто алгоритм роботи нейромережевих моделей, датасети для тренування моделей розпізнавання мовлення та сам процес тренування моделі. Проаналізовано процес завантаження моделей на комп'ютерні системи та оптимізацію використання пам'яті та обчислень. Також було пояснено різницю між прямим проходом на графічному та центральному процесорах.

В другому розділі здійснено вибір та обґрунтування моделі розпізнавання мовлення та моделі для створення машинного перекладу. Розглянуто метрики оцінки точності машинного перекладу та протестовано моделі машинного перекладу за цими метриками, виконано порівняння результатів.

Третій розділ присвячено розробці системи автоматичного субтитрування, порівнянню роботи розробленої програми на різних конфігураціях комп'ютерів, також розглянуто можливості вдосконалення системи на прикладі андервольту центрального та графічного процесорів.

Ключові слова: великі мовні моделі, відеокарти, процесори, машинний переклад.

ANNOTATION

Omelchuk D. A system for automatic subtitle generation and translation based on neural network models. Manuscript.

Bachelor's Qualifying Work in the field of study «Computer Engineering», specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The qualifying work consists of an introduction, three chapters, conclusions, a list of references, and an appendix.

The first chapter is dedicated to an overview of the subject area, explaining the relevance of the chosen topic, reviewing the algorithm of neural network models and datasets used for training speech recognition models, and the model training process itself. The deployment of models on computer systems is analyzed alongside optimization techniques for memory usage and computational efficiency. Additionally, it explains the difference between a forward pass executed on a GPU versus a CPU.

The second chapter focuses on the selection and justification of a speech recognition model and a model for machine translation. The metrics for evaluating the accuracy of machine translation are also considered and tested machine translation models using these metrics, results of testing are compared.

The third chapter is dedicated to the development of an automated subtitle creation system, comparing the performance of the developed program on different computer system configurations, and discussing the possibilities for improving the system using undervolting of central and graphics processing units.

Keywords: large language models, graphics cards, processors, machine translation.

ЗМІСТ

ВСТУП

Актуальність теми. Глобалізація є позитивним явищем, але одним з її наслідків є те, що медіа може бути наявним лише однією мовою. Моделі штучного інтелекту можуть допомогти вирішити дві проблеми: створити субтитри до медіа в форматі відео і перекласти текст мовою читача. Але у моделей штучного інтелекту існує своя проблема – не кожен комп'ютер здатний працювати з ними через відсутність достатньої кількості відео- та оперативної пам'яті або відсутність можливості використовувати велику кількість електроенергії на переклад.

Метою роботи є створення програми субтитрування з функцією перекладу створених субтитрів за допомогою мовних моделей, її тестування та порівняння на різних конфігураціях та пошук способу оптимізації енергоспоживання графічного та центрального процесорів.

Об'єкт дослідження − моделі розпізнавання мовлення та перекладу.

Предмет досліджень – нейромережеві моделі: OpenAI Whisper та Facebook Seamless-M4T-v2-Large.

Завдання, які необхідно виконати:

− дослідити метрики оцінки точності машинного перекладу;

− розробити програму створення та перекладу субтитрів;

− перевірити швидкодію розробленої програми на різних конфігураціях комп'ютерних систем та оптимізувати роботу графічного та центрального процесора за допомогою андервольту;

− підібрати найточнішу модель розпізнавання мовлення.

Апробація роботи. Результати роботи доповідалися на Міжнародній науково-практичній конференції «Цифрова трансформація: виклики та стратегії», Луцьк, 25 лютого 2025 року.

Публікації. Опубліковано тези [1].

РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ АВТОМАТИЧНОГО СУБТИТРУВАННЯ ВІДЕО

1.1 Принципи навчання моделей розпізнавання мовлення

Навчання моделей розпізнавання мовлення є складним і багатоетапним процесом. Сучасні моделі розпізнавання мовлення (Speech-to-Text, STT) здебільшого навчаються з використанням методів глибокого навчання, що вимагає великих обсягів даних та значних обчислювальних ресурсів. Розглянемо докладніше основні підходи до навчання.

Одним з ключових методів є навчання з учителем (Supervised Learning). Цей підхід передбачає використання розміченого набору даних, де кожному аудіозапису відповідає його точна текстова транскрипція. Модель навчається на цих даних, зіставляючи вхідні акустичні ознаки з вихідними символами (фонемами, складами або цілими словами). В процесі навчання модель налаштовує свої внутрішні параметри (ваги нейронної мережі) таким чином, щоб мінімізувати різницю між своїми передбаченнями та істинними транскрипціями. Зазвичай для цього використовується функція втрат, яка кількісно оцінює помилку моделі, і алгоритм оптимізації (наприклад, стохастичний градієнтний спуск), який ітеративно оновлює ваги мережі [2].

Важливим аспектом навчання моделей, особливо тих, що базуються на рекурентних нейронних мережах (RNN), є використання алгоритму Connectionist Temporal Classification (CTC). Традиційні підходи до навчання вимагають точного вирівнювання між аудіосигналом і відповідною йому послідовністю символів у транскрипції. Це означає, що для кожного фрейму аудіо необхідно вказати, якому символу він відповідає. Створення такого вирівнювання вручну є дуже трудомістким процесом, особливо для великих обсягів даних. CTC вирішує цю проблему, вводячи додатковий символ «blank» (порожній символ), який позначає відсутність розпізнаного символу в даному фреймі. Модель, навчена з використанням CTC, видає послідовність символів та «blank», яка може містити

повторення символів. Наприклад, для слова «hello» вихід моделі може бути «hhe-ll-l-o», де «-» позначає «blank». Алгоритм CTC обчислює ймовірність всіх можливих вирівнювань, які відповідають даній транскрипції, і оптимізує модель таким чином, щоб максимізувати сумарну ймовірність правильних вирівнювань. Це дозволяє навчати модель без необхідності явного вирівнювання аудіо та тексту [3].

Останнім часом набувають популярності методи самонавчання (SelfSupervised Learning). Ці методи дозволяють навчати моделі на великих обсягах нерозмічених аудіоданих, що значно розширює можливості навчання, оскільки нерозмічених даних зазвичай набагато більше, ніж розмічених. Одним із прикладів такого підходу є Wav2Vec 2.0. В процесі самонавчання модель навчається передбачати приховані частини аудіосигналу. Наприклад, частина аудіофреймів може бути випадковим чином замаскована, і модель повинна передбачити їх вміст на основі контексту. Це змушує модель вивчати корисні представлення аудіосигналу, які потім можуть бути використані для вирішення конкретних завдань, таких як STT. Після попереднього навчання на нерозмічених даних модель може бути донавчена на невеликій кількості розмічених даних, що дозволяє досягти високої точності розпізнавання навіть при обмежених ресурсах. Інший приклад, це контрастне навчання (contrastive learning), де модель навчається розрізняти схожі та несхожі аудіосегменти [4].

Якість та обсяг даних, на яких навчається модель STT, мають вирішальне значення для її продуктивності. Для навчання сучасних моделей використовуються великі набори даних, що містять тисячі годин аудіозаписів та відповідні їм текстові транскрипції. Ці датасети можуть бути як загального призначення, так і спеціалізованими, наприклад, для розпізнавання мовлення в певній предметній області або в умовах сильного шуму.

Одним із найвідоміших і широко використовуваних датасетів є LibriSpeech. Він містить близько 1000 годин аудіозаписів англійською мовою, отриманих з аудіокниг, що читаються різними дикторами. LibriSpeech добре підходить для

навчання та оцінки моделей STT, оскільки він містить чисте мовлення з відносно невеликою кількістю шуму та варіацій акустичних умов.

Іншим важливим проєктом є Common Voice від Mozilla. Це ініціатива зі збору даних для різних мов світу, включаючи українську. Common Voice дозволяє волонтерам записувати свої голоси, читаючи запропоновані речення, а також перевіряти транскрипції, зроблені іншими учасниками. Цей проект спрямований на створення відкритих та доступних наборів даних для навчання моделей розпізнавання мовлення для різних мов, особливо тих, для яких існує дефіцит розмічених даних.

Ще одним прикладом є датасет TED-LIUM, який містить аудіозаписи виступів з конференцій TED. Ці виступи характеризуються різноманітністю тем, стилів мовлення та акцентів, що робить TED-LIUM корисним для навчання моделей, стійких до різних акустичних умов.

Для навчання моделей, призначених для розпізнавання телефонних розмов, часто використовується датасет Fisher English Training Speech. Він містить велику кількість записів телефонних розмов англійською мовою, що характеризуються наявністю шуму, спотворень сигналу та розмовної лексики.

Крім перерахованих вище, існує багато інших датасетів, як комерційних, так і загальнодоступних, які використовуються для навчання та оцінки моделей STT. Вибір конкретного датасету залежить від завдань, які повинна вирішувати модель, а також від доступності даних для цільової мови та акустичних умов.

1.2 Структура нейромережевої моделі: ваги, активації, графи обчислень

Сучасні нейромережеві моделі, особливо ті, що використовуються для розпізнавання мовлення та машинного перекладу, є складними структурами, які складаються з мільйонів чи навіть мільярдів параметрів (ваг). Ці параметри, як правило, представлені числами з плаваючою крапкою (float32 або float16), хоча існують і більш екзотичні формати.

Ваги (Weights) – це числові параметри, що визначають силу зв’язків між нейронами в мережі. Вони зберігаються у вигляді багатовимірних масивів – тензорів. Розмірність і кількість цих тензорів залежать від архітектури моделі. Розглянемо детальніше.

У повнозв’язному шарі ваги зберігаються у вигляді матриці W розміром m x n , де n – кількість вхідних нейронів, а m – кількість вихідних нейронів. Кожен елемент Wij цієї матриці представляє вагу зв’язку між i-м вихідним нейроном і j-м вхідним нейроном. Додатково може існувати вектор зміщення (bias) b розміром m, який додається до результату множення вхідного вектора на матрицю ваг.

У згортковому шарі ваги зберігаються у вигляді набору ядер згортки (фільтрів). Кожне ядро – це невелика матриця, яка застосовується до локальних областей вхідних даних. Наприклад, для 2D згортки ядро може мати розмір k x k x cin, де k – розмір ядра (наприклад, 3x3), а cin – кількість вхідних каналів (наприклад, 3 для RGB зображення). Кількість ядер визначає кількість вихідних каналів (cout). Таким чином, загальний тензор ваг для згорткового шару має розмір cout x k x k x cin. На рисунку 1.1 зображено порядок згорткових шарів [5].

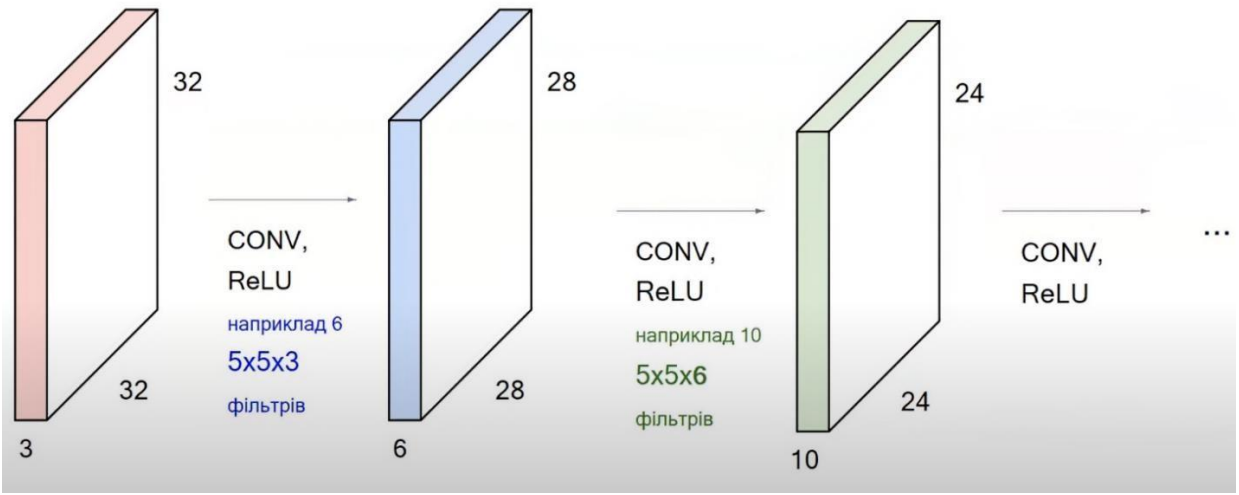

Рисунок 1.1 – Приклад архітектури згорткових шарів [5]

Ваги в рекурентних шарах мають більш складну структуру, тому що вони повинні враховувати зв’язки між нейронами в різні моменти часу. Наприклад, для простої рекурентної нейронної мережі використовуються три матриці ваг: Wxh – для зв’язку між вхідним вектором і прихованим станом, Whh – для зв’язку між прихованим станом на попередньому кроці і поточним прихованим станом і Why – для зв’язку між прихованим станом і вихідним вектором. Для LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit) кількість матриць ваг ще більша, оскільки вони мають додаткові механізми (ворота), що керують потоком інформації. На рисунку 1.2 зображено порядок рекурентних шарів [5].

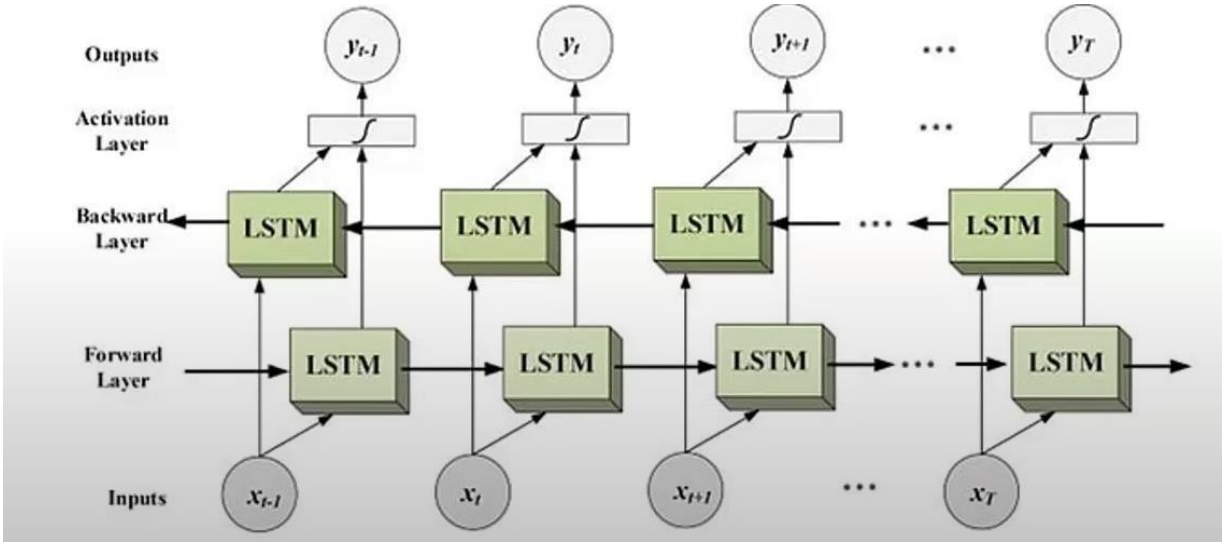

Рисунок 1.2 – Зображення роботи рекурентних шарів [5]

У моделях архітектури Transformer ключовим компонентом є механізм уваги. Тут ваги представлені матрицями, що використовуються для обчислення «ваг уваги» між різними елементами вхідної послідовності. Зазвичай використовуються три матриці: Wq для запитів, Wk для ключів і Wv для значень [6]. На рисунку 1.3 зображено представлення ваг в цій архітектурі [7].

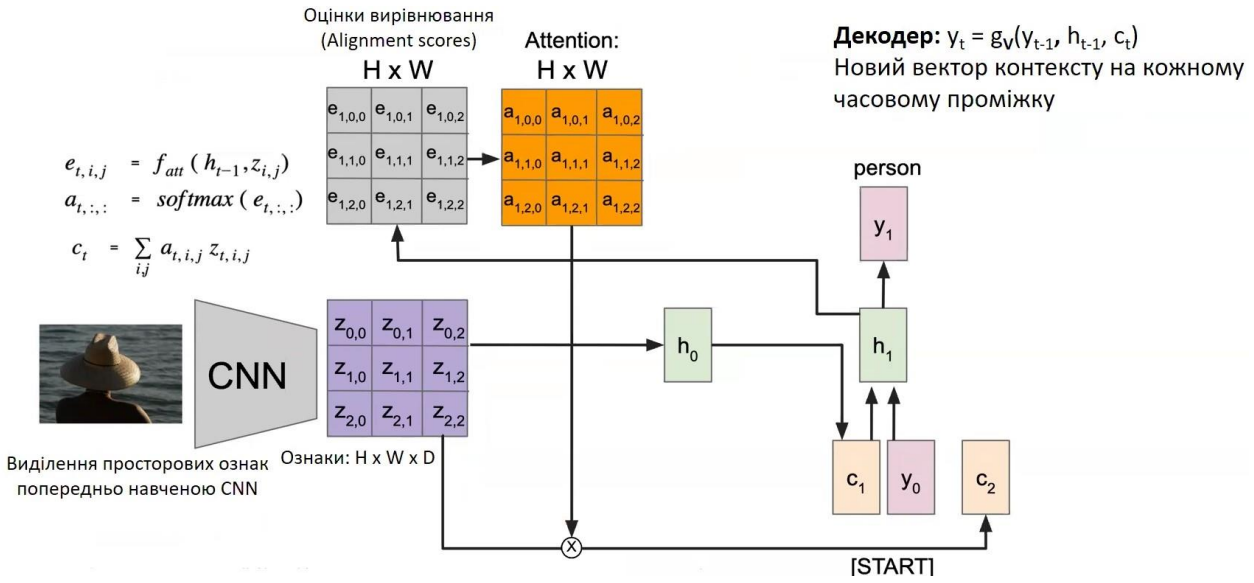

Рисунок 1.3 – Представлення ваг уваги в архітектурі Transformer [7]

Активації – це вихідні значення нейронів після застосування до них функції активації. Функція активації вносить нелінійність у модель, дозволяючи їй вивчати складні залежності. Приклади функцій активації:

– ReLU (Rectified Linear Unit): f(x) = max(0, x). Найпопулярніша функція активації завдяки своїй простоті та ефективності;

– Sigmoid: f(x) = 1 / (1 + exp(-x)). Стискає значення в діапазон [0, 1];

– Tanh (Hyperbolic Tangent): f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x)). Стискає значення в діапазон [-1, 1];

– Softmax: використовується на вихідному шарі для задач класифікації. Перетворює вектор значень на вектор ймовірностей, сума яких дорівнює 1.

Активації, як і ваги, зберігаються у вигляді тензорів. Їх розмірність залежить від розміру вхідних даних, розміру пакета (batch size) та архітектури моделі. Активації обчислюються під час прямого проходу (forward pass) і можуть зберігатися в пам’яті для використання під час зворотного проходу (backward pass).

Нейромережева модель може бути представлена у вигляді орієнтованого ациклічного графа (Directed Acyclic Graph, DAG), де вузли – це операції (математичні функції), а ребра – це тензори, що передаються між операціями.

Цей граф визначає послідовність обчислень, необхідних для отримання вихідних значень моделі. Сучасні фреймворки глибокого навчання (PyTorch, TensorFlow) автоматично будують граф обчислень на основі коду моделі. Граф може бути статичним (визначеним перед виконанням) або динамічним (змінюватися під час виконання) [8].

1.3 Завантаження моделі в пам’ять

Процес завантаження моделі включає декілька етапів:

– ваги моделі, як правило, зберігаються в серіалізованому форматі на диску (наприклад, HDF5, PyTorch .pth, ONNX, TensorFlow SavedModel). При завантаженні моделі дані зчитуються з файлу;

– дані, які були зчитані з файлу, десеріалізуються, тобто перетворюються з серіалізованого формату у внутрішнє представлення фреймворку глибокого навчання (наприклад, у вигляді об’єктів PyTorch або TensorFlow);

– десеріалізовані дані (тензори ваг) розміщуються в відео та оперативній пам’яті комп’ютера. Якщо модель велика, вона може не поміститися в цю всю пам’ять повністю, і тоді можуть використовуватися різні техніки, такі як memory mapping (відображення файлу в адресний простір процесу) або завантаження частини моделі на вимогу. Завантаження моделі на вимогу працює на великих мовних моделях виду Mixture-of-Experts, які складаються з менших моделей котрі треновані для певних задач. При обробці промпту використовуються моделі, створені для задачі, описаної в ньому;

– деякі фреймворки виконують додаткову ініціалізацію після завантаження ваг, наприклад, створюють додаткові структури даних, необхідні для виконання моделі.

Обсяг відео- та оперативної пам’яті, необхідний для моделі, приблизно дорівнює розміру файлу моделі. Варто враховувати додаткову пам’ять для

активацій та контексту, якщо розглядається використання великих мовних моделей [9].

1.4 Прямий прохід

Під час прямого проходу вхідні дані проходять через мережу, і обчислюються вихідні значення моделі. Цей процес включає виконання послідовності операцій, визначених графом обчислень.

1.4.1 Прямий прохід на центральному процесорі

CPU (центральний процесор) має відносно невелику кількість ядер (зазвичай від 4 до 64, в серверних процесорах може бути більше), кожне з яких є складним і здатним виконувати широкий спектр інструкцій. CPU має складну систему кешування (L1, L2, L3 кеш), яка прискорює доступ до даних.

При виконанні нейромережевих моделей на CPU, операції над тензорами (множення матриць, згортки тощо) виконуються, як правило, з використанням бібліотек лінійної алгебри, таких як BLAS (Basic Linear Algebra Subprograms) і LAPACK (Linear Algebra PACKage). Ці бібліотеки оптимізовані для виконання операцій лінійної алгебри на CPU і можуть використовувати різні техніки, такі як векторизація (SIMD, Single Instruction Multiple Data) і розпаралелювання (з використанням декількох ядер CPU).

Основним обмеженням CPU для нейромережевих обчислень є відносно невелика кількість ядер, що обмежує можливості з паралельного виконання обчислень.

1.4.2 Прямий прохід на графічному процесорі

GPU (графічний процесор) має тисячі ядер, які є більш простими, ніж ядра CPU, але здатні виконувати однотипні операції над великими масивами даних дуже ефективно. GPU має свою власну високошвидкісну пам’ять (VRAM, Video RAM), яка використовується для зберігання тензорів (ваг, активацій та вхідних даних).

При виконанні на GPU, тензори копіюються з оперативної пам’яті (RAM) в пам’ять GPU (VRAM). Обчислення виконуються паралельно на тисячах ядер GPU з використанням спеціалізованих бібліотек, таких як CUDA (для NVIDIA GPU) або ROCm (для AMD GPU). Ці бібліотеки надають API для програмування GPU і дозволяють розробникам писати спеціальні програми (ядра), які виконуються на GPU. cuDNN (CUDA Deep Neural Network library) – це бібліотека примітивів для глибоких нейронних мереж, яка надає високооптимізовані реалізації операцій, що часто використовуються в нейромережевих моделях (згортки, пулінг, функції активації тощо).

CUDA Streams дозволяють виконувати декілька операцій на GPU асинхронно, що може покращити використання ресурсів GPU і зменшити час виконання.

Обмеженням GPU є обсяг пам’яті, який зазвичай менший, ніж обсяг оперативної пам’яті. Якщо модель не поміщається у відеопам’ять повністю, необхідно використовувати різні техніки, такі як розбиття моделі на частини, що виконуються на різних GPU (model parallelism), або використання механізмів обміну даними між RAM і VRAM. Також вузьким місцем може бути швидкість передачі даних між CPU і GPU [10].

1.5 Зворотний прохід

Під час навчання нейромережевої моделі після прямого проходу, на якому обчислюються вихідні значення моделі для заданих вхідних даних, виконується зворотний прохід. Метою зворотного проходу є обчислення градієнтів функції втрат за параметрами (вагами) моделі. Ці градієнти показують, як потрібно змінити кожен параметр моделі, щоб зменшити значення функції втрат, тобто покращити точність моделі.

Зворотний прохід базується на принципі зворотного поширення помилки (backpropagation) та правилі ланцюга з диференціального числення. Правило ланцюга дозволяє обчислювати похідну складної функції (композиції функцій) як

добуток похідних окремих функцій. У контексті нейромережевої моделі функція втрат є складною функцією від вхідних даних, ваг моделі та активацій. На рисунку 1.4 показано схему того, як саме проходить зворотнє поширення по часу [11].

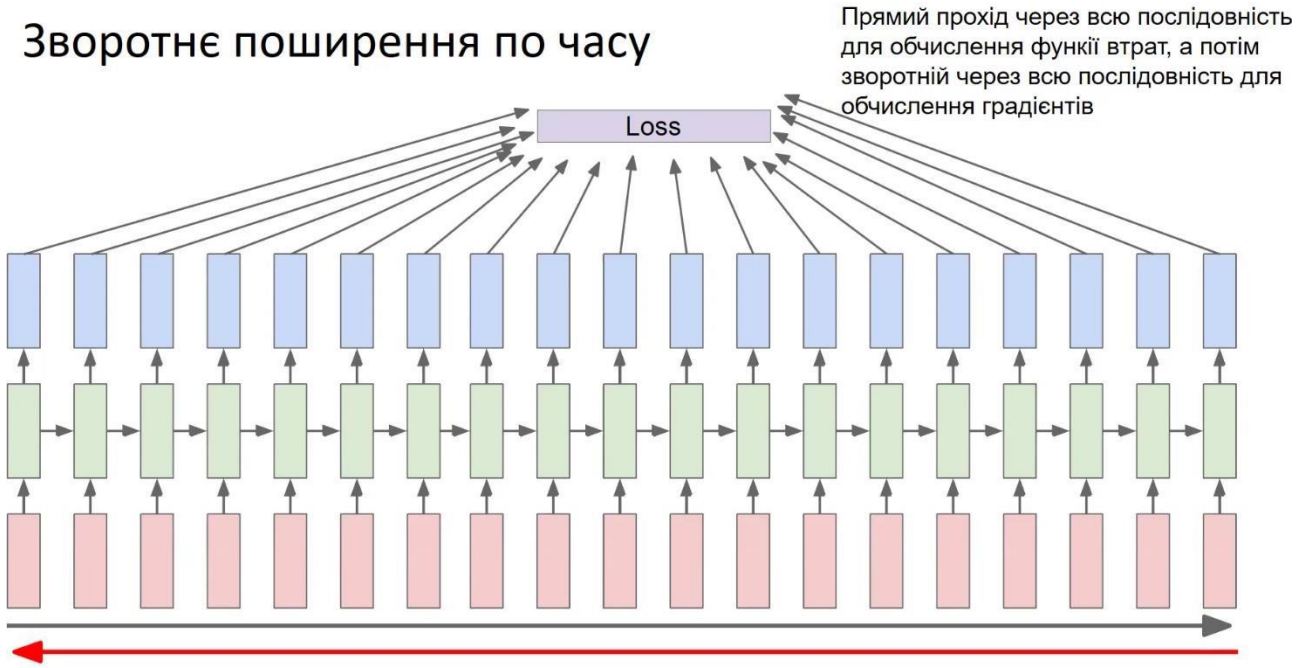

Рисунок 1.4 – Приклад зворотнього поширення [11]

Процес обчислення градієнтів виглядає наступним чином: спочатку обчислюється значення функції втрат, яка кількісно оцінює різницю між передбаченнями моделі та істинними значеннями (мітками). Вибір функції втрат залежить від типу задачі (класифікація, регресія тощо). Наприклад, для багатокласової класифікації часто використовується крос-ентропія (cross-entropy loss), а для регресії – середньоквадратична помилка (mean squared error). Обчислюється похідна функції втрат за вихідними активаціями останнього шару моделі. Ця похідна показує, як зміниться значення функції втрат при зміні вихідних активацій. Обчислені градієнти поширюються назад по шарах моделі, від вихідного шару до вхідного. Для кожного шару обчислюються:

− градієнт функції втрат за вихідними активаціями даного шару (показує, як зміниться значення функції втрат при зміні вихідних активацій даного шару);

− градієнт функції втрат за вхідними активаціями даного шару (показує, як зміниться значення функції втрат при зміні вхідних активацій даного шару. Він використовується для подальшого поширення градієнта на попередній шар);

− градієнт функції втрат за вагами даного шару (показує, як зміниться значення функції втрат при зміні ваг даного шару. Він використовується для оновлення ваг).

При обчисленні градієнтів для кожного шару використовується правило ланцюга. Наприклад, для повнозв'язного шару градієнт функції втрат за вагами обчислюється як добуток градієнта функції втрат за вихідними активаціями даного шару на транспонований вектор вхідних активацій. Градієнт за вхідними активаціями обчислюється як добуток градієнта функції втрат за вихідними активаціями на транспоновану матрицю ваг.

Для обчислення градієнтів необхідно також обчислювати похідні функцій активації. Наприклад, похідна ReLU (f(x) = max(0, x)) дорівнює 1 для x > 0 і 0 для x <= 0. Похідна sigmoid (f(x) = 1 / (1 + exp(-x))) дорівнює f(x) * (1 - f(x)).

Після обчислення градієнтів функції втрат за вагами всіх шарів, ваги оновлюються з використанням алгоритму оптимізації (наприклад, стохастичного градієнтного спуску (SGD), Adam, RMSprop). Алгоритм оптимізації визначає, як саме використовувати градієнти для оновлення ваг. Наприклад, у найпростішому варіанті SGD ваги оновлюються за формулою (1.1):

$$w=w-\eta*\nabla Q(w), \qquad (1.1)$$

де w – вага, η – швидкість навчання (learning rate), ∇Q(w) – градієнт функції втрат за вагою w.

Обчислення градієнтів під час зворотного проходу також включає операції над тензорами (множення матриць, обчислення похідних), аналогічні тим, що виконуються під час прямого проходу. Тому зворотний прохід також може бути ефективно прискорений на GPU. Сучасні фреймворки глибокого навчання (PyTorch, TensorFlow) автоматично обчислюють градієнти, використовуючи

механізм автоматичного диференціювання (automatic differentiation). Розробнику не потрібно вручну виводити формули для градієнтів, достатньо визначити граф обчислень (тобто послідовність операцій, що виконуються під час прямого проходу), а фреймворк автоматично побудує граф для обчислення градієнтів і виконає зворотний прохід [12].

1.6 Оптимізація використання пам’яті та обчислень

Для підвищення ефективності використання апаратних ресурсів, як CPU, так і GPU, при роботі з нейромережевими моделями застосовується ряд технік оптимізації. Ці техніки спрямовані на зменшення обсягу пам’яті, необхідної для зберігання моделі та проміжних результатів (активацій), а також на прискорення обчислень.

Одним із підходів є використання чисел з плаваючою крапкою половинної точності (FP16, float16) замість стандартної точності (FP32, float32). Це дозволяє вдвічі зменшити обсяг пам’яті, необхідної для зберігання ваг та активацій моделі. Крім того, сучасні графічні процесори (GPU) часто мають спеціалізовані апаратні блоки, такі як тензорні ядра в архітектурі NVIDIA, які значно прискорюють виконання операцій над даними у форматі FP16. Однак, використання FP16 може призвести до зниження точності обчислень, особливо під час обчислення градієнтів у процесі навчання. Тому часто застосовується змішана точність (mixed precision training), коли більшість операцій виконується з використанням FP16 для швидкості, а критично важливі для точності операції, наприклад, обчислення функції втрат та оновлення ваг, виконуються з використанням FP32.

Іншим підходом є квантування (quantization), яке передбачає перехід від чисел з плаваючою комою до цілих чисел, наприклад, 8-бітних (INT8) або навіть 4-бітних (INT4). Квантування дозволяє ще більше зменшити обсяг необхідної пам’яті (до чотирьох разів для INT8 порівняно з FP32) і може значно прискорити обчислення, особливо на спеціалізованих пристроях, таких як TPU (Tensor

Processing Unit) від Google, або на CPU з підтримкою відповідних інструкцій (наприклад, AVX-512) [13]. Проте, квантування, як правило, призводить до більш значної втрати точності, ніж використання FP16. Тому застосовуються різні методи квантування, такі як статичне квантування (post-training quantization), коли модель квантується після навчання, або динамічне квантування (quantization-aware training), коли модель навчається з урахуванням майбутнього квантування [14].

Ще одним методом оптимізації є прунінг (pruning), який полягає у видаленні з моделі найменш значущих зв'язків (ваг), значення яких близькі до нуля. Це дозволяє зменшити розмір моделі, скоротити обсяг обчислень і, як наслідок, зменшити обсяг необхідної пам'яті, а також прискорити процес виконання моделі.

Існують різні стратегії прунінгу: видалення окремих ваг, видалення цілих нейронів або фільтрів (для згорткових мереж), а також структурований прунінг, коли видаляються групи ваг. Після прунінгу модель зазвичай донавчають (finetuning), щоб відновити або навіть покращити її точність [15].

Дистиляція знань (knowledge distillation) – це техніка, яка дозволяє передати знання від великої, складної моделі (вчителя) до меншої, компактнішої моделі (учня). Замість того, щоб навчати учня безпосередньо на вихідних даних, його навчають на «м'яких мітках» (soft targets), які представляють собою розподіл ймовірностей, передбачених вчителем. Це дозволяє учню отримати більше інформації, ніж при навчанні на звичайних «жорстких мітках» (hard targets), і досягти продуктивності, близької до продуктивності вчителя, при значно меншому розмірі [16].

Пакетна обробка (batch processing) є стандартною практикою при навчанні та використанні нейромережевих моделей. Замість обробки даних по одному прикладу дані обробляються групами (пакетами, батчами). Це дозволяє більш ефективно використовувати ресурси графічного процесора, тому що операції над великими матрицями виконуються значно швидше, ніж над окремими

векторами. Розмір пакета (batch size) є важливим гіперпараметром, який впливає як на швидкість навчання, так і на узагальнюючу здатність моделі.

Асинхронне копіювання даних між центральним і графічним процесором дозволяє приховати затримки, пов’язані з передачею даних. Поки графічний процесором виконує обчислення над поточним пакетом даних, центральний може готувати наступний пакет і передавати його у відеопам’ять – пам’ять графічного процесора. Це дозволяє більш ефективно використовувати обчислювальні ресурси і зменшити загальний час виконання.

Градієнтне акумулювання (gradient accumulation) використовується, коли розмір пакета обмежений обсягом доступної відеопам’яті. Замість того, щоб оновлювати ваги моделі після кожного пакета, градієнти обчислюються для декількох пакетів і накопичуються. Оновлення ваг відбувається лише після накопичення градієнтів за певну кількість кроків. Це дозволяє ефективно використовувати більші розміри пакетів, навіть якщо вони не поміщаються у відеопам’ять цілком [17].

РОЗДІЛ 2 ОГЛЯД НЕЙРОМЕРЕЖЕВИХ МОДЕЛЕЙ ТА МЕТРИК ОЦІНКИ

2.1 Поняття механізму уваги

Механізм уваги (attention mechanism) є ключовим компонентом сучасних нейромережевих моделей машинного перекладу. Він дозволяє моделі динамічно фокусуватися на найбільш релевантних частинах вхідного речення при генерації кожного слова перекладу.

Наприклад, перекладається речення «The cat sat on the mat» з англійської на українську. При генерації слова «кицька», декодувальнику корисно звернути особливу увагу на слово «cat» у вхідному реченні. Механізм уваги обчислює «ваги уваги» для кожного слова вхідного речення, які показують, наскільки це слово важливе для генерації поточного слова перекладу. Ці ваги обчислюються на основі схожості між поточним станом декодувальника і станами кодувальника, що відповідають різним словам вхідного речення. Потім ваги уваги використовуються для обчислення зваженої суми станів кодувальника, яка і є контекстним вектором, що використовується декодувальником для генерації наступного слова.

Багатостороння увага (multi-head attention) розширює цей принцип, дозволяючи моделі одночасно обчислювати кілька різних наборів ваг уваги. Це дозволяє моделі враховувати різні аспекти взаємозв’язку між словами, наприклад, граматичні зв’язки, семантичну близькість тощо[18].

2.2 Датасети для навчання моделей машинного перекладу

Успіх моделей машинного перекладу, як і будь-яких інших моделей, що базуються на глибокому навчанні, значною мірою залежить від якості та кількості доступних даних. Для навчання цих моделей необхідні великі обсяги паралельних корпусів – колекцій текстів, де кожен текст однією мовою має відповідний йому переклад іншою мовою. Такі паралельні дані дозволяють

моделі вивчити відповідності між словами, фразами та синтаксичними структурами різних мов.

Одним з найважливіших ресурсів, що надає доступ до різноманітних паралельних корпусів, є проєкт OPUS – відкрита колекція, яка постійно розширюється і включає паралельні тексти з найрізноманітніших джерел. Сюди входять субтитри до фільмів та телепередач (наприклад, OpenSubtitles), офіційні документи Європейського Союзу (Europarl, DGT-TM), релігійні тексти (наприклад, Біблія), матеріали веб-сайтів та багато іншого. Завдяки широкому спектру джерел, OPUS надає дані для багатьох мовних пар і різних стилів тексту [19].

Щорічні змагання з машинного перекладу, що проводяться в рамках Workshop on Machine Translation (WMT), також відіграють важливу роль у розвитку галузі. WMT надає стандартизовані набори даних для навчання та тестування, що дозволяє дослідникам порівнювати ефективність різних моделей та підходів. Ці набори даних, як правило, включають новинні тексти, а також інші жанри, і охоплюють різні мовні пари [20].

United Nations Parallel Corpus є ще одним цінним ресурсом, що містить офіційні документи Організації Об’єднаних Націй, перекладені шістьма офіційними мовами (англійською, арабською, іспанською, китайською, російською та французькою). Цей корпус відрізняється високою якістю перекладу та охоплює широкий спектр тем, пов’язаних з міжнародними відносинами, правом, економікою тощо.

Важливими ресурсами, особливо для оцінки моделей на низькоресурсних мовах (тобто мовах, для яких доступно відносно мало паралельних даних), є датасет flores-101 та його розширена версія flores-200. Flores (Facebook Low Resource) – це набір даних, спеціально розроблений для оцінки якості машинного перекладу для мов з обмеженими ресурсами. Він містить речення, витягнуті з Вікіпедії, перекладені на понад 100 мов у датасеті flores-101 та 200 мов у випадку flores-200, включаючи багато мов, які не представлені в інших великих паралельних корпусах. Flores-101 і Flores-200 не призначені для навчання

моделей з нуля через їхній відносно невеликий розмір, але ідеально підходять для оцінки та порівняння моделей, особливо тих, що попередньо навчені на великих багатомовних наборах даних [21].

Крім перелічених, існує безліч інших паралельних корпусів, як загальнодоступних, так і комерційних, які використовуються для навчання та оцінки моделей машинного перекладу. Вибір конкретного набору даних або їх комбінації залежить від конкретних цілей, цільових мов та доступних ресурсів.

2.3 Архітектури нейронних мереж для розпізнавання мовлення

Рекурентні нейронні мережі (RNN) були одними з перших архітектур, які базуються на глибоких нейронних мережах, успішно застосованих для STT. Вони обробляють вхідні дані послідовно, враховуючи попередній контекст при обробці кожного елемента послідовності (в даному випадку – фреймів аудіосигналу). Це дозволяє RNN моделювати часові залежності в мовленні. Однак, стандартні RNN мають проблеми з навчанням на довгих послідовностях через затухання/вибух градієнтів [22].

Довга короткочасна пам’ять (LSTM) – це спеціальний тип RNN, розроблений для вирішення проблеми затухання/вибуху градієнтів. LSTM використовують спеціальні блоки пам’яті з «воротами» (gates), які контролюють потік інформації, дозволяючи мережі зберігати інформацію на довгих проміжках часу. LSTM значно покращили продуктивність STT систем порівняно зі стандартними RNN [23].

Двонаправлені RNN (BRNN) та LSTM (BLSTM) обробляють вхідну послідовність у двох напрямках – від початку до кінця та в зворотньому напрямку. Це дозволяє мережі враховувати як минулий, так і майбутній контекст при розпізнаванні кожного фрейму. BRNN/BLSTM зазвичай забезпечують вищу точність, ніж однонаправлені RNN/LSTM. На рисунку 2.1 зображено різницю між RNN та LSTM [11].

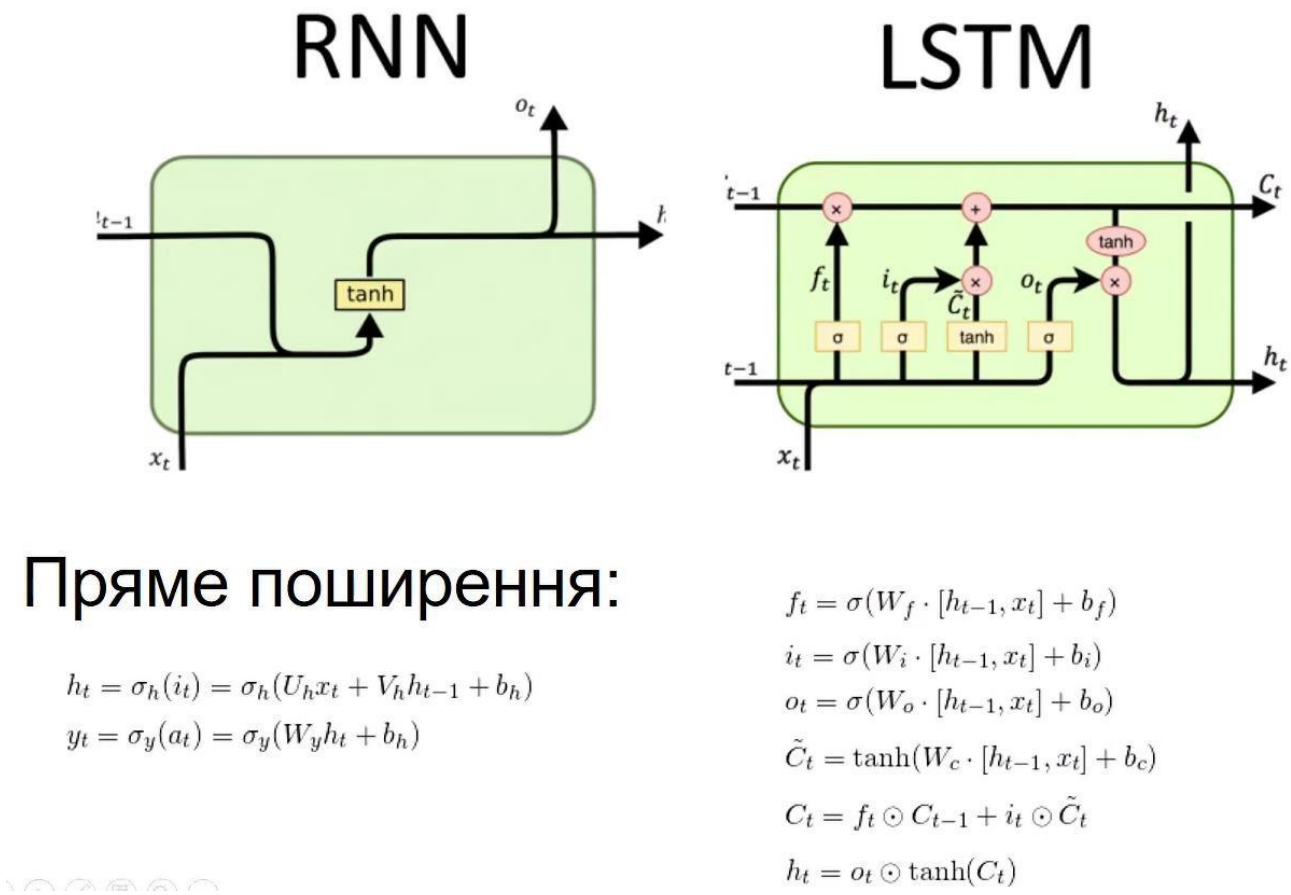

Рисунок 2.1 – Різниця між RNN та LSTM [11]

Архітектура Transformer, спочатку розроблена для машинного перекладу, також знайшла широке застосування в STT. Transformer використовує механізм уваги (attention mechanism), який дозволяє мережі фокусуватися на різних частинах вхідної послідовності при обробці кожного елемента. Це дозволяє Transformer ефективно обробляти довгі послідовності та моделювати складні залежності між різними частинами аудіосигналу. Transformer не використовує рекурентність, що дозволяє паралелізувати обчислення і прискорити навчання [24].

Conformer – це архітектура, яка поєднує в собі переваги Transformer та CNN (згорткових нейронних мереж). Conformer використовує згорткові модулі для вилучення локальних ознак з аудіосигналу, а модулі Transformer – для моделювання глобальних залежностей. Таке поєднання дозволяє Conformer досягати високої точності розпізнавання мовлення.

Wav2Vec 2.0 – це фреймворк для самонавчання (self-supervised learning) представлень мовлення. Wav2Vec 2.0 навчається на великих обсягах нерозмічених аудіоданих, маскуючи частини аудіосигналу та змушуючи модель передбачати приховані ділянки. Wav2Vec 2.0 демонструє високу ефективність, особливо в умовах обмежених розмічених даних [25].

Whisper – це розроблена OpenAI система автоматичного розпізнавання мовлення (ASR), навчена на 680 000 годин багатомовних і багатозадачних даних під наглядом, зібраних з Інтернету. Модель побудована на архітектурі Transformer (encoder-decoder). Whisper демонструє високу стійкість до акцентів, фонового шуму та технічної мови. Ця модель також дозволяє транскрибувати кількома мовами, а також перекладати ці мови на англійську [26].

Для реалізації системи автоматичної генерації та перекладу субтитрів було обрано модель OpenAI Whisper. Цей вибір зумовлений кількома причинами:

− Whisper демонструє одні з найкращих результатів розпізнавання мовлення серед доступних моделей, включаючи стійкість до шумів, акцентів та різних мов;

− Whisper підтримує розпізнавання та транскрипцію багатьох мов, що є ключовим для створення універсальної системи генерації субтитрів;

− використання архітектури Transformer забезпечує ефективну обробку довгих аудіопослідовностей та можливість паралелізації обчислень, що важливо для порівняльного аналізу продуктивності на CPU та GPU;

− існують готові реалізації моделі та бібліотеки для її використання на прикладі офіційної – whisper, так і створених користувачами, як whisper_timestamped, що спрощує інтеграцію в програмну систему.

Таким чином, Whisper є оптимальним вибором для розробки системи, яка має забезпечувати високу якість розпізнавання мовлення та ефективно використовувати обчислювальні ресурси без потреби в додатковому тренуванні моделі.

2.4 Огляд моделей машинного перекладу

На даний момент існує велика кількість моделей машинного перекладу, які відрізняються архітектурою, розміром, мовами, на яких вони навчені, та якістю перекладу. Тестуватись будуть наступні моделі:

− Helsinki-NLP/opus-mt-en-uk: модель з сімейства OPUS, навчена на переклад з англійської на українську мову. Вона базується на архітектурі Transformer і використовує дані проєкту OPUS [27];

− facebook/nllb-200-distilled-1.3B та facebook/nllb-200-distilled-600M – це дистильовані версії великої багатомовної моделі NLLB–200 від Facebook (Meta). Ці моделі підтримують переклад між 200 мовами [28];

− Qwen/Qwen2.5-7B-Instruct: велика мовна модель з відкритим кодом, яка може бути використана для різних завдань, включаючи машинний переклад [29];

− Meta-llama/Llama-3.1-8B-Instruct: модель серії Llama від Meta, оптимізована для виконання інструкцій та діалогового режиму, може бути використана і для перекладу [30];

− Microsoft/phi-4-mini–instruct: компактна модель від Microsoft, яка може бути використана для виконання інструкцій, в потенціалі може бути використана і для перекладу [31];

− lang-uk/Dragoman: модифікована модель mistralai/Mistral-7B-Instructv0.1, що була дотренована на спеціально створеному корпусі англійськоукраїнської мовної пари [32];

− facebook/seamless-m4t-v2-large: модель від Facebook (Meta), призначена для перекладу мовлення та тексту між багатьма мовами. Вона підтримує як Speech-to-Text, так і Text-to-Text переклад [33];

− mistralai/Mistral-7B-Instruct-v0.3: модель від Mistral AI, оптимізована для виконання інструкцій та діалогового режиму, також з потенціалом використання для перекладу [34].

2.5 Метрики оцінки якості машинного перекладу

Оцінка якості машинного перекладу є ключовим аспектом розробки та порівняння систем МТ. Оскільки автоматичний переклад є складним завданням, що включає лексичну точність, граматичну коректність, стилістичну відповідність та збереження змісту, не існує єдиної ідеальної метрики. Натомість використовується комплекс автоматичних метрик, які оцінюють різні аспекти якості, порівнюючи згенерований системою переклад (candidate translation) з еталонними перекладами (reference translations), створеними людьми.

2.5.1 BLEU

BLEU (Bilingual Evaluation Understudy) є однією з найпоширеніших і найбільш ранніх метрик для оцінки якості машинного перекладу. В основі BLEU лежить ідея порівняння n-грам (послідовностей з n слів) згенерованого перекладу з n-грамами в еталонних перекладах. Зазвичай використовуються n-грами довжиною від 1 до 4.

BLEU обчислює модифіковану точність n-грам. Для кожної n-грами в згенерованому перекладі визначається, чи зустрічається вона в якомусь з еталонних перекладів. Якщо так, то підраховується кількість збігів, але не більше, ніж максимальна кількість разів, коли ця n-грама зустрічається в одному з еталонних перекладів (це запобігає завищенню оцінки за рахунок повторення одних і тих самих n-грам). Сума цих кількостей ділиться на загальну кількість nграм у згенерованому перекладі.

Отримана точність для n-грам різної довжини потім усереднюється з використанням геометричного середнього. Крім того, BLEU включає «штраф за короткість» (brevity penalty), який знижує оцінку для занадто коротких перекладів. Це пов’язано з тим, що короткий переклад може мати високу точність n-грам, просто включаючи кілька правильних слів, але при цьому пропускаючи більшу частину змісту оригіналу.

BLEU є відносно простою в обчисленні та добре корелює з людською оцінкою, особливо при оцінці великих обсягів тексту (на рівні корпусу). Однак, BLEU має і недоліки. Вона погано враховує порядок слів усередині n-грам, не чутлива до семантичних та синтаксичних помилок, що не впливають на збіг nграм, і може давати некоректні оцінки для окремих речень. Також, BLEU не враховує синоніми [35].

2.5.2 TER

TER (Translation Edit Rate) – це метрика, яка вимірює кількість редагувань, необхідних для того, щоб перетворити згенерований переклад на один з еталонних перекладів. Під редагуваннями розуміються вставки, видалення, заміни та перестановки (зсуви) слів. Ця метрика обчислюється як відношення суми кількості редагувань до довжини еталонного перекладу (або середньої довжини, якщо еталонних перекладів декілька). Чим менше значення TER, тим краще якість перекладу, оскільки це означає, що потрібно менше редагувань для досягнення збігу з еталоном. TER, на відміну від BLEU, орієнтована на помилки, а не на точність. Вона краще враховує порядок слів, ніж BLEU. Однак, TER також має свої обмеження. Вона не враховує семантичну еквівалентність слів і фраз (наприклад, синоніми), і може бути надмірно чутливою до незначних відмінностей у формулюваннях, які не впливають на зміст [36].

2.5.3 cHRF++

cHRF++ (character n-gram F-score) – це метрика, яка, на відміну від BLEU та TER, працює на рівні символів, а не слів. Вона обчислює F-міру (середнє гармонійне точності та повноти) для символьних n-грам, зазвичай використовуючи n від 1 до 6.

Використання символьних n-грам робить cHRF++ більш стійкою до морфологічних варіацій слів. Наприклад, для різних форм одного і того ж слова (наприклад, «кіт», «кота», «коту») будуть спільні символьні n-грами, що дозволить метриці врахувати їх схожість. Крім того, cHRF++ використовує зважування n-грам, надаючи більшу вагу довшим n-грамам, що сприяє кращому врахуванню порядку символів.

Формула для обчислення cHRF++ складніша, ніж для BLEU або TER, і включає обчислення точності та повноти для символьних n-грам, а потім обчислення F-міри з використанням параметра β, який визначає відносну вагу точності та повноти. Зазвичай використовується cHRF++ з β=2, що надає більшу вагу повноті.

cHRF++ краще корелює з людською оцінкою для мов з багатою морфологією, ніж BLEU [37].

2.5.4 Інші метрики. Обгрунтування вибору метрик для порівняльного аналізу в роботі

Крім BLEU, TER та cHRF++, існує ряд інших метрик, які також використовуються для оцінки якості машинного перекладу.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) розроблена для подолання деяких недоліків BLEU. Вона враховує не тільки точні збіги слів, але й збіги на основі стемів (базових форм слів), синонімів (використовуючи WordNet) та парафраз. METEOR обчислює F-міру, як і cHRF++, але використовує іншу формулу, яка враховує точність і повноту з урахуванням штрафу за неправильний порядок слів [38].

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) спочатку була розроблена для оцінки якості автоматичного реферування (automatic summarization), але також знайшла застосування в машинному перекладі. ROUGE, як випливає з назви, орієнтована на повноту (recall), тобто на те, яка частина слів з еталонного перекладу була відтворена в згенерованому перекладі. Існує кілька варіантів ROUGE, наприклад, ROUGE-N (яка вимірює повноту nграм) та ROUGE-L (яка вимірює довжину найбільшої спільної підпослідовності)
[39].

2.5.5 Результати метрик та обгрунтування вибору моделі для перекладу

Для порівняльного аналізу продуктивності системи автоматичної генерації та перекладу субтитрів у даній роботі будуть використані метрики BLEU, TER та cHRF++. Вони були обрані через те, що вони є одними з найбільш поширених і

загальновизнаних метрик в області машинного перекладу та їх використання дозволить порівняти результати даної роботи з результатами інших досліджень. Також, ці три метрики оцінюють різні аспекти якості перекладу: BLEU – точність на рівні слів (n-грам), TER – кількість редагувань, необхідних для виправлення перекладу та cHRF++ – точність на рівні символів, стійкість до морфологічних варіацій. cHRF++ є особливо корисною метрикою для оцінки перекладу на українську мову, оскільки вона краще враховує багату морфологію української мови, ніж BLEU.

На рисунках 2.2-2.4 наведено результати метрик BLEU, cHRF++ та TER.

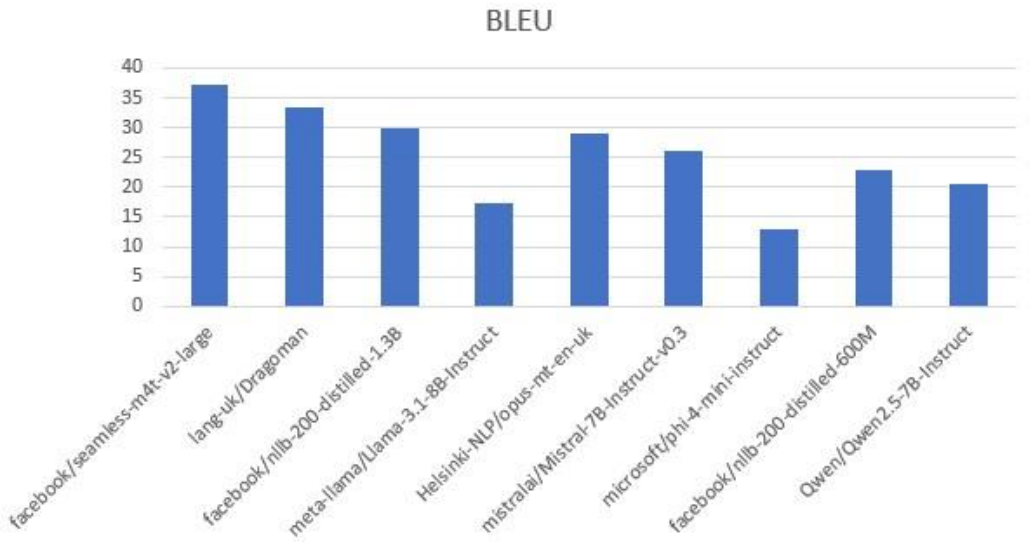

Рисунок 2.2 – Результати метрики BLEU

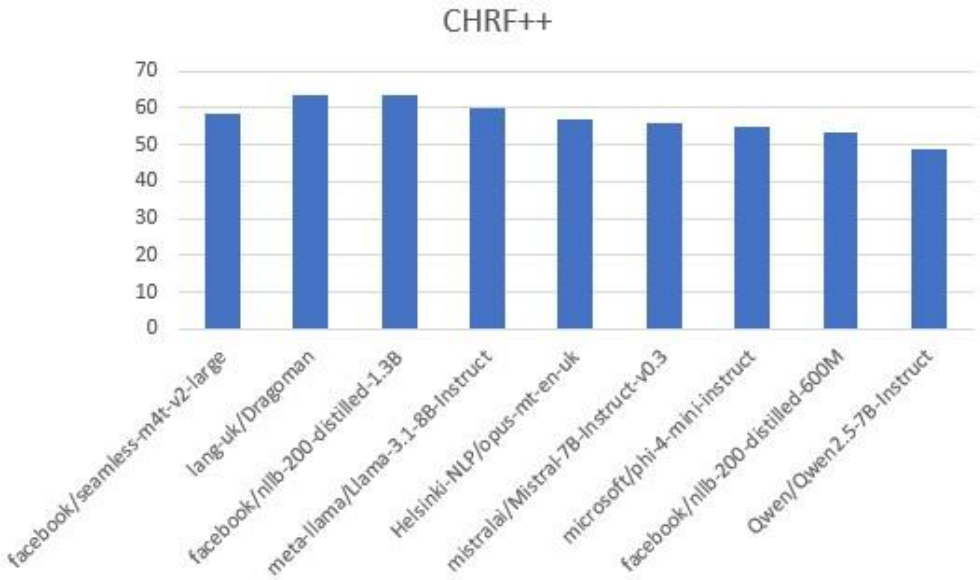

Рисунок 2.3 – результати метрики cHRF++

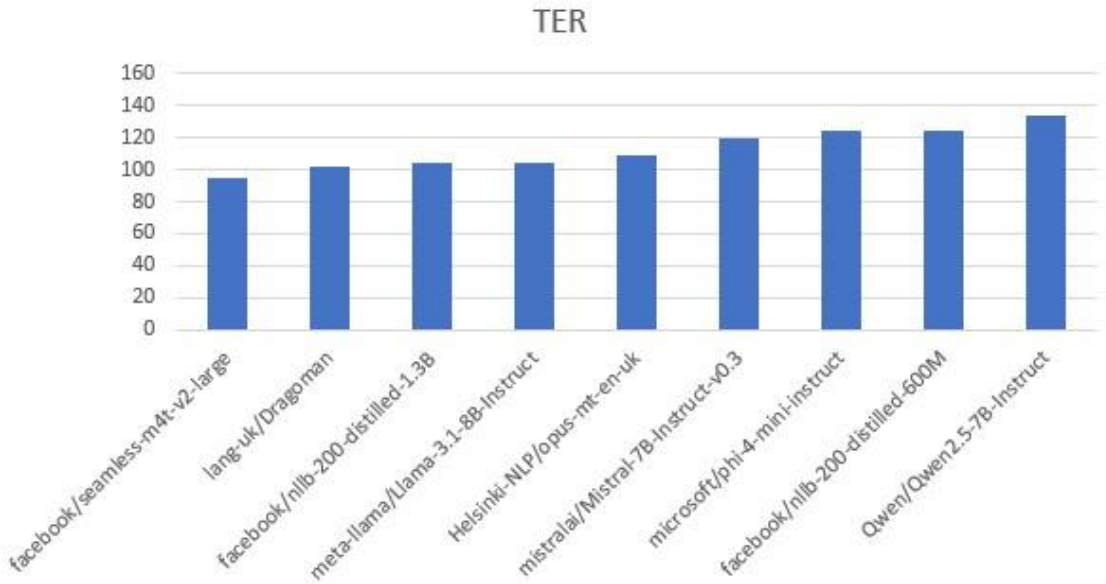

Рисунок 2.4 – Результати метрики TER

За результатами тестування обраних моделей було встановлено, що модель facebook/seamless-m4t-v2-large є найоптимальнішою через її відносно високу швидкість роботи та високі оцінки в обраних метриках BLEU, cHRF++ та TER.

РОЗДІЛ 3 РОЗРОБКА ТА АНАЛІЗ ПРОДУКТИВНОСТІ НА РІЗНИХ

КОМП'ЮТЕРНИХ СИСТЕМАХ

3.1 Розробка архітектури системи

Система розроблялася мовою Python через велику кількість бібліотек, які допоможуть реалізувати різноманітний функціонал: користувацький інтерфейс, використання різних моделей штучного інтелекту, а також програвання відеофайлів.

Система складатиметься з трьох файлів: main.py, video_player.py та subtitle_generator.

Головний файл main.py ініціалізує інтерфейс користувача, який створено в файлі video_player.py. Коли відео обробляється, тобто створюються субтитри, використовується код з файлу subtitle_generator.py, де знаходяться функції створення та перекладу субтитрів. Вміст файлу main.py зображено в лістингу 3.1.

Лістинг 3.1 – Файл main.py

```
import sys import
io
from PyQt5.QtWidgets import QApplication
from video_player import VideoPlayer def
main():
    app = QApplication(sys.argv)
player      =      VideoPlayer()
player.show()
sys.exit(app.exec_())         if
__name__ == '__main__':
  main()
```

кінець лістингу 3.1

3.2 Розробка інтерфейсу користувача

Вивід відео реалізовано за допомогою бібліотеки vlc, яка використовує популярний додаток для перегляду відео VLC Player. Розглядалася бібліотека cv2, але при тестуванні було виявлено проблему: відео не синхронізовано з аудіо, тому

ця бібліотека використовується для задач, пов’язаних з комп’ютерним зором (наприклад, для перевірки наявності об’єктів або для аналізу стану об’єктів, які знімає камера).

Інтерфейс створено за допомогою бібліотеки PyQt5. Добавлена кнопка вибору файлу на комп’ютері, де прийматимуться відео у найпоширенішому форматі MP4, а також кнопка програвання відео, яка ставить це відео на паузу. Код функції відкриття файлу зображено в лістингу 3.2.

Лістинг 3.2 – Функція відкриття файлу

```
def open_file(self):
     filename, _ = QFileDialog.getOpenFileName(self,
                                               "Open Video",
                                               "",
                                            "Video Files (*.mp4)")
     if filename:
          media = self.instance.media_new(filename)
          self.player.set_media(media)
          self.player.set_hwnd(int(self.video_label.winId()))
          self.play_button.setEnabled(True)
```

кінець лістингу 3.2

Добавлена полоска прогресу відео, яку можна перемотати, якщо потрібно повернутись до якогось моменту або пропустити.

Було добавлено галочку, яка позначатиме, чи потрібно перекладати створені субтитри. Якщо потрібно, то вводиться мова в поле введення. Для покращення досвіду користувача добавлено автозаповнення мови та прогресбар, який показує на якому етапі знаходиться обробка відео. Через специфіку моделі Whisper оновлення прогрес-бару в реальному часі не можливе, тому що модель обробляє весь звук з відео за один раз. Код ініціалізації моделі Whisper Turbo та обробки відео наведено в лістингу 3.3.

Лістинг 3.3 – Запуск моделі OpenAI Whisper

```
    model = whisper.load_model("turbo")   result =
model.transcribe(self.video_path,  word_timestamps=True)
```

кінець лістингу 3.3

В теорії, ділення на аудіофайли є можливим, але це знизить точність розпізнавання тексту, бо буде існувати ймовірність того, що фрагменти аудіо будуть містити не цілі слова. Саме тому, як тільки субтитри готові, прогрес-бар дійде до 100 %. Але переклад відбувається в циклі, який дозволяє добавляти оновлення прогрес-бару в реальному часі. Цикл перекладу субтитрів показано в лістингу 3.4.

Лістинг 3.4 – Цикл перекладу субтитрів та зміни значення прогрес-бару

```
for idx, (start, text) in enumerate(self.subtitles):
    if idx % 10 == 0:
    self.progress_update.emit(int((idx / len(self.subtitles)) *
100)) translation =
translator(text)[0]['translation_text']
translated_subtitles.append((start, translation))
```

кінець лістингу 3.4

3.3 Розробка функцій створення та перекладу субтитрів

Головна частина програми – створення субтитрів та переклад. Буде використовувалась бібліотека whisper_timestamped замість звичайної бібліотеки whisper через важливу перевагу – наявність параметру word_timestamps, яке при значенні true додає до кожного сказаного слова значення часу, коли саме це слово було сказано. Через це переклад по цілим реченням є можливим. Спеціально для об'єднання слів у речення було написано функцію, яку зображено в лістингу 3.5.

Лістинг 3.5 – Функція поділу слів на речення

```
def build_segments_from_words(words,
     max_pause=1,
     max_words=35):
     segments, buf = [], [] for
     idx, w in enumerate(words):
          buf.append(w)
          next_start = words[idx + 1]['start'] if idx + 1 < len(words)
     else None long_pause = next_start is not None and (next_start -
     w['end'] > max_pause)
          end_punc   = re.search(r'[.?!。？！]$', w['word'])
     too_long   = len(buf) >= max_words      last_word  = idx
     == len(words) - 1      if end_punc or long_pause or
     too_long or last_word: text = ''.join(b['word'] for b in
     buf).strip()
                segments.append((buf[0]['start'],
text))            buf = []      return segments
```

кінець лістингу 3.5

Вона закінчує речення, якщо між словами є пауза в 1 секунду, або якщо кількість слів в реченні перевищує 35, або якщо модель Whisper оприділила сама, що речення закінчилось, тобто поставила крапку, знак питання або знак оклику. Також були добавлені крапка, знак питання та знак оклику, які використовуються лише в японській мові, коли як решту мов оброблятимуть перші три символи.

Важливою функцією є очищення пам'яті після закінчення використання кожної моделі, тому що створення та переклад субтитрів не виконуються одночасно. Якщо моделі обробляються графічним процесором, то очиститься відеопам'ять, а коли центральним, то очищається оперативна пам'ять. Код, який відповідатиме за очищення пам'яті показано в лістингу 3.6.

Лістинг 3.6 – Очищення пам'яті

```
     del model if
     torch.cuda.is_available():
          torch.cuda.empty_cache()
     else:
          gc.collect()
```

кінець лістингу 3.6

Роботу моделі OpenAI Whisper показано в лістингу 3.3, ще було підключено модель facebook/seamless-m4t-v2-large. Для цього використано бібліотеку transformers та ініціалізовано модель, код для цього зображено в лістингу 3.7.

Лістинг 3.7 – Запуск моделі facebook/seamless-m4t-v2-large

```
from transformers import pipeline
translator = pipeline("translation", model="facebook/seamless-
                      m4t-v2-large", src_lang="auto",
                       tgt_lang=tgt_code,
           device=0)
```

кінець лістингу 3.7

По рядку src_lang="auto" зрозуміло, що модель сама здатна оприділити мову, з якої потрібно буде виконати переклад, що також покращить користувацький досвід та позбавить потреби вказувати мову у відео. Device=0 показує те, що модель буде завантажена на пристрій з ядрами CUDA. У випадку систем, на яких тестуватиметься програма – це графічні процесори Nvidia Geforce RTX 2070 та RTX 3090. При тестуванні на процесорах, достатньо змінити 0 на 1. Сама функція перекладу зображена в лістингу 3.8.

Лістинг 3.8 – Цикл перекладу субтитрів моделлю facebook/seamless-m4t-v2-large

```
translated_subtitles = [] for start,
     text in self.subtitles:
          translation = translator(text)[0]['translation_text']
          translated_subtitles.append((start, translation))
      self.subtitles = translated_subtitles
```

кінець лістингу 3.8

Додатково було добавлено переклад за допомогою Google Translate API, якщо в користувача недостатньо відеопам’яті для роботи моделі facebook/seamless-m4t-v2-large. Для цього потрібна бібліотека deep_translator, а

саме її частина GoogleTranslator. Переклад за допомогою Google Translate зображено в лістингу 3.9.

Лістинг 3.9 – Цикл перекладу субтитрів за допомогою Google Translate API

```
translator = GoogleTranslator(source='auto',
                              target=self.target_language)
translated_subtitles = [] total =
len(self.subtitles) for i, (start, text) in
enumerate(self.subtitles):
      translated_text = translator.translate(text)
      translated_subtitles.append((start, translated_text))
      progress = int((i+1)/total * 100)
      self.progress_update.emit(progress) self.subtitles =
      translated_subtitles
```

кінець лістингу 3.9

3.4 Конфігурація комп’ютерних систем

Розроблену програму було протестовано на різних системах: на ноутбуці з відеокартою Nvidia Geforce RTX 2070 (8 ГБ відеопам’яті GDDR6, 2304 CUDAядер), процесором Intel Core i7-10750H (6 ядер, 12 потоків, максимальна частота на всіх ядрах 4.2 ГГц) і 16 ГБ оперативної пам’яті та на комп’ютері з відеокартою Nvidia Geforce RTX 3090 (24 ГБ відеопам’яті GDDR6X, 10496 CUDA-ядер), процесором Intel Core i9-10900F (10 ядер, 20 потоків, максимальна частота на всіх ядрах 4.6 ГГц), 32 ГБ оперативної пам’яті. Досліджувалося швидкодія програми.

Програма була протестована на звичаних налаштуваннях процесора та відеокарти, а потім було виконано андервольт з ціллю оптимізації роботи. Як зображено на рисунку 3.1, напругу процесора i7-10750H було зменшено на 80 мВ через обмеження BIOS ноутбука. Хоч відеочіп у цього процесора був присутній, його андервольт заблокований також через BIOS ноутбука.

Name	Mode	Voltage	Offset
CPU Core	Adaptive	Default	-0.0801
Intel GPU	Adaptive	Default	+0.0000
CPU Cache	Adaptive	Default	-0.0801
iGPU Unslice	Adaptive	Default	+0.0000
System Agent	Adaptive	Default	+0.0000

Рисунок 3.1 – Андервольт центрального процесора i7-10750H

Знизивши напругу центрального процесора i9-10900F на 115 мВ, комп'ютер час від часу вимикався, тому було прийнято рішення залишити значення в 110 мВ. Як і в центральному процесорі, було проведено андервольт лише ядер та кешу, андервольт System Agent не знижує температур і приводить до вимкнення комп'ютер при невеликих значеннях андервольту. Графічного ядра в цьому процесорі немає, про що свідчить індекс F в назві процесора, тому відповідно андервольт графічного ядра є неможливим. Напругу центрального процесора i9-10900F було зменшено на 110 мВ, що видно на рисунку 3.2.

Name	Mode	Voltage	Offset
CPU Core	Adaptive	Default	-0.1104
Intel GPU	Adaptive	Default	+0.0000
CPU Cache	Adaptive	Default	-0.1104
iGPU Unslice	Adaptive	Default	+0.0000
System Agent	Adaptive	Default	+0.0000

Рисунок 3.2 – Андервольт центрального процесора i9-10900F

Стандартну криву частот та напруги графічного процесора Nvidia Geforce RTX 3090 з BIOS від відеокарти моделі ASUS ROG STRIX RTX 3090 GAMING показано на рисунку 3.3, а змінену криву після проведення андервольту на рисунку 3.4.

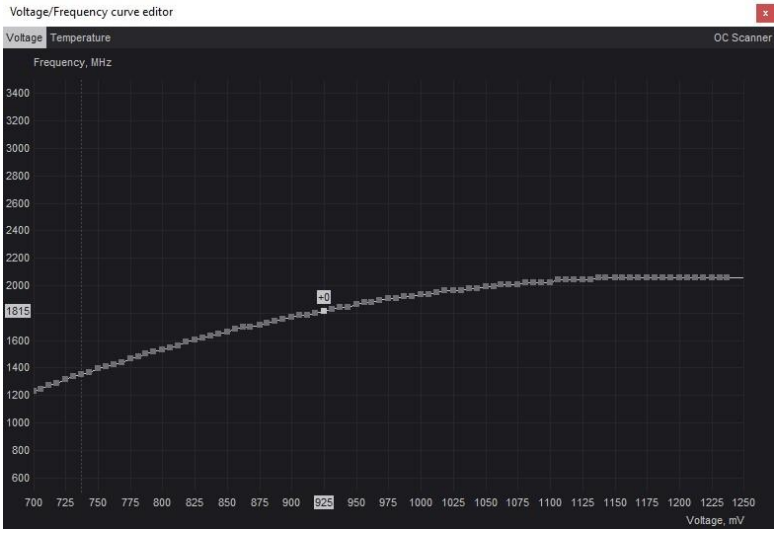

Рисунок 3.3 – Стандартна крива частот та напруги RTX 3090

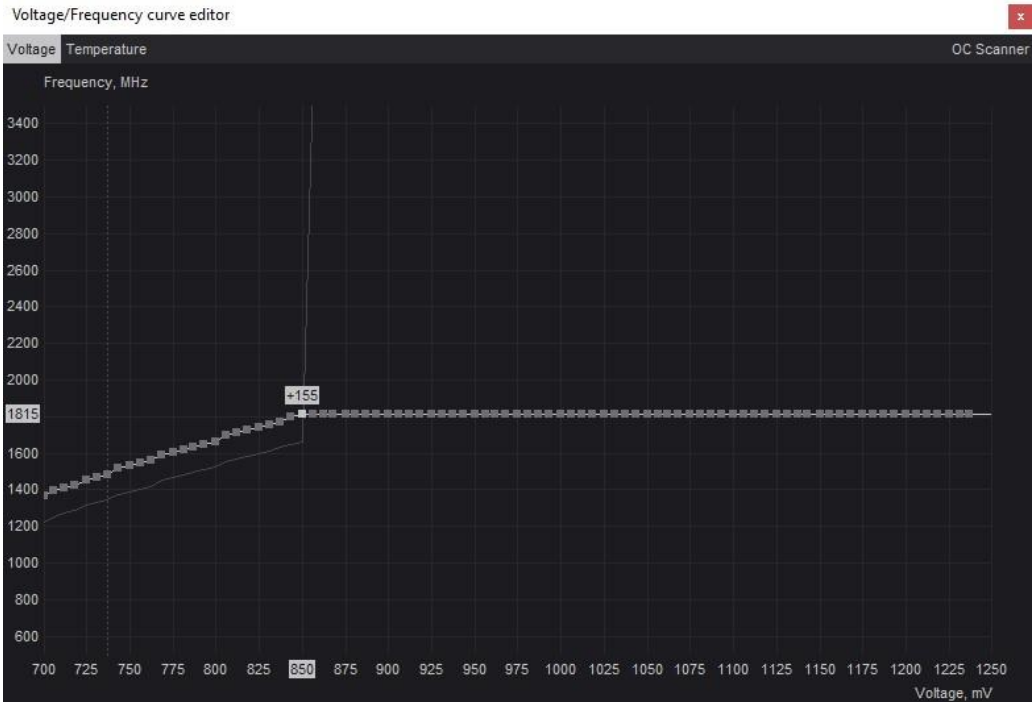

Рисунок 3.4 – Крива частот та напруги RTX 3090 після андервольту

Також було проведено андервольт і графічного процесора RTX 2070. Стандартна крива частот та напруги зображена на рисунку 3.5, а крива після андервольту – на рисунку 3.6.

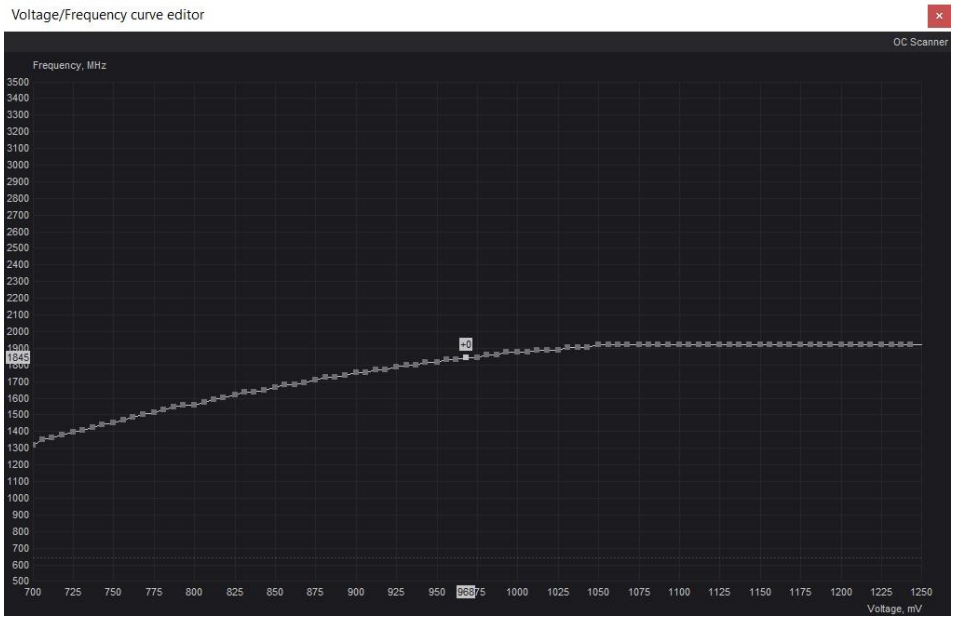

Рисунок 3.5 – Стандартна крива частот та напруги RTX 2070

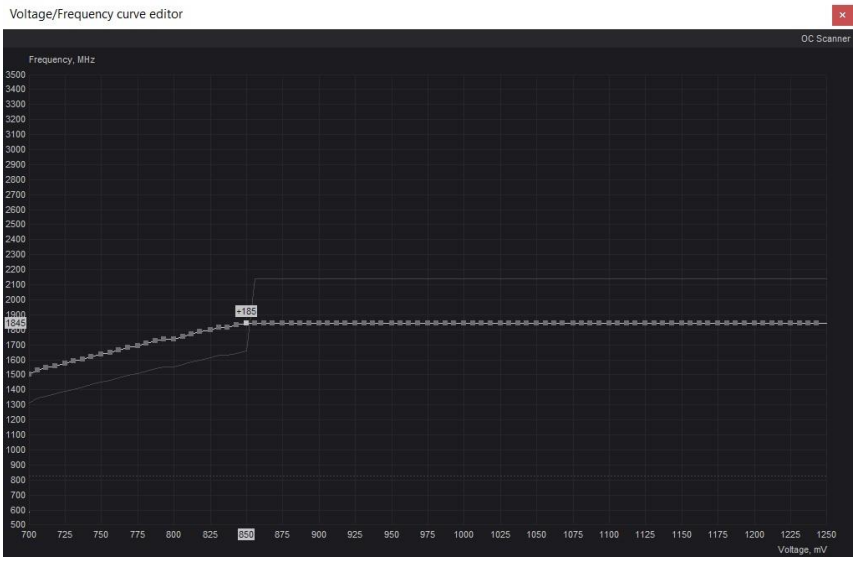

Рисунок 3.6 – Крива частот та напруги RTX 2070 після андервольту

Також частоту відеопам’яті відеокарти RTX 2070 було збільшено на 800 МГц, а RTX 3090 на 1000 МГц.

Для перевірки стабільності андервольту використовувався стрес-тест OCCT 3D Adaptive. Частоти відеопам’яті збільшувались доти, доки результати бенчмарку 3Dmark Steel Nomad не почали зменшуватись, тобто не почались з’являтись помилки при записі в неї та не почалось використання функції коригування помилок, яке затрачало час на створення кадру та кінцевий рахунок

не зменшився. Андервольт центральних процесорів виконувався через BIOS, а графічних за допомогою програми MSI Afterburner.

3.5 Тестування програми

Для тестування швидкодії було обрано відео довжиною рівно 5 хвилин англійською мовою. Переклад виконувався з англійської на українську мову. Для запису статистики по частотам та енергоспоживанню компонентів систем використовувалась програма HWiNFO версії 8.24 [40]. Версії драйверів відеокарт були одинаковими, 566.36. Результати наведені в додатку А в таблиці А.1.

Отже, найбільш енергоефективним способом виявився запуск на ноутбуці з відеокартою без використання Seamless-M4T-v2-large через те, що переклад за допомогою Google Translate не використовує відеокарту, а посилає запити на веб-сайт Google, де вже виконується переклад, який є менш точним за переклад моделлю, обраною нами. Найефективніший переклад за допомогою SeamlessM4T-v2-large зайняв 3 місце на відеокарті RTX 2070 з різницею в 0.58 Вт-год від першого місця, що свідчить про те, що переклад за допомогою штучного інтелекту може бути майже таким ефективним, як переклад через сервіс Google Translate.

Також можна побачити, що у всіх випадках, за виключенням двох, андервольт допоміг енергоспоживання системи. Виключеннями є результати андервольту процесора i7-10750H як при перекладі за допомогою Seamless-M4Tv2-large, так і Google Translate. Андервольт збільшив середню частоту процесора у випадку перекладу через Seamless-M4T-v2-large на 388 МГц, а при використанні Google Translate на 419 МГц, але це відповідно збільшило і енергоспоживання на 0.69 і 0.47 Вт-год відповідно.

Взагалом, андервольт зменшив енергоспоживання відеокарти RTX 2070 при перекладі за допомогою моделі facebook/seamless-m4t-v2-large на 30 %, що є найбільшим приростом в енергоефективності серед наведених даних, хоч і збільшив енергоспоживання процесора при такому ж методі на 11 %. Відеокарті

RTX 3090 та процесору i9-10900F андервольт домоміг у всіх випадках, і зменшив використання електроенергії на 17 %. В таблиці 3.2 можна побачити результати замірів часу виконання різних стадій обробки відео програмою використовуючи найоптимальнішу модель та Google Translate на комп'ютері з графічним процесором Nvidia Geforce RTX 3090 з центральним процесором Intel Core i910900F і ноутбуком з графічним процесором Nvidia Geforce RTX 2070 і центральним процесором Intel Core i7-10750H.

В таблиці 3.2 зображено загальний час, який було затрачено на переклад різними конфігураціями комп'ютерних систем та способами перекладу.

Таблиця 3.1 – Час, затрачений на кожну зі стадій обробки: створення субтитрів та їх переклад

Компонент комп. системи	Спосіб перекладу, с	Час на створення субтитрів, с	Час на переклад субтитрів, с	Загальний час, с
Nvidia Geforce RTX 3090	facebook/seamless-m4t-v2large	22.52	67.43	89.95
Nvidia Geforce RTX 3090	Google Translate	22.36	17.88	40.24
Nvidia Geforce RTX 2070	facebook/seamless-m4t-v2large	30.72	78.81	109.52
Nvidia Geforce RTX 2070	Google Translate	31.18	30.66	61.84
Intel Core i910900F	facebook/seamless-m4t-v2large	157.76	208.21	365.97
Intel Core i910900F	Google Translate	156.46	14.6	171.06
Intel Core i710750H	facebook/seamless-m4t-v2large	295.66	280.37	576.04
Intel Core i710750H	Google Translate	290.62	21.17	311.79

За результатами можна визначити, що модель facebook/seamless-m4t-v2large працює приблизно однаково як і на відеокартах з 8 ГБ відеопам’яті, так і з 24 ГБ, оскільки розмір моделі Whisper є меншим за 6 ГБ, а модель facebook/seamless-m4t-v2-large займає 3.5-4 ГБ відеопам’яті. Вони завантажені у відеопам’ять в різний час, отже нам не потрібна дорога комп’ютерна система для забезпечення швидкого та точного перекладу. При використанні моделі OpenAI Whisper Turbo достатньо мати до 6 ГБ відеопам’яті. Якщо швидкість не принципова, то можна скористатися варіантами Medium (використовує до 5 ГБ відеопам’яті) або Small (потребує до 2 ГБ відеопам’яті). Для процесорів найкращим варіантом буде використання Google Translate для пришвидшення роботи на 40-50 %. Незважаючи на знижену точність перекладу, програма буде займати до 6 ГБ оперативної пам’яті на комп’ютері та значно швидше оброблятиме відео. Переклад субтитрів до відео довжиною 5 хвилин відбувається в 13-14 разів швидше з Google Translate, ніж з великою мовною моделлю.

ВИСНОВКИ

За результатами виконання кваліфікаційної роботи можна зробити відповідні висновки:

Досліджено метрики оцінки точності машинного перекладу BLEU, cHRF++ та TER та виявлено, що найоптимальнішою моделлю штучного інтелекту для перекладу є facebook/seamless-m4t-v2-large через найвищі оцінки в усіх метриках з трьох, і лише в одній метриці оцінка однієї з моделей була близькою.

Розроблено програму створення та перекладу субтитрів на одну з 95 мов світу. Після розробки отримано 3 Python-файли, які складають з себе програму субтитрування. Перший файл main.py виводить інтерфейс користувача. Другий файл video_player.py відповідає за ініціалізацію всіх кнопок в програмі та вивід відео з субтитрами. Останній файл subtitle_generator.py має функції, де виконується створення субтитрів та їх переклад.

Перевірено швидкодію розробленої програми на різних конфігураціях комп'ютерних систем та оптимізовано роботу графічного та центрального процесора за допомогою андервольту з подальшим покращенням результатів у більшості випадків. Андервольт відеокарти RTX 2070 знизив енергоспоживання на 30 %, а процесора i9-10900F та відеокарти RTX 3090 на 17 %. Андервольт не знизив енергоспоживання лише у випадку процесора i7-10750H, тобто в одному випадку з чотирьох. Окрім зниження енергоспоживання, знизився і час, який потрібен для створення субтитрів для обох центральних процесорів. Субтитри створюються приблизно одинакову кількість часу за допомогою моделі facebook/seamless-m4t-v2-large на обох графічних процесорах, але час збільшився у випадку перекладу за допомогою Google Translate API.

Підібрано найточнішу модель розпізнавання мовлення – OpenAI Whisper Turbo через найвищу точність роботи, простоту використання, відсутність потреби в додатковому тренуванні, швидкість роботи та розмір, який дозволяє

використовувати цю модель на графічних процесорах з відносно невеликою кількістю відеопам’яті (2-6 ГБ залежно від варіанту моделі).

Досліджено роботу моделей штучного інтелекту на центральному та графічному процесорах та визначено найоптимальніші сценарії роботи: користувачам з графічними процесорами, які мають більше 4 ГБ відеопам’яті та більше рекомендовано використовувати для перекладу модель facebook/seamless-m4t-v2-large, а решта користувачів можуть використовувати переклад за допомогою Google Translate API. Модель розпізнавання мовлення OpenAI Whisper Small здатна працювати на графічних процесорах з 2 ГБ відеопам’яті, що є мінімальною вимогою до відеопам’яті для оптимальної швидкості роботи програми. Рекомендовано використовувати відеокарту з 6 ГБ відеопам’яті та більше для того, щоб використовувати модель OpenAI Whisper Turbo для найшвидшого та найточнішого результату.

Розроблену систему можна використовувати для перегляду відео формату MP4 мовами, якими користувач не володіє або якщо в нього є проблеми зі слухом та прослуховування відео не є можливим.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Омельчук Д., Мельник К., Мельник П. Аналіз методів оцінки машинного перекладу та факторів, що впливають на їх вибір. *Цифрова трансформація: виклики та стратегії* : матеріали міжнар. науково-практ. конф., м. Луцьк, 25 лют. 2025 р. Луцьк, 2025. С. 163-165.

2. Supervised learning. *Scikit-learn*. URL: https://scikitlearn.org/stable/supervised_learning.html (дата звернення: 30.03.2025).

3. GeeksforGeeks. Connectionist temporal classification – geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/connectionist-temporalclassification/ (дата звернення: 30.03.2025).

4. IBM. What Is Self-Supervised Learning? IBM. *IBM – United States*. URL:

https://www.ibm.com/think/topics/self-supervised-learning (дата звернення: 30.03.2025).

5. Школа штучного інтелекту. Л4. Згорткові нейронні мережі, 2024. *YouTube*. URL: https://www.youtube.com/watch?v=bddWlGuoKsU (дата звернення: 31.03.2025).

6. The Essential Guide to Neural Network Architectures. *V7 AI Document Processing & Data Labelling*. URL: https://www.v7labs.com/blog/neural-networkarchitectures-guide (дата звернення: 31.03.2025).

7. B M. A deep dive into activation functions: a comprehensive guide for neural network beginners. *Medium*. URL: https://bit.ly/435sFL1 (дата звернення: 31.03.2025).

8. Школа штучного інтелекту. Л8. Механізм уваги і трансформери, 2024. *YouTube*. URL: https://www.youtube.com/watch?v=yIYFv0zJByI (дата звернення: 17.04.2025).

9. Optimizing LLMs for speed and memory. *Hugging Face – The AI community building the future*. URL: https://shorturl.at/pXY6q (дата звернення: 27.04.2025).

10. Forward Propagation, Backward Propagation, and Computational Graphs – Dive into Deep Learning 1.0.3 documentation. *Dive into Deep Learning – Dive into Deep Learning 1.0.3 documentation*. URL: https://d2l.ai/chapter_multilayerperceptrons/backprop.html (дата звернення: 27.04.2025).

11. Школа штучного інтелекту. Л7. Рекурентні нейронні мережі, 2024. *YouTube*. URL: https://www.youtube.com/watch?v=LiNGo4k6WLY (дата звернення: 28.04.2025).

12. Andrej Karpathy. The spelled-out intro to neural networks and backpropagation: building micrograd, 2022. *YouTube*. URL: https://www.youtube.com/watch?v=VMj-3S1tku0 (дата звернення: 27.04.2025).

13. Quantization. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/docs/transformers/main_classes/quantization (дата звернення: 27.04.2025).

14. Gemma 3 QAT models: bringing state-of-the-art AI to consumer gpus– google developers blog. *Home – Google Developers Blog*. URL: https://bit.ly/4jTYCf4 (дата звернення: 27.04.2025).

15. LLM pruning & distillation: Minitron approach SuperAnnotate. *SuperAnnotate*. URL: https://www.superannotate.com/blog/llm-pruning-distillationminitron-approach (дата звернення: 27.04.2025).

16. Distilling step-by-step: outperforming larger language models with less training. *Google Research – Explore Our Latest Research in Science and AI*. URL: https://bit.ly/43zYmfo (дата звернення: 27.04.2025).

17. Performing gradient accumulation with Accelerate. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/docs/accelerate/usage_guides/gradient_accumulation (дата звернення: 27.04.2025).

18. GeeksforGeeks. Multi-Head attention mechanism – geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/multi-head-attentionmechanism/ (дата звернення: 30.03.2025).

19. OPUS corpora. *OPUS Corpora*. URL: https://opus.nlpl.eu/ (дата звернення: 27.04.2025).

20. WMT. *Machine Translate*. URL: https://machinetranslate.org/wmt (дата звернення: 27.04.2025).

21. Gsarti/flores_101 datasets at hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/datasets/gsarti/flores_101 (дата звернення: 27.04.2025).

22. What is RNN? Recurrent neural networks explained – AWS. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/what-is/recurrent-neural-network/ (дата звернення: 27.04.2025).

23. Long Short-Term Memory (LSTM). *NVIDIA Developer*. URL: https://developer.nvidia.com/discover/lstm (дата звернення: 27.04.2025).

24. Explain the transformer architecture (with examples and videos). *AIML.com*. URL: https://aiml.com/explain-the-transformer-architecture/ (дата звернення: 27.04.2025).

25. Wav2Vec2. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/docs/transformers/model_doc/wav2vec2 (дата звернення: 27.04.2025).

26. GitHub openai/whisper: robust speech recognition via large-scale weak supervision. *GitHub*. URL: https://github.com/openai/whisper (дата звернення: 27.04.2025).

27. Helsinki-NLP/opus-mt-en-uk hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/Helsinki-NLP/opus-mten-uk (дата звернення: 27.04.2025).

28. Facebook/nllb-200-distilled-1.3B hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/facebook/nllb-200distilled-1.3B (дата звернення: 27.04.2025).

29. Qwen/Qwen2.5-7B-Instruct hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/Qwen/Qwen2.5-7BInstruct (дата звернення: 27.04.2025).

30. Meta-llama/Llama-3.1-8B-Instruct hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/meta-llama/Llama-3.18B-Instruct (дата звернення: 27.04.2025).

31. Microsoft/Phi-4-mini-instruct hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/microsoft/Phi-4-miniinstruct (дата звернення: 27.04.2025).

32. GitHub lang-uk/dragoman. *GitHub*. URL: https://github.com/languk/dragoman (дата звернення: 27.04.2025).

33. Facebook/seamless-m4t-v2-large hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/facebook/seamless-m4tv2-large (дата звернення: 27.04.2025).

34. Mistralai/Mistral-7B-Instruct-v0.3 hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/mistralai/Mistral-7BInstruct-v0.3 (дата звернення: 27.04.2025).

35. BLEU testing with kolena. *Developer Guide Testing with Kolena*. URL: https://docs.kolena.com/metrics/bleu/ (дата звернення: 27.04.2025).

36. Tamchyna A. MT metrics explained: A visit to the MT metric zoo. *Medium*. URL: https://bit.ly/3EZgxC5 (дата звернення: 27.04.2025).

37. Evaluating GenAI Performance with CHRF++. *Predactica*. URL: https://predactica.com/blog/genai-performance-measurement-tool-chrf/ (дата звернення: 27.04.2025).

38. METEOR testing with kolena. *Developer Guide Testing with Kolena*. URL: https://docs.kolena.com/metrics/meteor/ (дата звернення: 27.04.2025).

39. ROUGE-N testing with kolena. *Developer Guide Testing with Kolena*. URL: https://docs.kolena.com/metrics/rouge–n/ (дата звернення: 27.04.2025).

40. Free download HWiNFO sofware installer & portable for windows, DOS. *HWiNFO*. URL: https://www.hwinfo.com/download/ (дата звернення: 27.04.2025).