Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РЕАЛІЗАЦІЯ СИСТЕМИ МАШИННОГО ПЕРЕКЛАДУ ДЛЯ МОВ ІЗ ОБМЕЖЕНОЮ КІЛЬКІСТЮ ДАНИХ З ВИКОРИСТАННЯМ FINE-TUNING ТРАНСФОРМЕРІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-2
Лисенко Д. О.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О. А.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Лисенку Данилу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Реалізація системи машинного перекладу для мов із обмеженою кількістю даних з використанням fine-tuning трансформерів

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 20 червня 2026 р.
3. Вихідні дані до роботи матеріали досліджень архітектури Transformer та механізмів уваги, офіційна документація та препринти моделей сімейства NLLB-200 (Meta AI), наукові публікації щодо методів параметрично ефективного донавчання (PEFT), відкриті паралельні корпуси (OPUS, WikiMatrix, Tatoeba) для боснійсько-українського напрямку, текстові масиви Вікіпедії для реалізації стратегії зворотного перекладу (Back-Translation), документація бібліотек PyTorch та HuggingFace (Transformers, PEFT), специфікації обчислювального рушія CTranslate2 для квантування моделей, документація фреймворків FastAPI (backend) та React/TypeScript (frontend), методологічні рекомендації щодо оцінювання якості перекладу за метриками BLEU, chrF та COMET.
4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасного стану NMT та еволюції архітектур.
2. Дослідження специфіки низькоресурсних мовних пар та флективних слов'янських мов.
3. Огляд методів оцінювання якості (BLEU, chrF, COMET, BERTScore).
4. Дослідження методів PEFT з математичним обґрунтуванням LoRA та DoRA.
5. Розроблення методології формування гібридного набору даних.
6. Експериментальні дослідження стратегій донавчання та архітектурних рішень.
7. Проєктування архітектури системи, програмна реалізація сервера з квантуванням INT8.
8. Реалізація алгоритмів збереження форматування .docx.
9. Розроблення реактивного інтерфейсу з підтримкою SSE.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) схеми порівняння архітектур моделей (RNN, Transformer, Seq2Seq, Decoder-only) та метрики COMET, UML-діаграми проєктування системи (прецеденти, класи, послідовності), діаграми діяльності та станів для процесів навчання, квантування і черг запитів, схеми архітектурної взаємодії та ієрархії компонентів системи, графік динаміки функції втрат під час донавчання, комп'ютерні ілюстрації роботи клієнтського інтерфейсу та процесу перекладу документів.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Кобилін О. А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 88 с., 33 табл., 22 рис., 1 дод., 52 джерела.

МАШИННИЙ ПЕРЕКЛАД, НИЗЬКОРЕСУРСНІ МОВИ, ОБРОБКА ПРИРОДНОЇ МОВИ, ТРАНСФОРМЕРИ, DORA, FINE-TUNING.

Об’єктом роботи є процес розроблення системи нейронного машинного перекладу для боснійсько-українського напрямку.

Метою роботи є створення вебзастосунку двонаправленого перекладу на базі архітектури DoRA та згенерованих наборів даних для підвищення морфологічної точності.

Використані методи: штучний інтелект, архітектура Transformer, параметрично ефективне донавчання (PEFT), веброзробка.

Практична цінність: розроблено оптимізований вебзастосунок на основі квантованої моделі NLLB-600M, що здатний коректно обробляти складну морфологію та перекладати файли із збереженням розмітки в умовах обмежених апаратних ресурсів.

Область застосування: високоточний машинний переклад повсякденних і спеціалізованих текстів, автоматизація перекладу файлів зі збереженням їхньої структури.

Результати дослідження отримані в межах міжнародного проєкту INITIATE за грантом No. 101136775 HORIZON WIDERA 2023 ACCESS 03.

DORA, FINE-TUNING, LOW-RESOURCE LANGUAGES, MACHINE TRANSLATION, NATURAL LANGUAGE PROCESSING, TRANSFORMERS.

The object of the work is the development of a neural machine translation system for the Bosnian-Ukrainian language pair.

The goal of work is to create a bidirectional translation web application based on the DoRA architecture and synthesized datasets to improve morphological accuracy.

Methods used: artificial intelligence, Transformer architecture, parameter-efficient fine-tuning (PEFT), web development.

Practical value: an optimized web application based on a quantized NLLB-600M model for both texts and .docx documents.

Applications: high-accuracy machine translation of everyday and specialized texts, automated file translation with structure preservation.

The results of this research were obtained under the international project INITIATE under the grant No. 101136775 HORIZON WIDERA 2023 ACCESS 03.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

API – Application Programming Interface (інтерфейс програмного застосунку)

BLEU – Bilingual Evaluation Understudy (метрика оцінювання якості машинного перекладу)

DoRA – Weight-Decomposed Low-Rank Adaptation (метод низькорангової адаптації з декомпозицією ваг)

GPU – Graphics Processing Unit (графічний процесор)

LLM – Large Language Model (велика мовна модель)

LoRA – Low-Rank Adaptation (метод низькорангової адаптації)

MLP – Multi-Layer Perceptron (багатошаровий персептрон)

NLP – Natural Language Processing (обробка природної мови)

PEFT – Parameter-Efficient Fine-Tuning (параметрично ефективне донавчання)

RNN – Recurrent Neural Network (рекурентна нейронна мережа)

VRAM – Video Random Access Memory (відеопам’ять)

ВСТУП

Проблема машинного перекладу для близькоспоріднених слов'янських мов, зокрема для боснійсько-українського напрямку, залишається надзвичайно актуальною. Насамперед це зумовлено дефіцитом якісних паралельних наборів даних. Існуючі багатомовні моделі часто втрачають семантичне узгодження, формуючи граматично некоректні конструкції під час генерації тексту. У той час як використання методів автоматичного збору інформації з відкритих джерел призводить до потрапляння в навчальну вибірку значного обсягу лінгвістичного шуму. Відповідно до цього, виникає складна спеціалізована задача створення систем, здатних ефективно обробляти флективну морфологію в умовах жорстких ресурсних обмежень.

Сучасний розвиток архітектури Transformer відкрив нові можливості для оброблення природної мови, і передові світові рішення у цій сфері демонструють високу точність трансляції змісту. Проте застосування масивних нейронних мереж вимагає значних обчислювальних потужностей. З огляду на те, що більшість існуючих систем оптимізовано для мов із великим обсягом доступних даних, якість оброблення низькоресурсних мовних пар залишається недостатньою. Таким чином, наявні підходи потребують суттєвої архітектурної та алгоритмічної адаптації для забезпечення коректної роботи на стандартному споживчому обладнанні.

Актуальність обраної теми полягає у необхідності розроблення ефективного програмного рішення для боснійсько-українського перекладу, яке здатне подолати вищезазначені обмеження. Здійснений аналіз вказує на доречність поєднання методів штучного синтезу даних із сучасними алгоритмами низькорангової адаптації нейронних мереж. Завдяки цьому підходу можливо досягти високої якості розуміння складної словозміни без потреби у розгортанні кластерів високопродуктивних серверів. Оптимізація процесу навчання є критично важливою для широкого впровадження технологій машинного перекладу.

Для розв’язання виявленої проблеми обґрунтовано використання комбінованого підходу до підготовки даних та процесу донавчання. Спочатку передбачається застосування генеративних моделей для морфологічної аугментації, що дозволяє компенсувати нестачу паралельних текстів. За базову архітектуру системи обрано модель NLLB з подальшою інтеграцією DoRA-адаптерів. Таке архітектурне рішення спрямоване на параметрично ефективне донавчання, у результаті якого мережа налаштовується на специфіку цільової мовної пари, мінімізуючи витрати апаратних ресурсів та зберігаючи високу швидкість висновування.

Практична реалізація розглянутих методів охоплює створення повноцінного програмного комплексу, що включає підсистему підготовки даних, ядро нейронного перекладу (Inference Engine) та відповідний вебінтерфейс. Створена система може бути використана для автоматизації перекладу технічної документації, засобів масової інформації, а також як самостійний компонент більших інформаційних систем для крос-мовної комунікації. Результати роботи закладають ґрунтовну базу для подальшого вдосконалення технологій оброблення низькоресурсних слов’янських мов.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Еволюція нейронних архітектур у машинному перекладі

1.1.1 Перехід від рекурентних мереж до архітектури Transformer

Протягом тривалого часу базовим підходом до вирішення завдань обробки природної мови, зокрема машинного перекладу, були рекурентні нейронні мережі (RNN) та їхні модифікації, такі як мережі довгої короткострокової пам’яті (LSTM). Основною концепцією цих архітектур є послідовне оброблення вхідних даних, де кожен наступний стан прихованого шару залежить від попереднього. Проте такий підхід має суттєві математичні та архітектурні обмеження. Найбільш критичною є проблема затухання градієнта (vanishing gradient problem), яка виникає під час зворотного поширення помилки крізь час. Оскільки градієнти обчислюються за допомогою ланцюгового правила множення, для довгих послідовностей значення градієнта експоненційно зменшується, що унеможливлює ефективне навчання на віддалених залежностях у тексті. Крім того, послідовна природа RNN повністю виключає можливість паралельних обчислень під час тренування, що робить процес навчання на великих наборах даних вкрай неефективним з точки зору використання апаратних ресурсів.

Революційним кроком у розвитку машинного перекладу стала поява архітектури Transformer, яка повністю відмовилася від рекурентності на користь механізму уваги (Self-Attention) [1, 2]. Цей механізм дозволяє моделі одночасно аналізувати всі елементи вхідної послідовності, встановлюючи зв’язки між словами незалежно від їхньої фізичної відстані у реченні. В основі механізму лежить обчислення трьох матриць: запитів (Query, Q), ключів (Key, K) та значень (Value, V). Ці матриці отримуються шляхом множення вхідних векторів слів (ембедінгів) на відповідні матриці ваг, що навчаються.

Математично механізм зваженої уваги (Scaled Dot-Product Attention) має такий вигляд:

$$Attention(Q,K,V) = soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (1.1)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмірність векторів ключів.

Для наочного розуміння відмінностей в обробці даних двома архітектурами розроблено відповідну блок-схему (рис. 1.1).

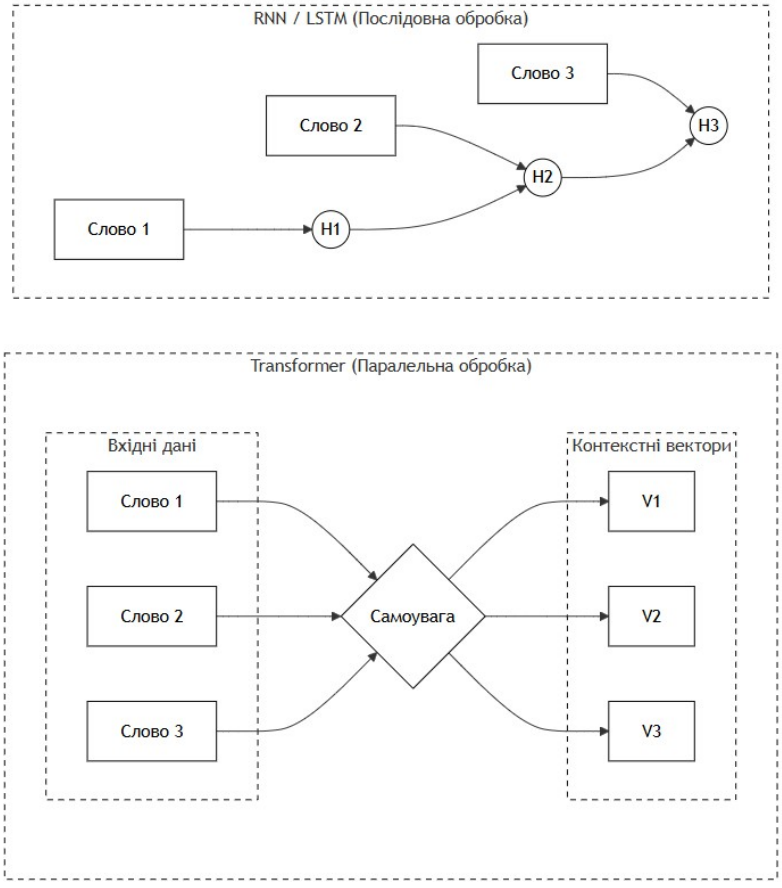


Рисунок 1.1 – Порівняння принципів обробки послідовностей в RNN та Transformer

Таким чином, перехід до архітектури Transformer вирішив проблему обробки довгих залежностей і відкрив шлях до створення масивних багатомовних моделей завдяки можливості ефективного розпаралелювання обчислень на графічних процесорах.

1.1.2 Огляд парадигм Seq2Seq та Decoder-only

Сучасні великі мовні моделі (LLM) здебільшого поділяються на дві основні архітектурні парадигми: моделі типу кодувальник-декодувальник (Encoder-Decoder або Seq2Seq) та суто декодерні моделі (Decoder-only) (рис. 1.2) [3]. Архітектурні розбіжності між цими парадигмами наведено на рисунку 1.3. Моделі Seq2Seq, такі як T5 або BART, мають чіткий поділ: кодувальник обробляє вхідний текст, перетворюючи його на щільне контекстне представлення, а декодувальник генерує вихідний текст на основі цього представлення. Суто декодерні моделі, такі як GPT, об’єднують ці процеси, прогнозуючи наступний токен виключно на основі попереднього контексту (авторегресійна генерація). Використання єдиного формату text-to-text дозволило стандартизувати ці задачі генерації, що суттєво полегшило їх архітектурну адаптацію.

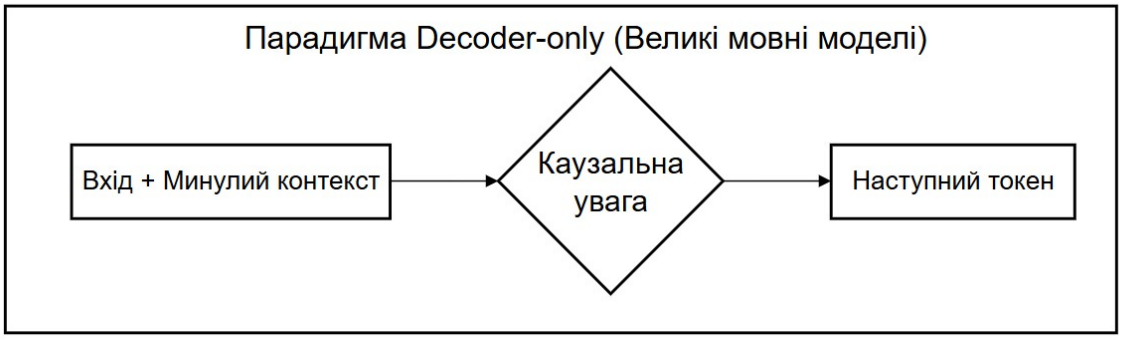


Рисунок 1.2 – Аріхтектура Decoder-only

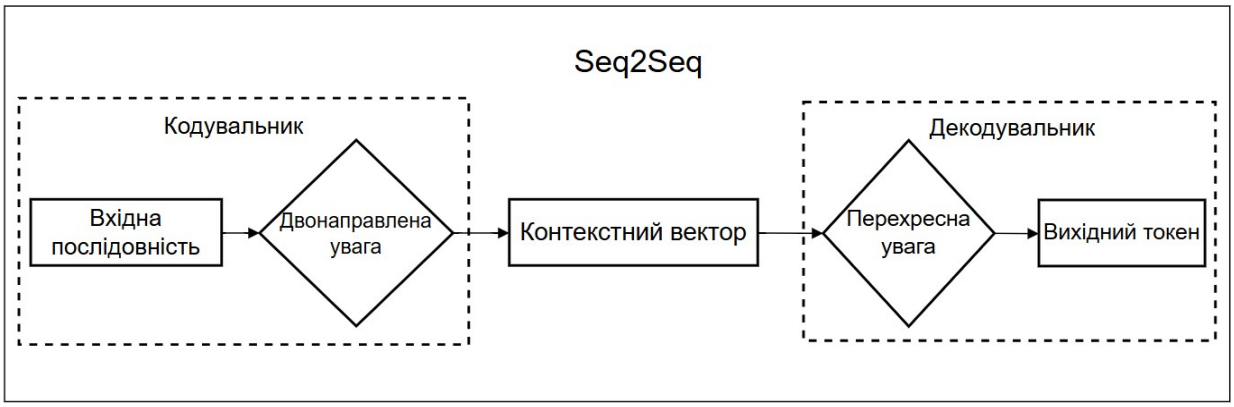


Рисунок 1.3 – Архітектура Seq2Seq

Для завдань машинного перекладу, особливо у випадку складних слов'янських мов, парадигма Encoder-Decoder має фундаментальну перевагу [4, 5]. Кодувальник використовує двонаправлену увагу, тобто кожне слово має доступ до повного контексту всього речення (як зліва, так і справа) ще до початку генерації перекладу. Це критично важливо для мов із вільним порядком слів, де семантична роль слова може визначатися закінченням, розташованим у кінці речення. Натомість суто декодерні моделі використовують однонаправлену (каузальну) увагу, що обмежує їхню здатність враховувати подальший контекст при побудові граматичних зв'язків.

Встановлено, що хоча моделі Decoder-only демонструють вражаючі результати в генерації тексту та виконанні загальних інструкцій, вони часто поступаються в точності морфологічного узгодження при перекладі близькоспоріднених мов. Це призводить до помилок у відмінках, родах та числах, що спотворює зміст. Порівняльний аналіз цих парадигм подано у таблиці 1.1.

З огляду на наведені фактори, для розроблення системи машинного перекладу між слов'янськими мовами обґрунтованим є вибір архітектури Seq2Seq.

Таблиця 1.1 – Порівняння архітектурних парадигм генеративних моделей

Характеристика	**Encoder-Decoder (Seq2Seq)**	**Decoder-only**
Механізм уваги для вхідних даних	Двонаправлений (повний контекст)	Однонаправлений (авторегресійний)
Спеціалізація	Переклад, узагальнення	Генерація тексту, діалогові системи
Стійкість до вільного порядку слів	Висока	Середня
Врахування морфологічних зв’язків	Високе	Потребує значного масштабування

1.1.3 Багатомовна модель NLLB-200

Розв’язання проблеми перекладу для низькоресурсних мов (до яких належить боснійська мова) вимагає спеціалізованих рішень. Одним із найуспішніших проєктів у цьому напрямку є розроблення моделі NLLB-200 (No Language Left Behind) від компанії Meta [6]. Ця архітектура здатна здійснювати прямий переклад між 200 мовами без використання англійської як проміжної ланки, що суттєво зменшує втрату семантичної інформації під час перекладу між близькоспорідненими мовами.

Архітектурною основою NLLB є модифікований Transformer. Важливою особливістю підходу Meta стала методика формування набору даних: для вилучення паралельних речень з інтернету застосовувався глобальний автоматичний видобуток даних на основі багатомовних векторних представлень LASER3. Завдяки цьому модель здобула здатність проєктувати знання з високоресурсних слов’янських мов (наприклад, чеської чи польської) на низькоресурсні (як боснійська).

Для забезпечення масштабованості оригінальна флагманська версія моделі використовує архітектуру Sparsely Gated Mixture-of-Experts (MoE) і містить 54 мільярди параметрів. Вона активує лише певну частину нейронної мережі (окремих «експертів) для конкретної мовної пари, що робить її ефективною, але надзвичайно вимогливою до апаратної пам’яті (потребує серверних кластерів). Для вирішення цієї проблеми розробники застосували процес дистиляції знань (knowledge distillation) та підготували лінійку щільних (dense) моделей меншого розміру, де активуються всі параметри одночасно. Порівняння основних моделей сімейства NLLB подано у таблиці 1.2.

Таблиця 1.2 – Порівняння масштабів моделей сімейства NLLB

Версія моделі	**Архітектура**	**Кількість параметрів**	**Вимоги до VRAM, ГБ**	**Оптимальне середовище застосування**
NLLB-54B	Mixture of Experts (MoE)	54,5 млрд	> 120	Промислові серверні кластери
NLLB-3.3B	Щільна (Dense)	3,3 млрд	~ 8-10	Високопродуктивні локальні станції
NLLB-1.3B	Щільна (Dense)	1,3 млрд	~ 4-6	Доступні хмарні сервери
NLLB-600M	Щільна (Dense)	600 млн	< 3	Споживче обладнання (Edge-пристрої)

З огляду на порівняльні характеристики, NLLB-600M є оптимальною базою для дослідницької розробки. Вона зберігає високий рівень розуміння багатомовності завдяки дистиляції від 54B-моделі і дозволяє інтегрувати адаптери для донавчання на звичайних споживчих відеокартах.

1.2 Методологія оцінювання якості та семантичної близькості

1.2.1 Поверхневі метрики

Оцінювання якості машинного перекладу є невід'ємною частиною розроблення будь-якої NLP-системи. Традиційно галузевим стандартом вважаються поверхневі метрики, що базуються на лексичному збігу між згенерованим перекладом та еталонним текстом (референсом). Найбільш поширеною є метрика BLEU (Bilingual Evaluation Understudy). Алгоритм її роботи полягає у підрахунку кількості *n*-грам (послідовностей з *n* слів), які збігаються у гіпотезі та еталоні, з подальшим застосуванням штрафу за стисліcть, якщо згенероване речення коротше за оригінальне [7].

Математично точність збігу *n*-грам (p_n) для метрики BLEU визначається так:

$$p_n = \frac{\sum_{C \in Candidates} \sum_{n-gram \in C} Count_{clip}\left(n-gram\right)}{\sum_{C \in Candidates} \sum_{n-gram \in C} Count\left(n-gram\right)}, \tag{1.2}$$

де $Count_{clip}$ – кількість збігів n-грами, обмежена максимальною кількістю її поява у еталонному реченні;

$Count$ – загальна кількість n-грам у згенерованому реченні.

Іншою популярною метрикою є *chrF* (Character *n*-gram *F*-score), яка працює на рівні символьних *n*-грам. На відміну від BLEU, яка оперує цілими словами і жорстко карає за зміну закінчення, *chrF* оцінює збіги підрядків символів (наприклад, коренів слів) [8]. Математично *chrF* розраховується як *F*-міра, що поєднує точність і повноту символьних збігів:

$$chrF = \frac{\left(1+\beta^2\right) \cdot chrP \cdot chrR}{\beta^2 \cdot chrP + chrR}, \tag{1.3}$$

де *chrP* – точність (відсоток символьних n-грам гіпотези, що містяться в еталоні);

chrR – повнота (відсоток символьних n-грам еталону, що містяться в гіпотезі);

β – параметр, що регулює вагу повноти відносно точності (зазвичай $\beta = 2$, щоб надати пріоритет повноті).

Цей підхід частково вирішує проблему морфологічно багатих мов, оскільки дозволяє зараховувати слова з різними закінченнями. Проте поверхневі метрики мають критичні загальні обмеження. Вони оцінюють виключно лексичну форму, ігноруючи семантичну правильність. Для слов'янських мов високий показник BLEU чи chrF може супроводжуватися катастрофічною втратою змісту [9]. Наприклад, переклад «Він бачить собаку» і «Він бачить собака» матиме високий лексичний збіг, хоча друге речення є граматично некоректним і спотворює об'єкт дії.

1.2.2 Векторні метрики

З огляду на недоліки поверхневого оцінювання, сучасна парадигма зміщується у бік нейронних метрик, що аналізують семантичну (смислову) близькість текстів. Такі метрики використовують попередньо навчені багатомовні моделі (наприклад, XLM-RoBERTa) для перетворення речень у багатовимірні вектори (ембедінги).

Метрика COMET (Crosslingual Optimized Metric for Evaluation of Translation) є сучасним стандартом оцінки, який корелює з людським сприйняттям на рівні понад 80%. На відміну від BLEU, COMET здатна розпізнавати правильний переклад навіть тоді, коли використано синоніми, які повністю відсутні в еталонному тексті, або коли змінено порядок слів зі збереженням змісту [10]. Архітектура COMET приймає на вхід одразу три елементи: джерельне речення, еталонний переклад та згенеровану гіпотезу.

Вони проходять через багатомовний кодувальник для отримання семантичних векторів, після чого обчислюються векторні різниці та поелементні добутки, які подаються у багатошаровий персептрон (MLP) для прогнозування фінальної оцінки. Загальну схему роботи цієї метрики наведено на рисунку 1.4.

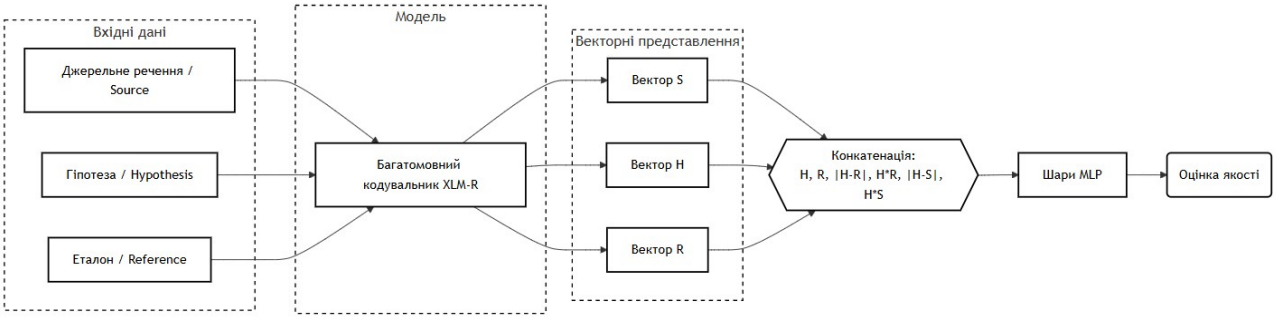


Рисунок 1.4 – Архітектура обчислення нейронної метрики COMET

Для автоматичної фільтрації наборів даних та оцінки їхньої якості часто застосовується математичний апарат косинусної подібності (Cosine Similarity). Оскільки векторні простори мовних моделей є просторами високої розмірності (наприклад, 1024 виміри), відстань між точками краще вимірювати не через евклідову метрику, а через кут між векторами. Косинусна подібність ігнорує амплітуду (довжину тексту) [7, 10], зосереджуючись виключно на напрямку. Розрахунок здійснюється за формулою:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{1.4}$$

де A_i, B_i – компоненти векторів двох речень;

$\|A\|, \|B\|$ – норми векторів.

Векторні інструменти є незамінними для відсіювання речень, що збігаються за структурою, але не відповідають за змістом (паралельне сміття). Проте їхнє масове обчислення на великих корпусах вимагає значних обчислювальних потужностей графічних процесорів (GPU).

1.3 Теоретичний огляд методів адаптації моделей

1.3.1 Математичне обґрунтування методів LoRA та DoRA

Повне донавчання (Fine-Tuning) всіх параметрів моделі розміром понад півмільярда параметрів потребує значних обчислювальних потужностей (десятків гігабайт VRAM). Для вирішення цієї проблеми розроблено методи параметрично ефективного донавчання (Parameter-Efficient Fine-Tuning, PEFT). Найбільш поширеним методом є LoRA (Low-Rank Adaptation).

Суть методу LoRA полягає в заморожуванні оригінальних ваг попередньо навченої моделі та додаванні до них невеликих обхідних матриць низького рангу . Замість оновлення масивної повної матриці параметрів, LoRA апроксимує її зміну за допомогою добутку двох значно менших матриць [11]. Перша матриця відповідає за стиснення вхідної розмірності до невеликого заданого рангу (так званого «вузького місця»), а друга матриця розгортає це стиснене представлення назад до оригінальної розмірності. Під час проходження сигналу через шар, вихідне значення формується як сума результату роботи базових заморожених ваг та результату роботи цього низькорангового адаптера. Це зменшує кількість параметрів, які необхідно навчати, на 90-98%. Однак дослідження траєкторій градієнтів показало, що LoRA має обмеження: адаптер одночасно відповідає як за напрямок зміни ваг, так і за їхню амплітуду (величину впливу). Це обмежує здатність адаптера імітувати поведінку повного донавчання, особливо при вивченні складних лінгвістичних винятків.

Для усунення цього недоліку розроблено метод DoRA (Weight-Decomposed Low-Rank Adaptation). Він базується на концептуальній декомпозиції матриці ваг на дві відокремлені складові: вектор амплітуди, який визначає загальну силу активації нейронів, та матрицю напрямку, яка вказує на семантичний зсув у просторі ознак. DoRA застосовує підхід низькорангової адаптації (аналогічний до LoRA) виключно до матриці напрямку [12, 13]. При цьому вектор амплітуди виноситься як окремий параметр, що тренується незалежно. Під час прямого проходу мережі оновлене значення ваги обчислюється шляхом нормалізації спрямованої матриці та її подальшого масштабування на вектор амплітуди. Таке розділення дозволяє моделі динамічно балансувати силу активації окремих нейронів незалежно від зміни їхнього семантичного напрямку. Встановлено, що завдяки розділенню амплітуди DoRA значно перевершує класичну LoRA в завданнях, які потребують тонкого граматичного узгодження і переведення мовних концептів у нові відмінкові форми.

1.3.2 Роль MLP-модулів у засвоєнні мовних флексій

При застосуванні методів PEFT ключовим рішенням є вибір цільових модулів: якщо стандартні імплементації LoRA часто обмежуються матрицями уваги (Q та V), що дієво для зміни загального стилю, то для фундаментальної зміни граматичної поведінки під час машинного перекладу цього абсолютно недостатньо, оскільки механізм уваги функціонує переважно як маршрутизатор токенів. Квантована адаптація та методи інтелектуального аналізу часових рядів дозволяють набагато ефективніше працювати зі складними параметричними зв’язками [14–16]. Сучасні дослідження доводять, що понад 60% параметрів зосереджено в шарах багатошарового персептрона (MLP), які відіграють роль лінгвістичної пам’яті (knowledge-value memories), де перший шар (up_proj або gate_proj)

активується на розпізнаний граматичний шаблон, а другий (down_proj) конвертує його в конкретний розподіл ймовірностей словника. Відповідно, для ефективного засвоєння складних морфологічних правил слов'янських мов необхідно змінювати саме механізм конвертації ознак у слова, що вимагає обов'язкового застосування стратегії all-linear (охоплення адаптерами всіх лінійних шарів), порівняльну ефективність якої наведено у таблиці 1.3.

Таблиця 1.3 – Вплив підключення адаптерів на різні модулі архітектури

Цільовий модуль	**Кількість параметрів, що навчаються**	**Ефективність засвоєння морфології**	**Ризик перенавчання**
Тільки Q, V (увага)	Дуже низька (~0,5%)	Низька	Мінімальний
Усі матриці уваги (Q, K, V, O)	Низька (~1,5%)	Середня	Низький
Шари MLP (up_proj, down_proj)	Середня (~2,5%)	Висока	Середній
All-linear (Увага + MLP)	Висока (~4%)	Дуже висока (максимальна гнучкість)	Середній (потребує регуляризації)

1.4 Постановка завдання

Проблема машинного перекладу для близькоспоріднених слов'янських мов залишається надзвичайно актуальною через дефіцит якісних паралельних корпусів та обмеження стандартних метрик оцінювання.

Існуючі багатомовні моделі часто втрачають семантичне узгодження, формуючи граматично некоректні конструкції, а використання автоматичного збору даних призводить до потрапляння в навчальну вибірку значного обсягу шуму. Відповідно, виникає гостра необхідність розроблення спеціалізованої системи на базі сучасних методів параметричної адаптації.

Об’єктом роботи є процес розроблення системи нейронного машинного перекладу для боснійсько-українського напрямку.

Метою роботи є створення вебзастосунку двонаправленого перекладу на базі архітектури DoRA та згенерованих наборів даних для підвищення морфологічної точності.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати існуючі архітектури нейронного машинного перекладу та методики оцінювання якості;

– дослідити вплив зашумлених паралельних даних на поверхневі метрики та виявити причини метричної невідповідності;

– розробити алгоритм комбінованого синтезу даних із застосуванням генеративних моделей для морфологічної аугментації;

– провести порівняльні експерименти стратегій навчання;

– обґрунтувати архітектуру системи на базі моделі NLLB з інтеграцією DoRA-адаптерів;

– реалізувати підсистему підготовки даних та виконати навчання моделі;

– розробити ядро нейронного перекладу (Inference Engine) та користувацький вебінтерфейс;

– протестувати систему на контрольних вибірках.

2 ЕКСПЕРЕМЕНТАЛЬНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Дослідження стратегій формування набору даних

2.1.1 Експеримент з автоматичним збором даних

Для формування початкового масиву з майже півмільйона неочищених пар речень у напрямку «боснійська-українська» було проведено автоматичне агрегування даних із трьох глобальних відкритих сховищ. Кількісний та якісний розподіл джерел сирих даних наведено у таблиці 2.1. Схожий підхід інтенсивно використовується при автоматизованій побудові та модифікації онтологій [17].

Таблиця 2.1 – Розподіл джерел сирої інформації для автоматичного збору

Джерело даних	**Характеристика текстового вмісту**	**Вилучені пари, шт**	**Частка від масиву, %**
OPUS (Open Parallel Corpus)	Локалізація відкритого ПО, субтитри, технічна документація	281215	60,0
WikiMatrix	Автоматично зіставлені речення зі статей багатомовної Вікіпедії	140608	30,0
TED2020	Транскрипції та аматорські переклади публічних виступів і лекцій	46870	10,0
Разом	Початковий невідфільтрований масив даних	468693	100,0

Для оптимізації отриманого масиву було спроєктовано та застосовано каскадний алгоритм фільтрації. Перший (базовий) рівень включав застосування швидких евристик: інструмента FastText для попередньої ідентифікації мови (LangID) з порогом впевненості 0,85, а також фільтра співвідношення довжини, який видаляв пари, де кількість символів одного речення перевищувала інше більше ніж у три рази. Другий (глибинний) рівень полягав у застосуванні нейронної моделі LaBSE (Language-agnostic BERT Sentence Embedding) для обчислення косинусної подібності багатовимірних векторів [18, 19]. Застосування налаштованого конвеєра продемонструвало стрімке скорочення корисного обсягу даних (табл. 2.2).

Таблиця 2.2 – Динаміка скорочення корпусу під час автоматичного фільтрування

Етап обробки даних	**Кількість збережених пар речень**	**Відсоток від початкового обсягу**
Сирий масив даних (Raw mining)	468693	100,0
Після LangID та швидких евристик довжини	50000	10,7
Після семантичного вирівнювання (поріг LaBSE > 0,75)	45283	9,6
Фінальний очищений масив (до верифікації)	32226	6,8

Встановлено, що 89,3% початкового масиву відхиляється вже на базових етапах алгоритмічного очищення. Використання подібних конвеєрів довело свою результативність в інструментаріях інтелектуального прийняття рішень для складних систем [20, 21]. Аналіз структури відхилених даних

показав специфічний розподіл аномалій, притаманних глобальним відкритим корпусам (табл. 2.3).

Таблиця 2.3 – Типологія відхилених даних на базовому рівні фільтрації

Причина відхилення алгоритмом	**Частка від відхилених, %**	**Опис виявленої проблеми**
Невідповідність мові (помилка LangID)	~ 65	Потрапляння хорватської та сербської лексики замість боснійської через значну структурну та лексичну спорідненість.
Порушення евристик довжини	~ 20	Суттєва розбіжність у довжині рядків, що свідчить про системну втрату вирівнювання на рівні абзаців.
Технічне та символьне сміття	~ 15	Наявність рядків без голосних літер, фрагменти програмного коду, залишки HTML-тегів, нерозпарсені URL-адреси.

Незважаючи на застосування жорсткого порогу відсікання векторної моделі LaBSE, у фінальному масиві з 32226 пар було виявлено наявність «паралельного сміття». Це пари речень, які формально отримали високу математичну оцінку подібності векторів, але фактично містять спотворений зміст або грубі морфологічні помилки. Типові приклади таких аномалій наведено у додатку А (табл. А.1).

Отримані експериментальні дані доводять, що традиційний підхід до збору паралельних текстів створює «ілюзію обсягу». Висока математична подібність векторних просторів (ембедінгів) ефективно працює для загальної кластеризації тем, але залишається абсолютно нечутливою до тонкощів слов’янської морфології. Це обґрунтовує необхідність застосування

додаткових етапів семантичного контролю, оскільки навчання нейромережі на таких даних призведе до закріплення хибних граматичних шаблонів.

2.1.2 Аналіз невідповідності метрик реальній якості

Для кількісної перевірки гіпотези про те, що поверхневі та векторні метрики не здатні адекватно оцінити якість перекладу для флективних мов, було проведено додатковий експеримент. Масив даних, який отримав високі бали векторної подібності, було передано на фінальну експертизу алгоритму автоматичного судді на базі великої мовної моделі (LLM-as-a-judge). Для автоматизованого контролю якості інтегровано модель Qwen2.5-1.5B, здатну виконувати завдання нульового пострілу (zero-shot) з високою точністю логічного висновку. Завдання алгоритму полягало у бінарній класифікації (Pass/Fail) на основі перевірки граматичної природності та відсутності спотворень оригінального змісту (табл. 2.4).

Таблиця 2.4 – Статистика відбракування даних на етапі автоматичної LLM-верифікації

Показник алгоритмічного оцінювання	**Кількісне значення**
Загальний обсяг даних до верифікації (з LaBSE > 0,75)	32226 пар
Обсяг валідованих даних (визнані суддею коректними)	26153 пар
Кількість відбракованих пар (містять критичні помилки)	6073 пар
Відсоток хибно позитивних оцінок векторної метрики	18,8%

Під час експерименту зафіксовано масштабне явище метричної невідповідності: майже кожне п’яте речення (18,8%), яке математичні

алгоритми вважали ідеальним еквівалентним перекладом, було відхилено мовною моделлю через спотворення суті. Типові приклади хибно позитивних спрацьовувань векторних метрик наведено у додатку А (табл. А.2).

Спроби вирішити проблему невідповідності метрик за допомогою традиційних інструментів синтаксичного аналізу (бібліотеки stanza, spaCy) виявилися неефективними. Незважаючи на те, що ці алгоритми мають високу точність у визначенні частин мови та побудові дерев залежностей, вони діють у межах однієї мови і не здатні виявляти крос-мовні семантичні зсуви або логічні суперечності [18, 21].

Крім того, перехід до використання генеративних моделей у ролі автоматичного судді має апаратні обмеження. Експерименти довели, що легковагові моделі (з кількістю параметрів до 500 мільйонів) не здатні впоратися з цим завданням через брак внутрішньої пам’яті для утримання складних логічних інструкцій. Тому для ефективного та точного очищення корпусу необхідне залучення моделей класу на 1,5 млрд та більше параметрів.

2.1.3 Порівняння генеративних моделей для синтезу морфології

Враховуючи високий рівень критичних помилок та шуму у відкритих джерелах, було прийнято рішення протестувати стратегію розширення навчальної вибірки за допомогою штучного синтезу даних. Експеримент полягав у порівняльному тестуванні здатності сучасних генеративних моделей суворо дотримуватися структурних інструкцій (Instruction Following) при пакетній генерації.

Моделям надавався ідентичний системний запит (промпт) із жорсткими обмеженнями: довжина згенерованого речення мала складати від 5 до 12 слів, вимагалася обов’язкова наявність активного дієслова, стиль мав

бути розмовним, а використання змішаної лексики (суржику) чи літер інших алфавітів суворо заборонялося. Результати буде наведено у таблиці 2.5.

Таблиця 2.5 – Кількісні результати дотримання інструкцій генеративними моделями

Назва моделі	**Згенеровано тестових пар, шт.**	**Відсоток чистих речень, %**
DeepSeek-Chat	1421	78,0
Claude 3 Haiku	1000	77,0
Gemini 3.0 Flash	1000	76,9
GPT-5	1500	76,3

При генеруванні одиничних прикладів усі протестовані моделі виконували завдання коректно. Проте ключовою проблемою виявилося масштабування. Під час масової безперервної генерації тисяч речень зафіксовано явище деградації контекстного вікна та зниження рівня генеративної уваги (attention allocation), що призводило до системних збоїв. Специфіка цих збоїв залежала від архітектури конкретної моделі (табл. 2.6).

Таблиця 2.6 – Характеристика типових збоїв генеративних моделей при масштабуванні

Назва моделі	**Домінуюча помилка під час масової пакетної генерації**
1	**2**
DeepSeek-Chat	Структурні збої синтаксису: порушення форматування виводу JSON-об’єктів (пропущені дужки або коми).
Claude 3 Haiku	Спрощення синтаксису: систематичне ігнорування нижнього ліміту довжини (генерація речень з 3-4 слів).

Продовження таблиці 2.6

1	2
Gemini 3.0 Flash	Втрата частин мови: тенденція до побудови називних речень без обов'язкового активного дієслова.
GPT-5	Неконтрольоване мовне змішування: ігнорування заборони на змішування мов, масова поява русизмів у тексті.

Встановлено, що структурні збої формату JSON (найбільш притаманні архітектурі DeepSeek) легко виправляються на етапі фінальної обробки за допомогою базових регулярних виразів. Натомість лексичні та синтаксичні спрощення, що генеруються іншими моделями, потребують складної алгоритмічної або експертної ручної перевірки. Тому доцільно розглядати розширені конвеєри та алгоритми геоінформаційної маршрутизації для підвищення структурної якості [22, 23]. З огляду на найвищий показник стабільності та простоту обробки результатів, базовим інструментом для автоматизованого синтезу даних було обрано модель DeepSeek.

2.1.4 Гіпотеза про перевагу комбінованого синтезу даних

Для вирішення фундаментальної проблеми дефіциту морфологічних варіацій в природних текстах було розроблено та обґрунтовано стратегію формування комбінованого навчального масиву. Запропонований підхід базується на поєднанні двох методів: керованої морфологічної аугментації та зворотного перекладу мономовних текстів (Back-Translation). Ця логічна трансформація векторів ознак також поширена у методах опису зображень [24].

Перший етап комбінованої стратегії полягав у морфологічній аугментації. За основу було взято початкове ядро з приблизно 300 високоякісних паралельних речень, які були відібрані та перевірені

експертним шляхом. Оскільки попередні тестування виявили нездатність генеративних моделей безпомилково генерувати всі граматичні форми за один масивний запит, процес генерації було декомпозовано.

Для кожного базового речення з ядра послідовно генерувалося 18 специфічних морфологічних варіацій за попередньо визначеною структурною сіткою, яка охоплювала всі можливі перехресні комбінації ключових граматичних категорій:

– граматичний час (3 варіації): минулий, теперішній, майбутній;
– граматична особа (3 варіації): перша, друга, третя;
– граматичне число (2 варіації): однина, множина.

Сумарно перетин цих категорій формував 18 унікальних граматичних форм (3 × 3 × 2) для кожної базової пари. Теоретичний обсяг згенерованого масиву становив близько 5400 пар. Однак через наявність незначної частки генеративного шуму (галюцинацій), притаманного мовним моделям під час роботи зі складною флективною морфологією, отриманий масив було додатково відфільтровано. У результаті алгоритмічного та ручного очищення було сформовано ідеальне ядро з рівно 5000 аугментованих пар. Принципи структурного пошуку та аналізу потоків дозволяють ефективно працювати з такими згенерованими масивами [25, 26].

Динаміку поетапного масштабування ядра наведено у таблиці 2.7. Для забезпечення параметричної симетрії під час двонаправленого навчання нейромережі, згенерований чистий масив було додатково розгорнуто (дзеркально відображено позиції мовних пар оригіналу та перекладу).

Другий етап формування комбінованого масиву базувався на зворотному перекладі (Back-Translation) мономовних текстів високої якості, вилучених з україномовної та боснійськомовної Вікіпедії. Цей підхід спрямований на подолання проблеми «машинного стилю», який виникає при навчанні виключно на класичних паралельних корпусах.

Таблиця 2.7 – Масштабування ядра набору даних через декомпозицію та аугментацію

Етап обробки або джерело текстових даних	**Кількісне значення**
Базове ядро (експертний відбір)	~ 300 пар
Аугментація через DeepSeek (18 варіацій) та фільтрація шуму	5000 пар
Реверсування напрямку для забезпечення симетрії навчання	5000 пар
Фінальний обсяг морфологічної бази	10000 пар
Мультиплікатор розширення бази	~ 330%

Під час алгоритмічної генерації оригінальні тексти від носіїв мови маркувалися спеціальною контрольною міткою $< high >$, тоді як їхні переклади, згенеровані проміжною моделлю, отримували мітку $< low >$. Така стратегія передбачає, що під час оптимізації ваг нейромережа буде постійно змушена перекладати текст із нижчої якості у вищу, паралельно засвоюючи ідеальну природну синтаксичну структуру цільової мови. Загальну архітектуру та кількісний розподіл сформованого комбінованого масиву наведено у таблиці 2.8.

Таблиця 2.8 – Структура фінального комбінованого навчального масиву

Компонент даних	**Цільове призначення в навчальному масиві**	**Обсяг, пар**
1	**2**	**3**
Аугментовані форми	Засвоєння правил відмінювання, узгодження родів та осіб	10000
ВТ українських текстів	Забезпечення природності синтаксису цільової української мови	40000

Продовження таблиці 2.8

1	2	3
ВТ боснійських текстів	Забезпечення природності синтаксису цільової боснійської мови	40000
Разом	Повний оптимізований комбінований корпус	90 000

2.2 Експериментальна перевірка архітектурних гіпотез

2.2.1 Експеримент Denoising Autoencoder

Першим етапом структурних випробувань стало тестування гіпотези шумопоглинаючого автокодувальника (Denoising Autoencoder). Ідея полягала у створенні монолінгвального адаптера, який навчався виключно відновлювати коректний текст цільовою мовою зі штучно замаскованого або зашумленого тексту, повністю абстрагуючись від процесу двомовного перекладу та не взаємодіючи з текстом оригіналу під час тренування. Після ізольованого навчання такого адаптера було проведено тестування його прямої інтеграції у стандартний цикл перекладу (табл. 2.9, табл. 2.10).

Таблиця 2.9 – Метрики прямого перекладу (bos-ukr) з інтегрованим «Denoising»-адаптером

Архітектура системи перекладу	**BLEU**	**chrF**	**Середній час генерації, с**
Базова модель NLLB-600M	54,21	71,84	1,22
Модель з стандартним двомовним DoRA-адаптером	53,56	72,90	0,48
Модель з монолінгвальним «Denoising»-адаптером	57,42	77,30	1,03

Таблиця 2.10 – Метрики зворотного перекладу (ukr-bos) з інтегрованим «Denoising»-адаптером

Архітектура системи перекладу	**BLEU**	**chrF**	**Середній час генерації, с**
Базова модель NLLB-600M	43,12	70,64	0,45
Модель з стандартним двомовним DoRA-адаптером	59,71	79,26	0,37
Модель з монолінгвальним «Denoising»-адаптером	56,02	75,06	0,81

Зафіксовано вкрай нестабільну поведінку архітектури: в одному напрямку перекладу (з боснійської на українську) метрики з «Denoising»-адаптером об’єктивно зростали, проте у зворотному напрямку показник лексичного збігу BLEU впав до 56,02 (порівняно з 59,71 у разі використання стандартного підходу).

Глибинний аналіз внутрішніх процесів нейромережі (зокрема, розподілу прихованих станів декодера) виявив математичну причину такої деградації. Це зумовлено явищем зсуву розподілу (distribution shift) векторних просторів [27]. Український шумопоглинаючий адаптер оптимізував свої ваги виключно під нормальний розподіл українських лексем та токенів. При прямій інтеграції в цикл перекладу на вхід цього адаптера через декодер починали надходити багатовимірні вектори безпосередньо від боснійського енкодера. Ця крос-мовна просторова невідповідність викликала конфлікт ознак, що змушувало декодер генерувати випадковий граматичний шум. Наприклад, коректна конструкція «Ta mala sela su bila potpuno prazna» перекладалася адаптером як граматично розірвана синтаксема «Ця мала село були повністю порожні».

2.2.2 Порівняння конвеєрних систем та прямої генерації

З метою ізоляції адаптера від крос-мовного впливу та усунення проблеми зсуву розподілу було спроєктовано конвеєрну систему постредагування (Pipeline). Процес було розділено на ізольовані кроки. На першому кроці базова модель генерує чорновий переклад цільовою мовою без втручання адаптерів. На другому кроці в отриманий чорновий переклад алгоритмічно вноситься цілеспрямований шум. Замість випадкового відбору, спеціальним токеном маскувалося близько 25% слів, щодо яких базова модель демонструвала найнижчий рівень впевненості (low confidence score) під час первинної генерації. Водночас виявлено, що через специфіку ймовірнісного розподілу до категорії замаскованих елементів періодично потрапляли лексеми з абсолютно коректним попереднім варіантом перекладу. Після цього підготовлений зашумлений мономовний текст пропускається через цільовий «Denoising»-адаптер для фінального постредагування та виправлення відмінків. Результати порівнянь наведені у таблиці 2.11.

Таблиця 2.11 – Порівняння конвеєрної обробки та наскрізної прямої генерації

Метод генерації (інференсу)	**BLEU**	**chrF**	**Час на речення, с**
Базова NLLB-600M	46,70	73,37	2,14
Пряма наскрізна генерація («End-to-End» з DoRA)	47,14	76,52	1,28
Двохетапний конвеєр постредагування (Pipeline)	37,16	63,65	6,42

Зафіксовано катастрофічне падіння метрики BLEU майже на 10 пунктів порівняно з базовою моделлю, а також збільшення часу виконання генерації

більш ніж у 5 разів. Така часова затримка є технічно обґрунтованою та зумовлена авторегресійною природою архітектури Transformer: замість генерації тексту за один прохід, складність обчислень зросла до $O(2N)$.

Проте найбільш критичною проблемою стала масова поява семантичних галюцинацій під час постредагування. Приклад такого спотворення наведено у таблиці 2.12.

Виявлено, що алгоритм постредагування діяв «наосліп». Не маючи доступу до оригінального боснійського контексту (векторів енкодера), «Denoising»-адаптер відновив прихований токен «[MASK]» випадковим словом «структури». Це сталося через те, що у навчальному корпусі української мови словосполучення «професійні структури» має значно вищу статистичну ймовірність (частотність появи), ніж вузькоспеціалізоване «професійні реставратори». Онлайн-моніторинг та методи структурної класифікації підтверджують важливість динамічного усунення таких зсувів [28, 29].

Таблиця 2.12 – Приклад виникнення галюцинації у конвеєрній системі постредагування

Етап обробки	**Згенерований або вхідний текст**	**Стан ключового слова**
Оригінальний текст (Боснійська)	«lokalne vlasti i stručni restauratori su uspješno»	restauratori
Крок 1: Чорновий переклад	«місцеві органи влади та професійні реставратори»	реставратори
Крок 2: Алгоритмічне зашумлення	«місцеві органи влади та професійні [MASK]»	«[MASK]»
Крок 3: Фінальний вивід адаптера	«місцеві органи влади та професійні структури»	структури (галюцинація)

2.2.3 Дослідження мульти-адаптерних стратегій

Як альтернативу конвеєрним системам було досліджено можливість впровадження модульної архітектури MAD-X («Multiple Adapters for Cross-lingual transfer»). Цей підхід передбачає використання окремого мовного адаптера для енкодера (оброблення боснійської мови), окремого для декодера (генерація українською) та проміжного адаптера завдання для узгодження логіки перекладу. Проте спроба програмної реалізації виявила критичні інженерні перешкоди. Градієнти оптимізуються синхронно для всієї мережі, що унеможливлює математичне відокремлення ваг після навчання, а поточні версії бібліотек (зокрема «peft») не підтримують динамічне перемикання матриць адаптерів усередині одного циклу генерації.

Вирішальним фактором відмови від мульти-адаптерних стратегій та сучасних великих мовних моделей («Decoder-only») стали жорсткі апаратні обмеження. Цільове обладнання мало лише 6 ГБ відеопам’яті (VRAM), з яких близько 1 ГБ постійно резервувалося операційною системою та фоновими процесами, а додатковий обсяг вимагався для розміщення контексту генерації (KV-cache). За таких умов завантаження архітектури MAD-X або масивних моделей (які потребують понад 8 ГБ VRAM) гарантовано призводило до переповнення пам’яті. Забезпечення стабільної продуктивності в таких умовах досягається саме комбінацією методів аналізу та діагностики [30–32]. З огляду на це, використання компактної базової моделі NLLB-600M у поєднанні з єдиним наскрізним DoRA-адаптером було визнано єдиним технічно можливим та ефективним варіантом.

2.2.4 Експеримент із глибиною інтеграції адаптерів

Після відмови від багатокомпонентних архітектур було досліджено вплив цільових модулів самої нейромережі на ефективність засвоєння

морфологічних правил. Традиційний підхід обмежує втручання виключно матрицями механізму уваги («q_proj», «v_proj»). У ході дослідження висунуто гіпотезу щодо застосування стратегії «all-linear» – повної інтеграції адаптерів у всі повнозв'язні лінійні шари багатошарового персептрона (MLP), включаючи матриці «up_proj» та «down_proj». Результати експериментів наведено у таблиці 2.13.

Зафіксовано суттєвий стрибок символьної метрики chrF на майже 9 пунктів (до рівня 79,01). Математично та архітектурно це пояснюється тим, що механізм уваги у моделі Transformer діє переважно як маршрутизатор токенів (визначає синтаксис та порядок слів), тоді як матриці MLP виконують функцію внутрішнього словника, конвертуючи приховані ознаки у конкретний лексичний розподіл.

Таблиця 2.13 – Вплив глибини інтеграції адаптерів на метрики якості (ukr-bos)

Цільові модулі інтеграції адаптерів	**BLEU**	**chrF**	**COMET**
Base NLLB-600M	46,88	70,38	95,83
Тільки механізм уваги («Attention only»)	51,63	74,01	95,45
Усі лінійні шари (стратегія «all-linear»)	60,17	79,01	96,33

Зміна закінчення у слов'янських мовах (наприклад, «великого» на «великому») означає генерацію абсолютно іншого токена. Без доступу до MLP-шарів модель фізично не мала параметричної здатності запам'ятати складні таблиці відмінювання. Для якісного розуміння математичних метрик було проведено аналіз лінгвістичної поведінки моделей на специфічних конструкціях, який наведено у додатку А (табл. А.3).

Інтеграція адаптерів у всі повнозв'язні шари не призвела до вибуху градієнтів завдяки використанню алгоритму DoRA. Цей метод математично декомпозує матрицю ваг на незалежні компоненти напрямку та амплітуди, що забезпечує стабільну нормалізацію процесу оновлення масивних матриць.

Водночас дослідження виявило межі параметричної ємності моделі. Незважаючи на успіх стратегії «all-linear», обсяг у 600 мільйонів параметрів залишається фізичним обмеженням. Аналіз показав, що архітектура продовжує стикатися з проблемами при крос-мовних розбіжностях роду слів. Наприклад, боснійське слово «olovka» (жіночий рід) та українське «олівець» (чоловічий рід) призводять до того, що модель намагається застосувати українські чоловічі закінчення до боснійських коренів, генеруючи неіснуюче слово «olovak». Це свідчить про те, що базова модель досягла своєї параметричної межі щодо глибини розуміння крос-мовних словників.

2.3 Обґрунтування остаточного вибору архітектурного рішення

2.3.1 Порівняння результатів проведених експериментів

Для забезпечення об’єктивності результатів, автоматичне та ручне оцінювання якості перекладу проводилося на спеціально сформованому контрольному наборі даних. Цей набір слугував інструментом критичного оцінювання і складався з 10 комплексних пар речень та додаткового стрес-тесту з 4 складнопідрядних конструкцій. Вибірка фокусувалася на перевірці найскладніших випадків слов’янського синтаксичного узгодження: зміні граматичного роду, правильному використанні енклітик та специфічному керуванні дієслів.

Щоб математично довести перевагу обраної гібридної стратегії, було проведено порівняння трьох ключових ітерацій еволюції нейромережі:

– базова модель (Base NLLB-600M) – без модифікацій та донавчання;

– проміжна модель (DoRA 62k) – навчена виключно на зібраних «сирих» парах з мережі Інтернет;

– фінальна гібридна модель (DoRA 90k BT) – навчена за стратегією «all-linear» на гібридному наборі даних (синтез + зворотний переклад).

Для оцінювання застосовувалися метрики: BLEU (загальний лексичний збіг), chrF (точність морфології на рівні символьних *n*-грам), BERT F1 (векторна оцінка токенів) та COMET (нейронна оцінка збереження глибокої семантики). Результати зведено у таблиці 2.14 та таблиці 2.15.

Таблиця 2.14 – Порівняння еволюції моделей (напрямок bos-ukr)

Етап еволюції архітектури	**BLEU**	**chrF**	**BERT F1**	**COMET**
1. Base NLLB-600M	52,88	73,33	94,58	96,23
2. DoRA 62k	57,41	77,17	95,12	96,68
3. DoRA 90k BT	58,06	79,41	95,25	96,77

Таблиця 2.15 – Порівняння еволюції моделей (напрямок ukr-bos)

Етап еволюції архітектури	**BLEU**	**chrF**	**BERT F1**	**COMET**
1. Base NLLB-600M	46,88	70,38	92,79	95,83
2. DoRA 62k	56,52	77,50	95,54	95,82
3. DoRA 90k BT	60,17	79,01	94,90	96,33

Детальний аналіз метрик свідчить, що перехід від базової моделі до навчання на очищених інтернет-парах (DoRA 62k) дає швидкий початковий приріст якості, але швидко досягає ліміту: точність морфології (chrF) зростає повільно. Застосування фінальної архітектури («DoRA 90k BT») демонструє абсолютну перевагу. Особливо показовим є безпрецедентний стрибок метрики chrF до 79,01 у напрямку «ukr-bos» (на майже 9 пунктів вище за базову). Це доводить, що застосування синтетичних парадигм змусило модель не просто підбирати схожі слова, а коректно їх відмінювати. Кількісне зростання метрик прямо підтверджується якісним вирішенням специфічних лінгвістичних проблем, зафіксованих під час тестування наведеного у додатку А (табл. А.4).

2.3.2 Вибір фінальної стратегії та моделі

На основі проведених комплексних експериментів із даними та архітектурами було сформовано та науково обґрунтовано остаточний вибір технологічного стеку для вирішення задачі нейронного машинного перекладу (табл. 2.16).

Відмова від сучасних великих мовних моделей (Decoder-only LLM) на користь NLLB-600M зумовлена жорсткими апаратними обмеженнями (моделі від 4 мільярдів параметрів вимагають понад 8-16 ГБ відеопам’яті) та схильністю таких архітектур до генерації суржику при роботі з низькоресурсними мовами.

Таблиця 2.16 – Специфікація фінального архітектурного рішення

Компонент системи	**Обране рішення**	**Технічне або наукове обґрунтування**
Базова архітектура	NLLB-600M (Encoder-Decoder)	Стійкість до міжмовного змішування (на відміну від малих LLM); мінімальні вимоги до VRAM; висока швидкість генерації.
Метод адаптації	Наскрізний DoRA-адаптер	Розділення ваг гарантує стабільність навчання; ізольований наскрізний прохід швидший і надійніший за конвеєр (Pipeline).
Глибина підключення	Усі лінійні шари (all-linear)	Механізм уваги керує синтаксисом, тоді як MLP-шари містять «словник». Доступ до MLP є обов’язковим для флективних мов.
Навчальний масив	Комбінований (BT та Аугментація)	Зворотний переклад забезпечує природність синтаксису; морфологічний синтез через генеративні моделі усуває дефіцит форм відмінювання.

Аналіз продуктивності та застосування адаптивних матричних мереж сприяють прийняттю оптимальних рішень в умовах лімітованих ресурсів [30, 31]. Модель NLLB, натренована на 200 мовах із чітким розділенням лексичних кордонів, слугує найкращим фундаментом [32].

Застосування стратегії «all-linear» стало можливим виключно завдяки методу DoRA, який розкладає ваги на амплітуду та напрямок, запобігаючи градієнтним вибухам та катастрофічному забуванню. Це дозволило моделі глибоко редагувати свої знання про морфологію на кожному рівні енкодера та декодера. Використання зворотного перекладу (Back-Translation) текстів з Вікіпедії усунуло «машинний» стиль, дозволивши нейромережі виступати в ролі шумопоглинаючого фільтра та вчитися на природній граматиці.

Надійність обраної гібридної архітектури підтверджується відстеженням динаміки функції втрат (Loss) під час фінального навчання. Тренування проводилося з параметрами швидкості навчання (Learning Rate) $2 \cdot 10^{-4}$ та розміром пакету (Batch Size) 64 протягом трьох епох (табл. 2.17).

Таблиця 2.17 – Динаміка метрик «Loss» під час фінального навчання

Етап оптимізаційного процесу	**Значення помилки**	**Характеристика процесу**
Initial Loss (початкові втрати)	1,84	Стан неадаптованої мережі до оптимізації
Final Train Loss (навчальні втрати)	1,05	Плавна конвергенція без різких стрибків
Final Eval Loss (втрати на валідації)	0,98	Показник близький до 1,0 свідчить про відсутність перенавчання

Стрімке та плавне падіння функції втрат, відсутність аномальних стрибків та досягнення низького показника на тестовій вибірці математично підтверджують, що архітектура є збалансованою. Модель не перенавчилася (overfitting) на штучних синтетичних шаблоних, а успішно та стабільно

засвоїла складний слов’янський лінгвістичний розподіл. Поєднання моделі NLLB-600M, повношарової адаптації DoRA та гібридного набору даних є найбільш обґрунтованим рішенням для двонаправленого перекладу між досліджуваними мовами.

2.4 Проєктування та моделювання застосунку

2.4.1 Об’єктно-орієнтоване моделювання

Основою для проєктування будь-якого програмного забезпечення є формування чітких вимог. Виходячи з результатів попередніх експериментів та мети роботи, було сформовано перелік функціональних та нефункціональних вимог до системи машинного перекладу.

Функціональні вимоги визначають безпосередню поведінку системи та ті дії, які користувач може виконати за допомогою розробленого інтерфейсу. Перелік функціональних вимог наведено у додатку А (табл. А.5).

Нефункціональні вимоги описують атрибути якості системи, такі як продуктивність, безпека та апаратні обмеження, що є критичними для розгортання моделі нейромережі. Їх специфікацію наведено у додатку А (табл. А.6).

Для наочної візуалізації сценаріїв розроблено UML-діаграму прецедентів, яка відображає межі системи та зв’язки між акторами (рис. 2.1).

Рисунок 2.1 – UML-діаграма прецедентів системи машинного перекладу

Об’єктно-орієнтоване моделювання (ООМ) дозволяє декомпозувати процес перекладу на зрозумілі та ізольовані сценарії взаємодії між зовнішніми акторами та внутрішніми сутностями системи. Основні сценарії взаємодії (прецеденти) між цими акторами специфіковано у таблиці 2.18.

Таблиця 2.18 – Специфікація основних прецедентів системи

Назва прецеденту	**Залучені актори**	**Механізм виконання**
Введення та спрямування	Користувач	Вибір напрямку перекладу та введення вихідного тексту або ініціація завантаження цільового документа.
Потокова генерація	Користувач, Модуль перекладу	Синхронна генерація тексту з поверненням результату частинами («chunk-by-chunk») для ефекту поступового друкування.
Фоновий переклад файлу	Користувач, Модуль перекладу, Файлова підсистема	Асинхронна фонове оброблення великих документів із поділом на атомарні речення та відображенням прогресу.
Реконструкція розмітки	Файлова підсистема	Ізоляція візуальних стилів шрифту перед генерацією та їх точна математична розмітка після перекладу тексту.

2.4.2 Схема архітектури та взаємодії компонентів

Застосунок проєктується на базі клієнт-серверної архітектури з п’ятьма ізольованими компонентами. «Інтерфейс клієнта» забезпечує взаємодію з користувачем, а «Шлюз API» виконує перевірку та маршрутизацію запитів.

«Процесор документів» відповідає за сегментацію файлів і реконструкцію візуальних стилів. «Менеджер черг» балансує навантаження на графічний процесор, утримуючи важкі фонові завдання в очікуванні під час надходження пріоритетних потокових запитів. Уся взаємодія з моделлю NLLB-600M та адаптерами DoRA інкапсульована у «Фабриці трансляторів», що гарантує незалежність бізнес-логіки від нейромережевого ядра.

Життєвий цикл запиту починається на шлюзі API, який направляє дані до процесора документів (у випадку файла) або напряму у вигляді тексту. Атомарні задачі надходять до менеджера черг, який після перевірки відеопам’яті ініціює авторегресійну генерацію у фабриці трансляторів. Результати повертаються ініціатору: тексти відправляються потоково для забезпечення ефекту поступового друкування, а перекладені фрагменти акумулюються процесором для збірки готового документа. Схему взаємодії відображено на рисунку 2.2.

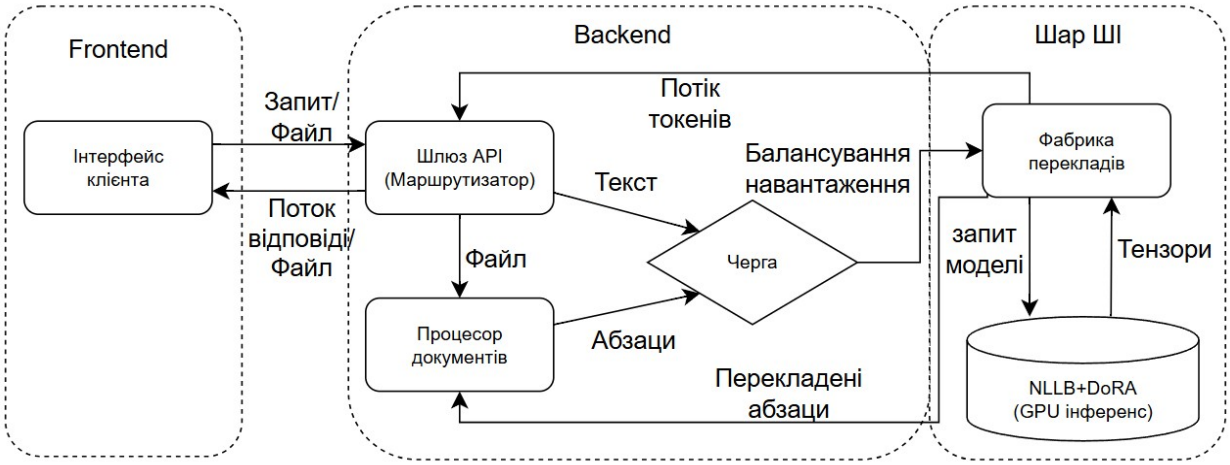


Рисунок 2.2 – Високорівнева схема архітектурної взаємодії компонентів

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Цей розділ кваліфікаційної роботи присвячено програмній реалізації запропонованої системи нейронного машинного перекладу. Розділ охоплює обґрунтування вибору інструментальних засобів, детальний опис потоків даних під час формування гібридного навчального набору, алгоритми інтеграції архітектури параметрично ефективного донавчання, розроблення програмного забезпечення серверної частини (бекенду) та клієнтського інтерфейсу, а також фінальне тестування і апробацію отриманих результатів.

3.1 Інструментальні засоби та загальна архітектура

На етапі проєктування складної інформаційної системи критично важливим є правильний вибір сукупності технологій, які повинні не лише відповідати функціональним вимогам застосунку, але й враховувати жорсткі апаратні обмеження. У цьому підрозділі наведено глибокий аналіз та обґрунтування обраних мов програмування, інструментів глибокого навчання та протоколів передачі даних.

3.1.1 Аналіз мов програмування та середовищ

Для реалізації обчислювального ядра, конвеєра підготовки даних та серверної частини розроблюваного застосунку було обрано мову програмування Python (версії 3.10+). Цей вибір обґрунтовується статусом Python як беззаперечного галузевого стандарту у сфері машинного навчання (Machine Learning) та обробки природної мови (NLP). Мова має найширшу екосистему оптимізованих бібліотек для роботи з багатовимірними масивами

(тензорами) та глибокими нейромережами, що дозволяє делегувати складні низькорівневі обчислення оптимізованим C++ модулям.

Розроблення алгоритмів, налагодження коду та тривале стрес-тестування архітектури системи здійснювалося на локальній робочій станції, оснащеній графічним прискорювачем NVIDIA RTX 3060M із 6 ГБ відеопам’яті (VRAM). Цей апаратний ліміт став ключовим чинником при виборі методів оптимізації та квантування. Однак фінальний прогін моделі, оптимізація гіперпараметрів та безпосереднє розгортання виконувалися на більш потужному обладнанні – графічному прискорювачі NVIDIA RTX 4070 (12 ГБ VRAM). Перехід на мікроархітектуру «Ada Lovelace» дозволив зберегти розраховані оптимальні параметри пакетної обробки (Batch Size), водночас суттєво пришвидшивши тензорні обчислення завдяки новішим тензорним ядрам четвертого покоління.

3.1.2 Вибір інструментів для навчання та роботи моделі

Базовим інструментом для роботи з нейромережами обрано PyTorch. На відміну від статичних графів обчислень у конкурентних рішеннях (наприклад, TensorFlow), PyTorch використовує динамічні обчислювальні графи, що забезпечує гнучкий низькорівневий контроль над градієнтами та ефективне управління пам’яттю GPU за допомогою апаратної архітектури CUDA. Для завантаження базової моделі NLLB-600M та управління процесом адаптації використано екосистему HuggingFace, зокрема бібліотеки «transformers» та «peft» (Parameter-Efficient Fine-Tuning).

Окремим і найбільш критичним завданням став вибір рушія для роботи нейромережі, тобто для процесу генерації перекладу у робочому (клієнтському) режимі. Хоча стандартна бібліотека «transformers» забезпечує повний функціонал, її авторегресійна генерація на споживчому обладнанні створює надмірне навантаження на VRAM та процесорний час. З огляду на

це, було проведено порівняльний аналіз доступних рушіїв генерації, результати якого подано у таблиці 3.1.

Таблиця 3.1 – Порівняння підходів до інференсу моделі NLLB-600M

Бібліотека / Обчислювальний рушій	**Формат збереження тензорних ваг**	**VRAM**	**Відносна швидкість генерації**	**Апаратна підтримка та оптимізація**
transformers (HuggingFace)	float32 (FP32)	~ 2,8 ГБ	1,0x	CPU / GPU (високі вимоги до пам’яті)
transformers (HuggingFace)	bfloat16 (BF16)	~ 1,5 ГБ	1,2x	GPU (сучасні архітектури Ampere/Ada)
CTranslate2 (OpenNMT)	int8 (Квантування)	~ 800 МБ	~ 3,5x - 4,0x	Оптимізовано для споживчих GPU/CPU

Як видно з наведених даних (табл. 3.1), спеціалізовану бібліотеку CTranslate2 було обрано як фінальний обчислювальний рушій. Використання математичного лінійного квантування ваг до формату INT8 дозволило зменшити споживання відеопам’яті до 800 МБ, забезпечивши приріст швидкості майже у 4 рази. Це звільнило критично важливі апаратні ресурси для можливості паралельної обробки кількох клієнтських запитів. Сутнісні принципи розробки механізмів грануляції та штучного інтелекту дозволяють досягати високого рівня компресії [33, 34].

3.1.3 Архітектура клієнт-серверної взаємодії

Для розроблення серверної частини (бекенду) обрано інструмент FastAPI. Його асинхронна природа, побудована на базі стандарту ASGI (Asynchronous Server Gateway Interface), ідеально підходить для управління неблокуючими потоками введення та виведення під час тривалих обчислень нейромережі.

Для забезпечення ефективного зворотного зв’язку з користувачем під час генерації тексту виникла потреба в надійному протоколі передачі даних. Порівняльний аналіз доступних мережевих протоколів наведено у таблиці 3.2.

Таблиця 3.2 – Порівняння протоколів передачі потокових даних

Характеристика	**WebSockets**	**Server-Sent Events**	**Long Polling**
Напрямок зв’язку	Двонаправлений (Full-duplex)	Однонаправлений (Server – Client)	Однонаправлений (Client – Server)
Специфікація стану	Stateful (високе навантаження)	Stateless (низьке навантаження)	Stateless
Оброблення розривів	Потребує ручної програмної реалізації	Вбудовано у стандарт сучасних браузерів	Вбудовано, ресурсоємно
Вибір для проєкту	Відхилено (надлишковий функціонал)	Обрано (ідеально для стрімінгу токенів)	Відхилено

На основі аналізу (табл. 3.2) технологію Server-Sent Events (SSE) було обрано як найефективнішу для однонаправленого стрімінгу згенерованих слів від нейромережі до клієнта. Клієнтський інтерфейс розроблено на базі

бібліотеки React 18 з використанням мови TypeScript для строгої типізації даних. Стилізація візуальних компонентів виконана за допомогою утилітарного фреймворку Tailwind CSS. Взаємодію між основними компонентами системи відображено на діаграмі компонентів (рис. 3.1).

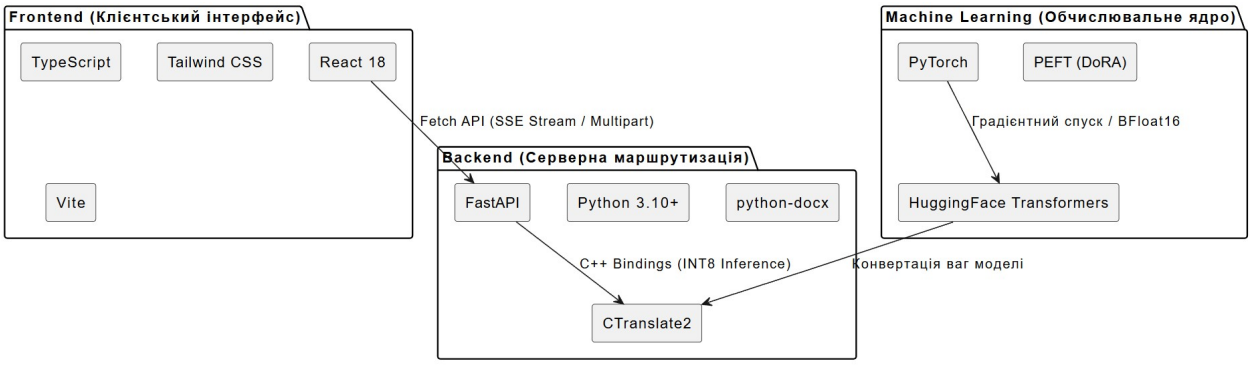


Рисунок 3.1 – Високорівневий технологічний стек та компоненти системи

Фізичне розгортання розробленої системи передбачає розміщення ресурсоємних обчислювальних вузлів (процесорів документів та тензорного рушія) на виділеному обладнанні, тоді як взаємодія здійснюється через легковагий веббраузер клієнта. Прийняття рішень у клієнт-серверних архітектурах спирається на інтелектуальні технології керування технологічними комплексами [35–37]. Цю архітектуру відображено на діаграмі розгортання (рис. 3.2).

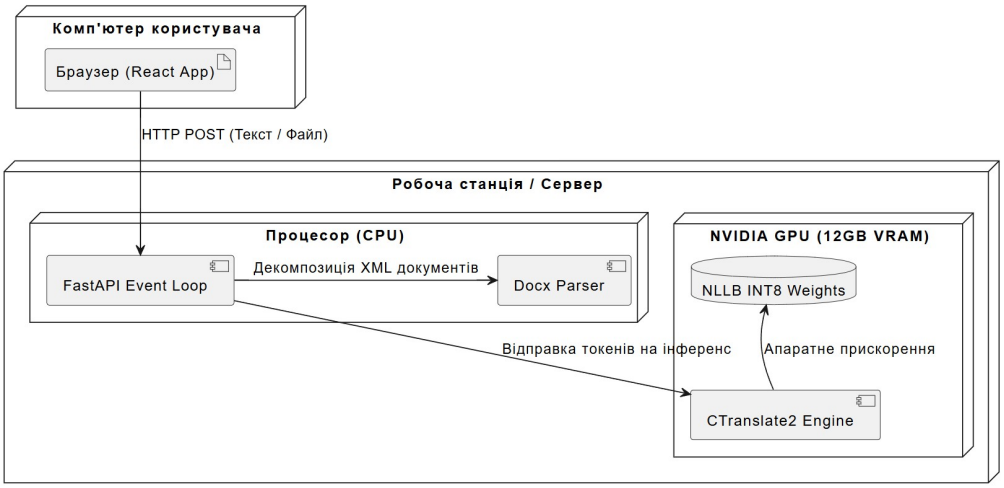


Рисунок 3.2 – Фізичне розгортання та розподіл обчислювальних потоків

Для наочної демонстрації налаштованого робочого середовища на рисунку 3.3 наведено стан серверних процесів та використання відеопам’яті під час ініціалізації обчислювального рушія.

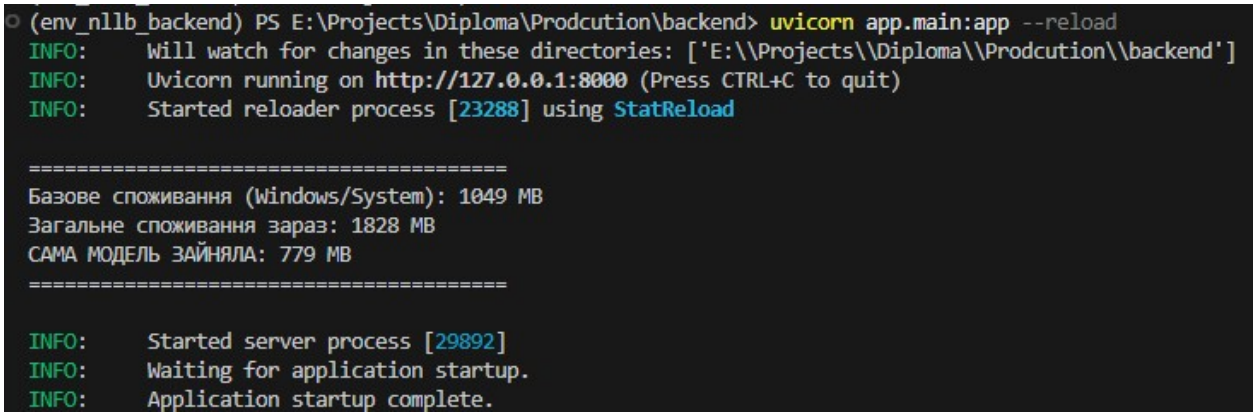


Рисунок 3.3 – Запуск серверу та відображення зайнятої пам’яті

3.2 Підготовка даних та реалізація конвеєра навчання

3.2.1 Архітектура потоку гібридних даних

На основі експериментальних досліджень (підрозділ 2.1) було прийнято рішення про відмову від використання «сирих» відкритих корпусів на користь гібридного масиву. Загальний обсяг фінального керованого набору даних склав 92000 унікальних пар речень [38]. Специфікацію сегментів набору наведено у таблиці 3.3.

Таблиця 3.3 – Специфікація фінального гібридного набору даних

Сегмент набору даних	**Джерело інформації**	**Обсяг**	**Спеціальний тег (Prompt)**	**Мета сегмента в навчанні**
1	**2**	**3**	**4**	**5**
Золоте ядро	Перевірені генерації	10000	quality: high |	Навчання ідеальній морфології

Продовження таблиці 3.3

1	2	3	4	5
Зворотний переклад (BT)	Проміжна модель NLLB-DoRA	80000	quality: low/high |	Засвоєння широкого синтаксису (40 000 Bos + 40 000 Ukr).
Завдання ідентичності	Власні назви, Математичні формули	2000	quality: high |	Запобігання помилковому перекладу імен власних (Identity Task).

Логіку формування цього масиву та послідовність взаємодії між розробленими Python-модулями візуалізовано на діаграмі послідовностей (рис. 3.4).

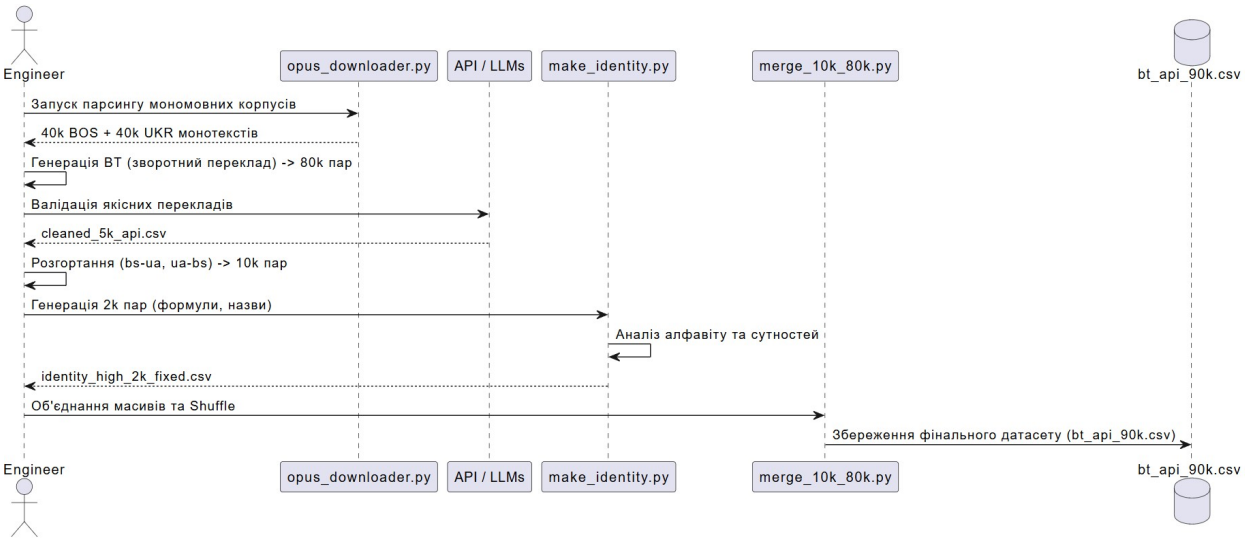


Рисунок 3.4 – Послідовність виконання модулів під час генерації набору даних

3.2.2 Реалізація створення та перевірки даних

Розглянемо детальну програмну реалізацію ключових етапів. Важливим кроком стало створення сегмента «Завдання ідентичності» (Identity Task), який запобігає перекладу імен власних або специфічних абревіатур. Базова багатомовна модель NLLB має тенденцію до надмірної адаптації англійських чи балканських назв під українську фонетику, що є критичною помилкою при обробці офіційної документації. Для вирішення цього було розроблено алгоритм автоматичного визначення цільової мови для збереження ідентичності (лістинг 3.1).

Лістинг 3.1 Алгоритм автоматичного визначення напрямку для збереження ідентичності:

```
def detect_direction(text):
    text = str(text)
    # Перевірка на наявність специфічних кириличних символів
    if re.search(r'[а-яА-ЯіїєґІЇЄҐ]', text):
        return "idt-ua"
    # Перевірка на наявність латиниці та специфічних балканських символів
    if re.search(r'[a-zA-ZčćđšžČĆĐŠŽ]', text):
        return "idt-bs"
    # Базове резервне правило для математичних формул та цифр
    return "idt-bs"
```

Наймасштабнішим та найбільш ресурсоємним етапом стала генерація 80000 пар зворотного перекладу (Back-Translation). Логіку роботи багатопотокового скрипта, що керував процесом масової генерації з використанням апаратного прискорення, наведено на діаграмі діяльності (рис. 3.5).

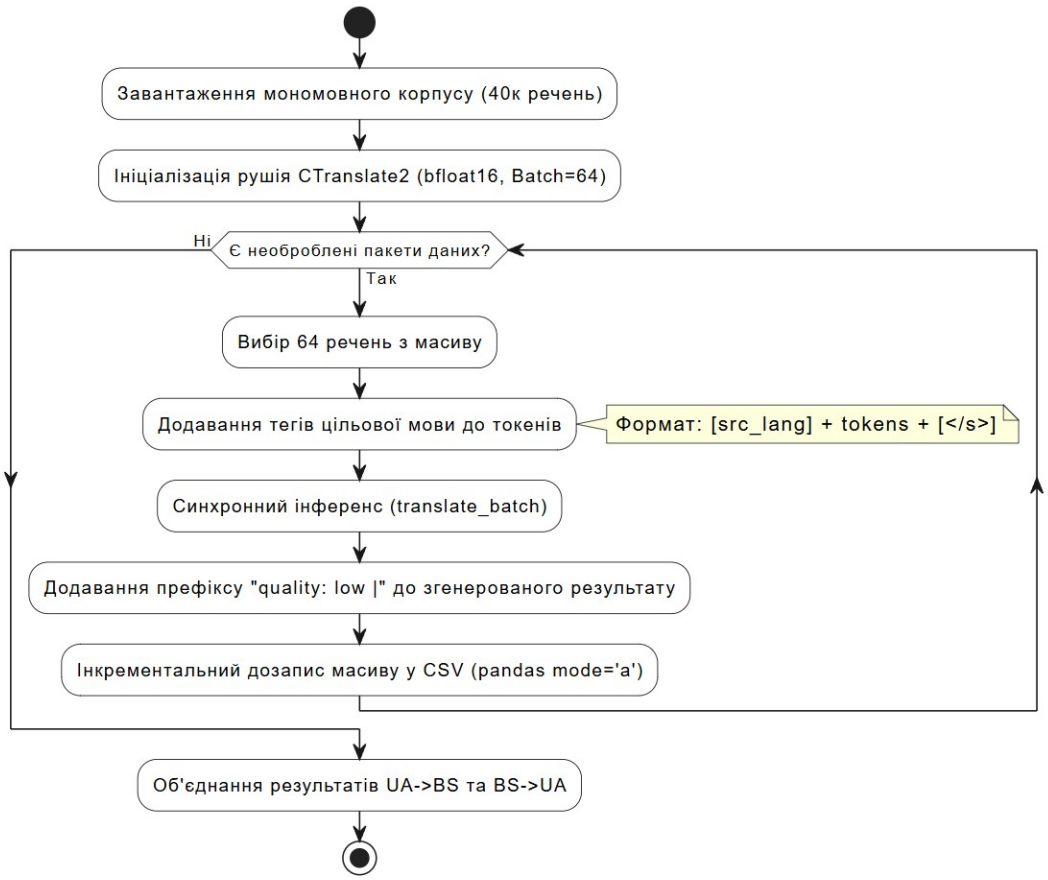


Рисунок 3.5 – Процес масової генерації зворотного перекладу

Під час масової генерації зворотного перекладу було виявлено серйозну технічну проблему: стандартна високорівнева функція токенізатора іноді алгоритмічно помилялася у визначенні вихідної мови для дуже коротких речень (складених з 1-2 слів). Щоб усунути це у визначеному просторі ознак, було розроблено метод process_chunk (лістинг 3.2), який вручну формує низькорівневу структуру токенів, жорстко форсуючи потрібну мову для обчислювального рушія.

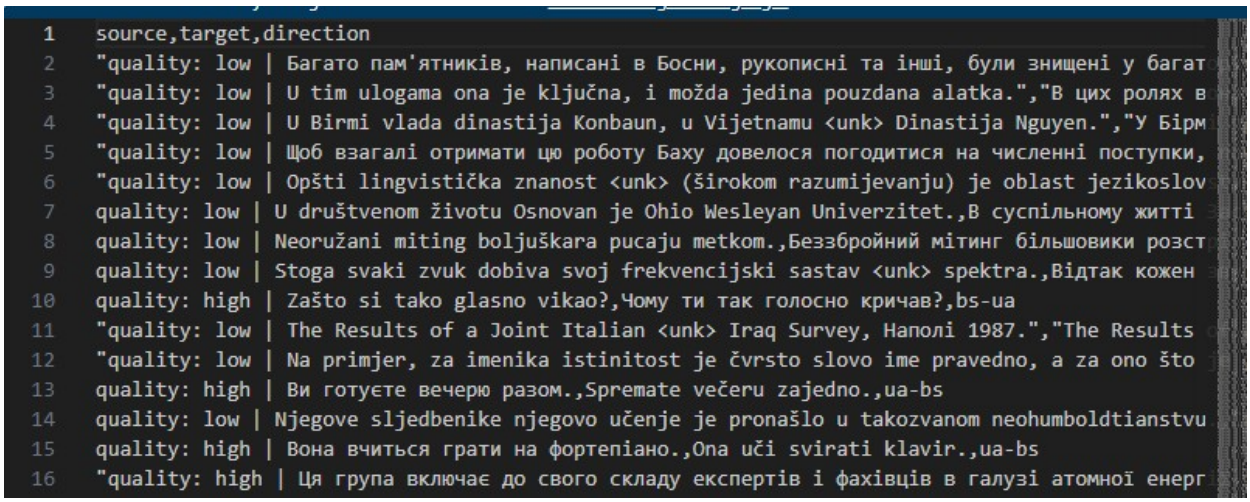
Лістинг 3.2 Запобігання помилкам токенізатора при масовій пакетній генерації:

```
def process_chunk(texts, translator, tokenizer, src_lang, tgt_lang):
    tokenized = []
    for t in texts:
        tokenizer.src_lang = src_lang
```

```
        # РУЧНИЙ МЕТОД: гарантуємо правильну структуру токенів без автовизначення
        pure_tokens = tokenizer.tokenize(t)
        # Додавання обов'язкових службових токенів початку та кінця послідовності
        source_tokens = [src_lang] + pure_tokens + ["</s>"]
        tokenized.append(source_tokens)

    # Пакетна генерація з оптимізованими параметрами променевого пошуку
    results = translator.translate_batch(
        tokenized,
        target_prefix=[[tgt_lang]] * len(texts),
        beam_size=4,
        repetition_penalty=1.2
    )
    # Декодування згенерованих ідентифікаторів назад у текст
    return [tokenizer.convert_tokens_to_string(r.hypotheses[0][1:]) for r in results]
```

Для візуального підтвердження коректності формування даних, на рисунку 3.6 наведено фрагмент згенерованого масиву даних, де чітко видно розрізнення між ідеальними парами (маркованими тегом quality: high |) та масивом зворотного перекладу (маркованим тегом quality: low |).



```
source,target,direction
"quality: low | Багато пам'ятників, написані в Босни, рукописні та інші, були знищені у багат
"quality: low | U tim ulogama ona je ključna, i možda jedina pouzdana alatka.","В цих ролях в
"quality: low | U Birmi vlada dinastija Konbaun, u Vijetnamu <unk> Dinastija Nguyen.","У Бірм
"quality: low | Щоб взагалі отримати цю роботу Баху довелося погодитися на численні поступки,
"quality: low | Opšti lingvistička znanost <unk> (širokom razumijevanju) je oblast jezikoslov
quality: low | U društvenom životu Osnovan je Ohio Wesleyan Univerzitet.,В суспільному житті
quality: low | Neoružani miting boljuškara pucaju metkom.,Беззбройний мітинг більшовики розст
quality: low | Stoga svaki zvuk dobiva svoj frekvencijski sastav <unk> spektra.,Відтак кожен
quality: high | Zašto si tako glasno vikao?,Чому ти так голосно кричав?,bs-ua
"quality: low | The Results of a Joint Italian <unk> Iraq Survey, Наполі 1987.","The Results
"quality: low | Na primjer, za imenika istinitost je čvrsto slovo ime pravedno, a za ono što
quality: high | Ви готуєте вечерю разом.,Spremate večeru zajedno.,ua-bs
quality: low | Njegove sljedbenike njegovo učenje je pronašlo u takozvanom neohumboldtianstvu
quality: high | Вона вчиться грати на фортепіано.,Ona uči svirati klavir.,ua-bs
"quality: high | Ця група включає до свого складу експертів і фахівців в галузі атомної енерг
```

Рисунок 3.6 – Приклад файлу для навчання нейромережі з мітками

3.2.3 Навчання та оптимізація процесу

Після успішного злиття всіх текстових сегментів розпочався процес наскрізного навчання нейромережі. Для уникнення критичної помилки переповнення пам’яті відеокарти (відомої як «CUDA Out of Memory») було здійснено тонке ручне налаштування гіперпараметрів навчання у конфігураторі HuggingFace. Перелік обраних параметрів та інженерне обґрунтування їх використання наведено у таблиці 3.4.

Таблиця 3.4 – Конфігурація гіперпараметрів навчання для обмежених ресурсів VRAM

Параметр конфігурації	**Значення**	**Інженерне обґрунтування вибору**
1	**2**	**3**
torch_dtype	bfloat16	Зменшує розмір матриць ваг вдвічі, зберігаючи необхідний динамічний діапазон чисел.
gradient_checkpointing	True	Видаляє проміжні активації з VRAM, вивільняючи близько 30%.

Продовження таблиці 3.4

1	2	3
per_device_train_batch_size	32	Визначено експериментально як максимальний розмір пакета для 12 ГБ VRAM.
gradient_accumulation_steps	2	Віртуально імітує більший розмір пакета (32 × 2 = 64) для забезпечення стабільності алгоритму.
target_modules	all-linear	Наказує алгоритму інтегрувати матриці у всі лінійні повнозв'язні шари мережі (MLP).

Незважаючи на програмну оптимізацію та використання графічного прискорювача NVIDIA RTX 4070, тривале авторегресійне навчання на масиві з понад 90 000 пар викликало критичне нагрівання апаратного забезпечення (понад 85 °C). Для забезпечення апаратної безпеки та запобігання зниженню частот (тепловому тротлінгу) було розроблено спеціальний клас-обгортку (лістинг 3.3). Цей алгоритм програмно інтегрується у внутрішній цикл бібліотеки HuggingFace та призупиняє обчислення для активного охолодження чипа.

Лістинг 3.3 Алгоритм програмної терморегуляції відеокарти:

```
class AdvancedCallback(TrainerCallback):
    def on_step_end(self, args, state, control, **kwargs):
        # Програмна пауза після кожного кроку оптимізації ваг
        time.sleep(current_sleep_time)
```

Завдяки застосуванню описаних методів оптимізації та контролю температури, процес машинного навчання пройшов без апаратних збоїв. Динаміка функції втрат (Loss) показала плавну математичну збіжність, що

підтверджується візуалізацією графіка втрат на тестовій та тренувальній вибірках (рис. 3.7). Ця стабільність корелює з результатами моделювання геоінформаційних структур та машинного перекладу рідкісних словоформ [39, 40].

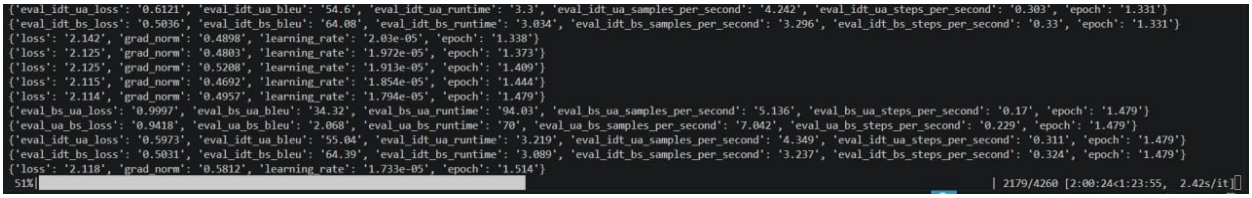


Рисунок 3.7 – Фрагмент тренування моделі, train_loss впав з 17 до 2,0 на середині навчання, і потім дійшов значення 1,3

3.3 Серверна частина для перекладу

Цей підрозділ присвячено детальній програмній реалізації серверної частини застосунку (бекенду), яка виконує роль прошарку між клієнтським вебінтерфейсом та оптимізованим обчислювальним рушієм нейромережі.

3.3.1 Налаштування нейромережі

Навчена гібридна модель, разом з інтегрованими вагами адаптерів DoRA, початково зберігалася у масивному форматі HuggingFace (тензори з плаваючою комою одинарної точності – FP32). Для забезпечення ефективного та швидкого використання у вебзастосунку в умовах обмежених ресурсів її необхідно було конвертувати в оптимізований формат (кваниування до INT8) для рушія CTranslate2. Таке вдосконалення архітектур доводить життєздатність застосування компресійних алгоритмів і методів просторового опису [41, 42]. Цей архітектурний перехід та взаємодію відображено на діаграмі послідовностей (рис. 3.8).

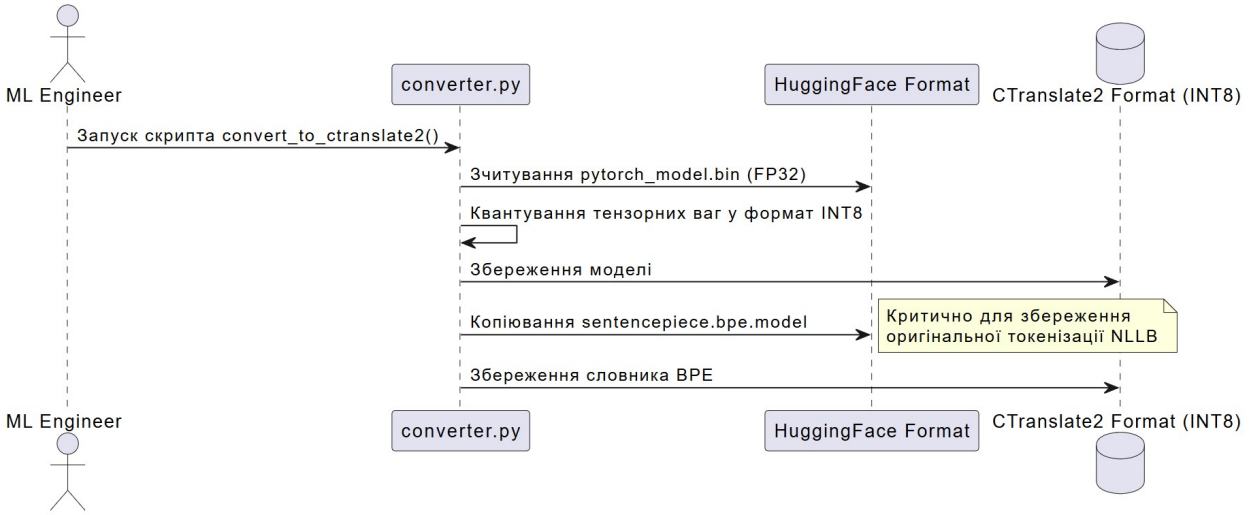


Рисунок 3.8 – Процес конвертації та квантування ваг моделі

Програмну реалізацію алгоритму лінійного квантування наведено в лістингу 3.4. Обов'язковим та критичним кроком є пряме копіювання файлу словника sentencepiece.bpe.model. Без нього рушій CTranslate2 втратить здатність правильно ділити незнайомі слова на підтокени.

Лістинг 3.4 Алгоритм конвертації та квантування моделі:

```
# Ініціалізація конвертера з обов'язковим копіюванням BPE моделі
converter = ctranslate2.converters.TransformersConverter(
    INPUT_MODEL_DIR,
    copy_files=["sentencepiece.bpe.model"]
)

# Виконання конвертації з HuggingFace у формат CTranslate2 та квантування
converter.convert(
    output_dir=OUTPUT_DIR,
    quantization="int8",
    force=True
)
```

3.3.2 Об’єктна структура та програмний інтерфейс

Архітектуру серверної частини спроєктовано навколо концепції ізольованих менеджерів, кожен з яких відповідає виключно за свою предметну область. Для забезпечення комунікації з клієнтським застосунком розроблено набір REST API маршрутів. Специфікацію розроблених ендпоінтів наведено у таблиці 3.5.

Таблиця 3.5 – Специфікація API ендпоінтів серверної частини

Ендпоінт (URL)	**Метод HTTP**	**Призначення програмного обробника**	**Формат Відповіді**
/health	GET	Контроль апаратних ресурсів GPU/CPU та загального статусу сервера.	JSON об’єкт
/translate/stream	POST	Приймає текстове тіло запиту та ініціює потоковий авторегресійний переклад.	SSE потік
/translate/file/stream	POST	Асинхронне фонове оброблення, розбиття на сегменти та переклад файлів.	SSE події
/translate/file/result	GET	Надання безпечного доступу до скачування реконструйованого бінарного файлу.	Binary Blob

Взаємодію ключових об’єктно-орієнтованих класів, які інкапсулюють бізнес-логіку обробки цих мережевих запитів, подано на UML-діаграмі класів (рис. 3.9).

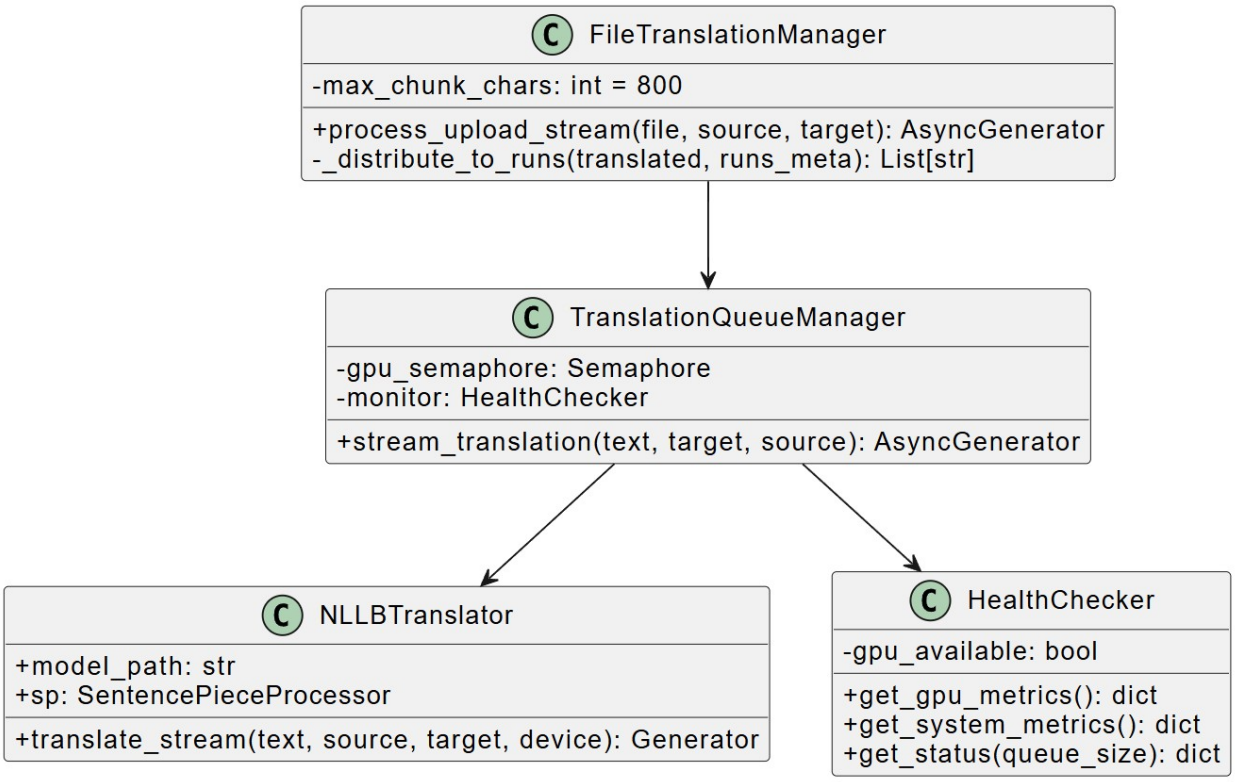


Рисунок 3.9 – Об’єктна структура основних модулів серверної частини

Головним захисним елементом системи виступає TranslationQueueManager, який забезпечує безпечну паралельну обробку мережевих запитів. Його логіку засновано на механізмі асинхронних семафорів (asyncio.Semaphore). Методи структурної ідентифікації об’єктів успішно використовують схожі принципи балансування потоків [43, 44]. Логіку роботи менеджера під час одночасного надходження кількох запитів від різних користувачів зображено на діаграмі діяльності (рис. 3.10).

Складним технічним викликом стала інтеграція синхронного C++ коду генерації (який міститься у бібліотеці CTranslate2) у повністю асинхронне середовище FastAPI. Прямий виклик методу при роботі заблокував би подієвий цикл (Event Loop) мови Python, що зробило б сервер недоступним для інших користувачів. Для вирішення цієї проблеми розроблено спеціальну асинхронну обгортку (лістинг 3.5), яка використовує метод run_in_executor для винесення важких тензорних обчислень у фоновий пул потоків операційної системи.

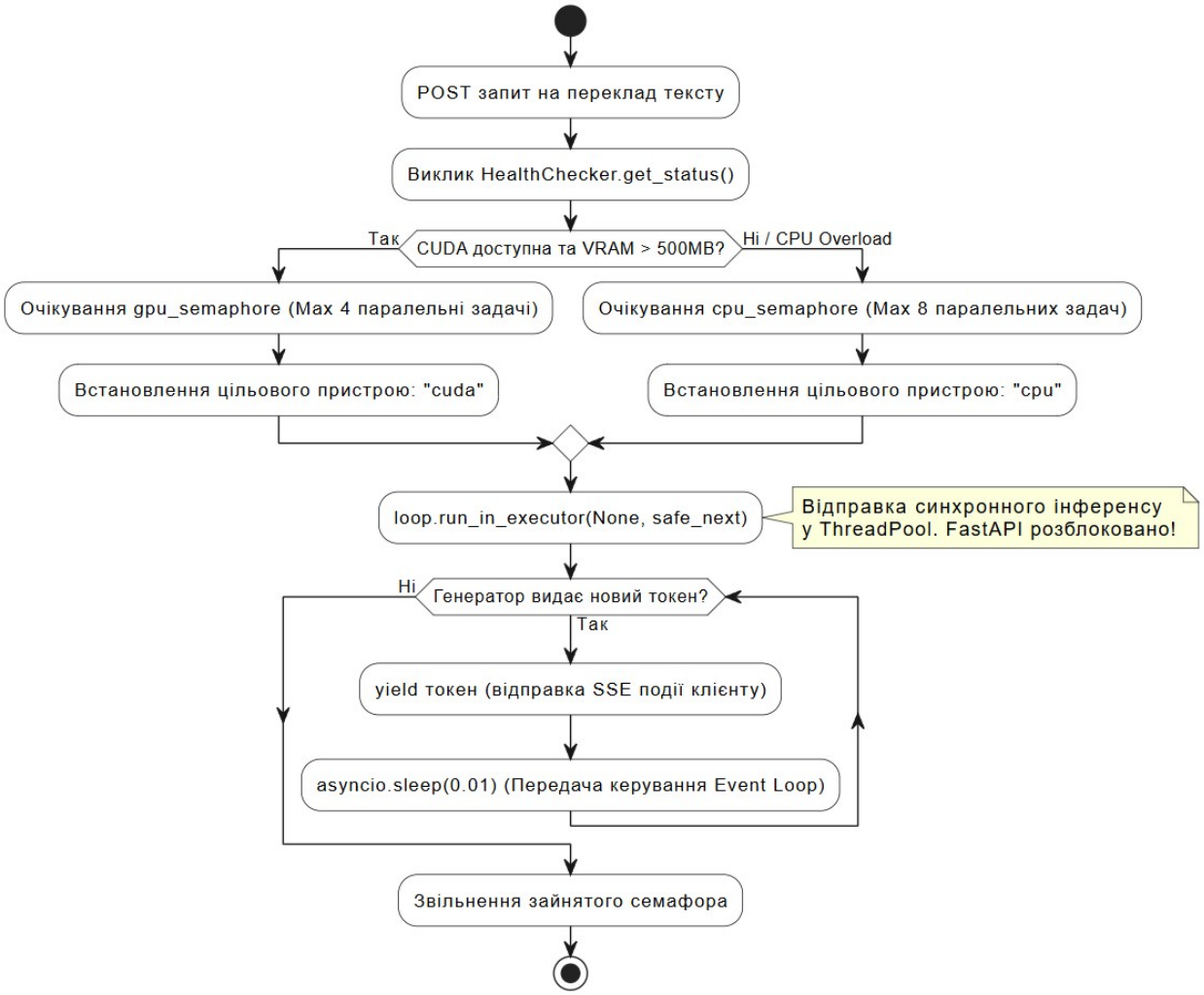


Рисунок 3.10 – Оброблення паралельних запитів у менеджері черг

Лістинг 3.5 Асинхронна обгортка для ізоляції блокуючих обчислень:

```
async def _execute_inference(self, text, source_lang, target_lang, device):
    loop = asyncio.get_running_loop()
    # Створення об’єкта синхронного генератора від рушія
    sync_gen = self.translator.translate_stream(text, source_lang, target_lang, device=device)

    def safe_next():
        try: return next(sync_gen)
        except StopIteration: return None
        except Exception as e: return e
```

```
while True:
    # Виконання блокуючого виклику у фоновому пулі потоків
    result = await loop.run_in_executor(None, safe_next)
    if result is None:
        break
    if isinstance(result, Exception):
        yield f" [Помилка інференсу: {result}] "
        break

    yield result
    await asyncio.sleep(0.01)
```

Для запобігання непередбачуваному падінню сервера через нестачу відеопам’яті (Out of Memory Error), розроблено клас HealthChecker (лістинг 3.6). Використовуючи низькорівневу системну бібліотеку pynvml, система безперервно відстежує рівень вільної пам’яті на графічному прискорювачі.

Лістинг 3.6 Динамічний моніторинг апаратних ресурсів сервера:

```
def get_status(self, queue_size: int = 0) -> Dict[str, Any]:
    gpu = self.get_gpu_metrics()
    sys = self.get_system_metrics()
    mode = "Normal"
    recommended_device = "cuda" if self.gpu_available else "cpu"

    # Запобігання критичним помилкам OOM на базі CUDA
    if self.gpu_available and gpu["free_vram"] < 500:
        mode = "Slow Mode (Low VRAM)"
        recommended_device = "cpu"
```

```
    elif sys["cpu_usage"] > 90:
        mode = "Overloaded"
        recommended_device = "cuda" if self.gpu_available else "cpu"

    return { "status": mode, "device": recommended_device }
```

Ще однією алгоритмічно складною задачею було оброблення форматів структурованих документів (зокрема стандарту .docx). Документи Word складаються з дрібних XML-фрагментів тексту (Runs), кожен з яких містить власний набір стилів. Для збереження цього візуального форматування під час перекладу розроблено алгоритм реконструкції розмітки (лістинг 3.7), який пропорційно переносить старі стилі на новий перекладений рядок, що гарантовано має іншу довжину в символах.

Лістинг 3.7 Алгоритм перенесення стилів у структурованих документах Word:

```
    def _distribute_to_runs(self, translated: str, runs_meta: List[dict]) -> List[str]:
        # Підрахунок загальної довжини оригінального фрагмента в символах
        total_orig = sum(len(r.get('text', '')) for r in runs_meta)
        if total_orig == 0:
            return [translated] + [''] * (len(runs_meta) - 1)

        tlen = len(translated)
        lengths = []
        acc = 0

        # Обчислення нової довжини пропорційно до старого форматування XML-вузлів
```

```
        for i, r in enumerate(runs_meta):
            if i == len(runs_meta) - 1:
                lengths.append(tlen - acc) # Залишок тексту потрапляє в останній фрагмент
            else:
                proportional = round(len(r.get('text', '')) / total_orig * tlen)
                lengths.append(proportional)
                acc += proportional

        # Нарізання перекладеного рядка згідно з розрахованими пропорціями
        allocated = []
        pos = 0
        for ln in lengths:
            allocated.append(translated[pos:pos+ln])
            pos += ln

        return allocated
```

3.3.3 Якісна оцінка результатів перекладу

Для комплексної оцінки результатів було проведено фінальне тестування на складних морфологічних конструкціях, що містили специфічне керування відмінків [45]. На рисунку 3.11 наведено скріншот успішного перекладу проблемного речення, який демонструє здатність інтегрованої гібридної моделі до коректного вибору словозмінних афіксів (наприклад, правильне застосування давального відмінка). Застосування систем ієрархічних ознак та ортогональних функцій гарантує високу точність такого розпізнавання [46, 47].

Розроблений алгоритм збереження форматування також довів свою ефективність на складних корпоративних документах. На рисунку 3.12 продемонстровано фрагмент файлу до та після програмного оброблення, де успішно збережено структуру таблиць та виділених курсивом фрагментів без втрати лінгвістичного контексту перекладених речень.

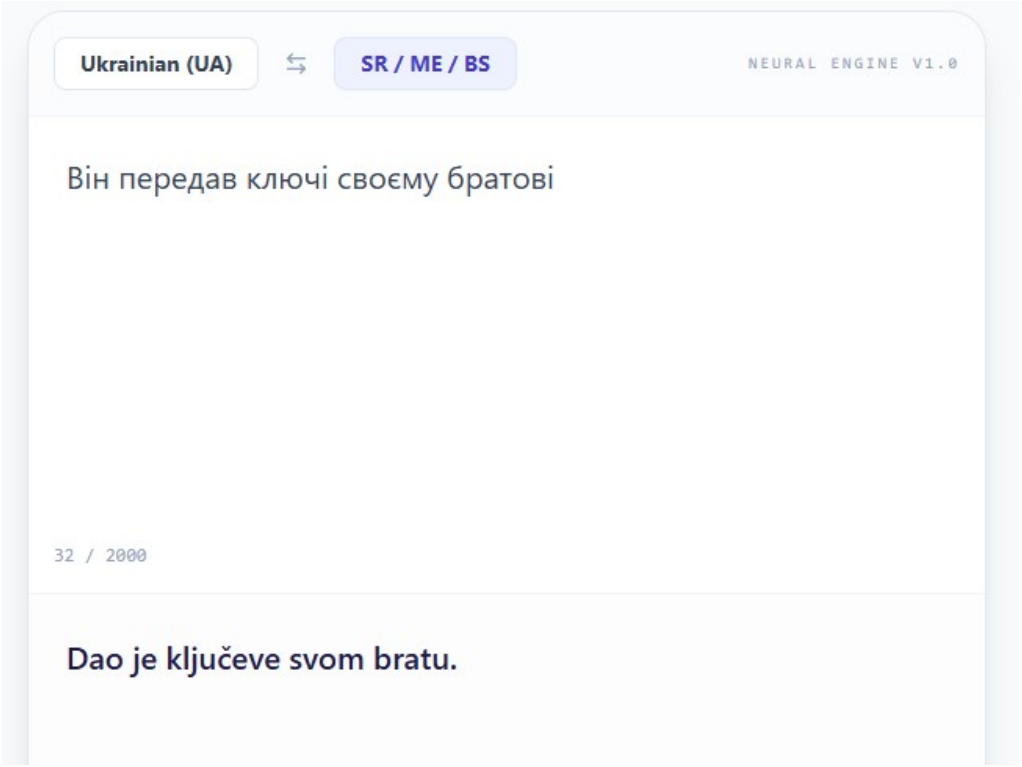


Рисунок 3.11 – Приклад складного перекладу

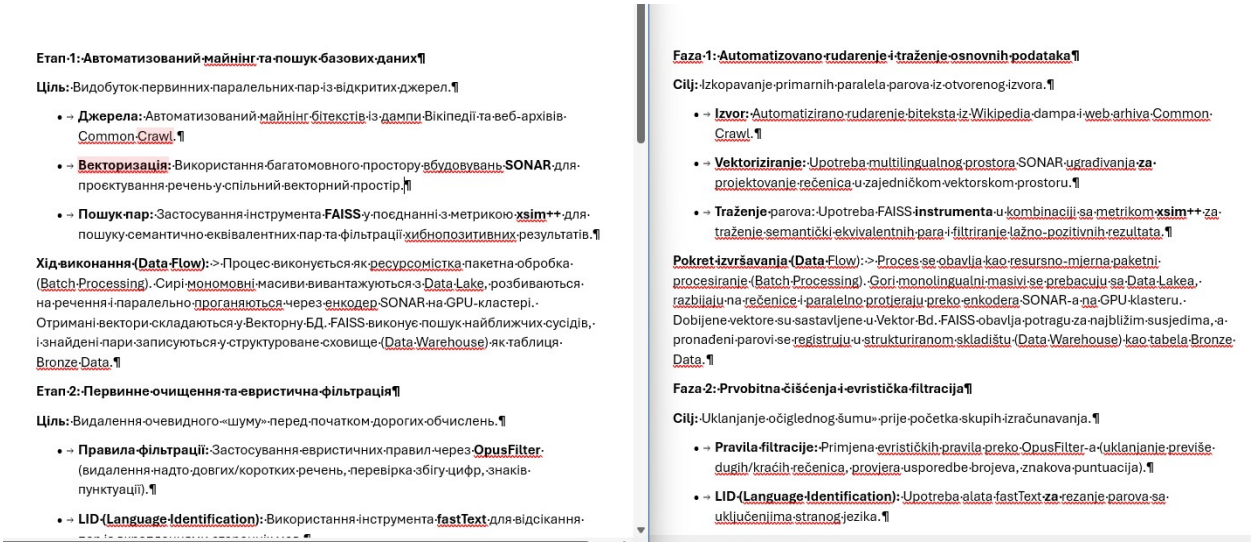


Рисунок 3.12 – Збереження форматування при перекладі документу

3.4 Клієнтська частина застосунку

Розроблення графічного інтерфейсу клієнта спрямоване на створення інтуїтивно зрозумілого візуального середовища для взаємодії кінцевого користувача з нейромережею. Основний акцент під час проєктування зроблено на чутливості інтерфейсу (високій реактивності) та надійному управлінні станами.

3.4.1 Вимоги до інтерфейсу та управління станом

Інтерфейс розроблено з використанням декларативного та компонентного підходу на базі React. Для забезпечення стабільної поведінки застосунку під час виконання паралельних мережевих запитів було застосовано суворий розподіл змінних стану між ізольованими модулями [48]. Специфікацію технологій управління станом наведено у таблиці 3.6.

Таблиця 3.6 – Специфікація технологій та управління станом у React

Технологія / виклик	**Змінна стану / Призначення**	**Інженерне обґрунтування вибору**
1	**2**	**3**
React 18	Декларативний рендеринг UI	Забезпечує миттєве перемалювання символів при отриманні текстових фрагментів (чанків).
useStreaming.ts	chunks, isStreaming	Кастомний виклик для прямого підключення по API (отримання побайтового потоку без зависань).

Продовження таблиці 3.6

1	2	3
App.tsx	abortControllerRef	Об’єкт, що дозволяє миттєво скасувати мережевий запит при зміні тексту користувачем.
FileUploader.tsx	jobId	Надійне зберігання ідентифікатора сесії для відкладеного скачування згенерованого файлу.
useHealth.ts	Статистика	Реалізує шаблони періодичного опитування (Polling) стану бекенду.

Ієрархію, рівні абстракції та функціональні зв’язки між цими візуальними компонентами й логічними хуками наочно відображено на діаграмі компонентів (рис. 3.13).

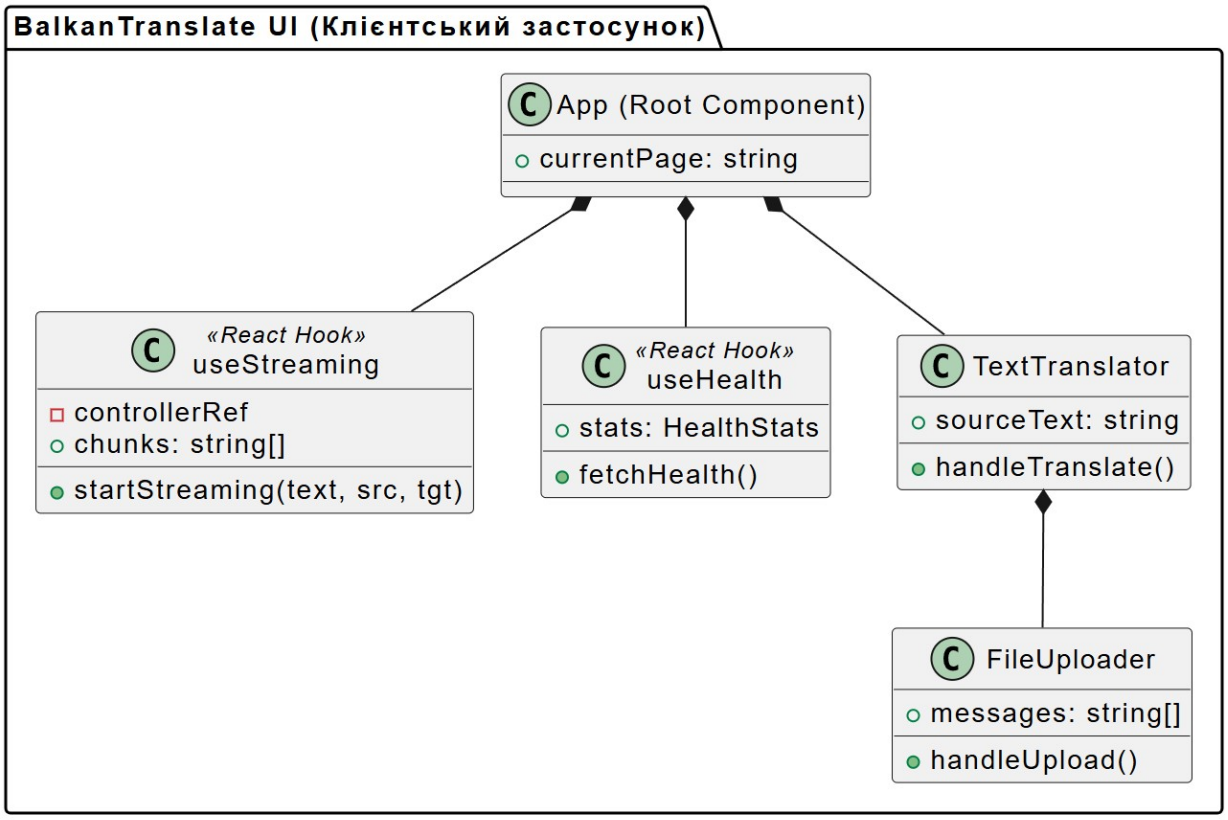


Рисунок 3.13 – Ієрархія компонентів клієнтського застосунку

3.4.2 Структура взаємодії компонентів та обробка потоків

Найбільш складним компонентом користувацького інтерфейсу є FileUploader, який бере на себе відповідальність за асинхронне завантаження, перевірку та обробку документів. З огляду на велику кількість проміжних кроків, його поведінку було формалізовано у вигляді машини станів (рис. 3.14).

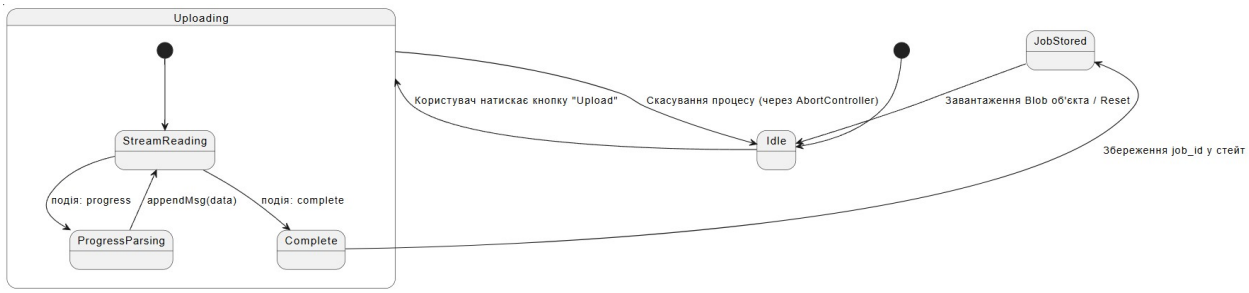


Рисунок 3.14 – Логіка роботи компонента завантаження файлів

Для безпосередньої обробки однонаправленого потоку згенерованих токенів від сервера розроблено спеціалізований виклик useStreaming (лістинг 3.8). Оскільки стандартні HTTP-клієнти, такі як Axios, діють за принципом очікування завершення всього запиту, вони категорично не підходять для потокової генерації тексту. Тому було використано стандартний інтерфейс Fetch API з прямим доступом до об’єкта ReadableStream [49].

Лістинг 3.8 Реалізація стрімінгового читання потоку даних:

```
const reader = resp.body?.getReader();
const decoder = new TextDecoder();
let buffer = ‘‘;

// Асинхронне зчитування байтів у міру їх надходження від сервера
while (true) {
```

```
const { done, value } = await reader.read();
if (done) break;
buffer += decoder.decode(value, { stream: true });

const parts = buffer.split('\n');
buffer = parts.pop() || ''; // Залишаємо неповні рядки у буфері для наступної ітерації

for (const part of parts) {
  const line = part.trim();
  if (line.startsWith('data:')) {
    const payload = line.replace(/^data:\s*/, '');
    if (payload === '[DONE]') { setIsStreaming(false); continue; }

    try {
      const p = JSON.parse(payload);
      if (p.text) setChunks((c) => [...c, p.text]);
    } catch {
      setChunks((c) => [...c, payload]);
    }
  }
 }
}
```

Щоб прозоро інформувати користувача про можливе перевантаження серверної системи та доступність графічного прискорювача, застосовано шаблон періодичного опитування (Polling). Спеціальний виклик useHealth (лістинг 3.9) опитує сервер кожні 5 секунд. Важливим інженерним елементом тут є використання функції clearInterval у фазі очищення (Cleanup), що

запобігає витоку пам’яті (Memory Leak) під час демонтажу (unmount) компонента або переходу користувача на іншу сторінку застосунку.

Лістинг 3.9 Шаблон періодичного опитування статусу бекенду:

```
export function useHealth(pollInterval = 5000) {
  const [stats, setStats] = useState<HealthStats>(null);
  const timerRef = useRef<number | null>(null);

  const fetchHealth = async () => {
    const res = await fetch('/health');
    const j = await res.json();
    setStats({ status: j.status, device: j.device });
  };

  useEffect(() => {
    fetchHealth(); // Початковий виклик під час монтування
    timerRef.current = window.setInterval(fetchHealth, pollInterval);

    return () => {
      // Фаза очищення (Cleanup): запобігання витоку оперативної пам’яті браузера
      if (timerRef.current) window.clearInterval(timerRef.current);
    };
  }, [pollInterval]);

  return { stats };
}
```

Після того як сервер успішно завершує переклад масивного документа, бінарні дані (об’єкт типу Blob) повертаються на клієнтську частину. Оскільки

з міркувань безпеки сучасні браузери блокують автоматичне завантаження файлів, які були отримані через фонові мережеві запити, для ініціації системного вікна збереження файлу використано підхід з ін'єкцією тимчасового HTML-елемента у DOM-дерево (лістинг 3.10).

Лістинг 3.10 Симуляція системного завантаження файлу браузером:

```
const downloadBlob = (data: Blob | string, filename: string, contentType?: string) => {
  const blob = typeof data === 'string' ? new Blob([data], { type: contentType || 'text/plain' }) : data;
  const url = URL.createObjectURL(blob);

  // Створення невидимого HTML-елемента посилання (тег <a>)
  const a = document.createElement('a');
  a.href = url;
  a.download = filename;
  document.body.appendChild(a); // Ін'єкція елемента в DOM-дерево
  a.click(); // Симуляція кліку мишею для відкриття діалогу ОС
  a.remove(); // Видалення елемента після натискання

  // Відкладене очищення пам'яті браузера для запобігання витокам
  setTimeout(() => URL.revokeObjectURL(url), 1000);
};
```

3.4.3 Тестування розробленого застосунку

Розроблений клієнтський інтерфейс успішно інтегрує складні обчислювальні процеси нейромережі у зручну, мінімалістичну вебоболонку. На рисунку 3.15 продемонстровано головний екран застосунку, де

користувач має змогу вводити текст та отримувати миттєвий потоковий відгук без перезавантаження сторінки [50-52].

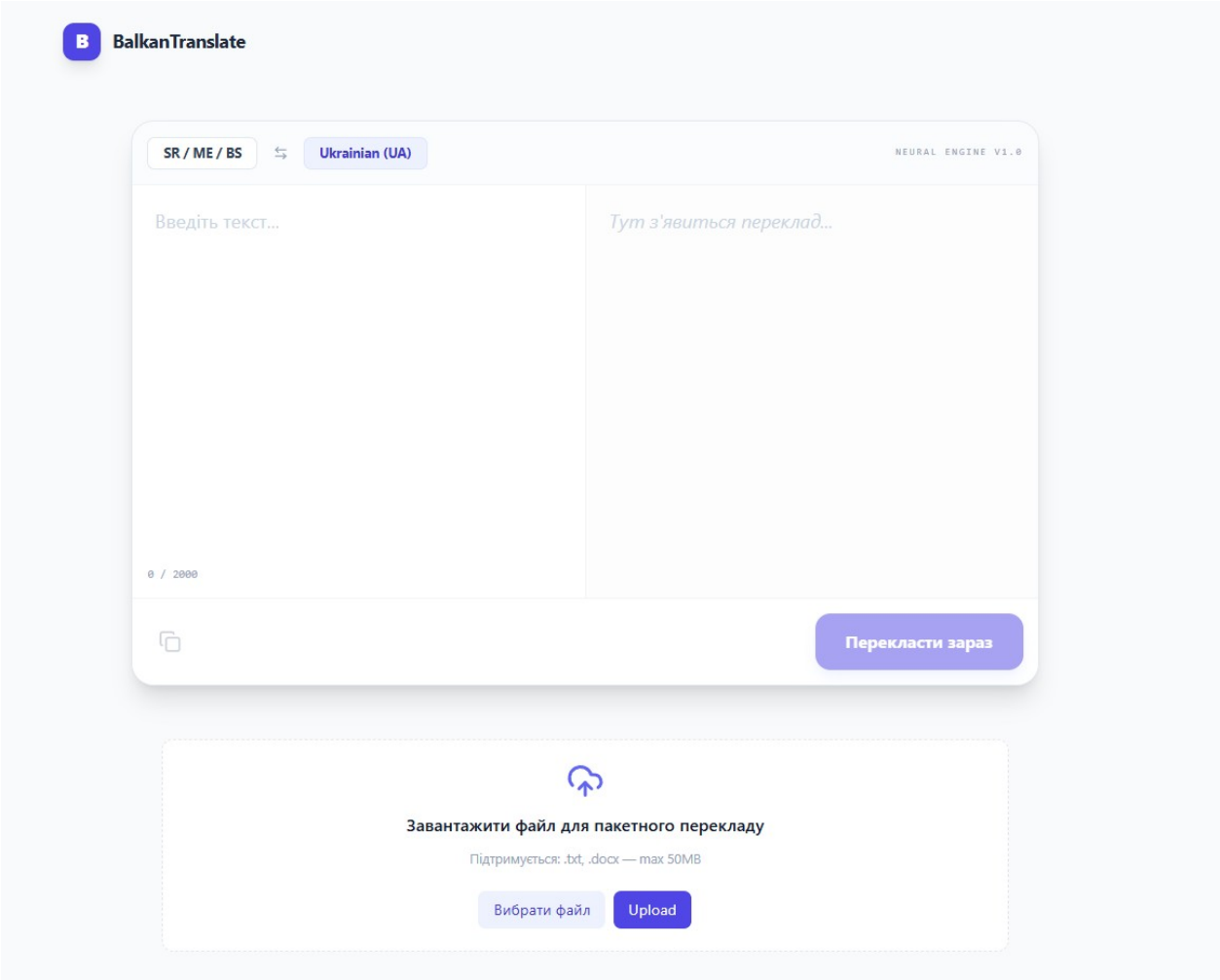


Рисунок 3.15 – Приклад інтерфейсу користувача

У верхньому правому куті реалізовано відображення індикатора здоров’я сервера, який динамічно змінює колір залежно від завантаженості відеокарти.

Окремий підмодуль системи забезпечує безшовну роботу з файлами. На рисунку 3.16 показано процес завантаження структурованого масивного документа. Система динамічно відображає процес обробки кожного абзацу через отримання подій формату SSE, що створює позитивний та прозорий користувацький досвід.

Застосування сучасних принципів розроблення реактивних інтерфейсів у поєднанні з надійним асинхронною серверною частиною дозволило створити стабільний, швидкий та відмовостійкий прикладний інструмент. Розроблене програмне забезпечення повністю відповідає поставленим вимогам і може використовуватися кінцевими споживачами для вирішення завдань двонаправленого машинного перекладу зі збереженням форматування документів.

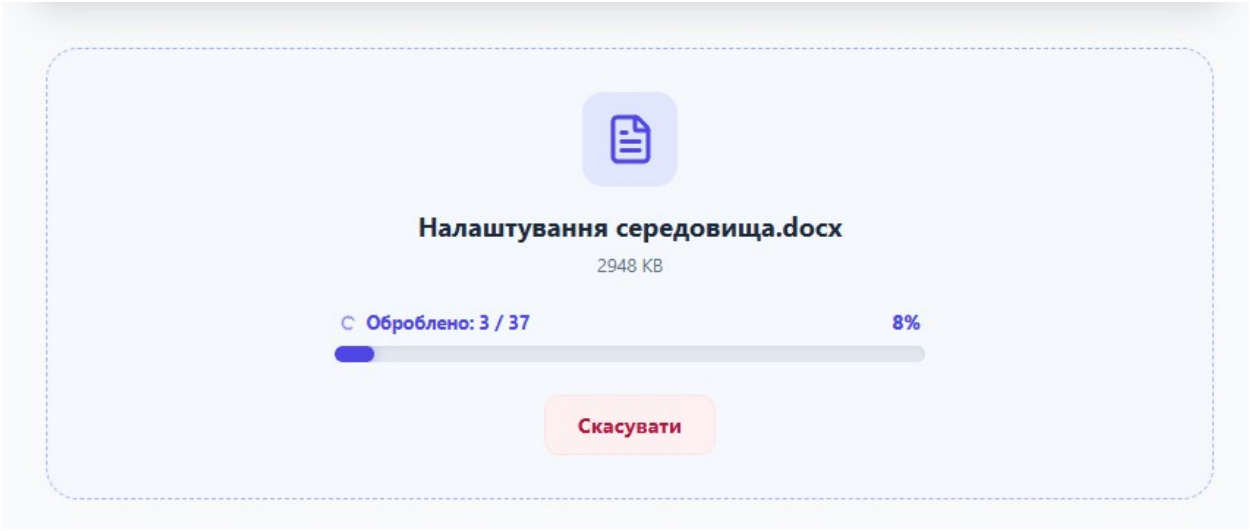


Рисунок 3.16 – Приклад інтерфейсу під час обробки

ВИСНОВКИ

У рамках кваліфікаційної роботи розроблено і реалізовано вебзастосунок для нейронного машинного перекладу низькоресурсних мов на основі архітектури Transformer. Робота охоплює повний цикл створення застосунку від аналізу наявних рішень до реалізації функціоналу із використанням сучасних інструментів, та вирішено такі завдання:

– проведено аналіз наявних архітектур нейронного машинного перекладу та методик оцінювання якості, що дало можливість встановити перевагу двонаправленого механізму уваги моделей типу кодувальник-декодувальник для флективних слов'янських мов;

– досліджено вплив зашумлених паралельних даних на поверхневі метрики та виявлено причини метричної невідповідності, що дозволило довести хибнопозитивність традиційних метрик майже у 20% випадків та обґрунтувати доцільність застосування мовних моделей у ролі автоматичних суддів;

– розроблено алгоритм комбінованого синтезу даних із застосуванням генеративних моделей для морфологічної аугментації, що дозволило сформувати оптимізований корпус обсягом 92000 пар речень та вирішити проблему дефіциту даних;

– проведено порівняльні експерименти стратегій навчання, що дало можливість експериментально підтвердити ефективність повношарової адаптації;

– обґрунтовано архітектуру системи на базі моделі NLLB з інтеграцією DoRA-адаптерів, що дозволило суттєво підвищити показник символьної точності перекладу;

– реалізовано підсистему підготовки даних та виконано навчання моделі на створеному навчальному корпусі;

– розроблено ядро нейронного перекладу (Inference Engine) та користувацький вебінтерфейс, завдяки чому споживання відеопам’яті оптимізовано та скорочено до мінімуму;

– протестовано систему на контрольних вибірках, що дало можливість підтвердити стабільну роботу розробленого програмного забезпечення на споживчому обладнанні.

Розглянуто наявні вебзастосунки зі схожим функціоналом та проаналізовано літературні джерела, що стосуються задач машинного перекладу для низькоресурсних мовних пар. Вивчено наявні методи параметричної адаптації та оцінювання якості перекладу, особливо ті, які орієнтовані на збереження складних морфологічних структур у флективних слов’янських мовах.

Для реалізації нейронного машинного перекладу використано базову модель NLLB з інтеграцією DoRA-адаптерів, а для оптимізації обчислень застосовано алгоритм лінійного квантування тензорних ваг до формату цілих чисел. Як інтерфейс обрано бібліотеку React, а вся логіка реалізована мовою програмування Python.

Проведено тестування працездатності системи, зокрема перевірку алгоритмів збереження вихідної розмітки під час перекладу структурованих документів, аналіз швидкості потокової генерації тексту, а також оцінювання точності та стабільності роботи ядра перекладу при високих навантаженнях.

Розглянуто перспективи покращення системи, які включають у себе оптимізацію алгоритмів квантування з метою прискорення обчислень, розширення функціоналу системи для оброблення інших форматів даних, а також можливість адаптації розробленої архітектури для інших низькоресурсних або близькоспоріднених мовних пар.

Результати кваліфікаційної роботи можуть бути корисними у сфері створення ресурсоефективного програмного забезпечення, придатного для локального використання в умовах обмежених обчислювальних потужностей без втрати якості оброблення складних морфологічних структур.

Результати роботи апробовано у вигляді статті та 4 тез доповідей:

– у науковому журналі «Information Processing Systems» [22];

– під час II Всеукраїнської науково-технічної конференції «Challenges and Solutions in Software Engineering» [5];

– під час II Міжнародної науково-практичної студентської конференції «IT Space of Today: Trends, Innovations and Development Prospects» [9];

– під час 30-го Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [19];

– під час XVI Всеукраїнської конференції молодих учених «Young Scientists 2026 – From Theory to Practice» [38].

Дослідження виконано в межах проєкту INITIATE за грантом No.101136775 HORIZON WIDERA 2023 ACCESS 03.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

2. Lialin, V., Deshpande, V., Yao, X., & Rumshisky, A. (2023). Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv*:2303.15647.

3. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1-67.

4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33, 1877-1901.

5. Lysenko, D. O., & Kobylyn, I. O. (2025). Research of the advantages of the Transformer architecture over the RNN/LSTM architecture in NLP. In *Challenges and Solutions in Software Engineering: Proceedings of the II All-Ukrainian Scientific and Technical Conference* (pp. 323–326). State University of Information and Communication Technologies.

6. Costa-jussà, M. R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., ... & Wang, C. (2022). No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv*:2207.04672.

7. Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-318).

8. Popović, M. (2015). chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation* (pp. 392-395).

9. Lysenko, D. O., & Kobylyn, I. O. (2025). Metrics for evaluating machine translation based on vector-based representations. In *IT Space of Today: Trends, Innovations and Development Prospects: Proceedings of the II International Scientific and Practical Student Conference* (pp. 278–280). V. N. Karazin Kharkiv National University.

10. Rei, R., Stewart, C., Farinha, C., & Lavie, A. (2020). COMET: A neural framework for MT evaluation. *arXiv preprint arXiv*:2009.09025.

11. Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv*:2106.09685.

12. Liu, S. Y., Shen, C. Y., Lee, H. Y. (2024). DoRA: Weight-Decomposed Low-Rank Adaptation. *arXiv preprint arXiv*:2402.09353.

13. Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv*:2305.14314.

14. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. *Information Technology and Management Science*, 23-28.

15. Bodyanskiy, Y., Vynokurova, O., Szymański, Z., Kobylin, I., & Kobylin, O. (2016). Adaptive robust models for identification of nonstationary systems in data stream mining tasks. In 2016 IEEE *First International Conference on Data Stream Mining & Processing (DSMP)* (pp. 263-268). IEEE.

16. Tvoroshenko, I. S., & Gorokhovatsky, V. O. (2019). Intelligent classification of biophysical system states using fuzzy interval logic. *Telecommunications and Radio Engineering*, 78(14), 1303-1315.

17. Tvoroshenko, I., Ahmad, M. A., Mustafa, S. K., Lyashenko, V., & Alharbi, A. R. (2020). Modification of Models Intensive Development Ontologies by Fuzzy Logic. *International Journal of Emerging Trends in Engineering Research*, 8(3), 939-944.

18. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... & Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv*:1911.02116.

19. Lysenko, D. O., & Kobylyn, I. O. (2026). Methods for creating datasets for small-resource language pairs. In *Radio Electronics and Youth in the XXI Century: Proceedings of the 30th International Youth Forum* (Vol. 7, pp. 19–21). Kharkiv National University of Radio Electronics.

20. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Technologies for Making Reliable Decisions on a Variety of Effective Factors using Fuzzy Logic. *International Journal of Advanced Computer Science and Applications*, 11(5), 43-50.

21. Tvoroshenko, I. S. (2004). Structure and functions of intelligent decision-making tools in complex systems. *Artificial Intelligence*, 4, 462-470.

22. Lysenko, D. O., Kobylyn, I. O., Putiatina, O. Ye., & Lysenko, O. A. (2026). Comprehensive approach to building a hybrid data processing pipeline in neural machine translation systems. *Information Processing Systems*, *1*(184). Ivan Kozhedub Kharkiv National Air Force University.

23. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. *Advanced Information Systems*, 9(1), 5-12.

24. Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). Image classification method modification based on model of logic processing of bit description weights vector. *Telecommunications and Radio Engineering*, 79(1), 59-69.

25. Tvoroshenko, I., & Zarivchatskyi, R. (2020). Analysis of existing methods for searching object in the video stream. Abstracts of VI International Scientific and Practical Conference «About the problems of science and practice, tasks and ways to solve them» (October 26-30, 2020). Milan, Italy, 500-505.

26. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing* (pp. 38-45).

27. Klein, G., Kim, Y., Deng, Y., Senellart, J., & Rush, A. M. (2017). OpenNMT: Open-source toolkit for neural machine translation. *arXiv preprint arXiv*:1701.02810.

28. Кобилін, І. О., & Ніколайчук, А. І (2024). Monitoring and diagnosing faults in online mode using time series data. *Системи обробки інформації*, 3(178), 27-32.

29. Кобилін, І. О., & Харченко, А. І. (2024). Classification techniques for computer vision. *Системи обробки інформації*, 3(178), 33-41.

30. Kharchenko, A. I., & Kobylin, I. O. (2024). Performance analysis of support vector machine for vehicle classification. *Ministry of Education and Science of Ukraine* VN Karazin Kharkiv National University, 101.

31. Бодянський, Є., Винокурова, О., Кобилін, І., & Мулета, П. (2017). Адаптивна матрична нейро-фаззі самоорганізовна мережа для кластеризації багатовимірних потоків даних. *Вісник Національного університету Львівська політехніка*, (864), 314-319.

32. Бодянський, Є. В., Винокурова, О. А., Ізонін, І. В., Кобилін, І. Ю., & Мулета, П. П. (2017). Кластеризація багатовимірних часових рядів на основі адаптивної матричної нейро-фаззі самоорганізовної мережі. *Intellectual Systems For Decision Making*, 247.

33. Творошенко, І. С. (2018). Особливості застосування сутнісних принципів штучного інтелекту до розробки ефективних механізмів моделювання складних систем. *Science and Technology of the Present Time*, 118-121.

34. Daradkeh, Y. I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L. A., & Ahmad, N. (2021). Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic. *IEEE Access*, 9, 13417-13428.

35. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ.

36. Творошенко, И. С. (2010). Анализ процессов принятия решений в интеллектуальных системах. *Системи обробки інформації*, (2), 248-253.

37. Кучеренко, Е. И., Филатов, В. А., Творошенко, И. С., & Байдан, Р. Н. (2005). Интеллектуальные технологии в задачах принятия решений технологических комплексов. *Восточно-Европейский журнал передовых технологий*, 2, 92-96.

38. Lysenko, D. O., & Kobylyn, I. O. (2026). Optimisation of large language model (LLM) deployment through modern methods of weight quantisation, activation and KV-cache. In *Young Scientists 2026 – From Theory to Practice: Proceedings of the XVI All-Ukrainian Conference of Young Scientists* (pp. 401–403). Ukrainian State University of Science and Technologies.

39. Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. *arXiv preprint arXiv*:1508.07909.

40. Творошенко, І. С., & Табашник, В. А. (2018). Розробка просторової моделі геоінформаційної підтримки людей з обмеженими можливостями. *Вісник наукових праць ХНУПС*, (1), 122-128.

41. Kobylin, O. A., Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). The application of non-parametric statistics methods in image classifiers based on structural description components. *Telecommunications and Radio Engineering*, *79*(10), 875-887.

42. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description. *Advances in Electrical and Electronic Engineering*, *21*(1), 19-27.

43. Кобилін, О., Вечірська, І., & Кравченко, О. (2024). Порівняння нейронних мереж типу RNN та LSTM. *Information Technology: Computer Science, Software Engineering and Cyber Security*, (3), 97–107.

44. Кобилін, О. А., & Путятіна, О. Є. (2024). Знешумлення зображень, зіпсованих дробовим шумом, у реальному часі. *Системи обробки інформації*, 1(176), 46–51.

45. Кобилін, О., Вечірська, І., & Афанасьєв, А. (2024). Аналіз існуючих моделей глибинного навчання в задачах обробки природної мови. *Information Technology: Computer Science, Software Engineering and Cyber Security*, (3), 63-76.

46. Bohdan, N., Tvoroshenko, I., Gorokhovatskyi, V., & Kobylin, O. (2025). Development of a hybrid method to enhance context memory for a chatbot application based on large language models. *International Journal of Academic Information Systems Research*, 9(10), 7-18.

47. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень: навч. посібник. Харків: ХНУРЕ.

48. Suprun, A., Tvoroshenko, I., Gorokhovatskyi, V., & Yakovleva, O. (2025). Development and research of a method for the combined use of large language models for text generation. *International Journal of Academic and Applied Research*, 9(10), 249-263.

49. Боденчук-Пастухов, Д. В., & Кобилін, І. О. (2026). Оцінка моделей трансформації машинного перекладу на низько-ресурсних, азійсько-уральських мовних парах. *Системи обробки інформації*, 1(184), 116-123.

50. Гороховатський В., Творошенко І. (2026) Продуктивні моделі аналізу даних у методах розпізнавання зображень: монографія. Харків: ХНУРЕ, 153 с.

51. Kobylin, I., Biehunova, V., Tsyban, D., & Kovalchuk, V. (2026). Measuring Textual Redundancy, Lexical Richness, and Veracity: a Multi-Metric Approach to Text Evaluation. *Information Technologies and Systems*, *8*(2), 25-45.

52. Nikolaichuk, A., Kobylin, O., Kobylin, I., & Putiatina, O. (2026). A comprehensive evaluation of transformer models for sentence-level semantic similarity in English and Ukrainian. *Avtomatizovani Systemy Upravlinnia ta Prylady Avtomatyky*, (189), 284-296.