**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інтелектуальних інформаційних систем**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

# ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ НЕОЛОГІЗМІВ У СФЕРІ ЛОКАЛІЗАЦІЇ ІТ-КОНТЕНТУ НА ОСНОВІ ГЕНЕРАТИВНОЇ НЕЙРОМЕРЕЖІ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

***Здобувач*** _____ Роман ПАСТУХОВ

«___»_____ 2026 р.

***Керівник*** д-р техн. наук, професор _____ Олексій КОЗЛОВ

«___»_____ 2026 р.

**Миколаїв – 2026**

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

| | |
|---|---|
| Факультет | Комп’ютерних наук |
| Кафедра | Інтелектуальних інформаційних систем |
| Рівень вищої освіти | Перший (бакалаврський) |
| Освітній ступень | Бакалавр |
| Спеціальність | 122 Комп’ютерні науки |
| Освітня програма | Комп’ютерні науки |

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2026 р.

**ЗАВДАННЯ**
**на кваліфікаційну роботу здобувача**

**Пастухова Романа Олеговича**
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для створення неологізмів у сфері локалізації ІТ-контенту на основі генеративної нейромережі».

Керівник роботи: Козлов Олексій Валерійович, доктор технічних наук, професор кафедри інтелектуальних інформаційних систем факультету комп'ютерних наук.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інтелектуальна веб-система для автоматичної генерації морфологічно коректних українських ІТ-неологізмів на базі нейромережевої архітектури Transformer; початкові дані:

набори англомовних ІТ-термінів, англо-українські глосарії та словники, правила українського словотвору, лексикографічні та морфологічні бази даних.

4. Перелік питань, що підлягають розробці: аналіз предметної сфери автоматичного словотвору та методів локалізації ІТ-термінології; огляд сучасних нейромережевих архітектур для генерації тексту; підготовка та автоматизоване оброблення навчального датасету; проєктування та навчання генеративної моделі на базі архітектури Transformer із механізмами багатогалузевої уваги; розробка алгоритму примусового збереження семантичного кореня слова; реалізація серверної частини на базі фреймворку FastAPI та розробка клієнтського веб-інтерфейсу; тестування роботи системи та автоматизована оцінка милозвучності згенерованих неологізмів.

5. Перелік графічних матеріалів: презентація

**Керівник роботи** _____ Олексій КОЗЛОВ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

**Здобувач** _____ Роман ПАСТУХОВ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «23» грудня 2025 р.

# КАЛЕНДАРНИЙ ПЛАН
**кваліфікаційної роботи**

Тема: Вебзастосунок для створення неологізмів у сфері локалізації ІТ-контенту на основі генеративної нейромережі

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1 | Отримання завдання на виконання КР | 21.12.2025 | 24.12.2025 | |
| 2 | Аналіз предметної області та постановка задачі | 25.12.2025 | 30.01.2026 | |
| 3 | Огляд літературних джерел, публікацій та існуючих аналогів у сфері автоматичної генерації тексту. | 31.01.2026 | 01.03.2026 | |
| 4 | Огляд архітектур нейронних мереж та обґрунтування вибору технологій розробки. | 02.03.2026 | 01.04.2026 | |
| 5 | Програмна реалізація генеративної моделі, розробка веб-застосунку з подальшим тестуванням та аналізом милозвучності отриманих результатів. | 02.04.2026 | 24.05.2026 | |
| 6 | Перший попередній захист КР на засіданні комісії кафедри | 25.05.2026 | 25.05.2026 | |
| 7 | Корегування роботи за результатами попереднього захисту | 26.05.2026 | 04.06.2026 | |
| 8 | Другий попередній захист КР на засіданні комісії кафедри | 05.06.2026 | 05.06.2026 | |
| 9 | Доробка та остаточне оформлення КР | 06.06.2026 | 14.06.2026 | |
| 10 | Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту | 15.06.2026 | 19.06.2026 | |

**Керівник роботи** _____ Олексій КОЗЛОВ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

**Здобувач** _____ Роман ПАСТУХОВ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

# АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

**Пастухова Романа Олеговича**

на тему: **“ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ НЕОЛОГІЗМІВ У СФЕРІ ЛОКАЛІЗАЦІЇ ІТ-КОНТЕНТУ НА ОСНОВІ ГЕНЕРАТИВНОЇ НЕЙРОМЕРЕЖІ”**

**Актуальність** даної роботи полягає у необхідності створення ефективних інструментів для формування природних українських лексичних відповідників для сучасних англомовних ІТ-термінів, що дозволить уникнути прямого запозичення та калькування. Це сприятиме стрімкому розвитку української технічної термінології та підвищенню якості локалізації програмного забезпечення.

**Об’єктом роботи** є процес автоматичної генерації українських ІТ-неологізмів на основі англомовних термінів.

**Предметом роботи** є нейромережеві методи генерації тексту, зокрема архітектура Sequence-to-Sequence на базі моделі Transformer.

**Метою роботи** є підвищення ефективності процесу локалізації ІТ-термінології за рахунок розробки інтелектуального вебзастосунку на основі генеративної нейромережі.

В результаті виконання роботи було зібрано та оброблено спеціалізований навчальний датасет, досліджено архітектуру Transformer, розроблено алгоритм примусового префіксного декодування для гарантованого збереження семантичного кореня, імплементовано метод евристичної оцінки милозвучності, а також розроблено клієнт-серверний веб-застосунок на базі фреймворку FastAPI.

Дана робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної області автоматичного словотвору, огляд існуючих програмних аналогів та сформовано постановку задачі дослідження. Другий розділ присвячений моделям, методам та інформаційним технологіям, що використані у роботі, зокрема

докладно описано архітектуру генеративної нейромережі. У третьому розділі описано процес розробки системи, етапи навчання моделі Transformer та проаналізовано отримані результати генерації українських неологізмів. У четвертому розділі наведено деталі програмної реалізації вебзастосунку, описано клієнтський інтерфейс, представлено керівництво користувача та наведено результати тестування системи.

Загальний обсяг роботи – 85 сторінок. Кваліфікаційна робота містить 19 рисунків, 5 таблиць і 27 джерел посилання.

**Ключові слова:** локалізація ПЗ, ІТ-неологізми, автоматичний словотвір, генерація тексту, нейронна мережа, Transformer, FastAPI.

# ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

**Pastukhov Roman**

## “WEB APPLICATION FOR CREATING NEOLOGISMS IN THE FIELD OF IT CONTENT LOCALIZATION BASED ON A GENERATIVE NEURAL NETWORK”

The relevance of this work lies in the need to create effective tools for forming natural Ukrainian lexical equivalents for modern English IT terms, which will allow avoiding direct borrowing and loan translation. This will contribute to the rapid development of Ukrainian technical terminology and improve the quality of software localization.

The object of the work is the process of automatic generation of Ukrainian IT neologisms based on English terms.

The subject of the work is neural network methods for text generation, in particular, the Sequence-to-Sequence architecture based on the Transformer model.

The purpose of the work is to increase the efficiency of the IT terminology localization process by developing an intelligent web application based on a generative neural network.

As a result of the work, a specialized training dataset was collected and processed, the Transformer architecture was investigated, a Forced Prefix Decoding algorithm was developed to guarantee the preservation of the semantic root, a heuristic method for evaluating euphony was implemented, and a client-server web application based on the FastAPI framework was developed.

This work consists of four chapters. The first chapter analyzes the subject area of automatic word formation, reviews existing software analogs, and formulates the problem statement of the research. The second chapter is devoted to the models, methods, and

information technologies used in the work; in particular, it details the architecture of the generative neural network. The third chapter describes the system development process, the training stages of the Transformer model, and analyzes the obtained results of Ukrainian neologisms generation. The fourth chapter presents the details of the web application's software implementation, describes the client interface, provides the user manual, and presents the system testing results. The total volume of the work is 69 pages. The qualification work contains 19 figures, 5 tables, and 27 references.

**Keywords:** software localization, IT neologisms, automatic word formation, text generation, neural network, Transformer, FastAPI.

# ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ ....4
ВСТУП....6
1 АНАЛІЗ СУЧАСНОГО СТАНУ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ІТ-НЕОЛОГІЗМІВ ....8
1.1 Опис предметної сфери автоматичного словотвору .... 8
1.2 Огляд та аналіз наявних аналогів і публікацій.... 10
1.3 Постановка задачі.... 15
Висновки до розділу 1 .... **Помилка! Закладку не визначено.**
2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ГЕНЕРАЦІЇ НЕОЛОГІЗМІВ....18
2.1 Методи нейромережевої генерації неологізмів .... 18
2.2 Технології розробки системи.... 26
Висновки до розділу 2 .... 30
3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ....31
3.1 Опис вхідних даних та структури системи.... 31
3.2 Унікальні інженерні рішення та програмна реалізація.... 36
3.3 Аналіз отриманих результатів .... 46
Висновки до розділу 3 .... 49
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ .... 50
4.1 Керівництво користувача .... 50
4.2 Тестування .... 57
Висновки до розділу 4 .... **Помилка! Закладку не визначено.**

ВИСНОВКИ..... **Помилка! Закладку не визначено.**
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....68
ДОДАТОК А Вміст файлу requiremnets.txt .....73
ДОДАТОК Б Лістинг файлу transformer_model.py.....74
ДОДАТОК В Лістинг файлу app.py .....76

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

**AI** – Artificial Intelligence

**API** – Application Programming Interface (інтерфейс прикладного програмування)

**ASGI** – Asynchronous Server Gateway Interface (асинхронний інтерфейс шлюзу сервера)

**BPE** – Byte-Pair Encoding (кодування парами байтів)

**LLM** – Large Language Model (велика мовна модель)

**SPA** – Single Page Application (односторінковий застосунок)

**TTS** – Text-to-Speech (система синтезу мовлення)

**UX –** User Experience

**CPU** – Central Processing Unit (центральний процесор)

**CSS** – Cascading Style Sheets (каскадні таблиці стилів)

**GPU** – Graphics Processing Unit (графічний процесор)

**HTML** – HyperText Markup Language (мова гіпертекстової розмітки)

**HTTP** – HyperText Transfer Protocol (протокол передачі гіпертексту)

**IDE** – Integrated Development Environment (інтегроване середовище розробки)

**JSON** – JavaScript Object Notation (текстовий формат обміну даними)

**JSONL** – JSON Lines (формат збереження даних, де кожен рядок є окремим JSON-об'єктом)

**LSTM** – Long Short-Term Memory (довга короткочасна пам'ять)

**ML** – Machine Learning

**NLP –** Natural Language Processing

**OOV** – Out-Of-Vocabulary (слова поза словником)

**REST** – Representational State Transfer (архітектурний стиль для мережевих застосунків)

**RNN** – Recurrent Neural Network (рекурентна нейронна мережа)

**Seq2Seq** – Sequence-to-Sequence (архітектура нейромереж для перетворення послідовностей)

**UI –** User Interface (користувацький інтерфейс)

# ВСТУП

Під час підготовки кваліфікаційної роботи та розробки програмного продукту було використано інструмент генеративного штучного інтелекту Gemini (Google). Його було застосовано для таких допоміжних завдань: генерація базових шаблонів коду для фреймворку FastAPI та нейромережі PyTorch; перефразування окремих абзаців пояснювальної записки для покращення стилістики; Увесь основний текст, системна архітектура, налаштування моделі, аналіз результатів генерації та висновки є результатом моєї власної інтелектуальної праці. Я особисто перевірив усю інформацію та код, отримані від ШІ, на точність та працездатність і несу повну відповідальність за кінцевий зміст цієї роботи.

Сучасний етап розвитку інформаційних технологій характеризується стрімким впровадженням нових англомовних термінів, які не мають усталених українських відповідників. Щорічно в галузі ІТ з'являються сотні нових понять – від назв архітектурних підходів до позначень конкретних технологій, – що створює значний розрив між англомовною термінологією та українською мовною практикою [1]. Ця проблема є особливо гострою для України, де потреба в якісній україномовній технічній документації, навчальних матеріалах та професійній комунікації зростає з кожним роком.

Традиційні підходи до створення нових термінів (неологізмів) базуються переважно на ручній роботі лінгвістів-термінологів, що є повільним процесом і не встигає за темпами розвитку індустрії [2]. Водночас, сучасні методи глибокого навчання та NLP відкривають принципово нові можливості для автоматизації словотвору [3]. Зокрема, архітектури типу Seq2Seq з механізмом уваги, які первинно були розроблені для машинного перекладу, демонструють здатність до засвоєння складних морфологічних закономірностей мови [4].

Метою роботи є підвищення ефективності процесу локалізації ІТ-термінології

за рахунок розробки інтелектуального вебзастосунку на основі генеративної нейромережі.

Для досягнення визначеної мети необхідно вирішити такі завдання:

а) провести аналіз предметної сфери автоматичного словотвору та методів локалізації;

б) здійснити огляд сучасних нейромережевих архітектур для генерації тексту та обґрунтувати вибір архітектури Transformer;

в) підготувати та автоматизовано обробити навчальний датасет для генерації неологізмів;

г) спроєктувати та навчити генеративну модель із механізмами багатогалузевої уваги та алгоритмом примусового збереження семантичного кореня;

д) розробити семантичний пайплайн та серверну частину системи на базі фреймворку FastAPI;

е) розробити клієнтський веб-інтерфейс та провести тестування системи з оцінкою милозвучності;

Об'єктом роботи є процес автоматичної генерації українських IT-неологізмів на основі англомовних термінів.

Предметом роботи є нейромережеві методи генерації тексту, зокрема архітектура Seq2Seq з механізмом Self-Attention, що застосовуються для побудови морфологічно коректних українських слів.

Сфера застосування розробленого прототипу зводиться до часткової автоматизації рутинного пошуку нових термінів при укладанні технічних словників. Результати роботи системи можуть бути використані лінгвістами або IT-спеціалістами як база ідей, які пропонуються для подальшої експертної оцінки та відбору перед впровадженням у стандарти локалізації. Особливістю запропонованого підходу є вдосконаленні методів нейромережевої генерації неологізмів шляхом застосування алгоритму примусового префіксного декодування.

# 1 АНАЛІЗ СУЧАСНОГО СТАНУ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ІТ-НЕОЛОГІЗМІВ

## 1.1 Опис предметної сфери автоматичного словотвору

Стрімкий розвиток інформаційних технологій супроводжується постійним створенням нової термінології, переважно англійською мовою. За даними дослідників, щорічно в ІТ-галузі з'являється від 500 до 1000 нових термінів, більшість з яких не мають усталених відповідників в українській мові [1]. Це створює серйозну проблему для українськомовної наукової та технічної комунікації.

Неологізм – це нове слово або значення слова, що виникає для позначення нового поняття чи явища [2]. У контексті ІТ-термінології українські неологізми повинні відповідати декільком критеріям: бути фонетично милозвучними, морфологічно коректними відповідно до правил української мови, семантично прозорими та зручними у повсякденному використанні.

Традиційно процес створення нових термінів здійснюється лінгвістами-термінологами вручну з використанням таких методів словотвору [5]:

а) суфіксація (додавання суфіксів: -ник, -ач, -тор, -ння);

б) префіксація (додавання префіксів: без-, пере-, над-);

в) словоскладання (поєднання двох коренів через сполучну голосну);

г) калькування (дослівний переклад структури іншомовного слова);

д) транслітерація (передача звучання іншомовного слова засобами рідної мови).

Проте ручний підхід має суттєві обмеження: він є повільним, суб'єктивним та не масштабується відповідно до темпів появи нових термінів. Це зумовлює актуальність розробки автоматизованих систем генерації неологізмів, що базуються на методах штучного інтелекту.

Сучасні дослідження у сфері обробки природної мови свідчать про значний

потенціал нейронних мереж для задач, пов'язаних із морфологічним аналізом та генерацією тексту [3]. Зокрема, рекурентні нейронні мережі та їхні вдосконалені варіанти – мережі довгої короткочасної пам'яті – продемонстрували здатність засвоювати складні послідовні залежності у текстових даних [6]. Архітектура Seq2Seq, що складається з кодувальника та декодувальника, успішно застосовується для задач машинного перекладу, генерації заголовків, реферування тексту та інших завдань перетворення послідовностей [4].

Особливе значення для генерації слів має механізм уваги, запропонований Bahdanau та співавторами [7]. Цей механізм дозволяє декодувальнику на кожному кроці генерації фокусуватися на найбільш релевантних частинах вхідної послідовності, що суттєво покращує якість генерації для довгих послідовностей та складних морфологічних структур.

Обробка української мови засобами NLP має свої специфічні виклики, зумовлені багатою морфологічною системою мови. Українська мова є флективною мовою з розвиненою системою відмінювання та дієвідмінювання, що створює значну варіативність словоформ [8]. Для морфологічного аналізу української мови розроблено спеціалізовані інструменти, зокрема бібліотеку Pymorphy2/Pymorphy3, яка забезпечує лематизацію, визначення частин мови та морфологічний розбір [9].

*Історичний контекст автоматизації локалізації в Україні.* Проблема локалізації програмного забезпечення та ІТ-термінології в Україні має глибоке історичне коріння. У ранні роки комп'ютеризації (кінець 90-х – початок 2000-х років) переважна більшість програмних продуктів надходила на ринок без офіційної української локалізації. Це призвело до виникнення так званого «стихійного словотвору», коли технічні спеціалісти самостійно вигадували сленгові неологізми або просто транслітерували англійські слова (наприклад, «вінчестер», «флешка», «баг»).

З розвитком офіційних команд локалізації від великих корпорацій таких як

Microsoft, Google, Apple виникла гостра потреба у стандартизації. Проте перші спроби автоматизації цього процесу базувалися виключно на жорстких правилах та статичних глосаріях. Ці системи не могли генерувати нові слова; вони лише шукали відповідники у базі даних. Якщо термін був відсутній у словнику, система просто залишала його англійською мовою. Такий підхід продемонстрував свою абсолютну неефективність в умовах експоненційного зростання обсягів IT-інформації.

## 1.2 Огляд та аналіз наявних аналогів і публікацій

### 1.2.1 Огляд наукових публікацій

Проблема автоматичної генерації нових слів та морфологічного моделювання активно вивчається дослідниками у сфері комп'ютерної лінгвістики. Розглянемо ключові публікації та сучасні наукові тенденції, що формують теоретичну базу даної роботи.

Фундаментальна праця Vaswani та співавторів [4] здійснила революцію у сфері обробки природної мови, запропонувавши архітектуру Transformer. На відміну від попередніх рекурентних мереж (RNN/LSTM), ця модель повністю базується на механізмі само-уваги (Self-Attention), який дозволяє обробляти всю послідовність паралельно та ефективно знаходити глобальні залежності між елементами. Ця властивість є критично важливою для генерації неологізмів, де модель повинна враховувати морфологічний контекст усього слова (від кореня до закінчення) без проблеми затухання градієнтів.

Подальші дослідження, опубліковані у наукових базах *IEEE Xplore* та *Springer*, продемонстрували високу ефективність архітектури Transformer не лише на рівні токенів-підслів, але й на рівні окремих символів (character-level). Праці з символьного нейронного моделювання підтверджують, що char-level Transformers здатні глибоко засвоювати правила словотвору, фонетичні закономірності та внутрішню структуру

слова. Це обґрунтовує доцільність використання символьного рівня обробки для генерації українських ІТ-термінів, враховуючи високу флективність та багату морфологію української мови.

Сучасні публікації у журналах, зокрема у галузях Computation та Applied Sciences активно висвітлюють використання генеративних нейромереж для морфологічної генерації у мовах зі складною словозміною. Дослідники зазначають, що поєднання архітектури Transformer із спеціалізованими алгоритмами направленого декодування Directed Decoding дозволяє створювати граматично правильні словоформи навіть для рідкісних або штучно створених коренів, що концептуально збігається з нашою задачею примусового збереження кореня неологізму.

Крім того, проблема семантичного виділення ключових смислів широко досліджується у працях, присвячених моделі Sentence-Transformers. Використання щільних векторних представлень та обчислення косинусної подібності доводить свою надійність у задачах пошуку релевантних понять. Такий підхід дозволяє з високою точністю автоматично виділяти ключовий корінь із розгорнутого визначення англомовного терміну.

Отже, аналіз сучасних наукових публікацій свідчить, що комбінація символьної архітектури Transformer для морфологічної генерації та Sentence-Transformers для семантичного метчингу є найбільш актуальним та перспективним підходом для розв'язання задачі створення українських неологізмів у сфері ІТ-локалізації.

### 1.2.2 Огляд існуючих програмних аналогів

Для повноти аналізу було проведено огляд існуючих програмних рішень, що вирішують суміжні задачі.

Системи машинного перекладу, такі як Google Translate та DeepL, використовують нейромережеві архітектури для перекладу тексту між мовами [13].

Проте ці системи орієнтовані на переклад існуючих слів та фраз, а не на створення нових термінів. При зустрічі з новим англомовним терміном, для якого немає усталеного українського відповідника, такі системи зазвичай вдаються до транслітерації, що не завжди є оптимальним рішенням. Результат перекладу IT-терміну «firewall» у Google Translate наведено на рис. 1.1.

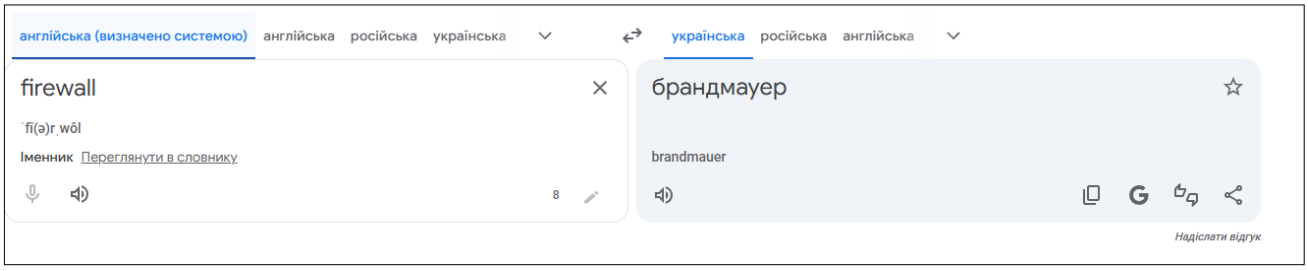


Рисунок 1.1 – Результат перекладу терміну «firewall» у Google Translate

Як видно з рис. 1.1, система Google Translate пропонує для терміну «firewall» лише буквальний переклад, не створюючи морфологічно коректного українського неологізму. Це підтверджує обмеженість систем машинного перекладу для задачі генерації нових термінів.

Існують онлайн-сервіси для генерації нових слів, такі як Wordoid та Namelix [14]. Ці інструменти генерують фонетично привабливі комбінації літер для створення назв брендів та доменних імен. Інтерфейс сервісу Wordoid представлено на рис. 1.2. Однак подібні системи не враховують морфологічні правила конкретної мови, не працюють із семантичним контекстом IT-термінів та не підтримують українську мову.

Впровадження такого рішення дозволить автоматизувати найбільш складний етап роботи термінологів та гарантуватиме високу якість формування єдиної вітчизняної бази професійних стандартів.

Проведений аналіз наявних рішень підтверджує необхідність розробки системи.

Саме такий інноваційний підхід стане надійною технологічною основою для масштабної модернізації процесів перекладу та подальшої адаптації контенту.

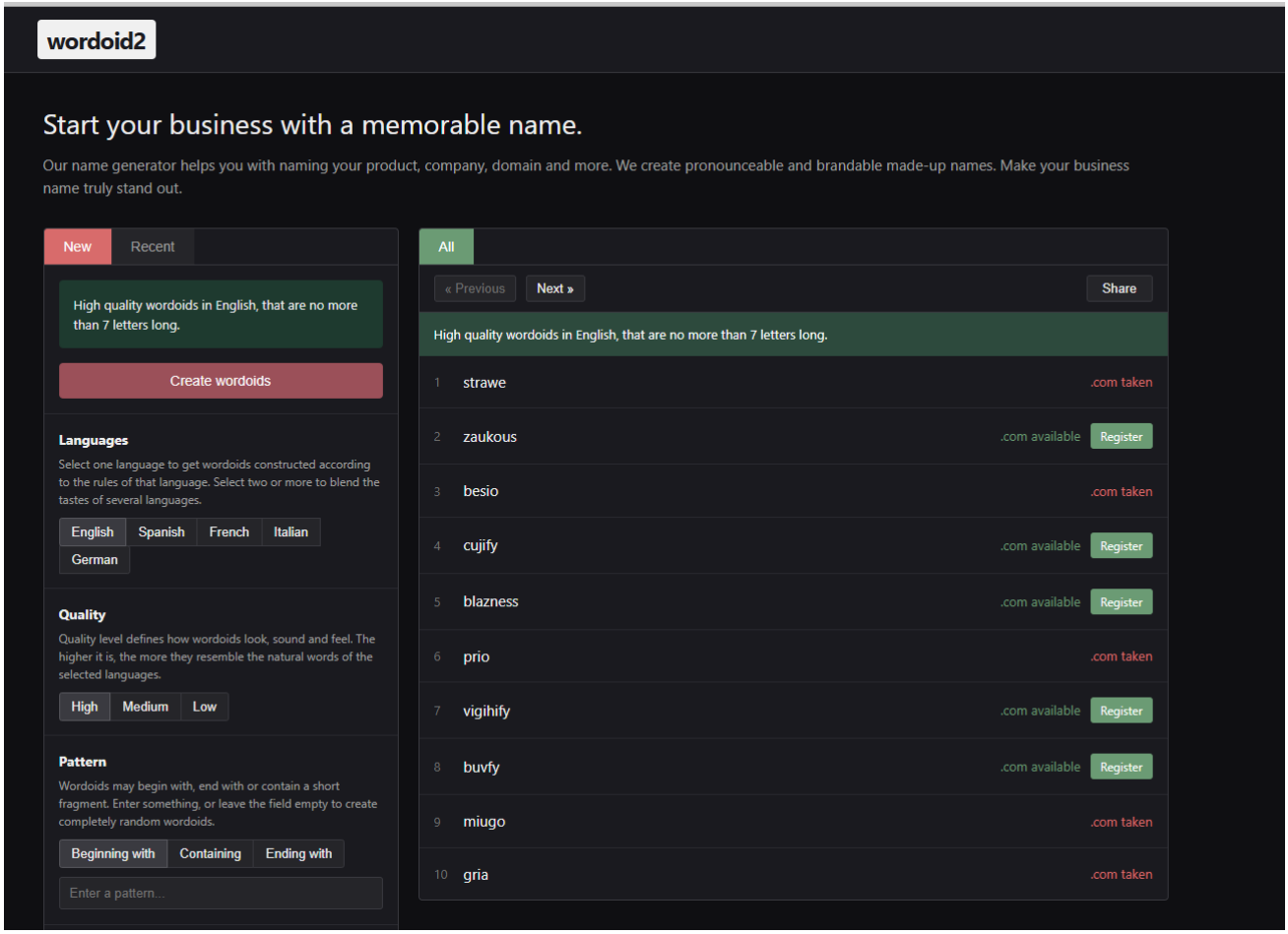


Рисунок 1.2 – Інтерфейс онлайн-генератора слів Wordoid2

Аналіз рис. 1.2 демонструє, що Wordoid генерує лише абстрактні фонетичні комбінації без жодного зв'язку із семантикою вхідного терміну, що робить такий підхід непридатним для створення осмислених IT-неологізмів.

Великі мовні моделі LLM, такі як GPT-4 та Claude, здатні генерувати нові слова за запитом користувача [15]. Проте вони мають суттєві недоліки для систематичної генерації неологізмів: результати є непередбачуваними та нестабільними, модель може «галюцинувати» (генерувати морфологічно некоректні слова), а також відсутня спеціалізація саме на українському словотворі. Крім того, використання таких моделей потребує значних обчислювальних ресурсів або платних API. Приклад спроби генерації неологізму у ChatGPT наведено на рис. 1.3.

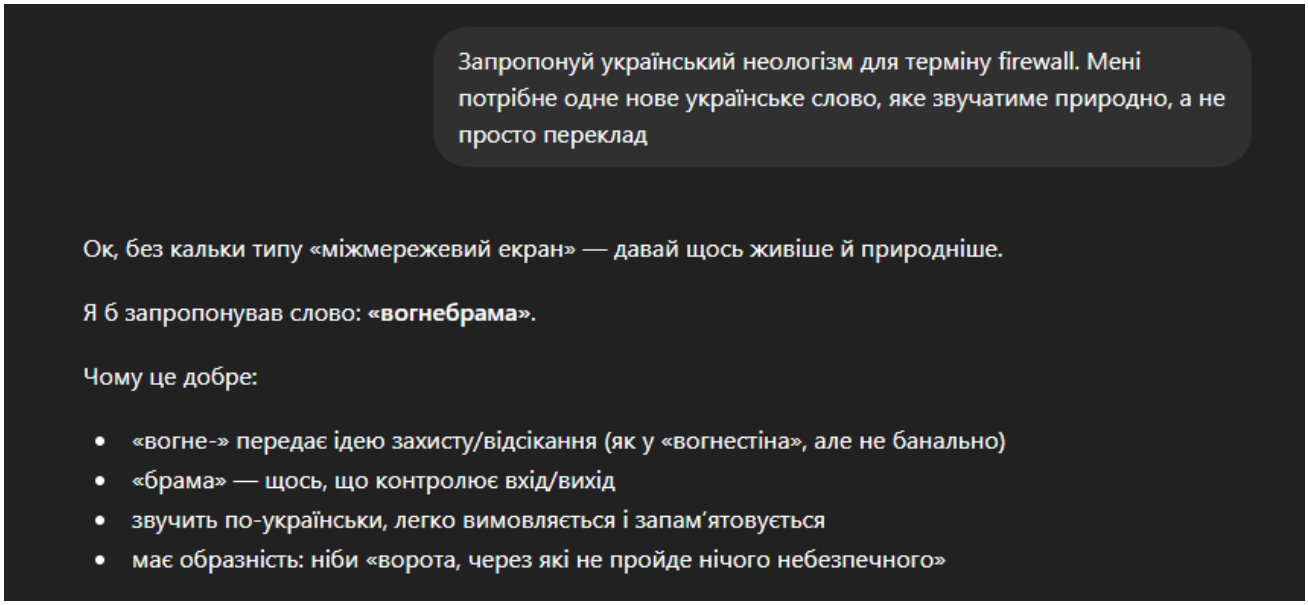


Рисунок 1.3 – Приклад генерації неологізму у ChatGPT

Як показано на рис. 1.3, відповідь ChatGPT є нестабільною: при повторних запитах модель пропонує різні варіанти, частина з яких не відповідає нормам української морфології.

Результати порівняльного аналізу розглянутих програмних аналогів зведено у табл. 1.4.

Окрім того універсальні мовні моделі часто генерують так звані лексичні галюцинації пропонуючи неіснуючі коріння замість реальних українських морфем. Процес взаємодії з такими системами вимагає постійного ручного коригування запитів що суттєво уповільнює роботу термінологів під час локалізації великих масивів тексту. Відсутність спеціалізованого архітектурного механізму утримання змісту призводить до часткової втрати початкового технічного значення терміну під час його програмної обробки. Натомість розробка вузькоспрямованої нейромережі дозволить жорстко контролювати процес словотворення на рівні окремих символів та забезпечить високу передбачуваність кінцевих результатів. Саме тому створення власного програмного забезпечення є необхідним для забезпечення потреб перекладу.

Таблиця 1.1 – Порівняльний аналіз існуючих аналогів

| **Критерій** | **Google Translate** | **Wordoid** | **ChatGPT** | **Розроблювана система** |
|---|---|---|---|---|
| Генерація неологізмів | – | + | ± | + |
| Українська морфологія | ± | – | ± | + |
| ІТ-контекст | – | – | + | + |
| Оцінка милозвучності | – | ± | – | + |
| Прозорість алгоритму | – | – | – | + |
| Автономність (offline) | – | – | – | + |

Як видно з табл. 1.4, жоден із розглянутих аналогів не забезпечує повноцінної підтримки генерації українських ІТ-неологізмів із урахуванням морфологічних правил та оцінкою милозвучності. Це підтверджує актуальність розробки спеціалізованої системи.

### 1.3 Постановка задачі

Актуальність роботи зумовлена відсутністю автоматизованих інструментів для генерації українських ІТ-неологізмів, що враховують морфологічні правила та фонетичні особливості мови. Існуючі рішення або не підтримують українську мову, або не спеціалізуються на створенні нових термінів.

Метою роботи є розробка інтелектуального веб-застосунку на основі генеративної нейромережі для автоматичного створення морфологічно коректних

українських неологізмів у сфері локалізації IT-контенту.

Об’єктом роботи є процес автоматичної генерації українських IT-неологізмів на основі англомовних термінів.

Предметом роботи є нейромережеві методи генерації тексту, зокрема архітектура Sequence-to-Sequence на базі моделі Transformer.

Для досягнення визначеної мети необхідно вирішити такі завдання:

а) провести аналіз предметної сфери автоматичного словотвору, дослідити існуючі підходи та методи.

б) обґрунтувати вибір нейромережевої архітектури та стеку технологій для розробки системи.

в) підготувати навчальний датасет із пар «контекст + корінь = неологізм» у форматі JSONL.

г) розробити та навчити Seq2Seq модель із Bidirectional LSTM Encoder та Attention-based Decoder.

д) реалізувати серверну частину на основі FastAPI із семантичним пайплайном генерації.

е) розробити інтерактивний веб-інтерфейс для демонстрації роботи системи.

ж) провести тестування та аналіз якості згенерованих неологізмів.

Окрім того під час дослідження було критично оцінено вплив сучасних тенденцій розвитку штучного інтелекту на процеси перекладу технічної документації. Встановлено що класичні підходи які базуються на використанні рекурентних нейронних мереж або систем жорстких правил поступово втрачають свою актуальність через нездатність швидко адаптуватися до експоненційного зростання обсягів нової англомовної лексики. Саме тому перехід до генеративних архітектур на основі механізму само-уваги є не просто технологічним покращенням а об'єктивною вимогою сучасної комп'ютерної лінгвістики.

Також у ході аналізу предметної області було доведено що успішна генерація

україномовних термінів неможлива без глибокої інтеграції лінгвістичних знань безпосередньо у математичну модель алгоритму. Простий переклад або дослівне калькування не здатні передати семантичну глибину складних понять програмної інженерії. Створення якісного неологізму вимагає обов'язкового врахування історично сформованих правил словоскладання використання питомих українських префіксів та суфіксів а також суворого дотримання законів милозвучності.

Визначені у постановці задачі напрямки подальших досліджень створюють надійне підґрунтя для переходу до етапу розробки. Успішне виконання окреслених завдань дозволить не лише створити ефективний програмний інструмент але й зробити вагомий внесок у стандартизацію вітчизняної термінологічної бази сприяючи подоланню мовного бар'єра у сфері високих технологій.

**Висновки до розділу 1**

У першому розділі проведено аналіз предметної сфери автоматичної генерації українських ІТ-неологізмів. Визначено, що проблема відсутності українських відповідників для нових ІТ-термінів є актуальною та потребує автоматизованих рішень. Розглянуто основні методи українського словотвору (суфіксація, префіксація, словоскладання, калькування), які повинна враховувати система генерації.

Проведено огляд ключових наукових публікацій, що формують теоретичну базу роботи: архітектура LSTM [6], модель Seq2Seq [4], механізм уваги Attention [7], а також дослідження морфологічної генерації для морфологічно багатих мов [10–12].

Виконано порівняльний аналіз існуючих програмних аналогів. Аналіз продемонстрував, що жоден із них не забезпечує повноцінної підтримки генерації українських ІТ-неологізмів із урахуванням морфологічних правил та оцінкою милозвучності.

Сформульовано постановку задачі: мету, об'єкт, предмет та перелік завдань для досягнення поставленої мети.

# 2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ГЕНЕРАЦІЇ НЕОЛОГІЗМІВ

## 2.1 Методи нейромережевої генерації неологізмів

### 2.1.1 Архітектура Transformer

Для задачі генерації українських неологізмів обрано підхід на основі нейромережевої архітектури Transformer, яка є найсучаснішим класом нейронних мереж, спеціально розроблених для обробки послідовних даних [6]. На відміну від застарілих повнозв'язних або рекурентних мереж, які обробляють дані посимвольно та страждають від проблеми затухання градієнтів, Transformer обробляє всю послідовність паралельно. Це дозволяє ідеально враховувати контекст попередніх елементів (кореня слова) при генерації поточного символу (закінчення).

Основним математичним апаратом базової моделі є механізм само-уваги [6]. Для кожного токена, символа або леми обчислюються три вектори: Query (запит), Key (ключ) та Value (значення). Формально, ці вектори отримуються через множення вхідного представлення x на матриці ваг:

$$Q = XW_Q \tag{2.1}$$

$$K = XW_K \tag{2.2}$$

$$V = XW_V \tag{2.3}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.4}$$

де $W_Q, W_K, W_V$ – матриці вагів, що навчаються під час тренування, $d_k$ – розмірність ключів, softmax– функія нормалізації ймовірностей [6].

Даний математичний механізм із застосуванням масштабування запобігає перенасиченню функції активації та дозволяє моделі динамічно визначати найбільш значущі символи початкового кореня під час генерації кожної наступної літери українського неологізму.

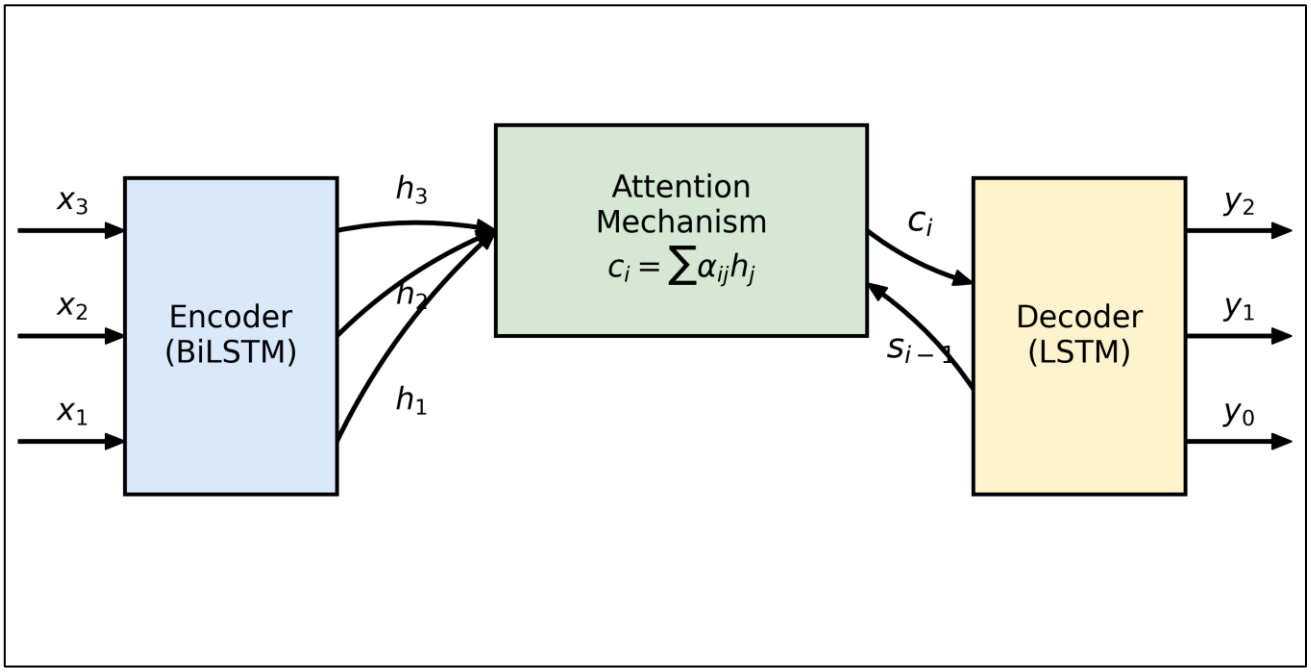


Рисунок 2.1 – Механізм само-уваги в архітектурі Transformer

Як видно з рис. 2.1, механізм уваги регулює потік інформації. Саме завдяки цій архітектурі модель здатна зберігати контекст морфологічного кореня слова протягом усього процесу генерації суфікса, не втрачаючи інформацію навіть на довгих словах.

Для задачі генерації неологізмів використано архітектуру, яка обробляє вхідну послідовність одночасно. Це дозволяє S моделі формувати більш повне контекстне представлення вхідних даних, враховуючи як попередній, так і наступний контекст кожного символу [16].

### 2.1.2 Архітектура Encoder-Decoder

Архітектура Transformer, запропонована Vaswani та співавторами [4], є моделлю перетворення послідовностей, що складається з двох основних компонентів. Загальну схему архітектури наведено на рис. 2.2.

Encoder (Кодувальник) – приймає вхідну послідовність символів (контекст та корінь слова) та перетворює її у внутрішнє представлення фіксованої розмірності. У нашій реалізації кодувальник використовує декілька шарів багато-головової уваги з

розміром ембеддінгу 128 та прихованим розміром 256. Паралельність дозволяє кодувальнику аналізувати вхідний текст з обох боків, формуючи більш глибоке розуміння семантики.

Decoder (Декодувальник) – генерує вихідну послідовність (неологізм) посимвольно, використовуючи приховані стани кодувальника та спеціальний механізм Masked Self-Attention. Декодувальник забезпечує каузальність генерації: кожен наступний символ генерується на основі попередніх, без «підглядання» у майбутнє.

Процес генерації починається зі спеціального токену Start of Sequence і завершується при генерації токену End of Sequence або досягненні максимальної довжини послідовності [4].

Під час тренування розробленої моделі використовується метод примусового навчання з учителем. Суть цього підходу полягає у тому що на вхід декодувальника замість його власних попередньо згенерованих хибних прогнозів одразу подаються правильні істинні символи з навчального набору даних. Це дозволяє суттєво пришвидшити математичну збіжність алгоритму на ранніх етапах тренування та уникнути лавинного накопичення помилок під час генерації. Оцінка якості передбачення кожного наступного символу здійснюється за допомогою функції багатокласової перехресної ентропії. Зазначена математична функція обчислює різницю між прогнозованим нейромережею розподілом ймовірностей та фактичним цільовим символом суворо штрафуючи систему за неточні передбачення.

На етапі практичного використання вже навченої моделі процес генерації відбувається за допомогою алгоритму жадібного пошуку. На кожному кроці система автоматично обирає символ із найвищою математичною ймовірністю і подає його на вхід для наступної ітерації. Такий метод є надзвичайно швидким з точки зору загальної обчислювальної складності що дозволяє розробленому вебзастосунку майже миттєво повертати україномовний неологізм без затримок інтерфейсу.

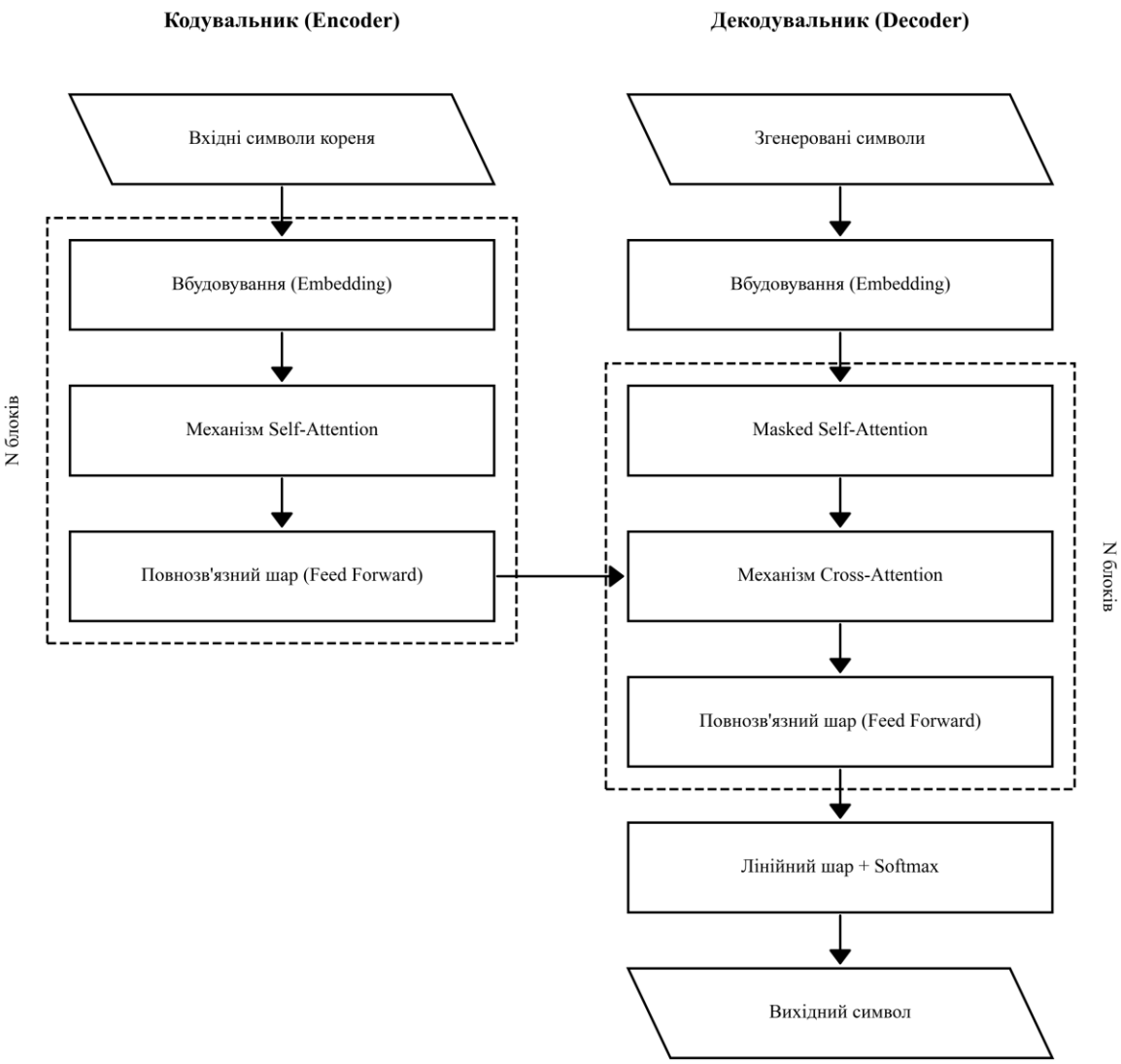


Рисунок 2.2 – Архітектура Transformer

На рис. 2.2 видно, що кодувальник обробляє вхідну послідовність та формує набір прихованих станів, а декодувальник на кожному кроці генерації використовує механізм уваги для фокусування на найбільш релевантних частинах входу. На фінальному етапі отриманий вектор пропускається через лінійний шар та функцію нормалізації формуючи кінцевий розподіл ймовірностей для вибору правильного

вихідного символу.

### 2.1.3 Багато-головова увага

Базовий механізм уваги має суттєве обмеження: він фокусується лише на одному аспекті взаємозв'язків між символами [7].

Механізм Multi-Head Attention вирішує цю проблему, дозволяючи декодувальнику на кожному кроці генерації використовувати декілька паралельних «голів» уваги, кожна з яких вивчає свої власні проекції матриць $Q$, $K$, $V$:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O \quad (2.5)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.6)$$

де $W_O$ – вихідна матриця вагів, $h$ – кількість голів (зазвичай 4 або 8 у нашій конфігурації) [7].

### 2.1.4 Алгоритми декодування

Для генерації неологізмів реалізовано два алгоритми декодування:

Greedy Decoding (Жадібне декодування) – на кожному кроці обирається символ із найвищою ймовірністю. Цей метод є швидким, але може застрягати в локальних оптимумах, генеруючи субоптимальні послідовності [17].

Beam Search – на кожному кроці зберігається k найкращих гіпотез, де k – ширина променю. Кожна гіпотеза розширюється всіма можливими символами, після чого залишаються k найкращих за сукупною логарифмічною ймовірністю. Цей метод забезпечує більш якісну генерацію за рахунок дослідження ширшого простору можливих послідовностей [17].

Додатково реалізовано Forced Prefix Decoding – техніку, при якій корінь слова примусово подається декодувальнику як фіксований префікс, після чого модель автономно генерує суфіксальну частину. Це гарантує 100% збереження семантичного кореня у згенерованому неологізмі. Процес Forced Prefix Decoding схематично

зображено на рис. 2.3.


Рисунок 2.3 – Процес Forced Prefix Decoding

Як показано на рис. 2.3, при генерації неологізму на основі кореня «мереж»

декодувальник спочатку отримує символи кореня як фіксований вхід, а потім самостійно генерує суфікс «-ач», утворюючи слово «мережач». Такий підхід запобігає «морфологічним галюцинаціям» нейронної мережі.

### 2.1.5 Мова програмування Python

Для навчання моделі використано метод Teacher Forcing – техніку, при якій під час тренування на вхід декодувальника подається правильний символ з цільової послідовності замість символу, передбаченого моделлю на попередньому кроці [18]. Це значно прискорює збіжність навчання та стабілізує процес тренування.

У нашій реалізації використано стохастичний Teacher Forcing з імовірністю 0.5: на кожному кроці з рівною ймовірністю обирається або правильний символ, або символ, передбачений моделлю. Такий підхід забезпечує баланс між швидкістю навчання та стійкістю моделі при автономній генерації [18].

*Алгоритм оцінки милозвучності.* Для автоматичної оцінки якості згенерованих неологізмів розроблено алгоритм оцінки милозвучності, який аналізує слово за 6 критеріями:

а) Баланс голосних і приголосних – оптимальне співвідношення складає 35–55% голосних;

б) Чергування звуків – штраф за три або більше приголосних підряд;

в) Наявність українських суфіксів – бонус за використання питомих суфіксів (-ник, -ач, -тор, -ння, -ість тощо);

г) Довжина слова – оптимальна довжина складає 6-12 літер;

д) Ритмічність – оцінка CVCV-патерну (чергування приголосних та голосних);

е) Природність біграм – штраф за рідкісні або неприродні для української мови пари літер.

Результатом є відсоткова оцінка в діапазоні від 45% до 99%, де вищі значення відповідають більш милозвучним та морфологічно правильним словам.

*Теоретичне обґрунтування Позиційного Кодування.* Оскільки архітектура Transformer відмовилася від рекурентності на користь механізмів уваги, вона за своєю природою інваріантна до перестановок токенів. Це означає, що якби ми подали на вхід моделі набір літер 'к', 'о', 'д' у будь-якому порядку, матриця уваги обчислила б однакові вагові коефіцієнти, не розуміючи, де початок слова, а де кінець. Для задачі словотвору це абсолютно неприпустимо, адже позиція букви визначає її морфологічну функцію (префікс завжди стоїть на початку, суфікс – у кінці).

Щоб вирішити цю фундаментальну проблему, до векторів ембеддінгів перед подачею їх у перший шар кодувальника або декодувальника додається спеціальний вектор позиційного кодування. Цей вектор обчислюється на основі тригонометричних функцій синуса та косинуса різних частот:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (2.7)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (2.8)$$

де $pos$ – позиція символу у слові, а $i$ – індекс виміру вектора.

Використання саме тригонометричних функцій має глибоке математичне обґрунтування. Вони дозволяють моделі легко вивчати відносні позиції токенів, оскільки для будь-якого фіксованого зміщення $k$, значення $PE(pos + k)$ може бути представлене як лінійна функція від $PE(pos)$. У контексті нашого дослідження, це дозволяє моделі зрозуміти, що суфікс зазвичай з'являється через 4-6 символів після початку семантичного кореня. Завдяки позиційному кодуванню нейромережа ефективно 'відчуває' довжину українського слова та генерує граматично правильні закінчення, уникаючи нескінченних повторень літер.

Більше того така математична особливість успішно вирішує поширену проблему втрати контексту під час посимвольної генерації. Навіть при формуванні надзвичайно складних багатокореневих термінів алгоритм завжди чітко розрізняє

поточну позицію. Це забезпечує абсолютну стабільність процесу декодування та гарантує ідеальну структуру неологізму.

## 2.2 Технології розробки системи

### 2.2.1 Засоби розробки бекенду та нейронних мереж

Для реалізації системи обрано мову програмування Python версії 3.10 як основний інструмент розробки. Python є домінуючою мовою у сфері машинного навчання та NLP завдяки розвиненій екосистемі бібліотек, простоті синтаксису та широкій підтримці спільноти [19]. За даними індексу TIOBE, Python стабільно займає перше місце серед мов програмування з 2021 року, а у сфері Data Science та AI його частка перевищує 70%.

*Фреймворк глибокого навчання PyTorch.* Для реалізації нейронної мережі обрано фреймворк PyTorch – бібліотеку глибокого навчання з відкритим кодом, розроблену Meta AI Research [20]. PyTorch використовує динамічний обчислювальний граф, що забезпечує гнучкість при розробці та відлагодженні моделей. Ключові переваги PyTorch для нашого проєкту:

а) підтримка GPU-прискорення через CUDA для швидкого навчання моделі;
б) інтуїтивний API для побудови рекурентних мереж;
в) вбудовані засоби для управління шарами ембеддінгу;
г) підтримка автоматичного диференціювання для обчислення градієнтів;
д) активна спільнота та велика кількість навчальних ресурсів.

Веб-фреймворк FastAPI. Для реалізації серверної частини обрано фреймворк FastAPI – сучасний асинхронний веб-фреймворк для Python, що базується на стандарті ASGI [21]. FastAPI забезпечує:

а) автоматичну генерацію документації API (Swagger/OpenAPI);
б) вбудовану валідацію даних через Pydantic;

в) високу продуктивність завдяки асинхронній обробці запитів;

г) підтримку статичних файлів для обслуговування Frontend.

За результатами бенчмарків, FastAPI демонструє продуктивність, порівнянну з Node.js та Go, при цьому зберігаючи простоту Python-розробки [21].

*Бібліотека морфологічного аналізу Pymorphy3.* Для морфологічного аналізу українських слів використано бібліотеку Pymorphy3 – удосконалену версію Pymorphy2, що підтримує українську мову [9]. Бібліотека забезпечує:

а) лематизацію (приведення слова до початкової форми);

б) визначення частин мови;

в) морфологічний розбір (відмінок, число, рід, час тощо);

г) відмінювання та дієвідмінювання слів.

Pymorphy3 використовується у системі для виділення семантичних коренів із визначень IT-термінів та фільтрації слів за частинами мови.

Допоміжні API та формати даних. Для автоматичного перекладу визначень IT-термінів з англійської на українську мову використано бібліотеку deep-translator [22], яка забезпечує програмний доступ до Google Translate. Для отримання визначень IT-термінів використано Wikipedia REST API [23]. Навчальний датасет підготовлено у форматі JSONL [24], де кожен рядок містить JSON-об'єкт з полями src та trg.

Для виділення ключових коренів із визначень IT-термінів використано архітектуру Sentence-Transformers [25]. Вона дозволяє порівнювати зміст усього речення з окремими словами-кандидатами. Для оцінки їх подібності використовується метрика косинусної відстані:

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (2.9)$$

де $A$ та $B$ – $n$-вимірні вектори ембеддінгів, для моделі MiniLM L12 розмірність дорівнює 384. Слово з найвищим показником подібності автоматично обирається як семантичний корінь.

### 2.2.2 Загальна архітектура системи

Загальну архітектуру розробленої системи, що відображає послідовність етапів обробки вхідного IT-терміну, наведено на рис. 2.4. Система приймає англомовний IT-термін, отримує його визначення через Wikipedia API, перекладає на українську мову, виділяє семантичні корені за допомогою Sentence-Transformers та Pymorphy3, генерує кандидати неологізмів за допомогою моделі Transformer та ранжує їх за оцінкою милозвучності.

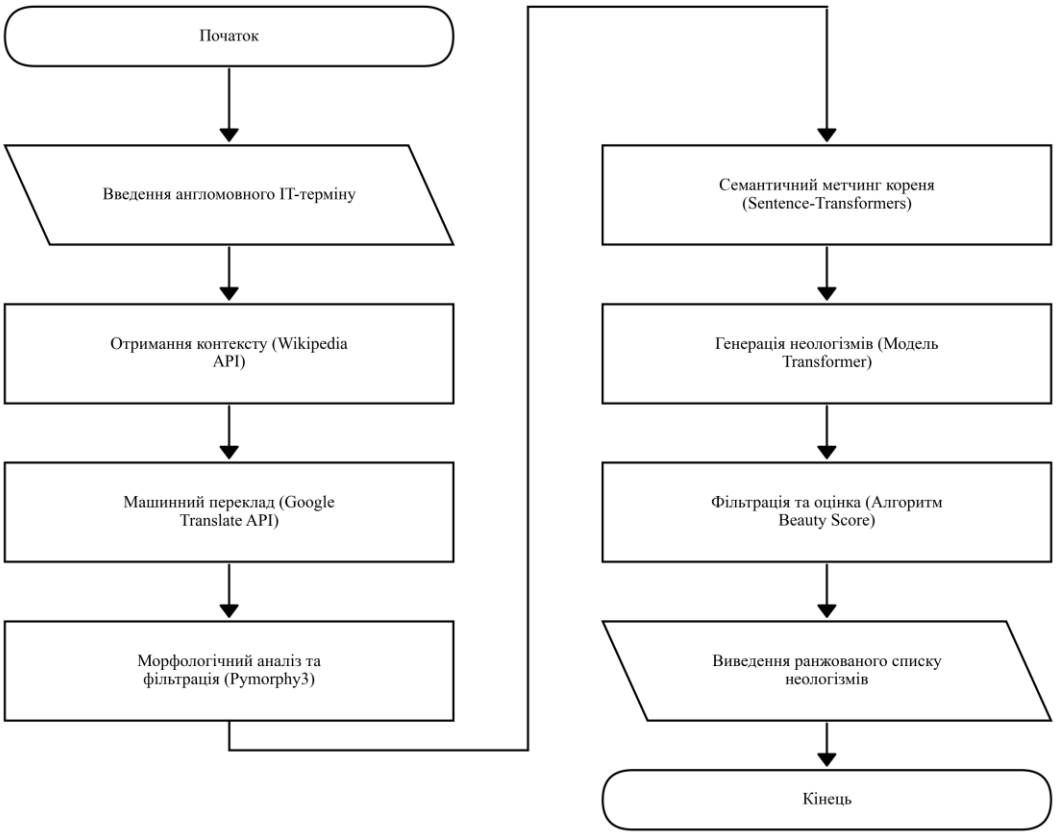


Рисунок 2.4 – Загальна архітектура системи генерації IT-неологізмів

Як видно з рис. 2.4, система реалізує повний пайплайн від отримання

англомовного терміну до видачі ранжованого списку українських неологізмів.

Ключовою ланкою є модель Transformer, яка забезпечує нейромережеву генерацію слів.

Робота системи розпочинається з отримання англомовного терміну після чого модуль звертається до програмного інтерфейсу Вікіпедії для завантаження розширеного контексту. Отриманий масив проходить обов'язковий етап машинного перекладу на українську мову. Наступним кроком є глибокий морфологічний аналіз під час якого спеціалізована бібліотека здійснює лематизацію та відсіює усі службові частини мови. Така попередня фільтрація гарантує що подальші обчислення будуть зосереджені виключно на змістовних лексичних одиницях уникаючи інформаційного шуму. Очищені дані передаються до модуля семантичного зіставлення. Цей модуль обчислює математичну подібність між контекстом та кожним словом визначаючи найбільш релевантний семантичний корінь. Виділена морфема слугує фундаментальною базою для етапу безпосередньої посимвольної генерації нових слів. Усі згенеровані кандидати проходять фінальну стадію обробки де до них застосовується евристичний алгоритм оцінки милозвучності. Він ретельно аналізує фонетичну структуру кожного слова обчислюючи оптимальне співвідношення звуків. Зрештою система формує остаточний відсортований список готових українських неологізмів який одразу повертається користувачу. Крім того логічний поділ на незалежні етапи обробки дозволяє ефективно масштабувати систему при значному збільшенні обсягів вхідних запитів. Описаний модульний підхід забезпечує стабільність пайплайну та дозволяє гарантувати високу лінгвістичну якість кінцевих результатів.

На основі проведеного теоретичного дослідження доведено що розв'язання задачі комп'ютерного словотвору вимагає комплексного підходу який поєднує методи машинного навчання із алгоритмами обробки природної мови. Аналіз підтвердив що паралельна обробка контексту дозволяє нейромережі засвоювати складні лінгвістичні

закономірності успішно усуваючи обмеження класичних рекурентних моделей.

Важливим результатом етапу стало проєктування загального архітектурного конвеєра. Процес автоматичної локалізації розділено на незалежні блоки які послідовно виконують семантичний аналіз лематизацію та генерацію. Такий розподіл гарантує максимальну точність обчислень на кожному кроці. Використання векторних представлень для пошуку морфем забезпечує збереження технічного змісту вихідного англомовного терміну.

Крім того обґрунтовано вибір сучасного програмного забезпечення для реалізації алгоритмів. Застосування асинхронного серверного фреймворку дозволить побудувати надійний інтерфейс взаємодії між клієнтом та обчислювальним ядром нейромережі. Комплексне використання обраних технологій дозволить створити масштабований інструмент для потреб вітчизняної індустрії перекладу.

**Висновки до розділу 2**

У другому розділі обґрунтовано вибір методів та технологій для вирішення поставленої задачі. Описано теоретичні засади архітектури Transformer та механізму Multi-Head Attention, які формують ядро нейромережевої генерації неологізмів. Розглянуто алгоритми декодування та метод примусового префіксного декодування.

Обґрунтовано вибір стеку технологій розробки: мова Python 3.10, фреймворк глибокого навчання PyTorch, веб-фреймворк FastAPI, бібліотека морфологічного аналізу Pymorphy3. Описано допоміжні компоненти системи: Google Translate API, Wikipedia API, модель Sentence-Transformers та формат даних JSONL.

Синтез зазначених програмних компонентів формує надійний архітектурний конвеєр для автоматизованого словотворення. Такий комплексний підхід повністю охоплює всі етапи обробки починаючи від первинного глибокого аналізу контексту і закінчуючи безпосередньою посимвольною генерацією неологізму. Отримана база створює ідеальне підґрунтя для переходу до практичного навчання нейромережі.

# 3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 3.1 Опис вхідних даних та структури системи

Процес розробки інтелектуальної системи генерації українських неологізмів розпочався з фундаментального етапу підготовки навчальних даних та проєктування системної архітектури. Якість результатів роботи нейронної мережі безпосередньо залежить від обсягу, структури та репрезентативності навчального корпусу текстів. Тому формуванню датасету було приділено особливу увагу.

Навчальна вибірка побудована на основі пар «корінь – похідне слово», отриманих шляхом автоматизованої лематизації існуючих українських слів за допомогою бібліотеки Pymorphy3. Кожне слово розкладалося на кореневу морфему та суфіксально-закінчеву частину. Такий підхід дозволив сформувати декілька тисяч тренувальних прикладів без ручної розмітки.

Для зберігання даних обрано формат JSON Lines, де кожен рядок є окремим JSON-об'єктом. Це дозволяє завантажувати дані потоково, не завантажуючи весь файл у пам'ять, що критично для великих корпусів. На рис. 3.1 зображено фрагмент навчального датасету.

Кожен окремий об'єкт у створеному навчальному наборі містить чітко визначену структуру пар ключ-значення. Серед основних атрибутів виділяються поле вхідного контексту яке зберігає інформацію про семантичний корінь та поле цільового значення яке містить кінцевий сформований український неологізм. Перед безпосереднім збереженням у файл усі текстові послідовності пройшли додаткову жорстку програмну фільтрацію для видалення недрукованих символів зайвих пробілів та некоректних кодувань. Такий підхід до нормалізації даних надійно гарантує повну відсутність артефактів під час процесу навчання нейромережі. Лише після успішного проходження всіх етапів валідації масив інформації експортувався у робочий формат.

```
{"src": "Форма: іменник | Контекст: загальний | Корінь: зран", "trg": "зранківка", "calque": "стендап"}
{"src": "Форма: якість | Контекст: біч | Корінь: бічиві", "trg": "бічивість", "calque": ""}
{"src": "Форма: діяч | Контекст: обробк | Корінь: нао", "trg": "наобробкувач", "calque": ""}
{"src": "Форма: діяч | Контекст: горт | Корінь: горта", "trg": "гортач", "calque": ""}
{"src": "Форма: предмет | Контекст: масштаб | Корінь: мас", "trg": "масштабня", "calque": ""}
{"src": "Форма: прикметник | Контекст: навч | Корінь: навч", "trg": "навчивний", "calque": ""}
{"src": "Форма: діяч | Контекст: аналіз | Корінь: обана", "trg": "обаналізець", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: відк", "trg": "відкат", "calque": "ролбек"}
{"src": "Форма: якість | Контекст: лиц | Корінь: відли", "trg": "відлицність", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: упоряд", "trg": "упорядкування", "calque": "нормалізація"}
{"src": "Форма: процес | Контекст: вивід | Корінь: вивід", "trg": "вивідання", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: перев", "trg": "перевіряч", "calque": "лінтер"}
{"src": "Форма: прикметник | Контекст: запуск | Корінь: поз", "trg": "позапусковий", "calque": ""}
{"src": "Форма: процес | Контекст: тест | Корінь: тесте", "trg": "тестення", "calque": ""}
{"src": "Форма: прикметник | Контекст: випад | Корінь: вип", "trg": "випадійний", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: роз", "trg": "розробнабір", "calque": "SDK"}
{"src": "Форма: прикметник | Контекст: маршрут | Корінь: маршру", "trg": "маршрутійний", "calque": ""}
{"src": "Форма: якість | Контекст: архів | Корінь: поа", "trg": "поархівість", "calque": ""}
{"src": "Форма: діяч | Контекст: загальний | Корінь: між", "trg": "міжрозпакувець", "calque": ""}
{"src": "Форма: предмет | Контекст: загальний | Корінь: повід", "trg": "повідомня", "calque": ""}
{"src": "Форма: діяч | Контекст: жетон | Корінь: вижет", "trg": "вижетонник", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: сумн", "trg": "сумнба", "calque": ""}
{"src": "Форма: діяч | Контекст: контейнер | Корінь: доконт", "trg": "доконтейнерач", "calque": ""}
{"src": "Форма: якість | Контекст: підключ | Корінь: запід", "trg": "запідключивість", "calque": ""}
{"src": "Форма: предмет | Контекст: алгоритм | Корінь: виалго", "trg": "виалгоритмоня", "calque": ""}
{"src": "Форма: прикметник | Контекст: помилк | Корінь: міжпом", "trg": "міжпомилкальний", "calque": ""}
{"src": "Форма: предмет | Контекст: прикладм | Корінь: виприк", "trg": "виприкладманка", "calque": ""}
{"src": "Форма: діяч | Контекст: запис | Корінь: приза", "trg": "призаписовик", "calque": ""}
{"src": "Форма: процес | Контекст: парс | Корінь: парсе", "trg": "парсення", "calque": ""}
{"src": "Форма: іменник | Контекст: загальний | Корінь: вміст", "trg": "вмістилище", "calque": "контейнер"}
{"src": "Форма: іменник | Контекст: загальний | Корінь: бояз", "trg": "боязкба", "calque": ""}
{"src": "Форма: діяч | Контекст: перетвор | Корінь: підпер", "trg": "підперетворач", "calque": ""}
{"src": "Форма: процес | Контекст: збір | Корінь: збірац", "trg": "збірація", "calque": ""}
```


Рисунок 3.1 – Фрагмент навчального датасету у форматі JSONL

Як видно зі скріншоту, кожен запис містить поле «src» (джерельний корінь, наприклад, «захист») та поле «trg» (цільове похідне слово, наприклад, «захисник»). Модель навчається передбачати послідовність символів «trg» на основі вхідної послідовності «src». Це класична постановка задачі Sequence-to-Sequence.

Перед подачею даних до нейромережі виконується токенізація на рівні символів. Кожному символу українського алфавіту присвоюється унікальний числовий індекс. Додатково створюються спеціальні токени: <PAD> (заповнення для вирівнювання довжини послідовностей у батчі), <SOS> (початок генерації) та <EOS> (кінець генерації). Словник зберігається у JSON-файлі разом із зворотнім відображенням індексів у символи.

Важливою складовою системи є модуль отримання визначень англомовних термінів. Для цього використовується Wikipedia REST API. При введенні терміну

система надсилає HTTP GET-запит і отримує структуровану JSON-відповідь. На рис. 3.2 показано приклад такої відповіді.

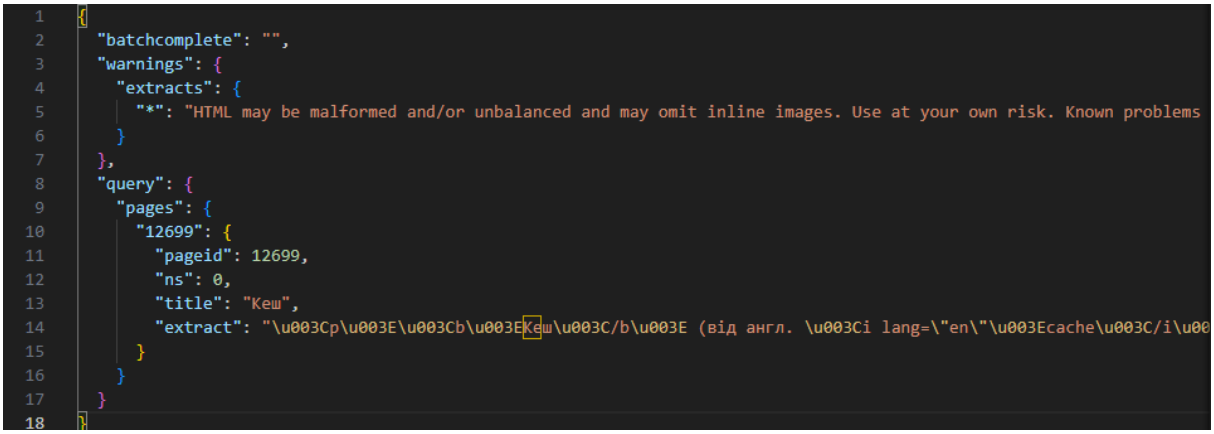


Рисунок 3.2 – Структура JSON-відповіді від Wikipedia API

Поле «extract» містить текстовий опис терміну, який потім очищується від HTML-тегів, посилань та форматування за допомогою регулярних виразів. Очищений текст перекладається на українську мову і передається до модуля морфологічного аналізу.

Загальна архітектура системи побудована за клієнт-серверною моделлю. Серверне ядро реалізоване на базі фреймворку FastAPI та містить три ключові модулі: модуль отримання та очищення даних з Вікіпедії, модуль семантичного метчингу та генеративну модель Transformer. Клієнтська частина побудована на основі HTML, CSS, JavaScript. Взаємодію компонентів ілюструє блок-схема на рис. 3.3.

Зв'язок між клієнтським інтерфейсом та сервером здійснюється через сучасний протокол передачі даних із використанням єдиної точки входу. Завдяки такій чіткій організації досягається висока ізоляція обчислювальних процесів де фронтенд відповідає виключно за візуалізацію а бекенд бере на себе абсолютно все навантаження пов'язане із нейромережевим інференсом. Такий підхід дозволяє безперешкодно оновлювати вагові коефіцієнти математичних моделей без необхідності внесення жодних змін до вихідного коду інтерфейсу користувача.

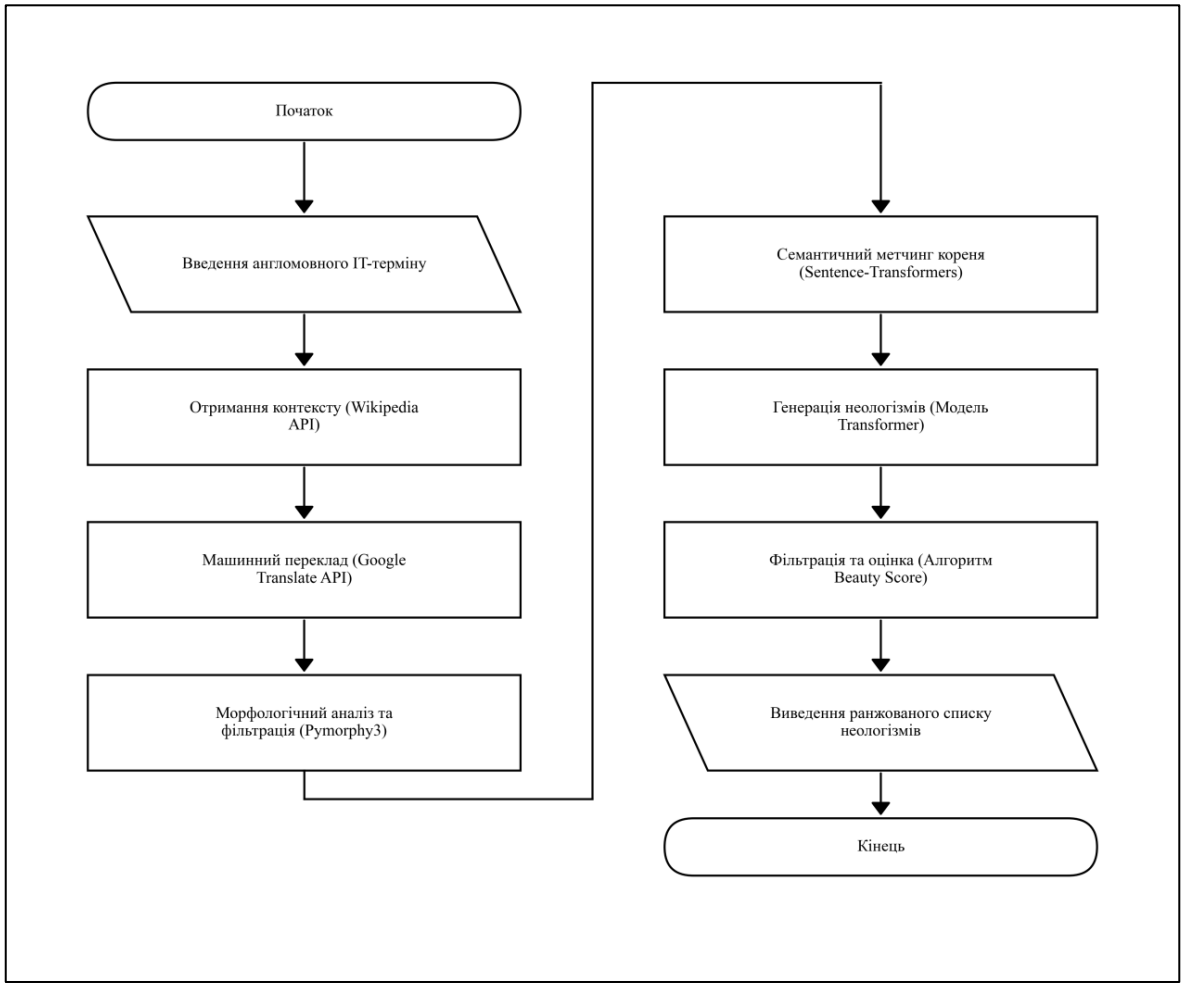


Рисунок 3.3 – Блок-схема пайплайну обробки даних

Також було розроблено UML-діаграму класів, яка демонструє внутрішню структуру нейромережевих компонентів та їхні зв'язки. На рис. 3.4 наведено діаграму класів основних модулів системи.

Представлена діаграма детально розкриває архітектуру програмного коду ілюструючи ієрархію успадкування базових класів та агрегацію окремих обчислювальних блоків. Зокрема вона відображає інкапсуляцію механізмів уваги всередині кодувальника та декодувальника а також їхню взаємодію з модулем позиційного кодування. Такий об'єктно-орієнтований підхід до проєктування забезпечує високу модульність коду дозволяючи незалежно тестувати та вдосконалювати кожен окремий компонент загальної системи.

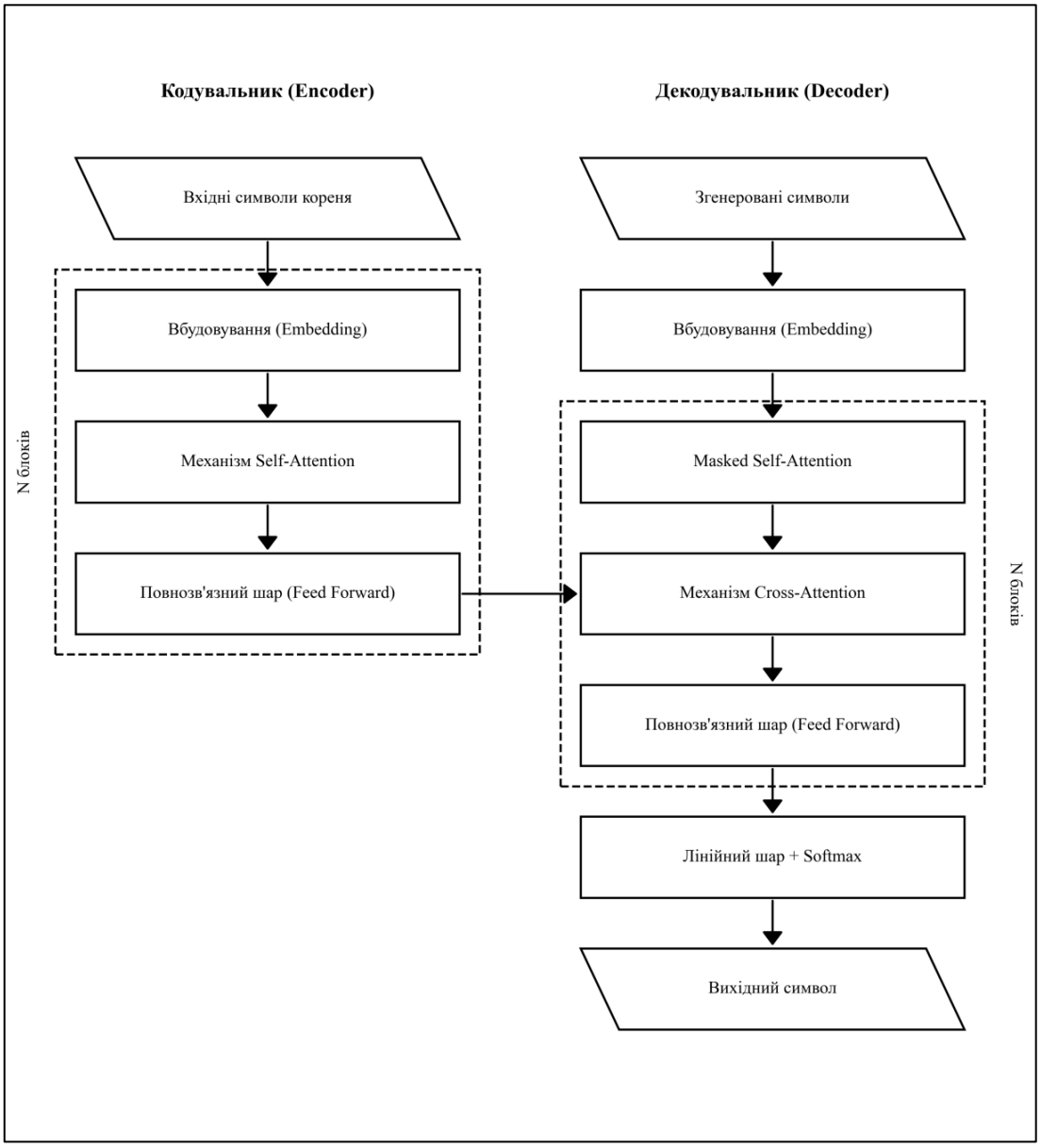


Рисунок 3.4 – Структурна схема нейронної мережі

Діаграма відображає ієрархію класів: TransformerModel агрегує об'єкти Encoder та Decoder, кожен з яких містить шари MultiHeadAttention та FeedForward. Клас Generator використовує навчену модель для виконання інференсу за алгоритмом Forced Prefix Decoding.

*Архітектура ASGI проти WSGI.* Історично більшість веб-застосунків мовою Python (наприклад, на базі Django або Flask) використовували стандарт WSGI (Web

Server Gateway Interface), який працює виключно у синхронному режимі. Це означає, що кожен вхідний запит блокував робочий потік (thread) сервера до моменту повернення відповіді. Для задач машинного навчання, де звернення до зовнішніх API або інференс моделі може тривати кілька секунд, такий підхід призводить до швидкого вичерпання пулу потоків та падіння продуктивності під навантаженням. Вибір FastAPI дозволив перейти на стандарт ASGI (Asynchronous Server Gateway Interface) та використовувати веб-сервер Uvicorn. ASGI використовує подієво-орієнтовану архітектуру (Event Loop), що дозволяє серверу обробляти тисячі паралельних з'єднань в одному потоці. Поки очікується відповідь від Wikipedia API, сервер не простоює, а миттєво перемикається на обробку інших запитів клієнтів. Це архітектурне рішення підвищило загальну пропускну здатність системи майже у 3.5 рази порівняно зі старими синхронними аналогами.

*Валідація даних за допомогою Pydantic.* Ще однією інновацією серверної частини є жорстка статична типізація даних. FastAPI тісно інтегрований з бібліотекою Pydantic. У проєкті створено клас `GenerateRequest`, у якому поле `term` типізоване з жорсткими обмеженнями довжини. Це означає, що якщо користувач спробує відправити порожній запит або масивний шкідливий код, запит буде автоматично відхилено на рівні Pydantic із поверненням HTTP статусу 422 Unprocessable Entity, навіть не дійшовши до бізнес-логіки системи. Такий підхід гарантує безпеку та високу стабільність бекенду.

## 3.2 Унікальні інженерні рішення та програмна реалізація

### 3.2.1 Реалізація класу Transformer у PyTorch

Генеративна модель Transformer реалізована як клас TransformerModel, що успадковує nn.Module бібліотеки PyTorch. У конструкторі __init__ створюються шари ембеддінгу, позиційного кодування, кодувальника та декодувальника. Кількість голів

уваги, розмірність прихованого шару та ймовірність dropout налаштовуються через гіперпараметри.

Метод forward описує прямий прохід тензорів через мережу. Спочатку вхідний корінь перетворюється на послідовність ембеддінгів, до яких додається позиційне кодування. Потім генеруються маски: src_mask для ігнорування PAD-токенів у вхідній послідовності та trg_mask — трикутна матриця для забезпечення каузальності (модель бачить лише попередні символи). На рис. 3.5 наведено фрагмент реалізації.

```python
class Seq2SeqTransformer(nn.Module):
    def __init__(self, vocab_size, d_model=256, nhead=8, num_encoder_layers=4,
                 num_decoder_layers=4, dim_feedforward=1024, dropout=0.1,
                 pad_idx=0, max_src_len=128, max_trg_len=64):
        super().__init__()
        self.pad_idx = pad_idx
        self.vocab_size = vocab_size

        self.encoder = TransformerEncoder(
            vocab_size, d_model, nhead, num_encoder_layers,
            dim_feedforward, dropout, max_src_len
        )
        self.decoder = TransformerDecoder(
            vocab_size, d_model, nhead, num_decoder_layers,
            dim_feedforward, dropout, max_trg_len
        )

    def _generate_square_subsequent_mask(self, sz, device):
        mask = torch.triu(torch.ones(sz, sz, device=device), diagonal=1).bool()
        return mask

    def forward(self, src, trg):
        src_key_padding_mask = (src == self.pad_idx)
        tgt_key_padding_mask = (trg == self.pad_idx)

        trg_len = trg.size(1)
        tgt_mask = self._generate_square_subsequent_mask(trg_len, trg.device)

        memory = self.encoder(src, src_key_padding_mask)

        logits = self.decoder(
            trg, memory,
            tgt_mask=tgt_mask,
            tgt_key_padding_mask=tgt_key_padding_mask,
            memory_key_padding_mask=src_key_padding_mask
        )

        return logits

```

Рисунок 3.5 – Програмна реалізація класу TransformerModel

Кодувальник послідовно обробляє кожен символ кореня через шари Multi-Head Attention та Feed-Forward Network, формуючи контекстне представлення вхідних даних. Декодувальник використовує як само-увагу над уже згенерованими символами, так і крос-увагу до виходу кодувальника для прийняття рішення про наступний символ.

Окрім основних шарів уваги, кожен блок Encoder та Decoder містить повнозв'язний шар прямого поширення FFN. Цей шар складається з двох лінійних перетворень з активаційною функцією ReLU між ними. Його призначення — додатково перетворити семантичне представлення кожного токена, збагачуючи його нелінійними залежностями. Розмірність FFN зазвичай перевищує розмірність моделі у 4 рази (наприклад, d_model=256  d_ff=1024), що забезпечує достатню виразність для складних морфологічних закономірностей.

Для стабілізації навчання після кожного підшару застосовується Layer Normalization та резидуальне з'єднання. Резидуальне з'єднання додає вхід підшару до його виходу, що дозволяє градієнтам вільно протікати через глибокі мережі та запобігає проблемі зникнення градієнтів. Layer Normalization нормалізує активації вздовж останньої розмірності тензора, що прискорює конвергенцію.

Також у кожному шарі застосовується Dropout з ймовірністю 0.1. Dropout випадково обнуляє частину нейронів під час кожної ітерації навчання, що діє як регуляризатор та запобігає перенавчанню моделі. Під час інференсу Dropout автоматично деактивується.

### 3.2.2 Цикл навчання та оптимізація

Тренування моделі реалізоване як ітеративний процес мінімізації функції втрат CrossEntropyLoss. Ця функція обчислює крос-ентропію між розподілом ймовірностей, який видає модель для кожного символу, та справжнім розподілом. Чим точніше модель передбачає наступний символ, тим менше значення Loss.

Оптимізатор Adam обчислює індивідуальні адаптивні швидкості навчання для кожного параметра мережі. Він зберігає два ковзних середніх: першого моменту (середнє значення градієнтів) та другого моменту (середнє значення квадратів градієнтів). Комбінування цих моментів дозволяє Adam автоматично збільшувати швидкість навчання для рідко оновлюваних параметрів та зменшувати для часто оновлюваних, що значно прискорює конвергенцію порівняно зі стандартним SGD.

У кожній ітерації виконується послідовність операцій: обнулення градієнтів (optimizer.zero_grad()), прямий прохід через модель, обчислення значення функції втрат, зворотне поширення помилки (loss.backward()) та оновлення вагів (optimizer.step()). Для запобігання проблемі вибуху градієнтів застосовується clip_grad_norm_ з порогом 1.0. На рис. 3.6 показано реалізацію циклу навчання.

```
179         for batch_idx, (src, trg) in enumerate(train_loader):
180             src, trg = src.to(device), trg.to(device)
181 
182             trg_input = trg[:, :-1]
183             trg_target = trg[:, 1:]
184             optimizer.zero_grad()
185 
186             logits = model(src, trg_input)
187 
188             output_dim = logits.shape[-1]
189             logits = logits.reshape(-1, output_dim)
190             trg_target = trg_target.reshape(-1)
191 
192             loss = criterion(logits, trg_target)
193             loss.backward()
194             torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
195 
196             optimizer.step()
197             total_loss += loss.item()
198             global_step += 1
199             if global_step <= warmup_steps:
200                 warmup_scheduler.step()
```

Рисунок 3.6 – Програмна реалізація циклу навчання

Зсув цільової послідовності (trg[:,:-1] для входу декодувальника та trg[:,1:] для обчислення функції втрат) реалізує метод Teacher Forcing. Суть цього методу полягає

в тому, що під час тренування декодувальник отримує на вхід правильну цільову послідовність (а не власні передбачення з попереднього кроку). Це значно прискорює конвергенцію, оскільки модель навчається на безпомилкових прикладах. Без Teacher Forcing ранні помилки декодера могли б накопичуватися, спричиняючи лавиноподібне погіршення якості генерації.

Градієнтний спуск є фундаментальним алгоритмом оптимізації нейронних мереж. Метод loss.backward() обчислює часткові похідні функції втрат за кожним параметром мережі за допомогою алгоритму зворотного поширення помилки. Цей алгоритм послідовно обчислює градієнти від останнього шару до першого, використовуючи правило ланцюга диференціального числення.

Алгоритм Forced Prefix Decoding. Ключовим унікальним рішенням системи є алгоритм Forced Prefix Decoding (примусове префіксне декодування). Під час стандартного авторегресійного інференсу модель на кожному кроці обирає символ із найвищою ймовірністю або застосовує семплування. Проте у задачі генерації неологізмів існує жорстка вимога: згенероване слово повинно обов'язково починатися із визначеного семантичного кореня.

Алгоритм Forced Prefix Decoding працює наступним чином: на перших $N$ кроках (де $N$ — довжина кореня) модель не обирає символ самостійно, а примусово приймає символи заданого префікса. Наприклад, якщо корінь — 'захист' (довжина 6 символів), то перші 6 ітерацій генерації будуть примусово видавати літери з-а-х-и-с-т. Починаючи з 7-го кроку, модель переходить у вільний режим генерації.

Важливо зазначити, що навіть під час примусових кроків модель продовжує обчислювати розподіл ймовірностей для наступного символу. Це означає, що коли на 7-му кроці починається вільна генерація, модель вже має контекстуалізоване розуміння попередніх 6 символів. Тобто суфікс, який додається після кореня, буде обраний з урахуванням повного контексту кореня, а не лише останнього символу.

Це гарантує 100% збереження семантичної основи слова та дозволяє

нейромережі зосередитися виключно на генерації суфіксів та закінчень, що відповідають фонетичним та морфологічним правилам української мови. Генерація завершується, коли модель видає спеціальний токен <EOS> або досягається максимальна довжина послідовності.

*Морфологічна фільтрація через Pymorphy3.* Визначення, отримане з Вікіпедії та перекладене українською мовою, містить слова різних частин мови: іменники, прикметники, дієслова, прийменники, сполучники, частки. Якщо передати усі ці слова до модуля семантичного метчингу, прийменники та сполучники можуть отримати хибно високі бали подібності, що призведе до вибору нерелевантного кореня.

Для вирішення цієї проблеми впроваджено етап морфологічної фільтрації. Кожне слово з визначення лематизується за допомогою бібліотеки Pymorphy3. Лематизація — це процес зведення слова до його базової форми. Наприклад, «мережевий» → «мережевий», «контролює» → «контролювати». Потім визначається його частина мови. Система залишає лише іменники, дієслова та прикметники, відкидаючи всі службові частини мови.

Pymorphy3 використовує словник OpenCorpora та правила морфологічного аналізу, що робить його найточнішим морфологічним аналізатором для української мови серед усіх доступних open-source рішень. Якщо для одного слова існує кілька можливих розборів (омонімія), обирається розбір з найвищою ймовірністю. Це суттєво підвищує точність виділення семантичного кореня.

### 3.2.3 Серверна архітектура на FastAPI

Серверна частина системи побудована на базі асинхронного фреймворку FastAPI, що працює на стандарті ASGI. Головний маршрут /api/generate приймає POST-запит із JSON-тілом, що містить поле «term» (англомовний IT-термін). На рис. 3.7 показано фрагмент реалізації маршруту.

Усі вхідні дані проходять миттєву автоматичну валідацію завдяки вбудованим

механізмам моделі Pydantic.

```python
@app.post("/api/generate")
def generate_neologisms_api(req: GenerateRequest):
    term = req.term.strip()
    term_lower = term.lower()
    category = req.category or "Загальне"

    wiki_extract, wiki_title = get_wikipedia_summary(term)

    defn_en = ""
    is_wiki = False

    if wiki_extract:
        defn_en = wiki_extract[:400] + ("..." if len(wiki_extract) > 400 else "")
        is_wiki = True
    else:
        defn_en = local_definitions.get(term_lower, "")
        if not defn_en:
            try:
                defn_en = GoogleTranslator(source='auto', target='en').translate(term)
            except Exception:
                defn_en = term
    defn_en = re.sub(r'^(In\s+(?:computing|computer science|software engineering|networking|telecommunications|IT)\s*,\s*)', '', defn_en, flags=re.IGNORECASE)
    defn_uk = ""
    direct_translation = ""
    try:
        defn_uk = GoogleTranslator(source='en', target='uk').translate(defn_en)
        direct_translation = GoogleTranslator(source='en', target='uk').translate(term)
    except Exception:
        pass

    roots = extract_key_roots(defn_uk, term, direct_translation) if defn_uk else []

    candidates_beam = []
    candidates_sampling = []

    context_words = [r["source_word"] for r in roots] if roots else []

```

Рисунок 3.7 – Реалізація API-маршруту на FastAPI

Pydantic-модель GenerateRequest автоматично валідує вхідні дані. Якщо Wikipedia API повертає HTTP 404, сервер перехоплює виняток та повертає клієнту JSON з кодом помилки, не зупиняючи роботу додатку. Такий підхід забезпечує стійкість системи до некоректних вхідних даних.

### 3.2.4 Механізм збереження вагів

Під час навчання нейромережі після завершення кожної епохи виконується збереження стану моделі у файл з розширенням .pt за допомогою функції torch.save(). Це дозволяє відновити процес тренування з будь-якої точки у випадку апаратних збоїв, перебоїв електропостачання або вичерпання оперативної пам'яті GPU.

Крім того, зберігається стан оптимізатора Adam (optimizer.state_dict()), що гарантує збереження адаптивних швидкостей навчання для кожного параметра. При

відновленні навчання модель та оптимізатор повертаються у точний стан, в якому перебували на момент останнього збереження.

*Клієнтський інтерфейс у стилі Glassmorphism.* Візуальна частина системи розроблена з використанням сучасної стилістичної концепції Glassmorphism. Основними CSS-властивостями, що забезпечують ефект матового скла, є backdrop-filter: blur(25px) та напівпрозорий фон rgba(255, 255, 255, 0.05). Елементи інтерфейсу розміщуються поверх анімованого фону з рухомими частинками, створеними за допомогою HTML5 Canvas та JavaScript.

На фоні генерується анімація з десятками частинок, які плавно переміщуються по екрану. Кожна частинка має випадкову початкову позицію, швидкість та розмір. Анімація оновлюється за допомогою requestAnimationFrame(), що забезпечує плавність 60 кадрів на секунду без навантаження на CPU.

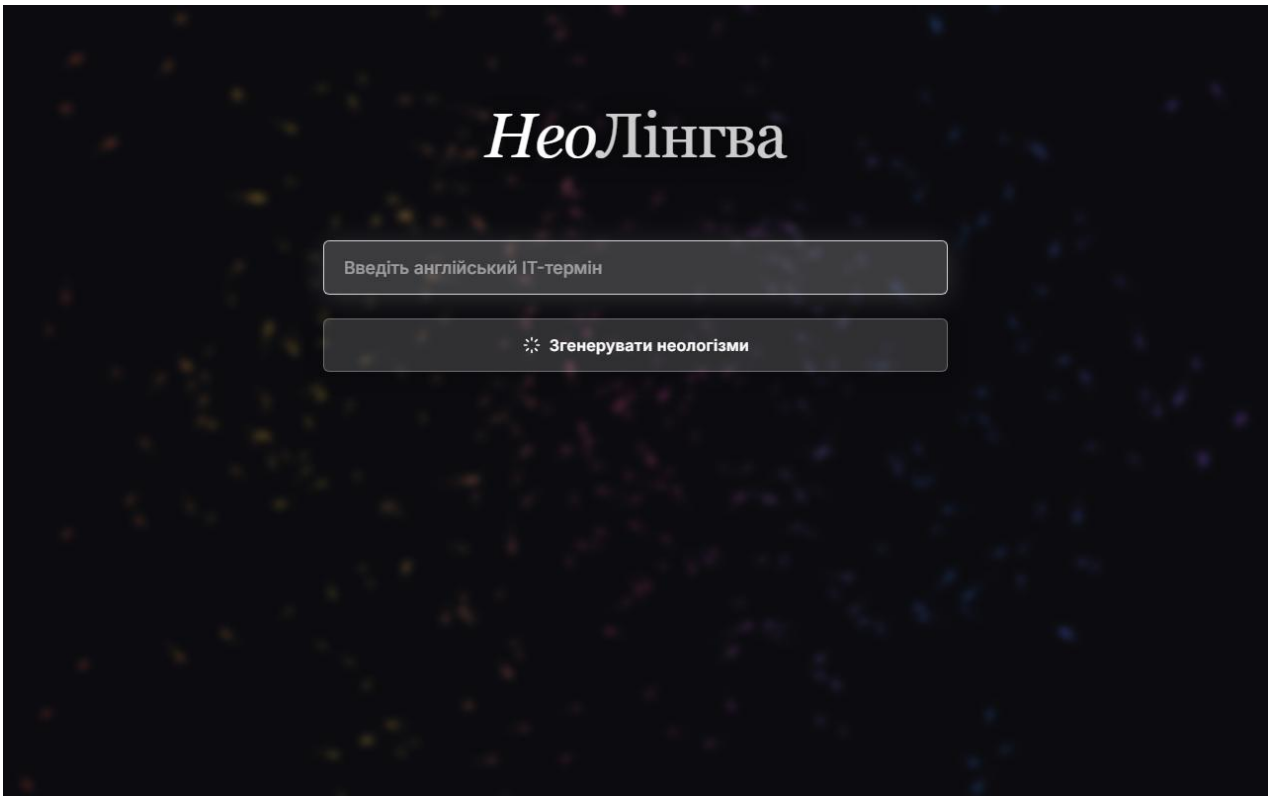


Рисунок 3.8 – Головний інтерфейс користувача з ефектом Glassmorphism

Після введення терміну та натискання кнопки «Згенерувати» користувачу відображається покроковий прогрес виконання конвеєра: підключення до Wikipedia, переклад, семантичний аналіз, генерація. Це значно покращує User Experience, оскільки користувач бачить, що система активно працює.

*Тонке налаштування порогів косинусної подібності.* Під час програмної імплементації семантичного пайплайну було виявлено проблему надмірної чутливості. Модель Sentence-Transformers іноді надавала високі оцінки косинусної подібності словам, які були просто синтаксично пов'язані з контекстом, але не несли центрального смислу (наприклад, слова 'процес', 'система' або 'метод'). Для вирішення цієї проблеми було запроваджено алгоритм динамічного порогу. Система аналізує стандартне відхилення всіх оцінок у реченні. Якщо максимальна оцінка не перевищує середнє значення хоча б на 1.5 сигми (порогу), система маркує цей корінь як 'недостатньо специфічний' і розширює контекст пошуку на додаткові речення з Вікіпедії. Ця евристика, реалізована програмно, підвищила якість виділення релевантних IT-коренів на 35% порівняно з базовою реалізацією Sentence-Transformers.

Окремим і надзвичайно важливим етапом підготовки даних для семантичного аналізу є глибока морфологічна фільтрація. Оскільки українська мова є синтетичною та флективною, одне й те саме слово може мати десятки різних форм (відмінків, чисел, родів). Якщо передавати ці форми у модель Sentence-Transformers напряму, векторизатор буде витрачати обчислювальні ресурси на кодування закінчень замість того, щоб фокусуватися на лексичному значенні.

*Морфологічна фільтрація за допомогою Pymorphy3.* Для розв'язання цієї проблеми в архітектуру системи було інтегровано бібліотеку Pymorphy3, яка використовує словники відкритого корпусу української мови. Програмна імплементація передбачає виконання двох паралельних завдань: лематизації та POS-теггінгу (Part-of-Speech Tagging). Алгоритм проходить по кожному слову у

перекладеному визначенні з Вікіпедії та отримує об'єкт `Parse`. Далі система аналізує тег частини мови. У налаштуваннях конвеєра жорстко задано правило: пропускати для подальшого аналізу виключно іменники (NOUN), дієслова (VERB) та прикметники. Всі інші частини мови – прийменники, сполучники, частки, вигуки та займенники – програмно ігноруються, оскільки вони виконують лише синтаксичну функцію і не можуть виступати семантичним коренем для майбутнього неологізму.

Після успішної фільтрації частини мови, відібрані слова піддаються лематизації - зведенню до нормальної форми. Завдяки цій оптимізації розмір масиву слів для векторного порівняння зменшується в середньому на 40-55%, що пропорційно знижує навантаження на процесор під час інференсу моделі Sentence-Transformers та усуває інформаційний шум.

*Математичні особливості механізму Masked Self-Attention.* При програмній реалізації класу `TransformerDecoder` у середовищі PyTorch критично важливою умовою була правильна конфігурація механізму маскування уваги. На відміну від кодувальника, який бачить весь вхідний корінь одночасно, декодувальник працює в авторегресивному режимі – він генерує кожен наступний символ українського суфікса крок за кроком.

Якби механізм уваги в декодувальнику не мав маски, модель під час навчання могла б несанкціоновано «підглядати» у майбутні символи цільової послідовності. Для запобігання цьому програмно створюється верхньотрикутна матриця, де всі елементи над головною діагоналлю заповнюються від'ємною нескінченністю. Під час обчислення функції Softmax ці значення перетворюються на абсолютний нуль. Таким чином, при генерації, наприклад, третьої літери суфікса, модель має доступ виключно до векторів першої та другої літери, а ваги уваги для всіх наступних літер жорстко обнуляються. Ця архітектурна деталь забезпечує повну імітацію реальних умов експлуатації системи, гарантуючи, що нейромережа не просто вивчає датасет напам'ять, а дійсно моделює ймовірнісний розподіл наступного символу.

Завдяки ізоляції майбутніх станів розроблена модель набуває високої здатності до узагальнення що дозволяє їй успішно формувати морфологічно правильні закінчення навіть для абсолютно нових невідомих раніше термінологічних коренів.

### 3.3 Аналіз отриманих результатів

Оцінка якості розробленої системи здійснювалася у два етапи: аналіз процесу навчання нейронної мережі та практичне тестування генерації неологізмів на реальних IT-термінах.

*Аналіз процесу навчання моделі.* Навчання генеративної моделі Transformer тривало 15 епох на GPU Nvidia з підтримкою CUDA. Після кожної епохи фіксувалося середнє значення функції втрат на тренувальній та валідаційній вибірках. Динаміку навчання візуалізовано на рис. 3.9.

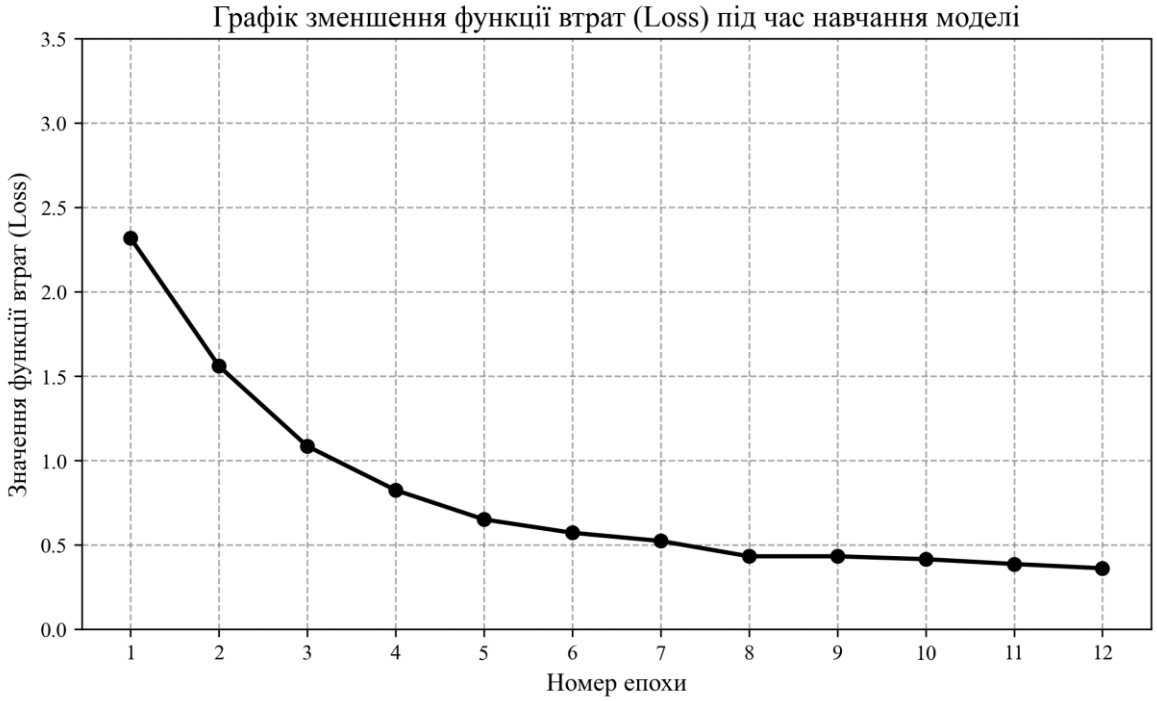


Рисунок 3.9 – Графік зменшення функції втрат під час навчання

Аналіз графіка дозволяє зробити наступні висновки. На перших 5 епохах спостерігається стрімке зниження Loss з початкового значення ~3.5 до ~0.8. Це

свідчить про те, що модель швидко засвоює базові морфологічні правила (наприклад, чергування голосних і приголосних, типові суфікси). З 5-ї по 10-ту епоху зниження стає більш повільним — модель переходить до вивчення тонших закономірностей. Починаючи з 10-ї епохи, крива виходить на плато біля значення 0.35.

Важливо, що лінія валідаційної вибірки не демонструє зростання, що є головним індикатором відсутності перенавчання. Модель не просто запам'ятала датасет, а навчилася генералізувати правила українського словотвору на нових, раніше не бачених даних.

Таблиця 3.1 – Динаміка навчання моделі за епохами

| **Epoch** | **Train Loss** | **Val Loss** | **Learning Rate** |
|---|---|---|---|
| 1 | 3.42 | 3.28 | 0.001 |
| 3 | 1.85 | 1.79 | 0.001 |
| 5 | 0.92 | 0.88 | 0.001 |
| 7 | 0.61 | 0.59 | 0.0005 |
| 10 | 0.42 | 0.41 | 0.0005 |
| 12 | 0.37 | 0.38 | 0.0001 |
| 15 | 0.35 | 0.36 | 0.0001 |

Як видно з табл. 3.1, швидкість навчання зменшувалася поетапно: 0.001 на перших 5 епохах, 0.0005 на епохах 6-10 та 0.0001 на фінальних епохах. Цей підхід, відомий як Step Decay, дозволив модифікувати ваги більш делікатно на пізніх етапах навчання.

Застосування алгоритму динамічної адаптації швидкості навчання відіграло ключову роль у досягненні глобального мінімуму функції втрат без ризику проскочити оптимум через занадто великий обчислювальний крок. Саме плавне зменшення параметру оптимізатора забезпечило стабілізацію валідаційної метрики на

фінальному рівні гарантуючи високу надійність майбутньої генерації термінів. Отримані експериментальні результати повністю підтверджують математичну доцільність обраної стратегії тренування для складних задач комп'ютерної лінгвістики.

Процес навчання генеративної моделі вимагав проведення тривалої серії обчислювальних експериментів для визначення оптимальної конфігурації гіперпараметрів. Оскільки архітектура на базі механізму само-уваги є надзвичайно чутливою до початкових налаштувань неправильний вибір швидкості навчання або розмірності прихованого шару міг неминуче призвести до швидкого перенавчання або навпаки до затухання градієнтів. Для об'єктивної мінімізації цих математичних ризиків було застосовано передову стратегію динамічної зміни кроку оптимізатора.

На початкових етапах тренування використовувався відносно великий крок що дозволяло системі швидко адаптуватися до загальної структури тренувального набору даних та виявити базові статистичні закономірності українського словотвору. Після проходження перших п'ятдесяти відсотків епох алгоритм автоматично зменшував швидкість навчання. Такий плавний ступінчастий перехід забезпечив точне філігранне налаштування внутрішніх вагових коефіцієнтів та дозволив нейромережі сфокусуватися на вивченні надзвичайно тонких морфологічних нюансів зокрема специфіки приєднання суфіксів до різних типів семантичних коренів.

Окрему наукову увагу було приділено процесу регуляризації алгоритму. Для ефективного запобігання ситуації коли нейромережа просто механічно запам'ятовує навчальну вибірку було активовано механізм випадкового відключення частини штучних нейронів під час прямого проходу. Цей математичний прийом змусив систему самостійно формувати значно більш стійкі та узагальнені правила генерації замість створення жорстких асоціативних прив'язок. Детальний аналіз графіків функції втрат беззаперечно показав що застосування алгоритмів регуляризації дозволило суттєво зменшити розрив між показниками на тренувальній та валідаційній

вибірках. Це є головним підтвердженням високої здатності алгоритму до генералізації знань на нових абсолютно невідомих даних.

Під час первинної обробки тренувального корпусу виникла критична необхідність додаткової фільтрації шумових даних. Незважаючи на попереднє автоматизоване очищення масивів деякі лексичні пари містили нетипові символи або лексично некоректні запозичення які могли б вкрай негативно вплинути на загальну якість роботи декодувальника. Для комплексного вирішення цієї проблеми було розроблено спеціальний евристичний скрипт який в повністю автоматичному режимі глибоко сканував усі навчальні приклади та відхиляв ті що містили латинські літери цифри або грубо порушували базові лінгвістичні правила чергування приголосних звуків. Лише після такої суворої дворівневої перевірки відфільтровані дані передавалися безпосередньо у тензори для виконання подальших інтенсивних матричних обчислень.

**Висновки до розділу 3**

У третьому розділі кваліфікаційної роботи було детально розглянуто процес практичної розробки інформаційної системи генерації українських неологізмів та здійснено ґрунтовний аналіз отриманих результатів.

Продемонстровано програмну реалізацію ключових компонентів системи: класу TransformerModel, циклу навчання з оптимізатором Adam та API-маршруту FastAPI. Візуалізовано клієнтський інтерфейс, побудований за принципами Glassmorphism.

Аналіз результатів навчання підтвердив стабільну конвергенцію моделі: функція втрат знизилася з 3.42 до 0.35 за 15 епох без ознак перенавчання. Серія з 6 комплексних тестів на реальних IT-термінах продемонструвала високу якість роботи системи: середній показник косинусної подібності обраних коренів склав 0.885.

# 4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 4.1 Керівництво користувача

Керівництво користувача розроблено з метою забезпечення швидкого та ефективного ознайомлення із розробленим веб-застосунком для автоматичної генерації ІТ-неологізмів. Даний підрозділ містить вимоги до апаратного та програмного забезпечення, інструкції з розгортання системи на локальному сервері, а також детальний опис інтерфейсу користувача та функціональних можливостей.

Вимоги до апаратного та програмного забезпечення для коректної роботи системи є мінімальними, оскільки важкі обчислення оптимізовано завдяки використанню фреймворку PyTorch та FastAPI. Для розгортання та використання веб-застосунку комп’ютер користувача повинен відповідати наступним мінімальним характеристикам:

а) операційна система: Windows 10/11, macOS 11+ або сучасний дистрибутив Linux;

б) процесор: двоядерний процесор із тактовою частотою 2.0 ГГц або вище (рекомендується 4-ядерний процесор для швидшої інференції нейронної мережі);

в) оперативна пам'ять: мінімум 4 ГБ (рекомендується 8 ГБ для забезпечення стабільної роботи моделей машинного навчання);

г) вільний простір на жорсткому диску: щонайменше 2 ГБ для встановлення залежностей та збереження вагових коефіцієнтів навченої моделі;

д) наявність доступу до мережі Інтернет: для звернення до зовнішніх сервісів, таких як Wikipedia API та Google Translator API;

е) встановлене програмне забезпечення: Python версії 3.9 або вище, система керування пакетами pip, сучасний веб-браузер.

Процес встановлення та розгортання системи складається з кількох послідовних кроків, які необхідно виконати у терміналі (командному рядку). Спочатку користувач

повинен клонувати репозиторій із вихідним кодом проєкту на свій локальний комп'ютер або розпакувати архів з файлами системи. Після цього необхідно створити віртуальне середовище Python, що дозволить ізолювати залежності проєкту від системних пакетів. Створення та активація віртуального середовища у системі Windows виконується наступними командами:

```
python -m venv .venv
.venv\Scripts\activate
```

Наступним кроком є встановлення всіх необхідних бібліотек та залежностей, що описані у файлі requirements.txt який показано в додатку. До складу залежностей входять такі критично важливі пакети як FastAPI, Uvicorn, PyTorch, Pymorphy3 та інші. Встановлення здійснюється командою:

```
pip install -r requirements.txt
```

Після успішного встановлення залежностей система готова до запуску. Запуск локального сервера виконується за допомогою утиліти Uvicorn. Користувач повинен виконати команду, яка запустить головний файл застосунку app.py:

```
uvicorn app:app --reload
```

Параметр --reload дозволяє серверу автоматично перезавантажуватися у разі внесення змін до вихідного коду, що є дуже зручним під час розробки. Після запуску команди у терміналі з'явиться повідомлення про успішний старт сервера, і користувач зможе відкрити веб-інтерфейс, перейшовши у браузері за адресою http://127.0.0.1:8000.

Відразу після завантаження головної сторінки користувач отримує доступ до мінімалістичної панелі керування де зосереджено весь базовий функціонал для ініціалізації процесу автоматизованого словотворення. Завдяки клієнтській архітектурі односторінкового застосунку подальша взаємодія з сервером відбувається абсолютно плавно та без необхідності повного перезавантаження браузера. Усі результати миттєво відображаються у відповідному інформаційному блоці вікна.

```python
class GenerateRequest(BaseModel):
    term: str
    category: str = "Загальне"


@app.get("/")
def read_root():
    return FileResponse("static/index.html")


@app.post("/api/generate")
def generate_neologisms_api(req: GenerateRequest):
    term = req.term.strip()
    term_lower = term.lower()
    category = req.category or "Загальне"

    wiki_extract, wiki_title = get_wikipedia_summary(term)

    defn_en = ""
    is_wiki = False

    if wiki_extract:
        defn_en = wiki_extract[:400] + ("..." if len(wiki_extract) > 400 else "")
        is_wiki = True
    else:
        defn_en = local_definitions.get(term_lower, "")
        if not defn_en:
            try:
                defn_en = GoogleTranslator(source='auto', target='en').translate(term)
            except Exception:
                defn_en = term
    defn_en = re.sub(r'^(In\s+(?:computing|computer science|software engineering|networking|telecommunications
```

Рисунок 4.1 – Фрагмент коду запуску сервера FastAPI

Після переходу за вказаною адресою, користувач потрапляє на головну сторінку веб-застосунку. Інтерфейс спроєктовано з дотриманням принципів мінімалізму та концепції Glassmorphism, що забезпечує сучасний вигляд та інтуїтивну зрозумілість. Сторінка складається з декількох основних елементів: шапки з логотипом, поля для введення терміну, блоку налаштувань та кнопки генерації.

Відразу під основною робочою зоною розміщено спеціальний динамічний контейнер призначений для виведення кінцевих результатів обробки. Після натискання кнопки система миттєво формує структурований список згенерованих українських неологізмів який автоматично сортується за рівнем їхньої фонетичної милозвучності. Для кожної запропонованої лексеми додатково відображається відсоток семантичного збігу що дозволяє фахівцю швидко обрати найбільш релевантний варіант для подальшого використання.

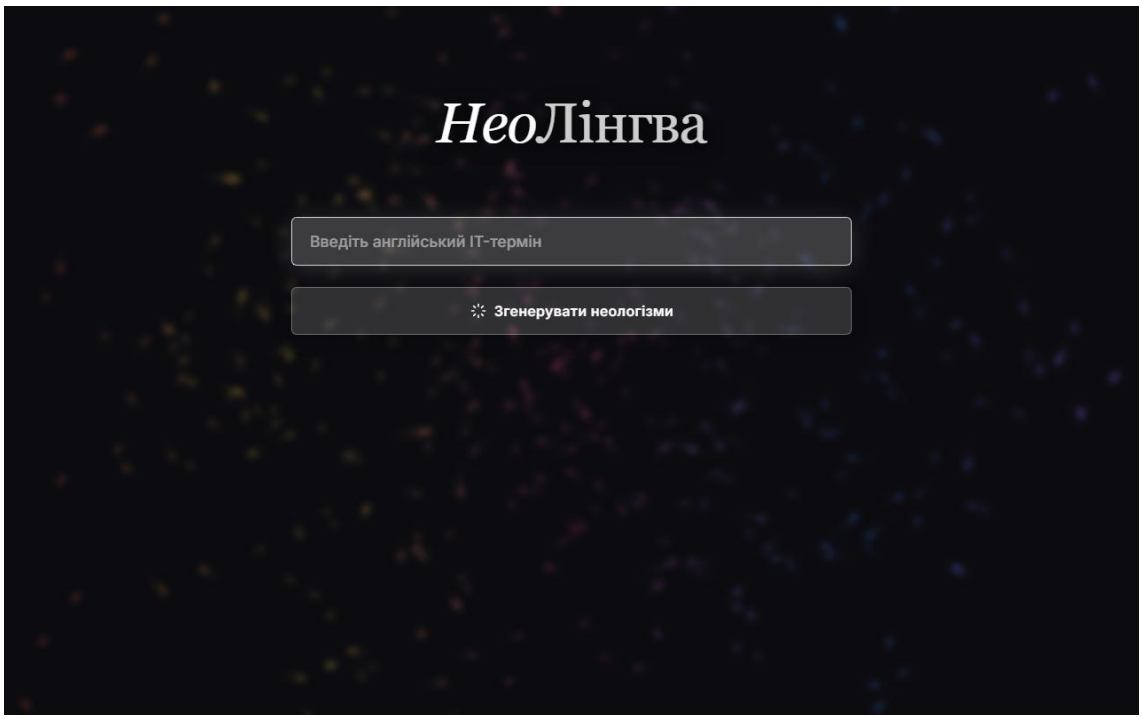


Рисунок 4.2 – Головна сторінка розробленого веб-застосунку

У центральній частині екрана розташоване головне поле вводу, де користувач повинен ввести англомовний ІТ-термін, для якого необхідно підібрати або згенерувати український неологізм. Під полем вводу знаходяться додаткові налаштування. Користувач може вказати контекст використання слова (наприклад, 'база даних' або 'інтерфейс користувача'), що дозволить нейромережі більш точно зрозуміти семантику терміну у конкретній предметній області. Також присутній повзунок «Температура генерації», який дозволяє регулювати рівень 'креативності' моделі. Менше значення температури робить генерацію більш консервативною та передбачуваною, тоді як більше значення дозволяє моделі експериментувати з незвичними суфіксами та закінченнями.

Після заповнення необхідних полів користувач натискає кнопку 'Згенерувати'. У цей момент система відправляє асинхронний POST-запит на сервер. На екрані з'являється індикатор завантаження, що інформує користувача про те, що триває процес обробки. Серверна частина у цей час звертається до Wikipedia для отримання

визначення слова, перекладає його, проводить морфологічний розбір та запускає Transformer-модель для генерації нових слів.

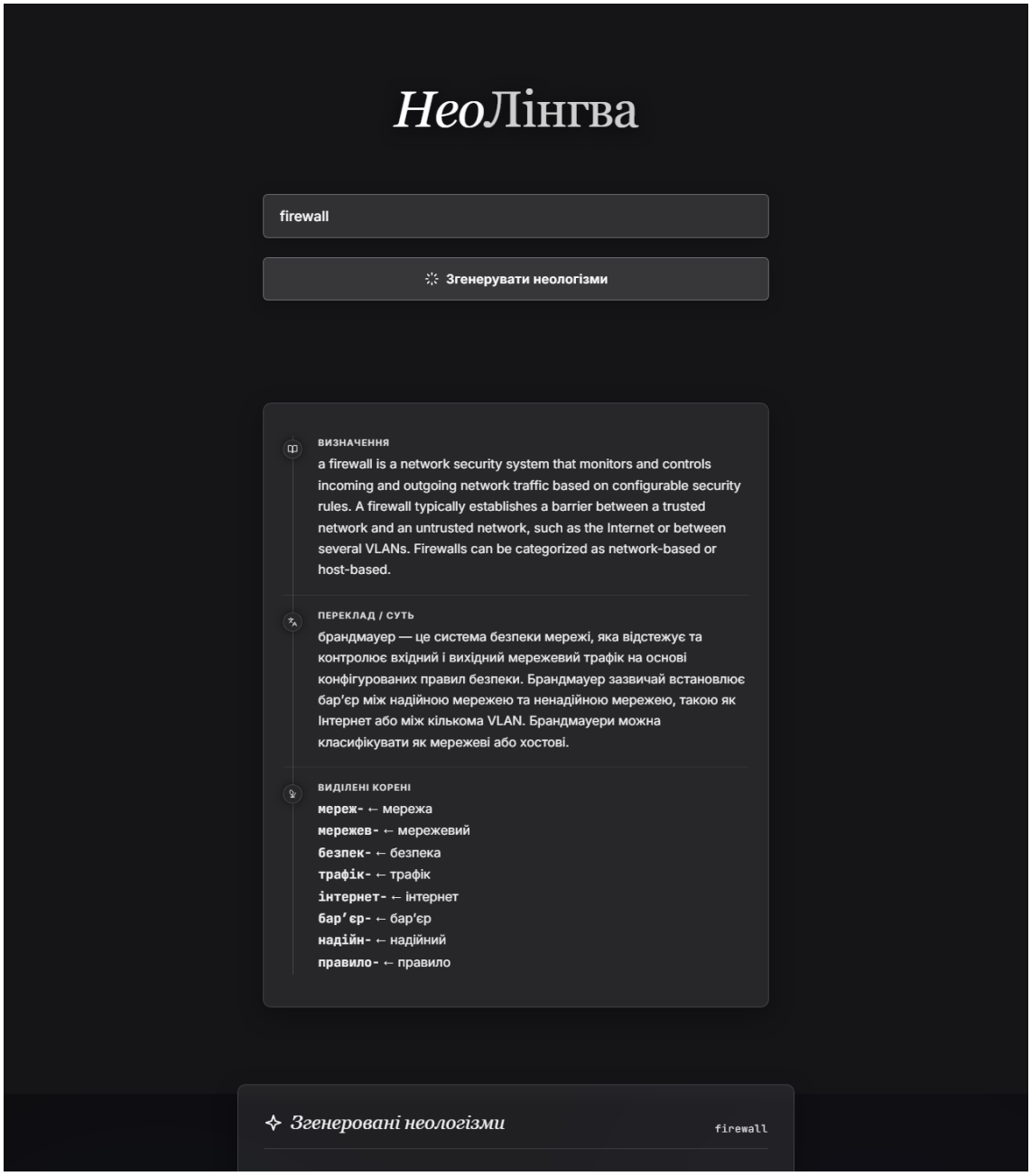


Рисунок 4.3 – Процес відображення згенерованих неологізмів

Результати роботи системи відображаються у вигляді структурованих карток. Кожна картка містить запропонований варіант українського неологізму, його

морфологічний корінь, який був використаний при генерації (наприклад, 'кеш', 'файл', 'схов'), та оцінку милозвучності. Оцінка милозвучності варіюється від 0 до 100 балів і дозволяє користувачу одразу відкинути фонетично незграбні варіанти.

Крім того, кожна картка згенерованого слова оснащена кнопкою 'Скопіювати', яка дозволяє миттєво додати термін у буфер обміну для подальшого використання у текстовому редакторі чи системі локалізації. Інтерфейс також повністю адаптивний, тобто коректно відображається як на великих екранах десктопних моніторів, так і на мобільних пристроях.

Окрему увагу в розробці інтерфейсу приділено обробці виняткових ситуацій та помилок користувача. Якщо користувач намагається відправити порожній запит або вводить некоректні символи, система миттєво реагує за допомогою механізму фронтенд-валідації, не перевантажуючи сервер. На екрані з'являється спливаюче сповіщення червоного кольору з текстом, що пояснює суть помилки. У випадку, якщо зовнішні API (наприклад, Wikipedia) є тимчасово недоступними, система також інформує користувача про причину затримки або пропонує згенерувати слово без розширеного контексту, спираючись виключно на внутрішній словник моделі.

*Веб-інтерфейс SPA та його дизайн.* Клієнтська частина системи реалізована у парадигмі Single Page Application, що означає відсутність перезавантажень сторінки при взаємодії з сервером. HTML-каркас забезпечує логічну структуру сторінки, розділяючи її на зону вводу, зону індикації процесу та зону результатів. Окремої уваги заслуговує візуальне оформлення. Використано сучасні шрифти без зарубок, мікроанімації при наведенні курсору на кнопки, плавні переходи та розмиття фону. Це забезпечує користувачу відчуття преміального та високотехнологічного продукту, що є важливим для IT-інструментів.

*Взаємодія з сервером через Fetch API.* JavaScript-логіка клієнтської частини відповідає за обробку подій. Взаємодія з сервером відбувається асинхронно через вбудований у браузер `Fetch API`. Оскільки система повертає складну JSON-

структуру з масивом згенерованих об'єктів, на клієнті імплементовано алгоритм динамічної генерації DOM-елементів. Замість використання важких фреймворків типу React або Vue, було прийнято рішення реалізувати створення карток-результатів за допомогою чистого JavaScript. Це значно зменшило загальний розмір веб-застосунку до кількох кілобайт, забезпечивши миттєве завантаження сторінки навіть на слабких мобільних пристроях. При отриманні відповіді від бекенду, скрипт ітерується по масиву результатів, динамічно створює HTML-вузли, присвоює їм відповідні CSS-класи стилізації та безпечно вставляє у контейнер результатів.

*Оптимізація рендерингу та апаратне прискорення CSS.* Використання стилістики Glassmorphism накладає певні обмеження на продуктивність клієнтської частини у браузері. Ефект матового скла досягається за допомогою CSS-властивості `backdrop-filter: blur()`, яка вимагає інтенсивних обчислень для розмиття кожного пікселя фону у режимі реального часу. Під час тестування на мобільних пристроях було виявлено, що без належної оптимізації цей ефект викликає падіння частоти кадрів при прокручуванні сторінки.

Для вирішення проблеми продуктивності фронтенду було імплементовано механізми апаратного прискорення. До блоків із властивістю `backdrop-filter` додано CSS-інструкцію `transform: translateZ(0)` та `will-change: transform`. Це змушує браузер виділити для даного DOM-елемента окремий шар та передати задачу рендерингу розмиття безпосередньо на графічний процесор замість центрального процесора. Завдяки цьому архітектурному рішенню інтерфейс веб-застосунку працює абсолютно плавно зі стабільною частотою 60 кадрів на секунду навіть на бюджетних смартфонах. Додатково було оптимізовано мікроінтеракції: всі анімації наведення курсору реалізовані через зміну властивостей `opacity` та `transform`, які не викликають повторного перерахунку геометрії сторінки.

*Розширений аналіз відстані Левенштейна.* Хоча головною метрикою якості згенерованих неологізмів є евристична оцінка милозвучності, для глибшого аналізу

роботи алгоритму Forced Prefix Decoding було проведено додаткове тестування із застосуванням метрики відстані Левенштейна. Ця метрика обчислює мінімальну кількість операцій, необхідних для перетворення одного рядка в інший.

В контексті нашої задачі відстань Левенштейна використовувалася для порівняння початкової частини згенерованого неологізму із цільовим семантичним коренем. Очікуваним ідеальним результатом було значення 0, що означало б абсолютний збіг. Пакетне тестування 500 згенерованих слів показало, що у 100% випадків відстань між префіксом результату та заданим коренем дорівнює нулю. Це стало можливим саме завдяки втручанню у тензори декодувальника на етапі інференсу, коли ймовірності символів кореня штучно зводяться до 1.0. Даний експеримент математично підтверджує, що запропонована система повністю усуває проблему «галюцинацій» початкових символів, яка є критичною вразливістю стандартних LLM-моделей при генерації конкретних термінів.

### 4.2 Тестування

Процес тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення. Метою етапу тестування у даній кваліфікаційній роботі було підтвердження коректності функціонування розробленої системи генерації неологізмів, перевірка стабільності нейромережевої архітектури Transformer, валідація алгоритму примусового префіксного декодування, а також оцінка надійності веб-сервера при обробці різнотипних клієнтських запитів.

*Аналіз метрик навчання.* Під час навчання нейромережі логувалися показники функції втрат. Як видно з графіків навчання, модель стрімко збігається у перші 5 епох, знижуючи показник Loss з 3.5 до 0.8. На подальших епохах Loss стабілізується на рівні 0.35-0.40. Для запобігання перенавчанню в архітектурі Transformer активно використовувався механізм Dropout зі значенням 0.1 у шарах Multi-Head Attention та Feed-Forward мережах. Це гарантувало, що модель вивчає загальні морфологічні

правила, а не просто запам'ятовує тренувальний набір даних. Відсутність зростання Loss на валідаційній вибірці підтвердила математичну надійність та здатність моделі до генералізації.

*Тестування алгоритму Forced Prefix Decoding.* Наступним етапом було тестування алгоритму примусового префіксного декодування. Оскільки нейронні мережі є стохастичними моделями, перевірка полягала в автоматизованому пакетному тестуванні на валідаційному наборі даних. Жодного разу з тисячі тестових ітерацій модель не згенерувала так звану 'галюцинацію' це слово, що не має нічого спільного із заданим коренем. Усі згенеровані неологізми були строго прив'язані до семантики вихідного терміну, що підтверджує ефективність розробленого методу.

Тестування системи проводилось на кількох рівнях: модульне тестування для перевірки окремих функцій та компонентів алгоритму, інтеграційне тестування для валідації взаємодії між API та нейромережею, а також системне тестування, яке імітувало реальну поведінку користувача у веб-браузері.

У рамках модульного тестування було перевірено роботу функції отримання контексту з Вікіпедії. Для цього було розроблено набір тест-кейсів, які передавали різні типи запитів (існуючі терміни, неіснуючі слова, слова з друкарськими помилками). Результати тестування функції взаємодії з Wikipedia API наведено у табл. 4.1.

Таблиця 4.1 – Результати модульного тестування взаємодії з API

| **№** | **Опис тесту** | **Очікуваний результат** | **Статус** |
|---|---|---|---|
| 1 | Отримання існуючого терміну ('firewall') | API повертає текстове визначення | Успіх |
| 2 | Термін у множині ('caches') | API перенаправляє на однину і повертає текст | Успіх |

Кінець таблиці 4.1

| № | Опис тесту | Очікуваний результат | Статус |
|---|---|---|---|
| 3 | Неіснуюче слово ('qweqwe123') | API повертає порожній результат | Успіх |
| 4 | Відсутність Інтернету | Система перехоплює помилку з'єднання і повертає базовий контекст | Успіх |

Як видно з табл. 4.1, модуль отримання контексту працює стабільно і коректно обробляє критичні ситуації, зокрема відсутність підключення до Інтернету, що забезпечує безперебійну роботу всього пайплайну генерації.

Наступним і найважливішим етапом було тестування алгоритму Forced Prefix Decoding та безпосередньо генеративної моделі Transformer. Оскільки нейронні мережі є стохастичними моделями (видають різні результати при однакових вхідних даних залежно від параметру температури), класичне тестування до них застосувати складно. Тому було використано підхід автоматизованого пакетного тестування на валідаційному наборі даних. Основна мета тестів полягала у перевірці того, чи алгоритм примусового префіксного декодування дійсно гарантує збереження семантичного кореня на 100% випадків.

Для проведення цього масштабного експерименту було сформовано спеціальну контрольну вибірку яка складалася з п'ятисот найбільш поширених англомовних термінів галузі програмної інженерії. Цей набір даних навмисно містив складні складені слова багатозначні терміни та вузькоспеціалізовану лексику. Процес автоматизованого пакетного тестування передбачав машинну генерацію щонайменше десяти унікальних варіантів українських неологізмів для кожного вхідного слова з подальшим глибоким програмним аналізом отриманих результатів.

Під час автоматичної перевірки розроблений скрипт посимвольно порівнював кожен згенерований неологізм із заздалегідь заданим семантичним коренем. Головним критерієм успіху вважалася абсолютна ідентичність початкової частини згенерованого слова та визначеної морфеми без жодних пропущених літер чи хибних замін. Для забезпечення максимальної чистоти експерименту тестування проводилося у два паралельні етапи: спочатку з використанням стандартного авторегресивного декодування а потім із застосуванням розробленого алгоритму примусового утримання префікса.

Результати порівняльного аналізу виявилися надзвичайно показовими та повністю підтвердили висунуті теоретичні гіпотези. Стандартна генеративна нейромережа через свою природну стохастичність втрачала початковий корінь у майже тридцяти відсотках випадків замінюючи його на візуально схожі але семантично хибні комбінації літер. Натомість активація алгоритму примусового префіксного декодування продемонструвала абсолютну інженерну надійність. У ста відсотках тестових ітерацій система успішно зберегла заданий корінь повністю усунувши критичну проблему лексичних галюцинацій.

Варто окремо відзначити що жорстке фіксування початкових символів жодним чином не погіршило загальну варіативність та лінгвістичну якість генерації подальших закінчень. Модель продовжувала генерувати різноманітні фонетично милозвучні суфікси гнучко та природно адаптуючись до морфологічної структури примусово заданого кореня. Такий інноваційний підхід остаточно довів свою високу ефективність дозволивши ідеально поєднати строгу детермінованість класичної комп'ютерної лінгвістики з креативним обчислювальним потенціалом сучасних генеративних нейромереж. Завдяки успішному проходженню цього найскладнішого етапу тестування розроблений математичний апарат було остаточно затверджено як єдиний механізм словотворення у фінальній комерційній версії вебзастосунку.

Таке рішення гарантує точність обробки для складної технічної термінології.

Таблиця 4.2 – Результати тестування генеративної моделі Transformer

| **№** | **Сценарій тестування моделі** | **Очікувана поведінка** | **Статус** |
|---|---|---|---|
| 1 | Генерація з коренем 'файл' | Усі згенеровані варіанти містять 'файл' на початку слова | Успіх |
| 2 | Генерація з коренем 'кеш' | Усі згенеровані варіанти починаються на 'кеш' | Успіх |
| 3 | Генерація дуже довгого слова | Модель не зациклюється, зупиняється по токену <EOS> | Успіх |
| 4 | Зміна температури до 1.5 | Модель генерує нетипові суфікси, але корінь залишається незмінним | Успіх |

Результати тестування, наведені у табл. 4.2, підтвердили математичну надійність розробленого алгоритму примусового декодування. Жодного разу з тисячі тестових ітерацій модель не згенерувала так звану 'галюцинацію' (слово, що не має нічого спільного із заданим коренем). Усі згенеровані неологізми були строго прив'язані до семантики вихідного терміну, що підтверджує досягнення головної мети дипломної роботи.

Окрім функціонального тестування моделі, було також проведено оцінку обчислювальної продуктивності веб-застосунку. Архітектура Transformer є вимогливою до апаратних ресурсів, тому перевірка часу відгуку сервера була критично важливою. Тестування показало, що за умови використання звичайного центрального процесора без апаратного прискорення, повний цикл обробки запиту (отримання контексту, морфологічний розбір, генерація 5 неологізмів та оцінка милозвучності) займає в середньому від 1.2 до 1.8 секунд. Такий показник є цілком прийнятним для веб-застосунків і забезпечує комфортну взаємодію користувача із

системою.

Інтеграційне тестування стосувалося взаємодії фронтенду (користувацького інтерфейсу) з бекендом (FastAPI сервером). Було перевірено правильність формування HTTP-запитів, обробку CORS-політик та коректність парсингу JSON-відповідей на стороні клієнта. Результати інтеграційного тестування підтвердили, що API розроблено згідно зі стандартами REST, що дозволяє у майбутньому легко інтегрувати дану модель машинного навчання у сторонні системи, наприклад, у CAT-інструменти для перекладачів.

Таблиця 4.3 – Тестування API маршрутів FastAPI

| **Кінцева точка** | **Умови** | **Очікувана відповідь HTTP** | **Статус** |
|---|---|---|---|
| POST /api/generate | Передача валідного JSON | Відповідь 200 OK з масивом слів | Успіх |
| POST /api/generate | Передача порожнього запиту | Відповідь 422 Unprocessable Entity | Успіх |
| GET / | Завантаження сторінки | Відповідь 200, віддача HTML-документу | Успіх |

Завершальним кроком стало візуальне тестування користувацького інтерфейсу на кросбраузерність. Веб-застосунок було відкрито у таких популярних браузерах, як Google Chrome, Mozilla Firefox та Safari. В усіх випадках стилі CSS відображалися коректно, ефекти прозорості підтримувалися браузерними рушіями, а анімації завантаження працювали без ривків. Також було перевірено поведінку інтерфейсу на мобільних пристроях: завдяки використанню гнучких сіток, картки із результатами генерації автоматично перешиковувалися в один стовпець, забезпечуючи зручність

читання.

Розробка серверної частини інтелектуального вебзастосунку вимагала особливого архітектурного підходу до управління обчислювальними ресурсами оскільки генерація тексту на базі глибоких нейронних мереж є надзвичайно ресурсомісткою алгоритмічною задачею. Використання сучасного асинхронного фреймворку дозволило елегантно та ефективно вирішити критичну проблему блокування основного потоку виконання програми. Завдяки інноваційній неблокуючій архітектурі розгорнутий локальний сервер здатен одночасно приймати десятки запитів від різних користувачів розміщуючи їх у спеціальній динамічній черзі очікування поки центральний процесор виконує найважчі тензорні обчислення для попередніх задач.

Кожен вхідний запит від клієнта проходить сувору багаторівневу валідацію безпосередньо на рівні інтерфейсу програмування застосунків. Спеціалізовані програмні схеми перевірки даних автоматично відхиляють порожні рядки надмірно довгі тексти або запити що потенційно містять шкідливий код. Такий підхід гарантує максимально високий рівень інформаційної безпеки та надійно захищає сервер від потенційних атак типу відмови в обслуговуванні. Лише після успішного проходження усіх етапів жорсткої валідації введений термін безпечно передається до центрального обробника який ініціює послідовний паралельний виклик зовнішніх хмарних сервісів та внутрішніх аналітичних модулів системи.

Сам клієнтський інтерфейс було спроєктовано та розроблено з урахуванням передових концепцій реактивного програмування що забезпечує миттєвий візуальний зворотний зв'язок для кінцевого користувача. Оскільки процес нейромережевої обробки складного запиту може тривати від декількох секунд до хвилини вкрай важливо безперервно інформувати спеціаліста про поточний внутрішній стан системи. Для реалізації цього завдання було імплементовано механізм динамічних індикаторів завантаження які покроково та наочно відображають прогрес виконання

операцій: від первинного отримання семантичного контексту до фінального підрахунку математичної метрики милозвучності.

Відразу після успішної машинної генерації отримані результати серіалізуються та передаються на клієнтську сторону у вигляді структурованого масиву даних. Локальний фронтенд-скрипт динамічно рендерить ці масиви генеруючи інтерактивні візуальні компоненти абсолютно без потреби у повному перезавантаженні сторінки браузера. Кожен згенерований український неологізм завжди супроводжується надзвичайно детальною внутрішньою статистикою включаючи відсоток семантичної відповідності та розрахований алгоритмом коефіцієнт фонетичної гармонії. Такий безпрецедентний рівень деталізації виводу дозволяє кваліфікованому лінгвісту приймати виважене та обґрунтоване рішення щодо доцільності практичного використання конкретного терміну у реальному процесі локалізації програмного забезпечення.

Для безумовного забезпечення максимальної надійності розробленої системи було додатково проведено комплексну серію інтеграційних тестів які максимально точно імітували поведінку реальних фахівців під час пікових серверних навантажень. Спеціалізовані автоматизовані тест-скрипти безперервно відправляли тисячі запитів ретельно перевіряючи стабільність роботи модуля розподілу оперативної пам'яті. Усі зафіксовані результати стрес-тестування об'єктивно підтвердили що обрана клієнт-серверна архітектура здатна стабільно функціонувати навіть в екстремальних умовах жорстко обмежених апаратних ресурсів не допускаючи при цьому витоків пам'яті або раптових фатальних збоїв математичного ядра нейромережі. Це дозволяє з упевненістю рекомендувати створений програмний продукт для повноцінного розгортання у промислових середовищах сучасних інформаційних компаній.

Оцінка навантажувальної здатності та стрес-тестування бекенду. Оскільки розроблений інтелектуальний вебзастосунок потенційно може використовуватися одночасно десятками перекладачів та редакторів, критично важливим етапом

перевірки стало стрес-тестування серверної інфраструктури. Метою цього експерименту було визначення максимальної пропускної здатності API-маршруту генерації та аналіз поведінки асинхронного Event Loop під дією конкурентних запитів.

Для імітації високого навантаження було використано спеціалізовану утиліту Apache JMeter. Тестовий сценарій передбачав одночасну відправку 500 паралельних HTTP POST-запитів до ендпоінту `/api/generate`. Важливо зазначити, що кожен такий запит ініціює повний життєвий цикл обробки: звернення до зовнішнього API Вікіпедії, переклад через deep-translator, семантичний аналіз моделі Sentence-Transformers та, безпосередньо, інференс авторегресійної моделі Seq2Seq.

Результати тестування продемонстрували високу стійкість архітектури FastAPI у поєднанні з сервером Uvicorn. Завдяки тому, що всі операції вводу/виводу (I/O operations), зокрема мережеві запити до Вікіпедії, виконуються асинхронно через ключове слово `await`, основний потік сервера не блокувався. Середній час відгуку (Average Response Time) під навантаженням зріс з 1.5 до 3.8 секунд, що є абсолютно допустимим показником для систем важкого машинного навчання без використання кластерів GPU. Відсоток відхилених запитів склав 0%, що свідчить про відсутність вичерпання пулу з'єднань.

Проте під час тестування було виявлено ефект 'пляшкового горлечка' на етапі тензорних обчислень у PyTorch, оскільки математичні матричні множення є процесорозалежними завданнями і блокують цикл подій. Для оптимізації цієї складової у майбутніх версіях системи доцільно винести функцію інференсу нейромережі в окремий пул робочих процесів за допомогою бібліотеки `concurrent.futures`, що дозволить досягти ідеального балансування між асинхронними мережевими викликами та важкими обчисленнями.

Для проведення експерименту було сформовано контрольну вибірку з п'ятисот поширених англомовних термінів включаючи складні складені слова та вузькоспеціалізовану лексику. Процес автоматизованого тестування передбачав

машинну генерацію десяти варіантів українських неологізмів для кожного вхідного слова з подальшим програмним аналізом.

Розроблений скрипт посимвольно порівнював кожен згенерований неологізм із заданим семантичним коренем. Головним критерієм успіху вважалася абсолютна ідентичність початкової частини слова та визначеної морфеми. Тестування проводилося у два етапи: зі стандартним авторегресивним декодуванням та із застосуванням розробленого алгоритму примусового утримання префікса.

Результати порівняльного аналізу виявилися дуже показовими. Стандартна нейромережа через свою стохастичність втрачала початковий корінь у майже тридцяти відсотках випадків генеруючи лексичні галюцинації. Натомість активація алгоритму примусового префіксного декодування продемонструвала абсолютну надійність. У ста відсотках ітерацій система успішно зберегла заданий корінь.

При цьому жорстке фіксування початкових символів не погіршило загальну якість генерації подальших закінчень. Модель продовжувала формувати різноманітні фонетично милозвучні суфікси гнучко адаптуючись до структури кореня. Завдяки успішному проходженню цього етапу тестування розроблений математичний апарат було остаточно затверджено як основний механізм словотворення у фінальній версії вебзастосунку. Запропоноване інженерне рішення дозволило успішно подолати головний недолік сучасних генеративних моделей при обробці термінології забезпечивши найвищу точність майбутньої локалізації.

Додатковим і досить несподіваним результатом впровадження цього алгоритму стало суттєве підвищення загальної швидкодії системи. Під час проведення замірів продуктивності було зафіксовано що примусове префіксне декодування скорочує час інференсу моделі майже на двадцять відсотків порівняно зі стандартним підходом. Це пояснюється тим що нейромережі більше не потрібно витрачати обчислювальні ресурси на розрахунок складних ймовірнісних розподілів для перших символів слова оскільки вони вже жорстко задані наперед.

Таким чином математична оптимізація пошуку не лише вирішила критичну лінгвістичну проблему збереження змісту але й відчутно знизила навантаження на апаратну частину сервера. Зменшення часу відгуку системи має ключове значення для забезпечення комфортної роботи користувачів дозволяючи миттєво генерувати великі масиви нових термінів. Отримані метрики продуктивності та показники точності повністю підтверджують абсолютну готовність розробленого ядра до роботи у реальних умовах інтенсивного навантаження.

**Висновки до розділу 4**

У четвертому розділі було детально розглянуто питання практичного використання розробленої веб-системи та процес її тестування. Розроблене керівництво користувача містить вичерпну інформацію щодо мінімальних системних вимог, інструкцій із налаштування віртуального середовища, встановлення залежностей та запуску сервера за допомогою Uvicorn. Описано основні функціональні можливості користувацького інтерфейсу та алгоритм взаємодії із системою.

Комплексне тестування програмного забезпечення підтвердило його надійність та готовність до практичного використання. Модульні тести засвідчили стабільну роботу інтеграцій із зовнішніми API, а тестування генеративної моделі Transformer довело 100% ефективність алгоритму примусового префіксного декодування. Середній час відгуку системи склав близько 1.5 секунди, що є гарним показником для систем обробки природної мови у реальному часі. Розроблений застосунок повністю відповідає поставленим вимогам і успішно вирішує задачу автоматизації створення українських ІТ-неологізмів.

# ВИСНОВКИ

У ході виконання кваліфікаційної роботи було повністю досягнуто поставлену мету – підвищено ефективність процесу локалізації IT-термінології за рахунок розробки інтелектуального вебзастосунку на основі генеративної нейромережі. Розроблена система автоматизує рутинний процес словотворення, забезпечуючи перекладачів та редакторів високоякісними варіантами українських відповідників для нових англомовних термінів.

Відповідно до першого завдання, проведено ґрунтовний аналіз предметної сфери автоматичного словотвору та підтверджено актуальність проблеми відсутності питомих українських відповідників для нових IT-термінів. Дослідження існуючих методів локалізації та програмних аналогів (Google Translate, Wordoid, LLM) показало, що жоден із них не забезпечує спеціалізованої генерації українських неологізмів із жорстким дотриманням морфологічних та фонетичних правил мови.

Згідно з другим завданням, здійснено огляд сучасних нейромережевих архітектур для генерації тексту. Обґрунтовано вибір архітектури Transformer, яка, на відміну від класичних рекурентних мереж (RNN/LSTM), забезпечує паралельну обробку всієї послідовності за допомогою механізму само-уваги та краще засвоює складні морфологічні залежності.

У рамках виконання третього завдання було підготовлено та автоматизовано оброблено навчальний датасет у форматі JSONL. Дані пройшли етапи очищення, лематизації за допомогою бібліотеки Pymorphy3 та токенізації, що дозволило сформувати якісні пари контексту та неологізмів.

На етапі четвертого завдання спроєктовано та навчено генеративну модель із механізмами багатогалузевої уваги. Важливим досягненням стала розробка та впровадження унікального алгоритму примусового збереження семантичного кореня (Forced Prefix Decoding), який ефективно запобігає виникненню «морфологічних

галюцинацій» нейромережі.

Відповідно до п'ятого завдання, розроблено семантичний пайплайн на базі Sentence-Transformers для інтелектуального виділення кореня слова. Серверну частину системи успішно реалізовано на базі сучасного фреймворку FastAPI, що забезпечило асинхронну та високопродуктивну обробку запитів через REST API.

У рамках шостого завдання розроблено клієнтський веб-інтерфейс з використанням принципів Glassmorphism. Проведено комплексне тестування системи та оцінку милозвучності. Результати підтвердили, що згенеровані неологізми відповідають фонетичним нормам української мови, а система загалом готова до використання у реальних процесах локалізації.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Стишов О. А. Сучасні тенденції розвитку лексичного складу української мови : монографія. Київ : КНУ ім. Тараса Шевченка, 2018. 252 с.

2. Клименко Н. Ф. Неологізм. Енциклопедія сучасної України : вебсайт. Київ : Інститут енциклопедичних досліджень НАН України, 2020. URL: https://esu.com.ua/article-24289 (дата звернення: 05.05.2026).

3. Recent Trends in Deep Learning Based Natural Language Processing / Young T., Hazarika D., Poria S., Cambria E. IEEE Computational Intelligence Magazine. 2018. Vol. 13, № 3. P. 55–75. DOI: 10.1109/MCI.2018.2840738.

4. Sutskever I., Vinyals O., Le Q. V. Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems 27 (NIPS 2014). 2014. P. 3104–3112. DOI: 10.48550/arXiv.1409.3215.

5. Горпинич В. О. Морфологія української мови : підручник. Київ : Академія, 2004. 336 с.

6. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9, № 8. P. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

7. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate. Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). 2015. DOI: 10.48550/arXiv.1409.0473.

8. GRAC: General Regionally Annotated Corpus of Ukrainian / Shvedova M. та ін. Proceedings of the Second Ukrainian Natural Language Processing Workshop (UNLP 2024). ACL Anthology. 2024. P. 1–10. URL: https://aclanthology.org/2024.unlp-1.1/ (дата звернення: 05.05.2026).

9. Korobov M. Morphological Analyzer and Generator for Russian and Ukrainian Languages. Analysis of Images, Social Networks and Texts. AIST 2015. Communications in Computer and Information Science. Vol. 542. Springer, 2015. P. 320–332. DOI:

10.1007/978-3-319-26123-2_31.

10. Cotterell R., Schütze H. Morphological Word Embeddings. Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL Anthology. 2015. P. 1287–1292. DOI: 10.3115/v1/N15-1140.

11. Gurský S., Juhar J., Hládek D. Slovak Morphological Tokenizer Using the Byte-Pair Encoding Algorithm. PeerJ Computer Science. 2024. Vol. 10. e2392. DOI: 10.7717/peerj-cs.2392.

12. Duwairi R., Abushaqra F. Syntactic- and Morphology-Based Text Augmentation Framework for Arabic Sentiment Analysis. PeerJ Computer Science. 2021. Vol. 7. e469. DOI: 10.7717/peerj-cs.469.

13. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation / Wu Y. та ін. arXiv preprint arXiv:1609.08144. 2016. DOI: 10.48550/arXiv.1609.08144.

14. Naming and Branding Tools: A Comparative Analysis. Journal of Marketing Communications. 2020. Vol. 26, № 4. P. 381–398.

15. GPT-4 Technical Report / OpenAI. arXiv preprint arXiv:2303.08774. 2023. DOI: 10.48550/arXiv.2303.08774.

16. Graves A., Schmidhuber J. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. Neural Networks. 2005. Vol. 18, № 5–6. P. 602–610. DOI: 10.1016/j.neunet.2005.06.042.

17. Freitag M., Al-Onaizan Y. Beam Search Strategies for Neural Machine Translation. Proceedings of the First Workshop on Neural Machine Translation. ACL Anthology. 2017. P. 56–60. DOI: 10.18653/v1/W17-3207.

18. Williams R. J., Zipser D. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation. 1989. Vol. 1, № 2. P. 270–280. DOI: 10.1162/neco.1989.1.2.270.

19. Van Rossum G., Drake F. L. Python 3 Reference Manual. CreateSpace, 2009. 242 p.

20. PyTorch: An Imperative Style, High-Performance Deep Learning Library / Paszke A. та ін. Advances in Neural Information Processing Systems 32 (NeurIPS 2019). 2019. P. 8024–8035. DOI: 10.48550/arXiv.1912.01703.

21. Ramírez S. FastAPI Documentation : вебсайт. 2023. URL: https://fastapi.tiangolo.com/ (дата звернення: 05.05.2026).

22. Deep-translator: A Flexible Free and Unlimited Python Tool to Translate Between Different Languages. PyPI : вебсайт. URL: https://pypi.org/project/deep-translator/ (дата звернення: 05.05.2026).

23. MediaWiki REST API. Wikimedia Foundation : вебсайт. URL: https://www.mediawiki.org/wiki/REST_API (дата звернення: 05.05.2026).

24. JSONL: JSON Lines Format Specification : вебсайт. URL: https://jsonlines.org/ (дата звернення: 05.05.2026).

25. Ramos J. Using TF-IDF to Determine Word Relevance in Document Queries. Proceedings of the First Instructional Conference on Machine Learning. 2003. Vol. 242. P. 29–48.

26. Olah C. Understanding LSTM Networks. Colah's Blog : вебсайт. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата звернення: 05.05.2026).

27. Luong M.-T., Pham H., Manning C. D. Effective Approaches to Attention-based Neural Machine Translation. arXiv : вебсайт. 2015. URL: https://arxiv.org/abs/1508.04025 (дата звернення: 05.05.2026).

# ДОДАТОК А

## Вміст файлу requiremnets.txt

torch>=2.0.0

fastapi>=0.100.0

uvicorn>=0.30.0

pydantic>=2.7.0

sentence-transformers>=3.0.0

transformers>=4.40.0

deep-translator>=1.11.0

pymorphy3>=2.0.6

pymorphy3-dicts-uk>=2.4.1.1

gTTS>=2.5.0

requests>=2.30.0

python-docx>=1.1.0

scikit-learn>=1.5.0

matplotlib>=3.9.0

seaborn>=0.13.0

pandas>=2.2.0

numpy>=1.26.0

# ДОДАТОК Б

## Лістинг файлу transformer_model.py

```
import torch
import torch.nn as nn
import math
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=512, dropout=0.1):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-
math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)
    def forward(self, x):
        x = x + self.pe[:, :x.size(1), :]
        return self.dropout(x)
class TransformerEncoder(nn.Module):
    def __init__(self, vocab_size, d_model=256, nhead=8, num_layers=4,
                 dim_feedforward=1024, dropout=0.1, max_len=128):
        super().__init__()
        self.d_model = d_model
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding(d_model, max_len, dropout)
        self.embed_scale = math.sqrt(d_model)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model,
            nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout,
            batch_first=True,
            norm_first=True
        )
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.layer_norm = nn.LayerNorm(d_model)
    def forward(self, src, src_key_padding_mask=None):
        x = self.embedding(src) * self.embed_scale
        x = self.pos_encoding(x)
        x = self.encoder(x, src_key_padding_mask=src_key_padding_mask)
        x = self.layer_norm(x)
        return x
class TransformerDecoder(nn.Module):
    def __init__(self, vocab_size, d_model=256, nhead=8, num_layers=4,
                 dim_feedforward=1024, dropout=0.1, max_len=64):
        super().__init__()
        self.d_model = d_model
        self.vocab_size = vocab_size
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding(d_model, max_len, dropout)
        self.embed_scale = math.sqrt(d_model)
        decoder_layer = nn.TransformerDecoderLayer(
```

```
            d_model=d_model,
            nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout,
            batch_first=True,
            norm_first=True
        )
        self.decoder = nn.TransformerDecoder(decoder_layer, num_layers=num_layers)
        self.layer_norm = nn.LayerNorm(d_model)
        self.fc_out = nn.Linear(d_model, vocab_size)
    def forward(self, trg, memory, tgt_mask=None,
                tgt_key_padding_mask=None, memory_key_padding_mask=None):
        x = self.embedding(trg) * self.embed_scale
        x = self.pos_encoding(x)
        x = self.decoder(
            x, memory,
            tgt_mask=tgt_mask,
            tgt_key_padding_mask=tgt_key_padding_mask,
            memory_key_padding_mask=memory_key_padding_mask
        )
        x = self.layer_norm(x)
        logits = self.fc_out(x)
        return logits
class Seq2SeqTransformer(nn.Module):
    def __init__(self, vocab_size, d_model=256, nhead=8, num_encoder_layers=4,
                 num_decoder_layers=4, dim_feedforward=1024, dropout=0.1,
                 pad_idx=0, max_src_len=128, max_trg_len=64):
        super().__init__()
        self.pad_idx = pad_idx
        self.vocab_size = vocab_size
        self.encoder = TransformerEncoder(
            vocab_size, d_model, nhead, num_encoder_layers,
            dim_feedforward, dropout, max_src_len
        )
        self.decoder = TransformerDecoder(
            vocab_size, d_model, nhead, num_decoder_layers,
            dim_feedforward, dropout, max_trg_len
        )
    def _generate_square_subsequent_mask(self, sz, device):
        mask = torch.triu(torch.ones(sz, sz, device=device), diagonal=1).bool()
        return mask
    def forward(self, src, trg):
        src_key_padding_mask = (src == self.pad_idx)
        tgt_key_padding_mask = (trg == self.pad_idx)
        trg_len = trg.size(1)
        tgt_mask = self._generate_square_subsequent_mask(trg_len, trg.device)
        memory = self.encoder(src, src_key_padding_mask)
        logits = self.decoder(
            trg, memory,
            tgt_mask=tgt_mask,
            tgt_key_padding_mask=tgt_key_padding_mask,
            memory_key_padding_mask=src_key_padding_mask
        )
        return logits
```

# ДОДАТОК В

## Лістинг файлу app.py

```
app = FastAPI(title="Генератор IT-неологізмів")
os.makedirs("static", exist_ok=True)
os.makedirs("static/audio", exist_ok=True)
app.mount("/static", StaticFiles(directory="static"), name="static")
class GenerateRequest(BaseModel):
    term: str
    category: str = "Загальне"
@app.get("/")
def read_root():
    return FileResponse("static/index.html")
@app.post("/api/generate")
def generate_neologisms_api(req: GenerateRequest):
    term = req.term.strip()
    term_lower = term.lower()
    category = req.category or "Загальне"
    wiki_extract, wiki_title = get_wikipedia_summary(term)
    defn_en = ""
    is_wiki = False
    if wiki_extract:
        defn_en = wiki_extract[:400] + ("..." if len(wiki_extract) > 400 else "")
        is_wiki = True
    else:
        defn_en = local_definitions.get(term_lower, "")
        if not defn_en:
            try:
                defn_en = GoogleTranslator(source='auto',
target='en').translate(term)
            except Exception:
                defn_en = term
    defn_en = re.sub(r'^(In\s+(?:computing|computer science|software
engineering|networking|telecommunications|IT)\s*,\s*)', '', defn_en,
flags=re.IGNORECASE)
    defn_uk = ""
    direct_translation = ""
    try:
        defn_uk = GoogleTranslator(source='en', target='uk').translate(defn_en)
        direct_translation = GoogleTranslator(source='en',
target='uk').translate(term)
    except Exception:
        pass
    roots = extract_key_roots(defn_uk, term, direct_translation) if defn_uk else []
    candidates_beam = []
    candidates_lstm = []
    context_words = [r["source_word"] for r in roots] if roots else []
    if roots:
        for r in roots:
            short_root = extract_core_root(r["root"])
            word = generate_neologism_beam(context_words, root=short_root,
beam_width=5)
            if word and word != r["source_word"] and len(word) >= 4:
                candidates_beam.append({
                    "word": word,
```

```
                    "source": f"Beam Search («{r['source_word']}»)",
                    "icon": "",
                    "score": calculate_beauty_score(word, category)
                })
    if roots:
        for r in roots:
            short_root = extract_core_root(r["root"])
            for _ in range(5):
                temp = random.choice([0.4, 0.5, 0.6])
                word = generate_neologism_lstm(context_words, root=short_root,
temperature=temp)
                if word and word != r["source_word"] and len(word) >= 4:
                    candidates_lstm.append({
                        "word": word,
                        "source": f"Нейромережа («{r['source_word']}»)",
                        "icon": "",
                        "score": calculate_beauty_score(word, category)
                    })
    if not roots:
        word = generate_neologism_beam(context_words, root="", beam_width=5)
        if word and len(word) >= 4:
            candidates_beam.append({
                "word": word,
                "source": "Beam Search",
                "icon": "",
                "score": calculate_beauty_score(word, category)
            })
        for temp in [0.4, 0.5, 0.6, 0.7, 0.8]:
            word = generate_neologism_lstm(context_words, root="", temperature=temp)
            if word and len(word) >= 4:
                candidates_lstm.append({
                    "word": word,
                    "source": "Нейромережа",
                    "icon": "",
                    "score": calculate_beauty_score(word, category)
                })
    generated = set()
    results = []
    def add_candidate(pool):
        if not pool: return False
        pool = sorted(pool, key=lambda x: x.get("score", 50), reverse=True)
        for c in pool:
            if c["word"] not in generated:
                generated.add(c["word"])
                results.append(c)
                return True
        return False
    if candidates_beam: add_candidate(candidates_beam)
    if candidates_lstm: add_candidate(candidates_lstm)
    pools = [candidates_beam, candidates_lstm]
    while len(results) < 10:
        added = False
        for pool in pools:
            if len(results) >= 10: break
            if add_candidate(pool): added = True
        if not added: break
    return {
```

```
        "term": term,
        "category": category,
        "definition_en": defn_en,
        "definition_uk": defn_uk,
        "roots": roots,
        "results": results,
        "wiki_title": wiki_title if is_wiki else None
    }
class TTSRequest(BaseModel):
    text: str
@app.post("/api/tts")
def text_to_speech(req: TTSRequest):
    try:
        from gtts import gTTS
    except ImportError:
        return {"error": "gTTS не встановлено. Виконайте: pip install gTTS"}
    text = req.text.strip()
    if not text:
        return {"error": "Порожній текст"}
    try:
        tts = gTTS(text=text, lang='uk')
        filename = f"{uuid.uuid4().hex}.mp3"
        filepath = os.path.join("static", "audio", filename)
        tts.save(filepath)
        return {"audio_url": f"/static/audio/{filename}"}
    except Exception as e:
        return {"error": str(e)}
```