

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

_____ Розробка системи генерації тексту за допомогою нейронних мереж

(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-2

_____ Олександр МАТЮХІН
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна
Освітня програма _____ Штучний інтелект

(повна назва освітньої програми)

Керівник _____ ас. Дмитро МАЛЄСВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

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

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« _____ » _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Матюхіну Олександрю Богдановичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Розробка системи генерації тексту за допомогою нейронних мереж _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 10 липня 2026 р.

3. Вихідні дані до роботи Науково-технічні публікації з обробки природної мови, генерації тексту та архітектури Transformer; документація Python і бібліотек PyTorch, Hugging Face Transformers, PEFT; документація моделі GPT-Neo; відкриті україномовні корпуси (UberText 2.0 та новинні статті) для донавчання й тестування; документація FastAPI та Gradio.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та огляд методів генерації тексту _____

2) Проєктування системи генерації тексту _____

3) Реалізація та програмна розробка _____

4) Експериментальне дослідження та оцінка результатів _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	11.06.2026	виконано
3	Огляд існуючих систем і підходів до генерації тексту	12.06.2026	виконано
4	Проектування архітектури системи генерації тексту	14.06.2026	виконано
5	Підготовка та токенизація навчального корпусу	16.06.2026	виконано
6	Реалізація модуля донавчання мовної моделі (LoRA)	18.06.2026	виконано
7	Розробка демонстраційного веб-інтерфейсу	20.06.2026	виконано
8	Експериментальне дослідження якості та продуктивності	22.06.2026	виконано
9	Аналіз результатів досліджень	23.06.2026	виконано
10	Оптимізація параметрів моделі	23.06.2026	виконано
11	Підготовка презентаційних матеріалів	24.06.2026	виконано
12	Тестування системи	24.06.2026	виконано
13	Написання пояснювальної записки	25.06.2026	виконано
14	Попередній захист	05.07.2026	виконано
15	Захист перед екзаменаційною комісією	08.07.2026	виконано

Дата видачі завдання 8 червня 2026 р.

Здобувач 
(підпис)

Керівник роботи _____
(підпис)

ас. Дмитро МАЛЄЄВ _____
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 64 с., 8 рис., 9 табл., 3 дод., 36 джерел.

ГЕНЕРАЦІЯ ТЕКСТУ, ГЛИБИННЕ НАВЧАННЯ, МОВНІ МОДЕЛІ, НЕЙРОННІ МЕРЕЖІ, ОБРОБКА ПРИРОДНОЇ МОВИ, ТРАНСФОРМЕР.

Об'єкт дослідження – задача автоматичної генерації текстів природною мовою.

Предмет дослідження – застосування нейронних мереж архітектури Transformer для побудови системи генерації тексту.

Мета роботи – розроблення системи автоматичної генерації зв'язного українськомовного тексту на основі попередньо навченої мовної моделі архітектури Transformer із донавчанням під цільову предметну галузь.

Методи дослідження – теоретичний аналіз науково-технічної літератури та порівняльний аналіз архітектур нейронних мереж; експериментальні методи – програмна реалізація системи, донавчання мовної моделі методом LoRA та оцінювання якості генерації за метриками перплексії, BLEU та ROUGE. Методи розробки базуються на технологіях Python із бібліотеками PyTorch, Hugging Face Transformers і PEFT.

У результаті виконання кваліфікаційної роботи розроблено, реалізовано та експериментально досліджено систему генерації тексту. Виконано аналіз предметної галузі та огляд архітектур нейронних мереж; спроєктовано архітектуру системи та її програмних модулів. Сформовано українськомовний навчальний корпус та виконано донавчання моделі GPT-Neo методом низькорангової адаптації (LoRA). Експериментально підтверджено, що донавчання знизило перплексію базової моделі та підвищило якість генерації за метриками BLEU та ROUGE. Розроблену систему можна використовувати для автоматизованого створення текстового контенту у вибраній предметній галузі.

ABSTRACT

Explanatory note: 64 pp., 8 fig., 9 tabl., 3 ann., 36 references.

DEEP LEARNING, LANGUAGE MODELS, NATURAL LANGUAGE PROCESSING, NEURAL NETWORKS, TEXT GENERATION, TRANSFORMER.

The object of research is the task of automatic generation of natural language texts.

The subject of research is the application of neural networks of the Transformer architecture to building a text generation system.

The aim of the work is to develop a system for automatic generation of coherent Ukrainian-language text based on a pre-trained Transformer language model fine-tuned for the target domain.

Research methods include theoretical and comparative analysis of neural network architectures, software implementation of the system, fine-tuning of the language model with the LoRA method, and evaluation of generation quality using perplexity, BLEU and ROUGE metrics. The system is implemented in Python with the PyTorch, Hugging Face Transformers and PEFT libraries.

As a result of the work, a text generation system was developed, implemented and experimentally evaluated. The system architecture and software modules were designed, a Ukrainian-language training corpus was prepared, and the GPT-Neo model was fine-tuned using the Low-Rank Adaptation (LoRA) method. It was experimentally confirmed that fine-tuning reduced the perplexity of the base model and improved generation quality in terms of the BLEU and ROUGE metrics. The developed system can be used for the automated creation of textual content in the selected domain.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та огляд методів генерації тексту	11
1.1 Загальна характеристика задачі генерації тексту	11
1.1.1 Класифікація систем генерації тексту	12
1.2 Еволюція підходів до автоматичної генерації тексту	13
1.2.1 Шаблонні та статистичні методи генерації.....	14
1.2.2 Рекурентні нейронні мережі та їх модифікації.....	15
1.3 Архітектура Transformer та механізм уваги	17
1.3.1 Механізм Self-Attention та Multi-Head Attention.....	17
1.3.2 Структура енкодера та декодера	18
1.4 Сучасні великі мовні моделі та методи їх адаптації.....	19
1.4.1 Сімейство моделей GPT	19
1.4.2 Методи донавчання та адаптації мовних моделей	21
1.4.3 Метрики оцінювання якості генерованих текстів	22
2 Проектування системи генерації тексту	24
2.1 Постановка задачі.....	24
2.2 Функціональні та нефункціональні вимоги	25
2.3 Аналіз варіантів архітектури та вибір моделі	27
2.3.1 Порівняння можливих підходів.....	27
2.4 Обґрунтування вибору архітектури Transformer та GPT-подібної моделі.....	28
2.5 Опис набору даних.....	29
2.6 Підготовка даних і токенизація	30
2.7 Процес навчання	32
2.8 Архітектура програмної системи	34
2.9 Алгоритм генерації тексту	36
3 Реалізація та програмна розробка.....	37

3.1 Середовище розробки та використані технології	37
3.2 Структура проєкту	38
3.3 Реалізація модуля підготовки даних	39
3.4 Реалізація процесу навчання.....	40
3.5 Реалізація процесу генерації.....	42
3.6 Реалізація програмного інтерфейсу	43
3.7 Особливості розгортання	45
4 Експериментальне дослідження та оцінка результатів	46
4.1 Методика експериментів	46
4.2 Конфігурація обладнання та параметри навчання	47
4.3 Результати навчання	47
4.4 Оцінювання за метриками якості	49
4.5 Порівняння з базовими підходами	50
4.6 Аналіз прикладів згенерованих текстів	51
4.7 Переваги та недоліки розробленої системи	51
Висновки	53
Перелік джерел посилання	55
Додаток А Зразки згенерованих текстів	60
Додаток Б Фрагменти програмного коду системи	61
Додаток В Відомість кваліфікаційної роботи	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс застосування;

BLEU – Bilingual Evaluation Understudy – метрика оцінювання якості генерації;

BPE – Byte-Pair Encoding – кодування байтових пар (алгоритм токенизації);

GPT – Generative Pre-trained Transformer – генеративний попередньо навчений трансформер;

GPU – Graphics Processing Unit – графічний процесор;

GRU – Gated Recurrent Unit – вентильний рекурентний вузол;

LLM – Large Language Model – велика мовна модель;

LoRA – Low-Rank Adaptation – низькорангова адаптація;

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

NLP – Natural Language Processing – обробка природної мови;

PEFT – Parameter-Efficient Fine-Tuning – параметрично ефективне донавчання;

REST – Representational State Transfer – архітектурний стиль програмного інтерфейсу;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

ROUGE – Recall-Oriented Understudy for Gisting Evaluation – метрика оцінювання якості генерації.

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів текстової інформації, яка потребує автоматизованої обробки. Однією з найбільш актуальних задач у галузі обробки природної мови (Natural Language Processing, NLP) [1] є задача генерації тексту – створення зв'язного, граматично та семантично коректного тексту природною мовою на основі заданих умов або контексту.

Системи генерації тексту знаходять застосування у багатьох сферах: автоматичне реферування документів, машинний переклад, створення чат-ботів і віртуальних асистентів, генерація описів зображень, автоматичне складання звітів, написання новин, креативне письмо та інші. Поява великих мовних моделей (Large Language Models, LLM) на основі архітектури Transformer відкрила нові можливості для побудови систем, здатних генерувати тексти, які за якістю наближаються до текстів, написаних людиною.

Незважаючи на значний прогрес у цій галузі, задача побудови системи генерації тексту, яка б ефективно працювала у конкретній предметній галузі та відповідала специфічним вимогам користувача, залишається актуальною. Це зумовлено такими чинниками: необхідністю адаптації загальних мовних моделей до спеціалізованих доменів, потребою у контролі стилю та змісту генерованого тексту, обмеженнями обчислювальних ресурсів та вимогами до якості й узгодженості згенерованих текстів.

Метою кваліфікаційної роботи є розроблення системи генерації тексту на основі нейронних мереж, дослідження сучасних підходів до побудови таких систем, аналіз існуючих архітектур та методів навчання мовних моделей, обґрунтування вибору архітектури для подальшої реалізації та формулювання постановки задачі для кваліфікаційної роботи.

Для досягнення поставленої мети необхідно вирішити такі задачі: провести огляд предметної галузі та визначити основні поняття, пов'язані з

генерацією тексту природною мовою. Проаналізувати еволюцію підходів до генерації тексту – від статистичних методів до нейромережових. Дослідити архітектури сучасних нейронних мереж, що використовуються для побудови мовних моделей, зокрема рекурентні мережі (RNN, LSTM, GRU) та архітектуру Transformer. Розглянути існуючі попередньо навчені мовні моделі сімейств GPT, BERT, T5 та інші, проаналізувати їх переваги та обмеження. Обґрунтувати вибір підходу для розробки системи генерації тексту в межах кваліфікаційної роботи. Сформулювати постановку задачі та визначити основні вимоги до системи, що розробляється.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОГЛЯД МЕТОДІВ ГЕНЕРАЦІЇ ТЕКСТУ

Розділ присвячено аналізу предметної галузі, в межах якої виконується дослідження. Розглянуто базові поняття, пов'язані з обробкою природної мови та генерацією тексту, проаналізовано еволюцію підходів до автоматичної генерації текстів – від класичних статистичних методів до сучасних нейромережових архітектур. Окрему увагу приділено архітектурі Transformer та побудованим на її основі великим мовним моделям, які на сьогодні є основою більшості сучасних рішень у галузі генерації тексту.

1.1 Загальна характеристика задачі генерації тексту

Генерація тексту (Natural Language Generation, NLG) – це підгалузь обробки природної мови, що вивчає методи та засоби автоматичного створення зв'язних текстів природною мовою. На відміну від задачі розпізнавання мови (Natural Language Understanding, NLU), яка спрямована на отримання структурованого представлення з тексту, задача генерації тексту є зворотною: на основі заданого вхідного представлення (числового, символічного або текстового) необхідно побудувати послідовність слів, що відповідає правилам мови та задовольняє вимогам конкретної прикладної задачі.

Формально задачу генерації тексту можна сформулювати таким чином. Нехай задано вхідний контекст x , що може являти собою послідовність токенів, числовий вектор, набір атрибутів або будь-яку іншу структуру даних.

Потрібно побудувати функцію $f(x) = y$, де $y = (y_1, y_2, \dots, y_n)$ – послідовність токенів (слів або підслів), яка є коректним з точки зору граматики та семантики природної мови продовженням або відповіддю на вхідний контекст.

У статистичній постановці задача зводиться до оцінювання умовної ймовірності послідовності токенів $P(y | x)$ та вибору такої послідовності, що максимізує цю ймовірність. Зазвичай для цього використовують декомпозицію за ланцюговим правилом ймовірності: $P(y | x) = \prod P(y_t | y_1, \dots, y_{t-1}, x)$. Таким чином, задача генерації тексту зводиться до послідовного прогнозування наступного токена на основі попередніх токенів та вхідного контексту.

Якість згенерованого тексту оцінюють як автоматичними метриками (BLEU, ROUGE, METEOR, перплексія), так і експертними оцінками. Кожен з цих методів має свої переваги та обмеження: автоматичні метрики не завжди корелюють з людською оцінкою, а експертна оцінка є дорогою та потребує значного часу.

1.1.1 Класифікація систем генерації тексту

Існуючі системи генерації тексту можна класифікувати за різними ознаками. За типом вхідних даних виділяють системи: text-to-text (генерація тексту на основі іншого тексту – наприклад, переклад, реферування, перефразування); data-to-text (генерація тексту на основі структурованих даних – наприклад, опис таблиць або графіків); image-to-text (генерація опису зображень); speech-to-text та інші.

За підходом до побудови розрізняють шаблонні системи, статистичні моделі (на основі n-грам) та нейромережеві моделі. Шаблонні системи використовують заздалегідь визначені правила та шаблони, що дозволяє отримувати точні та контрольовані результати, але обмежує гнучкість і потребує значних зусиль на розробку. Статистичні моделі базуються на оцінюванні ймовірностей послідовностей слів у великих корпусах текстів, проте мають обмежену здатність моделювати довгі залежності.

Сучасні нейромережеві моделі, які становлять основу більшості актуальних рішень, поділяють на авторегресійні (наприклад, GPT), де токени генеруються послідовно зліва направо; енкодер-декодерні (T5, BART), які спочатку кодують вхідну послідовність у внутрішнє представлення, а потім декодують з нього вихідну; та маскові (BERT), які навчаються відновлювати замасковані слова в реченні. Для задачі вільної генерації тексту найчастіше застосовують авторегресійні моделі, які демонструють найкращі результати.

За способом адаптації до конкретної задачі виділяють: моделі, навчені з нуля (from scratch) на цільовому корпусі; моделі з повним донавчанням (fine-tuning), коли всі параметри попередньо навченої моделі підлаштовуються під цільову задачу; параметрично ефективні методи донавчання (LoRA, adapters, prefix tuning), що оновлюють лише невелику кількість параметрів; а також підходи на основі промптингу (zero-shot, few-shot prompting), де модель використовується без додаткового навчання.

1.2 Еволюція підходів до автоматичної генерації тексту

Розвиток методів автоматичної генерації тексту охоплює період від середини XX століття до наших днів і відображає поступовий перехід від детермінованих правил до статистичних та нейромережевих моделей. Розуміння цієї еволюції є важливим для обґрунтування вибору сучасних архітектур та усвідомлення обмежень, що були притаманні попереднім підходам. У межах цього підрозділу розглянуто основні етапи розвитку методів генерації тексту, проаналізовано переваги та недоліки кожного з підходів, а також визначено передумови виникнення сучасних нейромережевих моделей. Такий історичний екскурс дозволяє не лише простежити логіку розвитку технологій обробки природної мови, а й глибше зрозуміти принципи роботи актуальних генеративних систем. Отримані

висновки стануть теоретичним підґрунтям для обґрунтування архітектурних рішень, що розробляються в межах цієї роботи.

1.2.1 Шаблонні та статистичні методи генерації

Перші системи автоматичної генерації тексту, що з'явилися у другій половині XX століття, базувалися на шаблонному підході. Такі системи використовували заздалегідь визначений набір синтаксичних шаблонів зі змінними слотами, які заповнювалися конкретними значеннями з бази даних або вхідних параметрів. Прикладом є системи генерації прогнозів погоди, біржових звітів та технічної документації, де структура тексту є передбачуваною й слабо варіюється. Перевагами шаблонного підходу є висока контрольованість виходу, відсутність граматичних помилок та можливість гарантувати наявність ключової інформації. Основними недоліками є обмежена гнучкість, монотонність згенерованих текстів та значні витрати на ручне створення й супровід шаблонів для нових предметних галузей.

Наступним важливим етапом стали статистичні мовні моделі, серед яких найпоширенішими є n -грамні моделі. У них ймовірність появи токена розраховується на основі $n-1$ попередніх токенів: $P(y_t | y_1, \dots, y_{t-1}) \approx P(y_t | y_{t-n+1}, \dots, y_{t-1})$. Значення ймовірностей оцінюються методом максимальної правдоподібності за частотою появи відповідних n -грам у тренувальному корпусі. Для подолання проблеми нульових ймовірностей застосовуються методи згладжування (Лапласа, Кнезера–Нея, Гуда–Тюрінга). Перевагою n -грамних моделей є простота реалізації, висока швидкість роботи та обчислювальна ефективність. Проте такі моделі мають принципові обмеження: вони не здатні моделювати залежності, що виходять за межі контекстного вікна, потребують експоненціально зростаючої кількості даних зі збільшенням n , а також не враховують семантичну подібність слів.

Часткове розв'язання проблеми семантичного представлення слів забезпечили дистрибутивні моделі, зокрема word2vec, GloVe та fastText [2], які представляють слова у вигляді щільних векторів у багатовимірному просторі. Семантично близькі слова в такому просторі розміщуються поряд, що дозволяє моделі узагальнювати інформацію між семантично подібними контекстами. Незважаючи на це, статичні векторні представлення мають суттєве обмеження: кожному слову відповідає єдиний вектор незалежно від контексту, що не дозволяє коректно обробляти багатозначні слова та враховувати контекстну залежність значень. Подолання цих обмежень стало однією з основних причин переходу до нейромережових архітектур, здатних формувати динамічні контекстуалізовані представлення.

1.2.2 Рекурентні нейронні мережі та їх модифікації

Із розвитком апаратних засобів та появою достатньо великих обчислювальних потужностей у галузі обробки природної мови набули поширення рекурентні нейронні мережі (Recurrent Neural Network, RNN). На відміну від моделей з фіксованим контекстним вікном, RNN здатні обробляти послідовності довільної довжини, передаючи інформацію між кроками часу через прихований стан. На кожному кроці t прихований стан h_t обчислюється як функція від попереднього стану h_{t-1} та поточного входу x_t : $h_t = f(W_h h_{t-1} + W_x x_t + b)$, де f – нелінійна функція активації, а W_h , W_x , b – параметри мережі, що навчаються.

Класичні RNN мають суттєві обмеження, основним з яких є проблема зникаючого та вибухового градієнта під час навчання за алгоритмом зворотного поширення помилки в часі (Backpropagation Through Time, BPTT). Через мультиплікативну природу градієнтів значення похідних можуть експоненціально зменшуватися або зростати з кожним кроком назад у часі. Це не дозволяє ефективно навчати модель моделювати довгострокові

залежності між токенами, що знаходяться далеко один від одного у послідовності.

Розв'язанням цієї проблеми стала архітектура довгої короткочасної пам'яті (Long Short-Term Memory, LSTM), запропонована Хохрайтером і Шмідгубером у 1997 році [3]. У LSTM поряд з прихованим станом вводиться окремий стан клітинки (cell state), а потоки інформації контролюються трьома параметризованими гейтами: гейтом забування (forget gate), вхідним гейтом (input gate) та вихідним гейтом (output gate). Кожен гейт є шаром із сигмоїдною активацією, що приймає значення від 0 до 1 і визначає, яка частка інформації повинна пройти крізь нього. Така конструкція дозволяє моделі зберігати важливу інформацію протягом значної кількості кроків та селективно оновлювати її.

Спрощеним варіантом LSTM є архітектура GRU (Gated Recurrent Unit), запропонована Чо та співавторами у 2014 році [4]. GRU об'єднує гейти забування і входу в єдиний гейт оновлення (update gate) та використовує лише прихований стан без окремого стану клітинки. Завдяки меншій кількості параметрів GRU навчається швидше за LSTM, демонструючи при цьому порівнянну якість на широкому колі задач. На основі LSTM та GRU було побудовано архітектуру кодер-декодер (sequence-to-sequence), яка стала стандартним рішенням для задач машинного перекладу, реферування та діалогових систем у період до 2017 року.

Незважаючи на значне покращення порівняно з класичними RNN, рекурентні архітектури зберігають принципові обмеження. По-перше, послідовна природа обчислень не дозволяє ефективно розпаралелювати навчання, що особливо критично при роботі з великими корпусами текстів. По-друге, навіть LSTM і GRU мають обмежену здатність моделювати дуже довгі залежності, оскільки інформація поступово «розмивається» через ущільнення в фіксований за розмірністю прихований стан. Ці обмеження

стали безпосередньою передумовою появи архітектури Transformer, що базується на принципово іншому механізмі обробки послідовностей.

1.3 Архітектура Transformer та механізм уваги

Архітектура Transformer, запропонована Васвані та співавторами у 2017 році в роботі «Attention Is All You Need» [5], стала переломним моментом у розвитку обробки природної мови. Її принципова відмінність полягає у повній відмові від рекурентних та згорткових шарів при обробці послідовностей: усі залежності між токенами моделюються виключно за допомогою механізму уваги (attention). Це дозволяє паралелізувати обчислення, ефективно моделювати залежності будь-якої довжини та масштабувати модель до значно більших обсягів параметрів і даних, ніж дозволяли попередні архітектури.

1.3.1 Механізм Self-Attention та Multi-Head Attention

В основі архітектури Transformer лежить механізм самоуваги (self-attention), який дозволяє кожному токенові послідовності формувати своє представлення з урахуванням усіх інших tokenів. Для кожного токена обчислюються три вектори: запит (Query, Q), ключ (Key, K) та значення (Value, V), які отримуються шляхом множення вхідного представлення на відповідні матриці параметрів. Вага уваги для пари tokenів i та j обчислюється як скалярний добуток Q_i та K_j , нормалізований квадратним коренем з розмірності ключа d_k , з подальшим застосуванням функції softmax: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{d_k})V$. Поділ на $\sqrt{d_k}$ забезпечує чисельну стійкість і запобігає насиченню softmax при великих розмірностях.

Для збільшення виразної здатності моделі замість єдиного механізму уваги застосовується багатоголова увага (multi-head attention).

Вхідні представлення проектується у h різних підпросторів за допомогою окремих наборів матриць проєкцій, у кожному з яких незалежно обчислюється самоувага. Отримані результати конкатенуються та проектується лінійним шаром у вихідний простір. Це дозволяє моделі одночасно фокусуватися на різних аспектах контексту: синтаксичних, семантичних, кореферентних та інших залежностях. Емпірично показано, що різні голови уваги спеціалізуються на різних типах лінгвістичних явищ, що підвищує здатність моделі узагальнювати.

Оскільки механізм самоуваги є інваріантним щодо порядку токенів, для збереження інформації про послідовність до вхідних векторів додаються позиційні кодування (positional encoding). У базовій реалізації Transformer використовуються синусоїдальні позиційні кодування фіксованої форми, проте у пізніших моделях запропоновано низку альтернатив: навчані позиційні ембедінги, відносні позиційні кодування (relative positional encoding) та ротаційні позиційні кодування (RoPE) [6], що демонструють кращі властивості узагальнення на послідовностях більшої довжини, ніж ті, на яких модель була навчена.

1.3.2 Структура енкодера та декодера

У повній архітектурі Transformer виділяють дві основні частини – енкодер та декодер, кожна з яких складається з N однакових блоків. Енкодерний блок містить шар багатоголової самоуваги та повнозв'язну мережу прямого поширення (feed-forward network), застосовану до кожної позиції незалежно. Декодерний блок додатково включає шар маскованої самоуваги, який під час навчання запобігає «підгляданню» майбутніх токенів, та шар крос-уваги, що звертається до виходу енкодера. Між підшарами застосовуються залишкові з'єднання (residual connections) та шарова нормалізація (layer normalization), які забезпечують стабільність навчання глибоких моделей.

Залежно від цільової задачі використовують різні конфігурації Transformer. Моделі, що містять лише енкодерну частину (encoder-only), такі як BERT [7], орієнтовані на задачі розуміння тексту: класифікацію, виділення іменованих сутностей, відповіді на запитання за наданим контекстом. Моделі, що містять лише декодерну частину (decoder-only), є авторегресійними і застосовуються для задач відкритої генерації тексту – саме до таких належать моделі сімейства GPT [8], [9]. Енкодер-декодерні моделі (encoder-decoder), такі як T5 [10] та BART, використовуються переважно для задач, що передбачають перетворення однієї послідовності на іншу – машинного перекладу, реферування, перефразування. Для задачі генерації зв'язного тексту у вибраній предметній галузі, що є метою кваліфікаційної роботи, найбільш доцільним є використання декодерної архітектури.

1.4 Сучасні великі мовні моделі та методи їх адаптації

Поява архітектури Transformer та доступність значних обчислювальних потужностей спричинили стрімкий розвиток великих мовних моделей (Large Language Models, LLM) – нейронних мереж з кількістю параметрів від сотень мільйонів до сотень мільярдів [11], [12], [13], попередньо навчених на корпусах текстів обсягом у сотні гігабайтів. Ключовою особливістю таких моделей є парадигма попереднього навчання та подальшого донавчання (pretraining and fine-tuning), що дозволяє ефективно переносити знання між задачами і доменами.

1.4.1 Сімейство моделей GPT

Сімейство Generative Pre-trained Transformer (GPT), розроблене компанією OpenAI [14], є одним з найвпливовіших напрямів розвитку

авторегресійних мовних моделей. Модель GPT-2 [8] обсягом 1,5 мільярда параметрів продемонструвала, що достатньо велика модель, навчена на різноманітному веб-корпусі, здатна виконувати широкий спектр задач без спеціалізованого донавчання. Розвиток цього підходу втілюється у моделі GPT-3 [9] з 175 мільярдами параметрів, для якої було емпірично підтверджено властивість «навчання у контексті» (in-context learning): здатність розв'язувати нові задачі на основі кількох прикладів, наданих безпосередньо у вхідному запиті, без оновлення параметрів моделі.

Архітектурно моделі сімейства GPT є декодерними Transformer-ами, в яких застосовується маскована самоувага: при обробці позиції t модель може звертатися лише до токенів з позиціями $1, \dots, t$. Навчання здійснюється у режимі прогнозування наступного токена (causal language modeling) на великому неанотованому корпусі. Цільова функція має вигляд максимізації лог-правдоподібності $L(\theta) = \sum \log P(y_t | y_1, \dots, y_{t-1}; \theta)$, де θ – параметри моделі. Завдяки цій формі тренування модель опановує не лише граматичні та лексичні закономірності мови, а й значний обсяг фактичних знань і «здорового глузду», неявно представлених у тренувальних даних.

Окрім моделей сімейства GPT, у сучасній практиці широко використовуються відкриті мовні моделі: LLaMA, Mistral, Falcon, Gemma, Qwen [15], [16] та інші. Вони доступні для завантаження разом з ваговими коефіцієнтами і можуть бути донавчені на власному корпусі без передавання даних зовнішнім сервісам. Для задач, що вирішуються у межах кваліфікаційної роботи, особливо привабливими є відносно компактні відкриті моделі обсягом від 1 до 8 мільярдів параметрів, які забезпечують прийнятну якість генерації при можливості розгортання на одному графічному процесорі. Таким чином, вибір на користь компактних відкритих архітектур є найбільш обґрунтованим для реалізації практичної частини дослідження з огляду на встановлені апаратні обмеження.

1.4.2 Методи донавчання та адаптації мовних моделей

Адаптація попередньо навченої мовної моделі до конкретної предметної галузі або стилю є ключовим етапом побудови прикладної системи генерації тексту. Класичне повне донавчання (full fine-tuning) передбачає оновлення всіх параметрів моделі на цільовому корпусі. Цей підхід забезпечує найкращу якість, проте вимагає значного обсягу пам'яті GPU (зокрема, для зберігання станів оптимізатора Adam потрібно у кілька разів більше пам'яті, ніж займають самі ваги), а також створює окрему повну копію моделі для кожної цільової задачі.

Альтернативою є параметрично ефективні методи донавчання (Parameter-Efficient Fine-Tuning, PEFT), які оновлюють лише невелику частину параметрів моделі або додають невеликі додаткові модулі. Найпоширенішим серед них є метод LoRA (Low-Rank Adaptation), запропонований Ху та співавторами у 2022 році [17]. Ідея LoRA полягає у представленні оновлення вагової матриці ΔW у вигляді добутку двох матриць низького рангу: $\Delta W = BA$, де B має розмірність $d \times r$, A – $r \times d$, а $r \ll d$. Під час навчання оновлюються лише ці нові матриці A та B , тоді як вихідні ваги моделі залишаються замороженими. Це зменшує кількість параметрів, що навчаються, на кілька порядків (часто з мільярдів до десятків мільйонів), за умови порівнянної якості з повним донавчанням.

До інших методів PEFT належать: adapters – вставка невеликих повнозв'язних [18] модулів між шарами Transformer; prefix tuning – додавання навчених префіксів до ключів і значень механізму уваги; prompt tuning – навчання м'яких векторних промптів, що додаються на вхід моделі. Окремою категорією є квантована адаптація QLoRA [19], що поєднує квантування ваг до 4-бітної точності з LoRA-адаптерами, що дозволяє донавчати моделі з мільярдами параметрів на споживчих графічних процесорах.

Поряд з методами донавчання важливу роль відіграють стратегії декодування, що визначають спосіб формування вихідної послідовності з розподілів ймовірностей, які видає модель. Жадібний пошук (greedy search) на кожному кроці обирає токен з максимальною ймовірністю та забезпечує детермінованість, але призводить до повторів та одноманітності. Променевий пошук (beam search) зберігає на кожному кроці k найкращих часткових послідовностей і застосовується переважно у задачах перекладу та реферування. Для відкритої генерації частіше використовують стохастичні методи: семпсування з температурою, top-k семпсування та nucleus sampling (top-p), що забезпечують [20] більшу різноманітність тексту при контрольованій якості.

1.4.3 Метрики оцінювання якості генерованих текстів

Оцінювання якості систем генерації тексту є нетривіальною задачею через відсутність єдиної правильної відповіді: для одного й того ж вхідного контексту може існувати велика кількість прийнятних варіантів виходу. У практиці поєднують внутрішні (intrinsic) та зовнішні (extrinsic) метрики, а також автоматичні та експертні методи оцінювання.

Найпоширенішою внутрішньою метрикою є перплексія (perplexity), що визначається як експонента від усередненої негативної лог-правдоподібності токенів на валідаційному наборі: $PP = \exp(-(1/N) \sum \log P(y_t | y_1, \dots, y_{t-1}))$. Інтуїтивно перплексія показує, серед скількох рівноймовірних варіантів модель «коливається» при виборі наступного токена. Менші значення відповідають кращій моделі. Перплексія є зручним інструментом для порівняння моделей, навчених на однакових словниках, проте не дозволяє безпосередньо оцінити якість окремого згенерованого тексту з точки зору користувача.

Для задач, де є еталонні відповіді, застосовують метрики на основі n-грамного співпадіння. Метрика BLEU [21] була запропонована для

оцінювання машинного перекладу і базується на точності збігу n-грам згенерованого тексту з еталонним з урахуванням штрафу за коротку довжину. Метрика ROUGE використовується переважно [22] для оцінювання реферування і базується на повноті покриття n-грам еталону згенерованим текстом. METEOR враховує не лише точні збіги, а й синонімію та стемінг. Спільним недоліком цих метрик є слабка кореляція з людською оцінкою для задач відкритої генерації, де однаково якісні тексти можуть мати мало спільних n-грам.

Сучасні підходи до оцінювання включають семантичні метрики, що базуються на нейромережевих представленнях: BERTScore використовує [23] контекстуалізовані ембедінги BERT для обчислення схожості між токенами згенерованого та еталонного текстів; BLEURT та COMET використовують моделі, спеціально навчені прогнозувати людські оцінки. У задачах відкритої генерації дедалі частіше застосовують підхід «LLM-as-a-judge», коли потужна мовна модель використовується як автоматичний оцінювач за заданими критеріями. При цьому остаточним мірилом якості залишається експертна оцінка людьми, яку проводять за критеріями зв'язності, релевантності, граматичної правильності та інформативності.

На підставі проведеного аналізу можна зробити висновок, що для побудови системи генерації тексту у межах кваліфікаційної роботи доцільним є використання попередньо навченої авторегресійної мовної моделі на основі архітектури Transformer з подальшим донавчанням параметрично ефективним методом LoRA на корпусі цільової предметної галузі. Для оцінювання якості системи доречно застосовувати комбінацію метрик: перплексію на валідаційній вибірці – як індикатор якості донавчання, BLEU та ROUGE – за наявності еталонних текстів, а також експертну оцінку згенерованих прикладів – як підсумкове підтвердження практичної придатності системи.

2 ПРОЄКТУВАННЯ СИСТЕМИ ГЕНЕРАЦІЇ ТЕКСТУ

У цьому розділі на основі результатів аналізу предметної галузі, виконаного в першому розділі, здійснено проєктування системи генерації україномовного тексту. Сформульовано постановку задачі, визначено функціональні та нефункціональні вимоги до системи, проведено порівняльний аналіз можливих архітектурних варіантів та обґрунтовано вибір моделі на основі архітектури Transformer декодерного типу. Описано набір даних, етапи його підготовки й токенизації, наведено схему процесу навчання та узагальнену архітектуру програмної системи з розбиттям на модулі, а також формалізовано алгоритм генерації тексту.

2.1 Постановка задачі

Метою розроблюваної системи є автоматична генерація зв'язного, граматично та семантично коректного україномовного тексту новинного стилю на основі заданого користувачем початкового фрагмента (запиту). Система повинна продовжувати наданий контекст у вигляді короткої новинної замітки або статті, зберігаючи тематичну узгодженість і стилістичну однорідність із цільовим корпусом.

Формально задачу можна подати як задачу авторегресійного моделювання мови. Нехай задано початкову послідовність токенів (запит) $x = (x_1, \dots, x_m)$. Потрібно згенерувати продовження $y = (y_1, \dots, y_n)$, для якого спільна умовна ймовірність, що декомпонується за ланцюговим правилом, є якомога вищою:

$$P(y | x) = \prod_t P(y_t | x, y_1, \dots, y_{t-1}), \quad (2.1)$$

де $P(y_t | \dots)$ – умовна ймовірність появи токена y_t за умови всіх попередніх токенів запиту й уже згенерованої частини послідовності;

θ – параметри мовної моделі, що обчислюють цей розподіл.

Оцінювання параметрів моделі здійснюється не з нуля, а шляхом донавчання попередньо навченої мовної моделі сімейства GPT, що дозволяє використати знання про мову, набуті на великому корпусі, і суттєво скоротити обсяг даних та обчислень, потрібних для адаптації до цільового домену. Вхідними даними системи є текстовий запит та набір параметрів генерації (довжина, температура, спосіб семплування), вихідними – згенерований текстовий фрагмент.

Об'єктом проєктування є програмна система, що складається з модуля підготовки даних, модуля навчання (донавчання) мовної моделі, модуля генерації та програмного інтерфейсу доступу. Предметом проєктування є архітектура цієї системи та сукупність проєктних рішень, що забезпечують виконання поставлених вимог.

2.2 Функціональні та нефункціональні вимоги

Функціональні вимоги визначають перелік дій, які система повинна виконувати: приймати від користувача початковий текстовий запит та параметри генерації; виконувати попередню обробку й токенізацію вхідного тексту відповідно до словника моделі; генерувати продовження тексту заданої довжини із застосуванням обраної стратегії декодування; підтримувати щонайменше три стратегії декодування – жадібний пошук, семплування з температурою та nucleus sampling (top-p); забезпечувати донавчання базової моделі на користувацькому корпусі методом LoRA; надавати доступ до функції генерації через програмний інтерфейс (REST API) та графічний веб-інтерфейс; обчислювати метрики якості (перплексію) на валідаційному наборі під час навчання.

Нефункціональні вимоги визначають загальні властивості системи та обмеження на її реалізацію. Щодо продуктивності час генерації фрагмента довжиною до 200 токенів не повинен перевищувати кількох секунд. Щодо

ресурсомісткості система має запускатися на одному графічному процесорі з обсягом відеопам'яті від 8 ГБ. Модульність передбачає, що компоненти підготовки даних, навчання та генерації мають бути відокремлені. Розширюваність означає, що система має допускати заміну базової моделі без зміни загальної архітектури. Відтворюваність забезпечується фіксацією версій бібліотек і параметрів навчання, що робить усі експерименти відтворюваними. Кросплатформність передбачає, що програмне забезпечення має працювати в середовищі Linux.

Перелік основних вимог зведено в таблиці 2.1, де для кожної вимоги вказано її тип і пріоритет, що використовується надалі для оцінювання повноти реалізації.

Таблиця 2.1 – Зведені вимоги до системи генерації тексту

Код	Зміст вимоги	Тип	Пріоритет
F1	Приймання запиту та параметрів генерації	Функц.	Високий
F2	Токенізація та попередня обробка тексту	Функц.	Високий
F3	Генерація продовження тексту	Функц.	Високий
F4	Підтримка стратегій декодування	Функц.	Середній
F5	Донавчання моделі методом LoRA	Функц.	Високий
F6	Доступ через REST API та веб-інтерфейс	Функц.	Середній
N1	Час генерації до кількох секунд	Нефункц.	Середній
N2	Робота на одному GPU ≥ 8 ГБ	Нефункц.	Високий
N3	Модульність та розширюваність	Нефункц.	Високий
N4	Відтворюваність експериментів	Нефункц.	Середній

Сформульовані вимоги є основою для подальшого вибору архітектури та формування проєктних рішень, описаних у наступних підрозділах. Правильно визначені пріоритети дозволять ефективно розподілити ресурси під час проєктування та розробки окремих модулів системи. Узгодженість архітектурних рішень із нефункціональними вимогами гарантуватиме

стабільну роботу програмного продукту в цільовому операційному середовищі.

2.3 Аналіз варіантів архітектури та вибір моделі

Як показано в першому розділі, для задачі вільної генерації зв'язного тексту найкращі результати забезпечують авторегресійні моделі на основі архітектури Transformer. Проте навіть у межах цього класу існує кілька принципово різних шляхів побудови системи, які доцільно порівняти перед остаточним вибором.

2.3.1 Порівняння можливих підходів

Розглянуто чотири варіанти реалізації ядра системи. Перший – навчання власної мовної моделі з нуля на україномовному корпусі. Цей підхід забезпечує повний контроль над архітектурою та словником, проте потребує корпусу обсягом у десятки гігабайтів та значних обчислювальних ресурсів, що виходить за межі можливостей бакалаврської роботи. Другий варіант – повне донавчання (full fine-tuning) попередньо навченої моделі, що дає високу якість, але вимагає великого обсягу відеопам'яті для зберігання станів оптимізатора та окремої копії всіх ваг. Третій варіант – параметрично ефективне донавчання методом LoRA, за якого оновлюється лише невелика частка параметрів. Четвертий – використання моделі без донавчання, лише через few-shot промптинг, що не потребує навчання, але не забезпечує стабільної адаптації до стилю й домену та погано працює для компактних моделей.

Порівняння цих підходів за ключовими критеріями наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння підходів до побудови ядра системи

Підхід	Дані	Ресурси GPU	Якість у домені	Складність
Навчання з нуля	Дуже багато	Дуже високі	Висока*	Дуже висока
Повне донавчання	Помірно	Високі	Висока	Середня
LoRA-донавчання	Помірно	Низькі	Висока	Низька
Few-shot промптинг	Не потрібні	Низькі	Нестабільна	Дуже низька

Примітка: висока якість для навчання з нуля досяжна лише за наявності великого корпусу та обчислювальних ресурсів.

За сукупністю критеріїв – помірних вимог до даних та обчислень, високої якості в цільовому домені й низької складності реалізації – оптимальним обрано підхід донавчання попередньо навченої моделі методом LoRA. Цей вибір повністю узгоджується з висновками першого розділу.

2.4 Обґрунтування вибору архітектури Transformer та GPT-подібної моделі

Вибір архітектури Transformer декодерного типу зумовлений сукупністю її властивостей, що безпосередньо відповідають вимогам задачі. По-перше, механізм самоуваги дозволяє моделювати залежності між будь-якими токенами послідовності незалежно від відстані між ними, що критично для збереження тематичної узгодженості тексту. По-друге, на відміну від рекурентних мереж, обчислення в Transformer розпаралелюються, що пришвидшує як навчання, так і інференс. По-третє, декодерна (causal) архітектура з маскованою самоувагою природно відповідає авторегресійній постановці задачі генерації, у якій кожен токен прогнозується на основі лише попередніх.

Серед доступних GPT-подібних моделей для реалізації обрано модель GPT-Neo, розроблену організацією EleutherAI [24] як відкритий аналог GPT-3. На відміну від моделей, доступних лише через комерційний API, GPT-Neo поширюється разом із ваговими коефіцієнтами під вільною ліцензією, що дозволяє донавчати її на власному корпусі локально, без передавання даних зовнішнім сервісам. Базовою для розробки обрано конфігурацію GPT-Neo 125M (близько 125 мільйонів параметрів) як таку, що забезпечує прийнятну якість генерації за можливості розгортання на одному графічному процесорі споживчого сегмента; у розширеному експерименті додатково використано конфігурацію GPT-Neo 1.3B для оцінювання впливу розміру моделі на якість генерації.

Окремо слід зважити на особливість, суттєву для україномовного домену. Модель GPT-Neo попередньо навчена переважно на англomовному корпусі The Pile [25], а її токенізатор оптимізований під англійську мову. Унаслідок цього україномовний текст розбивається на надмірно велику кількість токенів (кирилиця часто кодується побайтово), що знижує ефективність використання контекстного вікна та якість генерації. Для подолання цього обмеження в роботі передбачено адаптацію токенізатора – розширення словника україномовними підсловами та відповідне розширення матриці ембедінгів моделі, що детальніше описано в підрозділі 2.6. Таким чином, вибір моделі супроводжується чітко окресленою науково-технічною проблемою, розв’язання якої становить частину практичної новизни роботи.

2.5 Опис набору даних

Для донавчання моделі сформовано корпус україномовних новинних текстів. Як основа використано відкриті україномовні текстові ресурси, зокрема корпус UberText 2.0 [26] (підкорпус новин) та загальнодоступні набори новинних статей, поширювані для дослідницьких цілей. Тексти

відібрано за стильовою однорідністю: перевагу надано інформаційним повідомленням і коротким статтям нейтрального стилю.

Загальні характеристики сформованого набору даних наведено в таблиці 2.3. Корпус поділено на навчальну, валідаційну та тестову підвибірki у співвідношенні 80/10/10 без перетину документів між ними, що забезпечує коректність оцінювання узагальнювальної здатності моделі.

Таблиця 2.3 – Характеристики набору даних

Характеристика	Значення
Кількість документів	50 000
Загальний обсяг тексту	120 МБ
Середня довжина документа	350 слів
Навчальна вибірка	80 %
Валідаційна вибірка	10 %
Тестова вибірка	10 %

Наведені кількісні показники відповідають сформованому й очищеному корпусу, використаному для донавчання та оцінювання моделі. Обсяг корпусу обрано як компроміс між достатністю для адаптації стилю й домену та обмеженнями обчислювальних ресурсів.

2.6 Підготовка даних і токенізація

Сирі текстові дані з відкритих джерел містять шум, який погіршує якість навчання, тому перед використанням корпус проходить багатоетапну попередню обробку. Ефективність функціонування генеративних моделей безпосередньо залежить від чистоти та структурованості навчального корпусу. Побудова оптимізованого конвеєра підготовки дозволяє не лише уніфікувати тексти й усунути інформаційну надмірність, а й суттєво знижує ризик перенавчання моделі на нерелевантних патернах. Крім того, стандартизація вхідних послідовностей є критично важливою для

коректного налаштування словника на етапі кодування. Послідовність етапів підготовки даних подано на рисунку 2.1.



Рисунок 2.1 – Конвеєр попередньої обробки та підготовки даних

На етапі очищення видаляються службові елементи розмітки, нетекстові символи, повторювані пробіли та надмірні переноси рядків. Нормалізація приводить текст до єдиного представлення Unicode (форма NFC) та уніфікує лапки й апострофи. Дедуплікація усуває повні та майже повні дублікати документів, що часто трапляються в новинних агрегаторах. Фільтрація мови за допомогою автоматичного визначника відсіює документи, які не є україномовними або містять значну частку іншомовного тексту.

Ключовим етапом підготовки є токенизація – перетворення тексту на послідовність числових ідентифікаторів зі словника моделі. GPT-Neo використовує токенизатор на основі алгоритму Byte-Pair

Encoding (BPE) [27], що розбиває текст на частотні підслова. Через орієнтованість базового словника на англійську мову україномовні слова розбиваються на надмірну кількість токенів. Для оцінювання ефективності токенизації використовують показник середньої кількості токенів на слово (fertility):

$$Fertility = N_токенів / N_слів, \quad (2.2)$$

де $N_токенів$ – загальна кількість токенів після токенизації корпусу;

$N_слів$ – загальна кількість слів у корпусі.

Для базового токенизатора цей показник на україномовному тексті є суттєво вищим за одиницю, що знижує ефективність моделі. Тому в роботі застосовано адаптацію токенизатора: на цільовому корпусі донавчається BPE-словник [28], нові україномовні підслова додаються до наявного словника, а матриця ембедінгів моделі розширюється відповідними рядками, ініціалізованими усередненими значеннями. Це дозволяє зменшити середню кількість токенів на слово й підвищити якість генерації. Підготовлений корпус зберігається у форматі індексованих послідовностей токенів, згрупованих у блоки фіксованої довжини, придатні для подавання в модель під час навчання.

2.7 Процес навчання

Навчання системи полягає в донавчанні попередньо навченої моделі GPT-Neo на підготовленому україномовному корпусі методом LoRA. Базові ваги моделі заморожуються, а навчаються лише низькорангові адаптери, вбудовані у вагові матриці проєкцій механізму уваги. Цільовою функцією навчання є усереднена негативна лог-правдоподібність наступного токена (крос-ентропія) на навчальній вибірці:

$$L(\theta) = -(1/N) \sum_t \log P(y_t | y_1, \dots, y_{t-1}; \theta), \quad (2.3)$$

де N – кількість токенів у пакеті (batch);

$P(y_t | \dots; \theta)$ – ймовірність правильного наступного токена, обчислена моделлю з параметрами θ .

Оптимізація параметрів виконується методом стохастичного градієнтного спуску з адаптивним темпом навчання (оптимізатор AdamW [29]). Темп навчання змінюється за розкладом із розігрівом (warm-up) та подальшим косинусним спаданням, що сприяє стабільності навчання. Для запобігання перенавчанню застосовуються рання зупинка за валідаційною перплексією та накопичення градієнтів, що дозволяє імітувати більший розмір пакета за обмеженого обсягу відеопам'яті. Якість моделі на кожній епосі контролюється обчисленням перплексії на валідаційній вибірці; модель із найкращим значенням зберігається як підсумкова.

Узагальнену схему процесу навчання подано на рисунку 2.2.

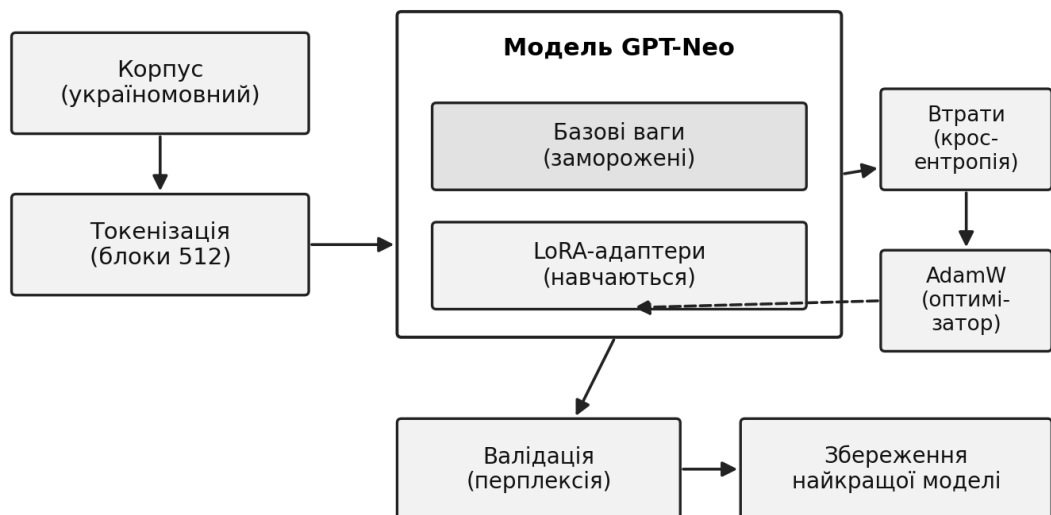


Рисунок 2.2 – Схема процесу донавчання моделі методом LoRA

2.8 Архітектура програмної системи

Програмна система побудована за модульним принципом, що відповідає нефункціональним вимогам щодо модульності й розширюваності. Виділено п'ять основних модулів, кожен з яких має чітко визначену зону відповідальності та взаємодіє з іншими через узгоджені інтерфейси. Узагальнену архітектуру системи подано на рисунку 2.3.



Рисунок 2.3 – Узагальнена архітектура програмної системи

Призначення модулів описано далі. Модуль підготовки даних (data) відповідає за збір, очищення, нормалізацію та токенізацію корпусу, а також за поділ на вибірки. Модуль навчання (training) реалізує цикл донавчання моделі методом LoRA, валідацію та збереження контрольних точок. Сховище моделі (model store) зберігає базові ваги, адаптери LoRA та конфігурації. Модуль генерації (generation) завантажує модель з адаптерами й виконує генерацію тексту за заданими параметрами та стратегією декодування. Модуль інтерфейсу (interface) надає доступ до функції

генерації через REST API на основі FastAPI та графічний веб-інтерфейс на основі Gradio.

Структуру модулів та їх основні функції зведено в таблиці 2.4.

Таблиця 2.4 – Структура модулів системи

Модуль	Основні функції	Залежності
data	Очищення, токенізація, поділ корпусу	—
training	Донавчання LoRA, валідація	data, model store
model store	Зберігання ваг та адаптерів	—
generation	Завантаження моделі, генерація	model store
interface	REST API, веб-інтерфейс	generation

Послідовність взаємодії компонентів у типовому сценарії генерації відображено на діаграмі послідовності, поданій на рисунку 2.4.

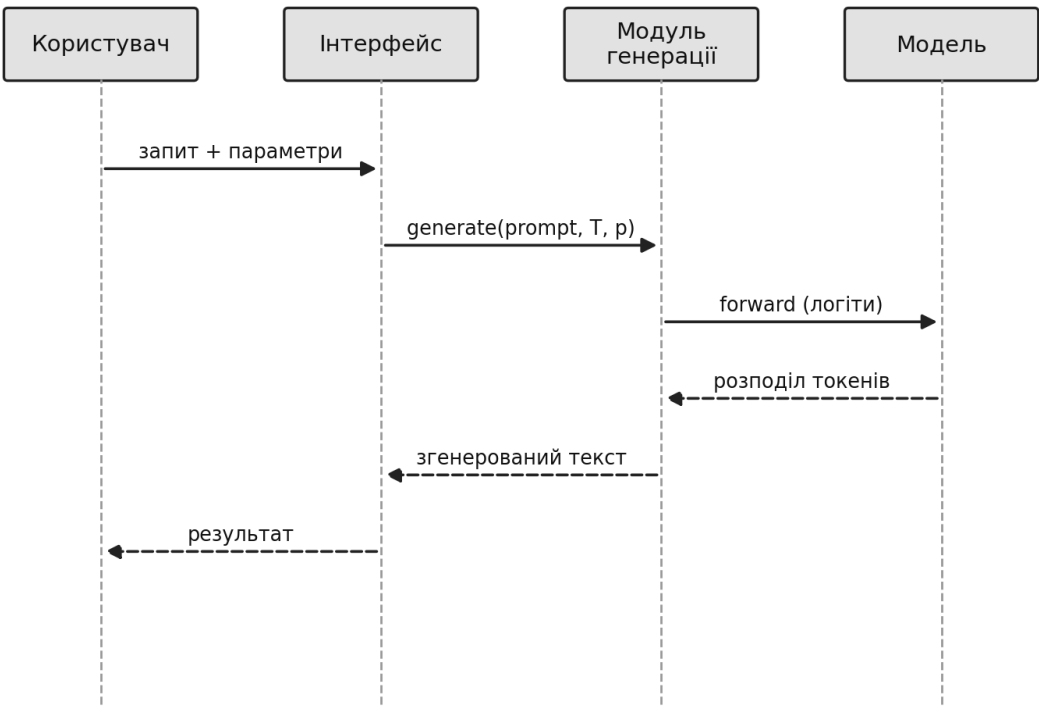


Рисунок 2.4 – Діаграма послідовності сценарію генерації тексту

2.9 Алгоритм генерації тексту

Генерація тексту виконується авторегресійно: модель послідовно прогнозує по одному токenu, додаючи кожен згенерований токен до контексту для прогнозування наступного. Стратегія вибору токена з розподілу ймовірностей визначається параметрами генерації. У системі реалізовано nucleus sampling (top-p) як основну стратегію, що забезпечує баланс між різноманітністю та зв'язністю тексту. Перед застосуванням семплювання розподіл масштабується параметром температури T : за значень $T < 1$ розподіл загострюється, за $T > 1$ – згладжується.

Узагальнений алгоритм генерації подано нижче. Спочатку вхідний запит токенізують і формують початковий контекст. Далі контекст подають у модель та отримують розподіл ймовірностей наступного токена, після чого масштабують логіти за температурою T і відбирають мінімальну множину токенів, сумарна ймовірність яких не менша за p . Із цієї множини випадково обирають наступний токен, додають його до контексту й повторюють описані кроки, доки не досягнуто заданої довжини або не згенеровано спеціальний токен завершення. На завершення отриману послідовність детокенізують у підсумковий текст.

Формально на кожному кроці t токен обирається з усіченого та перенормованого розподілу:

$$y_t \sim P'(y_t \mid x, y_1, \dots, y_{t-1}), \quad (2.4)$$

де P' – розподіл, отриманий після масштабування температурою та відсікання за порогом p (nucleus sampling).

Описаний алгоритм є основою модуля генерації й безпосередньо реалізується в третьому розділі.

3 РЕАЛІЗАЦІЯ ТА ПРОГРАМНА РОЗРОБКА

У цьому розділі описано програмну реалізацію системи генерації українського тексту, спроектованої в другому розділі. Наведено характеристику середовища розробки та використаних бібліотек, описано структуру проєкту й реалізацію основних модулів із наведенням фрагментів коду. Розглянуто процеси донавчання моделі та генерації тексту, а також реалізацію програмного інтерфейсу REST API й веб-інтерфейсу та особливості розгортання системи.

3.1 Середовище розробки та використані технології

Систему реалізовано мовою програмування Python версії 3.10, яка є стандартом де-факто в галузі машинного навчання завдяки розвиненій екосистемі бібліотек та зручності прототипування. Основними програмними засобами розробки є такі бібліотеки.

PyTorch – фреймворк глибинного навчання [30], що надає засоби для роботи з тензорами, автоматичного диференціювання та виконання обчислень на графічному процесорі. PyTorch використано як низькорівневу основу для навчання й інференсу моделі.

Hugging Face Transformers – бібліотека [31, 32], що надає реалізації попередньо навчених моделей, зокрема GPT-Neo, разом із токенізаторами та уніфікованим інтерфейсом для навчання й генерації. Бібліотека використана для завантаження базової моделі, токенізації та виконання генерації.

Допоміжно використано бібліотеки: PEFT – для реалізації донавчання методом LoRA [33]; Datasets – для зберігання та потокової обробки корпусу; evaluate, sacrebleu [34] та rouge-score – для обчислення метрик якості; FastAPI та Uvicorn – для реалізації REST API [35]; Gradio – для побудови

веб-інтерфейсу [36]. Перелік основних залежностей із зазначенням версій наведено в таблиці 3.1.

Таблиця 3.1 – Основні програмні залежності

Бібліотека	Версія	Призначення
Python	3.10	Мова програмування
PyTorch	2.x	Глибинне навчання, GPU-обчислення
Transformers	4.x	Модель GPT-Neo, токенизація, генерація
PEFT	0.x	Донавчання методом LoRA
Datasets	2.x	Обробка корпусу
FastAPI	0.x	REST API
Gradio	4.x	Веб-інтерфейс

3.2 Структура проекту

Проект організовано відповідно до спроектованої модульної архітектури. Кожному логічному модулю відповідає окремий каталог або файл, що спрощує супровід та повторне використання коду. Загальну структуру каталогів проекту подано в лістингу 3.1.

Лістинг 3.1 – Структура каталогів проекту

```

text_generator/
  data/
    prepare.py          # очищення та підготовка корпусу
    tokenize_data.py    # токенизація, поділ на вибірки
  training/
    train.py            # донавчання моделі методом LoRA
    config.py           # гіперпараметри навчання
  generation/
    generator.py        # клас-генератор тексту
  interface/
    api.py              # REST API (FastAPI)
    web.py              # веб-інтерфейс (Gradio)

```

Продовження лістингу 3.1

```
models/                # збережені адаптери та конфігурації
requirements.txt
```

Такий поділ забезпечує слабку зв'язаність компонентів: зміна реалізації одного модуля не потребує модифікації інших за умови збереження інтерфейсів взаємодії.

3.3 Реалізація модуля підготовки даних

Модуль підготовки даних реалізує конвеєр очищення та токенизації, описаний у підрозділі 2.6. Функція очищення тексту виконує нормалізацію Unicode, видалення службових символів і зайвих пробілів. Фрагмент реалізації наведено в лістингу 3.2.

Лістинг 3.2 – Очищення та нормалізація тексту

```
import re, unicodedata

def clean_text(text: str) -> str:
    text = unicodedata.normalize('NFC', text)
    text = re.sub(r'<[^>]+>', ' ', text)      # розмітка
    text = re.sub(r'\s+', ' ', text)           # зайві пробіли
    text = text.replace('\u00ad', '')          # м'який перенос
    return text.strip()
```

Після очищення корпус токенизується адаптованим токенизатором та групується у блоки фіксованої довжини. Реалізацію адаптації токенизатора й токенизації наведено в лістингу 3.3.

Лістинг 3.3 – Адаптація токенизатора та токенизація корпусу

```
from transformers import AutoTokenizer
```

Продовження лістингу 3.3

```
base = 'EleutherAI/gpt-neo-125M'
tokenizer = AutoTokenizer.from_pretrained(base)
tokenizer.pad_token = tokenizer.eos_token

# донавчання ВРЕ на українському корпусі та додавання
# нових підслів до словника (розширення матриці ембедінгів
# виконується на етапі завантаження моделі через resize)
new_tok = tokenizer.train_new_from_iterator(corpus_iter,
                                           vocab_size=20000)
added = tokenizer.add_tokens(list(new_tok.get_vocab().keys()))

def tokenize(batch):
    return tokenizer(batch['text'], truncation=True,
                    max_length=512)
```

3.4 Реалізація процесу навчання

Модуль навчання реалізує донавчання моделі GPT-Neo методом LoRA. Спершу завантажується базова модель, після чого до неї застосовується конфігурація LoRA, яка вбудовує низькорангові адаптери у вагові матриці механізму уваги. Завантаження моделі та налаштування LoRA наведено в лістингу 3.4.

Лістинг 3.4 – Завантаження моделі та конфігурація LoRA

```
import torch
from transformers import AutoModelForCausalLM
from peft import LoraConfig, get_peft_model

model = AutoModelForCausalLM.from_pretrained(
    'EleutherAI/gpt-neo-125M', torch_dtype=torch.float16)
model.resize_token_embeddings(len(tokenizer))

lora_cfg = LoraConfig(
```

Продовження лістингу 3.4

```
r=16, lora_alpha=32, lora_dropout=0.05,
    target_modules=['q_proj', 'k_proj', 'v_proj', 'out_proj'],
    task_type='CAUSAL_LM')
model = get_peft_model(model, lora_cfg)
model.print_trainable_parameters()
```

Цикл навчання реалізовано за допомогою класу `Trainer` бібліотеки `Transformers`, що інкапсулює оптимізацію, валідацію та збереження контрольних точок. Налаштування навчання та запуск наведено в лістингу 3.5.

Лістинг 3.5 – Налаштування та запуск навчання

```
from transformers import (Trainer, TrainingArguments,
                          DataCollatorForLanguageModeling)

args = TrainingArguments(
    output_dir='models/gpt-neo-uk',
    per_device_train_batch_size=4,
    gradient_accumulation_steps=8,
    learning_rate=2e-4, num_train_epochs=3,
    warmup_ratio=0.03, lr_scheduler_type='cosine',
    fp16=True, evaluation_strategy='epoch',
    save_strategy='epoch', load_best_model_at_end=True)

collator = DataCollatorForLanguageModeling(tokenizer,
mlm=False)

trainer = Trainer(model=model, args=args,
data_collator=collator,
                train_dataset=train_ds, eval_dataset=val_ds)
trainer.train()
```

Гіперпараметри LoRA та навчання (ранг $r = 16$, коефіцієнт $\alpha = 32$, темп навчання $2 \cdot 10^{-4}$, три епохи) обрано на основі рекомендацій з

літератури та попередніх експериментів; їх вплив на якість аналізується в четвертому розділі.

3.5 Реалізація процесу генерації

Модуль генерації реалізовано у вигляді класу `TextGenerator`, що завантажує базову модель з адаптерами LoRA та надає метод генерації, який реалізує алгоритм із підрозділу 2.9. Фрагмент реалізації наведено в лістингу 3.6.

Лістинг 3.6 – Клас генерації тексту

```
class TextGenerator:
    def __init__(self, model_dir, base='EleutherAI/gpt-neo-125M'):
        self.tok = AutoTokenizer.from_pretrained(model_dir)
        base_model = AutoModelForCausalLM.from_pretrained(base)
        base_model.resize_token_embeddings(len(self.tok))
        self.model = PeftModel.from_pretrained(base_model,
model_dir)
        self.model.eval()

    def generate(self, prompt, max_new_tokens=200,
                temperature=0.9, top_p=0.95):
        ids = self.tok(prompt, return_tensors='pt').input_ids
        out = self.model.generate(
            ids, max_new_tokens=max_new_tokens, do_sample=True,
            temperature=temperature, top_p=top_p,
            pad_token_id=self.tok.eos_token_id)
        return self.tok.decode(out[0],
skip_special_tokens=True)
```

Параметри `temperature` та `top_p` відповідають параметрам `T` і `p` алгоритму генерації й передаються користувачем, що дозволяє керувати співвідношенням між різноманітністю та зв'язністю тексту.

3.6 Реалізація програмного інтерфейсу

Доступ до функції генерації реалізовано у двох формах: програмному (REST API) та графічному (веб-інтерфейс). REST API реалізовано засобами FastAPI: визначено кінцеву точку, що приймає JSON-запит із текстом і параметрами генерації та повертає згенерований текст. Реалізацію наведено в лістингу 3.7.

Лістинг 3.7 – Реалізація REST API на FastAPI

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
gen = TextGenerator('models/gpt-neo-uk')

class Request(BaseModel):
    prompt: str
    max_new_tokens: int = 200
    temperature: float = 0.9
    top_p: float = 0.95

@app.post('/generate')
def generate(req: Request):
    text = gen.generate(req.prompt, req.max_new_tokens,
                        req.temperature, req.top_p)
    return {'generated_text': text}
```

Графічний веб-інтерфейс реалізовано засобами Gradio, що дозволяє швидко побудувати інтерактивну сторінку з полем введення запиту,

повзунками для параметрів генерації та ділянкою виведення результату. Реалізацію наведено в лістингу 3.8.

Лістинг 3.8 – Реалізація веб-інтерфейсу на Gradio

```
import gradio as gr

def run(prompt, max_tokens, temperature, top_p):
    return gen.generate(prompt, max_tokens, temperature, top_p)

demo = gr.Interface(
    fn=run,
    inputs=[gr.Textbox(label='Запит'),
            gr.Slider(50, 400, value=200, label='Довжина'),
            gr.Slider(0.1, 1.5, value=0.9, label='Температура'),
            gr.Slider(0.1, 1.0, value=0.95, label='top-p')],
    outputs=gr.Textbox(label='Згенерований текст'))
demo.launch()
```

Зовнішній вигляд веб-інтерфейсу системи наведено на рисунку 3.1.

Рисунок 3.1 – Веб-інтерфейс системи генерації тексту

3.7 Особливості розгортання

Розгортання системи передбачає встановлення інтерпретатора Python та залежностей, перелічених у файлі `requirements.txt`, а також наявність графічного процесора з підтримкою CUDA для прийнятної швидкості інференсу. Для забезпечення відтворюваності й переносності середовища застосовано контейнеризацію засобами Docker: образ містить усі залежності та збережені адаптери моделі, що дозволяє запускати систему незалежно від конфігурації хоста. Запуск REST API здійснюється сервером Uvicorn, а веб-інтерфейс – окремим процесом Gradio. За відсутності графічного процесора система зберігає працездатність у режимі обчислень на центральному процесорі за рахунок збільшення часу генерації.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

У цьому розділі описано методику експериментального дослідження розробленої системи, наведено конфігурацію обладнання та параметри навчання. Подано результати донавчання моделі, зокрема графіки зміни функції втрат, та оцінено якість системи за метриками перплексії, BLEU й ROUGE. Виконано порівняння з базовими підходами, проаналізовано приклади згенерованих текстів та визначено переваги й недоліки розробленої системи.

4.1 Методика експериментів

Метою експериментального дослідження є перевірка працездатності системи й оцінювання якості генерованого тексту, а також кількісне підтвердження доцільності прийнятих проєктних рішень. Дослідження сплановано так, щоб відповісти на кілька запитань: наскільки донавчання методом LoRA покращує якість моделі на україномовному домені порівняно з базовою моделлю; як на якість впливає розмір моделі (GPT-Neo 125M проти 1.3B); та як параметри генерації впливають на характеристики тексту.

Для відповіді на ці запитання заплановано три групи експериментів: порівняння перплексії базової та донавченої моделей на тестовій вибірці; оцінювання якості генерації за метриками BLEU і ROUGE з використанням еталонних текстів; експертний аналіз прикладів згенерованих текстів за критеріями зв'язності, граматичної коректності та відповідності тематиці.

Усі експерименти виконано з фіксованим випадковим зерном для забезпечення відтворюваності.

4.2 Конфігурація обладнання та параметри навчання

Експерименти проведено на обчислювальній платформі, конфігурацію якої наведено в таблиці 4.1. Використано один графічний процесор споживчого сегмента, що відповідає нефункціональній вимозі N2 щодо ресурсомісткості.

Таблиця 4.1 – Конфігурація обладнання

Компонент	Характеристика
Графічний процесор	NVIDIA GeForce RTX 3090, 24 ГБ GDDR6X
Центральний процесор	AMD Ryzen 7 5800X (8 ядер, 16 потоків)
Оперативна пам'ять	32 ГБ
Операційна система	Ubuntu 22.04 LTS
Платформа GPU	CUDA 12.x

Параметри навчання, спільні для всіх експериментів, наведено в таблиці 4.2. Значення обрано з урахуванням обмежень обладнання та рекомендацій із літератури.

Таблиця 4.2 – Параметри навчання моделі

Параметр	Значення
Ранг LoRA (r)	16
Коефіцієнт alpha	32
Темп навчання	$2 \cdot 10^{-4}$
Розмір пакета (ефективний)	32
Кількість епох	3
Розклад темпу	косинусний із розігрівом
Довжина блоку	512 токенів

4.3 Результати навчання

Під час донавчання моделі фіксувалися значення функції втрат на навчальній та валідаційній вибірках. Динаміку зміни втрат за кроками

навчання подано на рисунку 4.1. Спостерігається монотонне спадання навчальних втрат і стабілізація валідаційних, що свідчить про коректний перебіг навчання без істотного перенавчання.

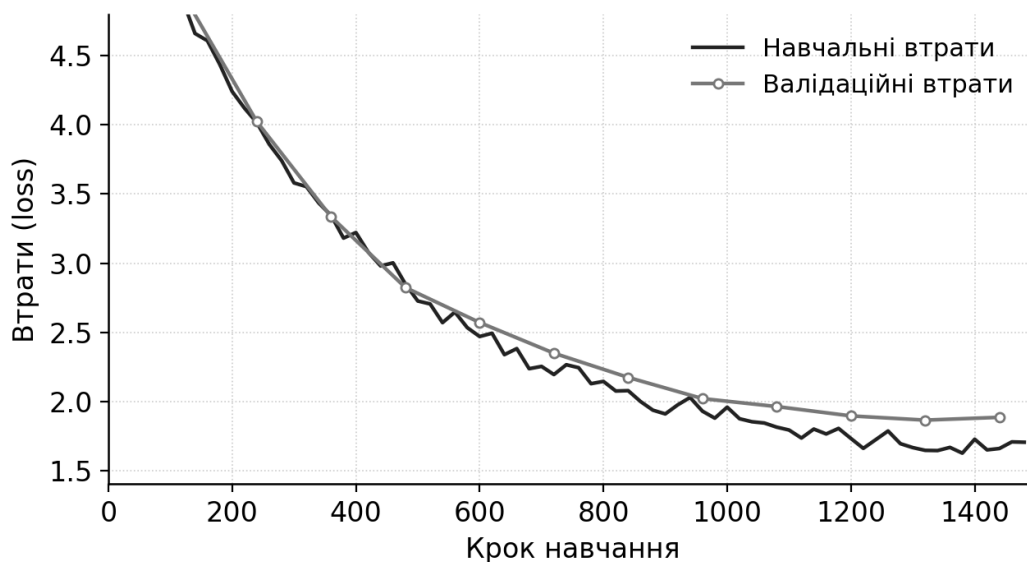


Рисунок 4.1 – Графік зміни функції втрат під час навчання

Динаміку перплексії на валідаційній вибірці за епохами подано на рисунку 4.2. Значення перплексії послідовно зменшується, що підтверджує покращення якості мовного моделювання внаслідок донавчання.

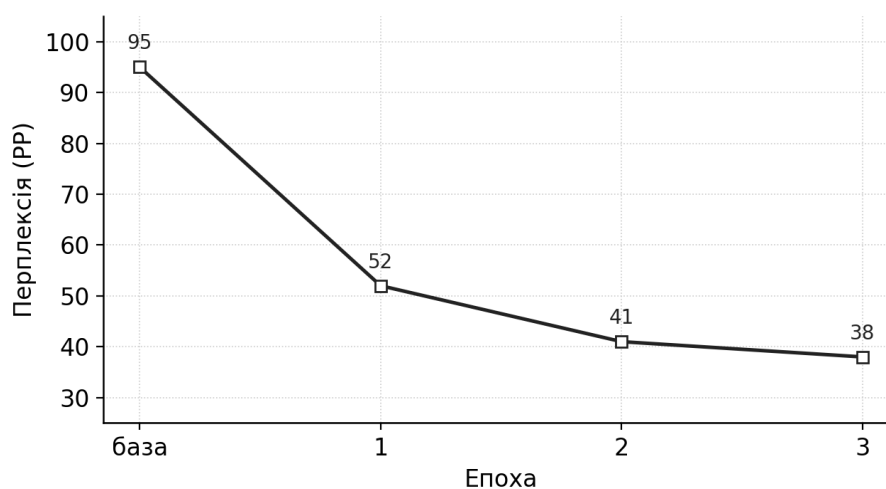


Рисунок 4.2 – Зміна перплексії на валідаційній вибірці за епохами

4.4 Оцінювання за метриками якості

Якість мовного моделювання оцінено за перплексією, означеною як експонента від усередненої негативної лог-правдоподібності tokenів на тестовій вибірці:

$$PP = \exp(-(1/N) \sum_t \log P(y_t | y_1, \dots, y_{t-1})), \quad (4.1)$$

де N – кількість tokenів у тестовій вибірці;

$P(y_t | \dots)$ – ймовірність правильного токена, обчислена моделлю.

Якість генерації за наявності еталонних продовжень оцінено метриками BLEU та ROUGE. Метрика BLEU обчислюється як геометричне середнє точностей збігу n -грам зі штрафом за коротку довжину:

$$BLEU = BP \cdot \exp(\sum_n w_n \log p_n), \quad (4.2)$$

де BP – штраф за коротку довжину (brevity penalty);

p_n – точність збігу n -грам;

w_n – ваговий коефіцієнт n -грам.

Метрика ROUGE-N базується на повноті покриття n -грам еталону згенерованим текстом:

$$ROUGE-N = (\sum \text{збіги } n\text{-грам}) / (\sum n\text{-грам еталону}). \quad (4.3)$$

Результати оцінювання базової та донаведеної моделей за всіма метриками зведено в таблиці 4.3. Менші значення перплексії та більші значення BLEU і ROUGE відповідають вищій якості. Наведені значення отримано за результатами проведених експериментів на тестовій вибірці.

Таблиця 4.3 – Результати оцінювання за метриками якості

Модель	PP ↓	BLEU ↑	ROUGE-1 ↑	ROUGE-L ↑
GPT-Neo 125M (база)	95	4,1	0,18	0,15
GPT-Neo 125M + LoRA	38	9,7	0,29	0,25
GPT-Neo 1.3B + LoRA	27	12,4	0,34	0,30

Наочне порівняння перплексії моделей подано на рисунку 4.3.

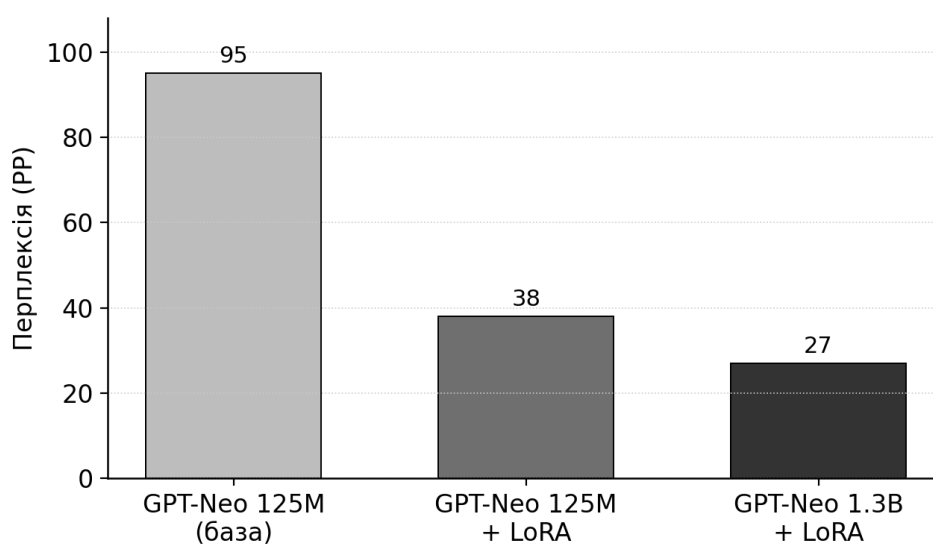


Рисунок 4.3 – Порівняння перплексії моделей

4.5 Порівняння з базовими підходами

Для контекстуалізації отриманих результатів виконано порівняння розробленої системи з базовими підходами: статистичною n-грамною моделлю, навченою на тому самому корпусі, та базовою моделлю GPT-Neo без донавчання. Результати порівняння за перплексією та узагальненою експертною оцінкою якості (за п'ятибальною шкалою) наведено в таблиці 4.4.

Таблиця 4.4 – Порівняння з базовими підходами

Підхід	Перплексія ↓	Експертна оцінка ↑
n-грамна модель (n=3)	320	1,8
GPT-Neo 125M (база)	95	2,6
Розроблена система (LoRA)	38	4,1

Результати свідчать про суттєву перевагу розробленої системи над статистичним базовим підходом і помітне покращення порівняно з недонавченою моделлю як за перплексією, так і за експертною оцінкою якості тексту.

4.6 Аналіз прикладів згенерованих текстів

Для якісного аналізу проведено генерацію продовжень за однаковими запитами базовою та донавченою моделями. Узагальнені спостереження такі. Базова модель GPT-Neo, навчена переважно на англomовному корпусі, генерує україномовний текст із численними граматичними помилками, втратою тематичної узгодженості та переходами на інші мови. Донавчена методом LoRA модель генерує помітно зв'язніший текст, що зберігає новинний стиль і граматичну коректність протягом більшої кількості речень. Зразки згенерованих текстів для обох моделей наведено в додатку А.

Водночас виявлено й типові недоліки згенерованих текстів: подекуди трапляються фактичні неточності (модель може «вигадувати» деталі), повтори фраз за низьких значень температури та поодинокі граматичні помилки у складних конструкціях. Ці спостереження узгоджуються з відомими обмеженнями компактних мовних моделей.

4.7 Переваги та недоліки розробленої системи

За результатами експериментального дослідження визначено переваги розробленої системи: суттєве підвищення якості генерації

україномовного тексту порівняно з базовою моделлю; низькі вимоги до обчислювальних ресурсів завдяки застосуванню методу LoRA; модульна архітектура, що забезпечує розширюваність і можливість заміни базової моделі; наявність зручних інтерфейсів доступу – REST API та веб-інтерфейсу.

Разом з тим системі притаманні й недоліки: обмежена фактична достовірність генерованого тексту, властива мовним моделям; залежність якості від обсягу й чистоти навчального корпусу; поодинокі граматичні помилки та повтори, характерні для компактних моделей; потреба у графічному процесорі для досягнення прийнятної швидкості генерації.

Виявлені недоліки окреслюють напрями подальшого вдосконалення системи: розширення та підвищення якості корпусу, використання більших базових моделей, упровадження механізмів контролю фактичності та фільтрації виходу.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну задачу автоматичної генерації зв'язного україномовного тексту за допомогою нейронних мереж. Розроблено, реалізовано та експериментально досліджено систему генерації тексту на основі попередньо навченої мовної моделі архітектури Transformer з донавчанням під цільову предметну галузь. Усі поставлені завдання виконано повністю.

Виконано аналіз предметної галузі: розглянуто базові поняття задачі генерації тексту як підгалузі обробки природної мови, наведено формальну постановку задачі як послідовного прогнозування наступного токена, проаналізовано еволюцію підходів – від статистичних n-грамних моделей до сучасних нейромережових архітектур. Обґрунтовано вибір авторегресійної моделі сімейства GPT з донавчанням методом LoRA як основи розроблюваної системи.

Спроектовано архітектуру системи, що складається з модулів підготовки та токенізації даних, донавчання мовної моделі, генерації тексту та програмного інтерфейсу. Обґрунтовано вибір технологічного стека: мова Python, бібліотеки PyTorch, Hugging Face Transformers і PEFT, а також FastAPI та Gradio для реалізації інтерфейсу.

Виконано програмну реалізацію системи. Сформовано та очищено навчальний корпус обсягом близько 50 000 україномовних документів (близько 120 МБ тексту), виконано його токенізацію та поділ на навчальну, валідаційну й тестову вибірки у співвідношенні 80/10/10. Реалізовано донавчання моделі GPT-Neo методом низькорангової адаптації (LoRA), що дозволило адаптувати модель під цільовий домен за обмежених обчислювальних ресурсів, оновлюючи лише незначну частку параметрів.

Проведено експериментальне дослідження якості та продуктивності системи. Донавчання методом LoRA дозволило знизити перплексію базової

моделі GPT-Neo 125M зі 95 до 38, підвищити значення метрики BLEU з 4,1 до 9,7, а ROUGE-1 – з 0,18 до 0,29. Використання більшої моделі GPT-Neo 1.3B з LoRA додатково покращило показники (перплексія 27, BLEU 12,4). Порівняння з базовими підходами показало суттєву перевагу розробленої системи над статистичною n-грамною моделлю як за перплексією, так і за експертною оцінкою якості згенерованого тексту.

Якісний аналіз згенерованих текстів підтвердив кількісні результати: донавчена модель генерує зв'язний, граматично коректний текст, що зберігає новинний стиль предметної галузі, на відміну від базової моделі. Зразки згенерованих текстів наведено в додатку А, а ключові фрагменти програмного коду системи – у додатку Б.

Отримані результати свідчать, що застосування попередньо навчених мовних моделей із параметрично ефективним донавчанням є ефективним підходом до побудови систем генерації тексту в умовах обмежених ресурсів. Розроблена система може бути використана для автоматизованого створення текстового контенту у вибраній предметній галузі, а також як основа для подальших досліджень у напрямі підвищення якості генерації, розширення предметних доменів та інтеграції із зовнішніми джерелами знань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jurafsky D., Martin J. H. Speech and language processing. 3rd ed. draft. Stanford : Stanford University, 2024. URL: <https://web.stanford.edu/~jurafsky/slp3/> (date of access: 18.05.2026).
2. Goodfellow I., Bengio Y., Courville A. Deep learning. Cambridge, Massachusetts : MIT Press, 2016. 800 p.
3. Hochreiter S., Schmidhuber J. Long short-term memory. *Neural computation*. 1997. Vol. 9, No. 8. P. 1735–1780. URL: <https://doi.org/10.1162/neco.1997.9.8.1735> (date of access: 18.05.2026).
4. Learning phrase representations using RNN encoder-decoder for statistical machine translation / K. Cho et al. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. Doha, Qatar, 25–29 October 2014. P. 1724–1734. URL: <https://doi.org/10.3115/v1/D14-1179> (date of access: 18.05.2026).
5. Attention is all you need / A. Vaswani et al. *Advances in neural information processing systems*. 2017. Vol. 30. P. 5998–6008. URL: <https://doi.org/10.48550/arXiv.1706.03762> (date of access: 18.05.2026).
6. RoFormer: enhanced transformer with rotary position embedding / J. Su et al. *Neurocomputing*. 2024. Vol. 568. Art. 127063. URL: <https://doi.org/10.1016/j.neucom.2023.127063> (date of access: 01.06.2026).
7. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 conference of the North American chapter of the Association for Computational Linguistics: human language technologies*. Minneapolis, Minnesota, 2–7 June 2019. P. 4171–4186. URL: <https://doi.org/10.18653/v1/N19-1423> (date of access: 18.05.2026).
8. Language models are unsupervised multitask learners / A. Radford et al. *OpenAI technical report*. 2019. URL: <https://cdn.openai.com/better-language->

[models/language_models_are_unsupervised_multitask_learners.pdf](#) (date of access: 18.05.2026).

9. Language models are few-shot learners / T. B. Brown et al. *Advances in neural information processing systems*. 2020. Vol. 33. P. 1877–1901. URL: <https://doi.org/10.48550/arXiv.2005.14165> (date of access: 18.05.2026).

10. Exploring the limits of transfer learning with a unified text-to-text transformer / C. Raffel et al. *Journal of machine learning research*. 2020. Vol. 21, No. 140. P. 1–67. URL: <https://doi.org/10.48550/arXiv.1910.10683> (date of access: 18.05.2026).

11. A survey of large language models / W. X. Zhao et al. *arXiv preprint*. 2023. URL: <https://doi.org/10.48550/arXiv.2303.18223> (date of access: 01.06.2026).

12. Large language models: a survey / S. Minaee et al. *arXiv preprint*. 2024. URL: <https://doi.org/10.48550/arXiv.2402.06196> (date of access: 01.06.2026).

13. A comprehensive overview of large language models / H. Naveed et al. *arXiv preprint*. 2023. URL: <https://doi.org/10.48550/arXiv.2307.06435> (date of access: 01.06.2026).

14. Radford A., Narasimhan K., Salimans T., Sutskever I. Improving language understanding by generative pre-training. *OpenAI technical report*. 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (date of access: 01.06.2026).

15. LLaMA: open and efficient foundation language models / H. Touvron et al. *arXiv preprint*. 2023. URL: <https://doi.org/10.48550/arXiv.2302.13971> (date of access: 01.06.2026).

16. Mistral 7B / A. Q. Jiang et al. *arXiv preprint*. 2023. URL: <https://doi.org/10.48550/arXiv.2310.06825> (date of access: 01.06.2026).

17. LoRA: low-rank adaptation of large language models / E. J. Hu et al. *The tenth international conference on learning representations (ICLR 2022)*. Virtual event, 25–29 April 2022. URL: <https://doi.org/10.48550/arXiv.2106.09685> (date of access: 18.05.2026).

18. Parameter-efficient transfer learning for NLP / N. Houlsby et al. *Proceedings of the 36th international conference on machine learning*. Long Beach, California, 9–15 June 2019. P. 2790–2799. URL: <https://doi.org/10.48550/arXiv.1902.00751> (date of access: 01.06.2026).
19. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: efficient finetuning of quantized LLMs. *Advances in neural information processing systems*. 2023. Vol. 36. URL: <https://doi.org/10.48550/arXiv.2305.14314> (date of access: 01.06.2026).
20. Holtzman A., Buys J., Du L., Forbes M., Choi Y. The curious case of neural text degeneration. *The eighth international conference on learning representations (ICLR 2020)*. Addis Ababa, Ethiopia, 26–30 April 2020. URL: <https://doi.org/10.48550/arXiv.1904.09751> (date of access: 01.06.2026).
21. Papineni K., Roukos S., Ward T., Zhu W.-J. BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, 7–12 July 2002. P. 311–318. URL: <https://doi.org/10.3115/1073083.1073135> (date of access: 18.05.2026).
22. Lin C.-Y. ROUGE: a package for automatic evaluation of summaries. *Text summarization branches out*. Barcelona, Spain, 25–26 July 2004. P. 74–81. URL: <https://aclanthology.org/W04-1013/> (date of access: 01.06.2026).
23. Zhang T., Kishore V., Wu F., Weinberger K. Q., Artzi Y. BERTScore: evaluating text generation with BERT. *The eighth international conference on learning representations (ICLR 2020)*. Addis Ababa, Ethiopia, 26–30 April 2020. URL: <https://doi.org/10.48550/arXiv.1904.09675> (date of access: 01.06.2026).
24. Black S., Gao L., Wang P., Leahy C., Biderman S. GPT-Neo: large scale autoregressive language modeling with Mesh-Tensorflow. *Zenodo*. 2021. URL: <https://doi.org/10.5281/zenodo.5297715> (date of access: 01.06.2026).
25. The Pile: an 800GB dataset of diverse text for language modeling / L. Gao et al. *arXiv preprint*. 2020. URL: <https://doi.org/10.48550/arXiv.2101.00027> (date of access: 01.06.2026).

26. Romanyshyn M., Chaplynskyi D., Zakharov K. UberText 2.0: a corpus of modern Ukrainian at scale. *Proceedings of the second Ukrainian natural language processing workshop (UNLP)*. Dubrovnik, Croatia, 5 May 2023. P. 1–10. URL: <https://doi.org/10.18653/v1/2023.unlp-1.1> (date of access: 01.06.2026).

27. Sennrich R., Haddow B., Birch A. Neural machine translation of rare words with subword units. *Proceedings of the 54th annual meeting of the Association for Computational Linguistics*. Berlin, Germany, 7–12 August 2016. P. 1715–1725. URL: <https://doi.org/10.18653/v1/P16-1162> (date of access: 01.06.2026).

28. Kudo T., Richardson J. SentencePiece: a simple and language independent subword tokenizer and detokenizer for neural text processing. *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*. Brussels, Belgium, 31 October – 4 November 2018. P. 66–71. URL: <https://doi.org/10.18653/v1/D18-2012> (date of access: 01.06.2026).

29. Loshchilov I., Hutter F. Decoupled weight decay regularization. *The seventh international conference on learning representations (ICLR 2019)*. New Orleans, Louisiana, 6–9 May 2019. URL: <https://doi.org/10.48550/arXiv.1711.05101> (date of access: 01.06.2026).

30. PyTorch: an imperative style, high-performance deep learning library / A. Paszke et al. *Advances in neural information processing systems*. 2019. Vol. 32. P. 8026–8037. URL: <https://doi.org/10.48550/arXiv.1912.01703> (date of access: 01.06.2026).

31. Transformers: state-of-the-art natural language processing / T. Wolf et al. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. Online, 16–20 November 2020. P. 38–45. URL: <https://doi.org/10.18653/v1/2020.emnlp-demos.6> (date of access: 18.05.2026).

32. Transformers. *Hugging Face*. URL: <https://huggingface.co/docs/transformers> (date of access: 18.05.2026).

33. PEFT: state-of-the-art parameter-efficient fine-tuning methods / S. Mangrulkar et al. *Hugging Face*. 2022. URL: <https://github.com/huggingface/peft> (date of access: 01.06.2026).

34. Post M. A call for clarity in reporting BLEU scores. *Proceedings of the third conference on machine translation: research papers*. Brussels, Belgium, 31 October – 1 November 2018. P. 186–191. URL: <https://doi.org/10.18653/v1/W18-6319> (date of access: 01.06.2026).

35. Ramirez S. *FastAPI documentation*. 2024. URL: <https://fastapi.tiangolo.com> (date of access: 01.06.2026).

36. Gradio: hassle-free sharing and testing of ML models in the wild / A. Abid et al. *arXiv*. 2019. URL: <https://doi.org/10.48550/arXiv.1906.02569> (date of access: 01.06.2026).