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)

Веб-додаток для генерації рецептів на основі наявних продуктів користувача з використанням методів обробки природної мови та рекомендаційних алгоритмів
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-3

Денис Пащенко
(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 ст. викл. Олександр Стьопін
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 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 Пащенку Денису Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-додаток для генерації рецептів на основі наявних продуктів користувача з використанням методів обробки природної мови та рекомендаційних алгоритмів
затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 24 червня 20 25 р.
3. Вихідні дані до роботи Дані про структуру та функціонал веб-додатку, інтерфейс користувача (введення інгредієнтів), програмний код на Python з використанням Flask, HTML. Набір тестових даних (комбінації інгредієнтів), сценарії обробки запитів, результат генерації рецептів за допомогою моделі GPT-2, оцінка змістовності, релевантності та практичності згенерованих відповідей

4. Перелік питань, що потрібно опрацювати в роботі
1) Теоретичні основи генерації тексту в задачах обробки природної мови
2) Обгрунтування вибору технологій та інструментів реалізації проєкту
3) Реалізація веб-додатку для генерації рецептів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі, обґрунтування	20.05.2025	виконано
	актуальності та вибір технологій		
3	Дослідження методів NLP та моделей,	23.05.2025	виконано
	інтеграція GPT-2		
4	Розробка архітектури та реалізація модуля	26.05.2025	виконано
	обробки даних		
5	Розробка інтерфейсу користувача та виведення	29.05.2025	виконано
	відповіді		
6	Тестування, експериментальна перевірка	31.05.2025	виконано
7	Аналіз результатів, формування висновків	02.06.2025	виконано
8	Написання та оформлення пояснювальної	04.06.2025	виконано
	записки		
9	Попередній захист	13.06.2025	виконано
10	Захист перед екзаменаційною комісією	24.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Олександр Стьопін
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 71 с., 20 рис., 1 табл., 1 дод., 22 джерела.

ВЕБ-ДОДАТОК, ГЕНЕРАЦІЯ ТЕКСТУ, ІНГРІДІЄНТИ, МОВНА МОДЕЛЬ, ОБРОБКА ПРИРОДНОЇ МОВИ, РЕКОМЕНДАЦІЙНІ СИСТЕМИ, РЕЦЕПТИ, GPT-2, FLASK.

Об’єкт дослідження – процеси генерації персоналізованих кулінарних рецептів на основі вхідних даних користувача із використанням технологій обробки природної мови та алгоритмів рекомендаційних систем.

Предмет дослідження – методи реалізації веб-додатків на основі мовних моделей (GPT-2) і рекомендаційних алгоритмів, здатних створювати рецепти страв із заданого набору інгредієнтів.

Мета роботи – розробити та реалізувати інтелектуальний веб-додаток, який на основі введених користувачем продуктів формуватиме кулінарні рецепти з використанням методів обробки природної мови та рекомендаційних технологій.

Методи дослідження – аналіз наукових джерел з обробки природної мови та генеративних моделей; застосування мовної моделі GPT-2 для генерації тексту; використання інструментів Flask для реалізації веб-інтерфейсу; експериментальна перевірка якості результатів на практичному прикладі.

Розроблено рекомендації щодо використання обробки природної мови та рекомендаційних алгоритмів у веб-додатках для генерації кулінарних рецептів. Реалізовано сервіс на основі GPT-2, що формує рецепти за введеними інгредієнтами. Результати підтвердили ефективність генеративного підходу до персоналізованих рекомендацій.

ABSTRACT

Bachelor’s thesis contains: 71 pp., 20 fig., 1 tabl., 1 ann., 22 references.

FLASK, GPT-2, INGREDIENTS, LANGUAGE MODEL, NATURAL LANGUAGE PROCESSING, RECIPE GENERATION, RECIPES, RECOMMENDER SYSTEMS, WEB APPLICATION.

Object of the research – the processes of generating personalized cooking recipes based on user input using natural language processing technologies and recommender system algorithms.

Subject of the research – methods of implementing web applications based on language models (GPT-2) and recommender algorithms capable of creating dish recipes from a given set of ingredients.

Purpose of the work – to develop and implement an intelligent web application that, based on the products entered by the user, generates cooking recipes using natural language processing methods and recommender technologies.

Research methods – analysis of scientific sources on natural language processing and generative models; use of the GPT-2 language model for text generation; application of Flask tools for implementing the web interface; experimental evaluation of result quality using a practical example.

The results – recommendations were developed for using natural language processing and recommender algorithms in web applications for recipe generation. A GPT-2-based service was implemented that forms recipes from the entered ingredients. The results confirmed the effectiveness of the generative approach to personalized recommendations.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – інтерфейс прикладного програмування;

FAQ – Frequently Asked Questions – поширені запитання;

GPT-2 – Generative Pre-trained Transformer 2 – трансформер, навчений генерувати текст;

GPU – Graphics Processing Unit – графічний процесор;

HTML – HyperText Markup Language – мова розмітки гіпертексту.

LM – Language Modeling – мовне моделювання;

NLP – Natural Language Processing – обробка природної мови;

QA – Quality Assurance – забезпечення якості;

RAG – Retrieval-Augmented Generation – генерація з доповненим пошуком;

WSGI – Web Server Gateway Interface – інтерфейс шлюзу веб-сервера.

ВСТУП

Сучасні інформаційні технології активно проникають у всі сфери людської діяльності, зокрема в галузь харчування та кулінарії. Одним із перспективних напрямів є використання штучного інтелекту для генерації персоналізованого контенту, зокрема кулінарних рецептів. У цьому контексті особливої актуальності набувають методи обробки природної мови (NLP) і генеративні мовні моделі, які здатні формувати текстові відповіді, наближені до людських.

Завдяки стрімкому розвитку автогенеративних моделей, таких як GPT-2, стало можливим створення веб-застосунків, які не лише аналізують вхідні дані, але й на їх основі генерують змістовні тексти у відповідному форматі. Це відкриває нові можливості для побудови сервісів, які допомагають користувачам, наприклад, формувати унікальні рецепти зі зручним інтерфейсом і мінімальним часом взаємодії.

Об’єктом дослідження даної роботи є процеси генерації кулінарних рецептів на основі користувацьких запитів з використанням мовних моделей. Предметом дослідження виступають методи та інструменти реалізації веб-додатків із підтримкою генерації тексту засобами NLP та рекомендаційних алгоритмів.

Метою кваліфікаційної роботи є розробка інтелектуального веб-додатку, який, спираючись на введені користувачем інгредієнти, автоматично формує кулінарні рецепти за допомогою GPT-2.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

– аналіз методів обробки природної мови та генерації тексту;

– обґрунтування вибору технологій реалізації, зокрема моделі GPT-2, бібліотеки Transformers, фреймворку Flask;

– побудова архітектури веб-додатку;

– реалізація модуля обробки вхідних даних користувача;

– виконання інференсу моделі та генерації відповідей;

– візуалізація результату на стороні клієнта;

– тестування системи на практичних прикладах та оцінка результатів.

Практична цінність роботи полягає в створенні функціонального прототипу системи, який демонструє можливості використання сучасних NLP-технологій у прикладному середовищі. Сервіс реалізовано за допомогою мови Python, фреймворку Flask та бібліотеки Transformers, що забезпечує інтеграцію з GPT-2. Проєкт також демонструє перспективність генеративного підходу порівняно з класичними системами рекомендацій.

1 ТЕОРЕТИЧНІ ОСНОВИ ГЕНЕРАЦІЇ ТЕКСТУ В ЗАДАЧАХ ОБРОБКИ ПРИРОДНОЇ МОВИ

1.1 Методи обробки природної мови у прикладних інтелектуальних системах

Обробка природної мови (NLP, natural language processing) – це підгалузь штучного інтелекту, що дозволяє комп’ютерам розуміти, інтерпретувати та опрацьовувати людську мову [1]. Інакше кажучи, NLP надає машинам здатність працювати з текстовими та голосовими даними так, як це роблять люди. Це є фундаментом багатьох прикладних інтелектуальних систем: від чат-ботів і голосових помічників (Apple Siri, Amazon Alexa) до пошукових систем та систем автоматичного перекладу [1]. Основні задачі NLP включають розпізнавання мовлення, класифікацію тексту, розуміння природної мови та генерацію тексту [2]. Саме остання задача – генерування зв’язного осмисленого тексту – є предметом розгляду в даній роботі.

Історично методи NLP еволюціонували від ранніх символічних підходів до сучасних нейронних мереж. Перші покоління NLP-систем спиралися на жорстко задані правила і лінгвістичні граматики (так звані символічні методи). Надалі, з появою достатніх обсягів цифрових текстів, набули розвитку статистичні методи – моделі на основі ймовірностей і машинного навчання, що навчалися на великих корпусах тексту. З середини 2010-х років домінуючим підходом стали нейронні мережі глибокого навчання. Такі мережі автоматично виокремлюють багаторівневі ознаки тексту та вчаться виконувати NLP-завдання із прикладами «входів-виходів», часто перевершуючи попередні алгоритми. Зокрема, сучасні контекстні моделі на основі механізму уваги (архітектура Transformer) встановили новий стандарт якості в NLP [3].

У типовій сучасній NLP-системі присутні етапи попередньої обробки даних (токенізація – розбиття тексту на токени, лематизація чи стемінг – нормалізація слів, синтаксичний аналіз речень тощо) та подальшого моделювання значення тексту за допомогою машинного навчання [3]. В результаті поєднання цих методів сучасні інтелектуальні системи здатні аналізувати текстову інформацію і генерувати корисні відповіді або дії на її основі.

Застосування методів NLP у прикладних системах охоплює широкий спектр. Наприклад, діалогові системи та чат-боти використовують NLP для розуміння запитів користувача (Natural Language Understanding) і формування відповідей (Natural Language Generation). Системи машинного перекладу автоматично перетворюють текст з однієї мови на іншу, аналізуючи структуру вхідного речення та генеруючи еквівалент цільовою мовою.

Пошукові системи застосовують NLP для інтерпретації змісту запиту і пошуку релевантних результатів, а також для ранжування документів за тематичною близькістю. Рекомендаційні системи можуть аналізувати описи товарів чи відгуки і добирати персоналізовані рекомендації для користувача. В усіх цих прикладах методи обробки мови інтегровані в інтелектуальний застосунок, підвищуючи його «розуміння» контексту і здатність спілкуватися з людиною природною мовою.

Отже, в прикладних інтелектуальних системах NLP виконує роль зв’язувальної ланки між людиною та машиною, забезпечуючи «розуміння» текстових даних. Сучасний розвиток глибокого навчання та контекстних моделей значно розширив можливості NLP-систем – від класифікації та вилучення інформації до повноцінної генерації зв’язного тексту на рівні, близькому до людського. Це створює підґрунтя для нових застосувань, зокрема генеративних систем, що автоматично пишуть тексти за заданими параметрами.

1.2 Застосування автогенеративних мовних моделей для текстового формування відповідей

Автогенеративні мовні моделі – це такі моделі штучного інтелекту, що здатні самостійно генерувати зв’язний текст, продовжуючи чи відповідаючи на наданий користувачем запит (текстовий контекст). Як правило, це мовні моделі великого масштабу, навчені на величезних корпусах текстових даних. Вони працюють за принципом авторегресії: генерують текст по одному слову (токену), кожного разу прогнозуючи наступне слово на основі всіх попередніх слів у контексті [4]. Прикметно, що такі моделі навчалися в режимі unsupervised learning на необмежених даних – тобто без ручної розмітки, просто намагаючись передбачити наступне слово в тексті. Це дозволяє їм опанувати статистичні закономірності мови і навіть певні фактичні знання про світ з тренувального корпусу.

Яскравим прикладом автогенеративної моделі є GPT-2 від OpenAI. Ця модель (Generative Pre-trained Transformer 2) являє собою великий трансформер із ~1.5 млрд параметрів, тренований на вибірці з 8 млн веб-сторінок обсягом ~40 ГБ тексту [5]. GPT-2 навчалась з простим завданням – передбачати наступне слово в тексті, даному попередній контекст [5]. Незважаючи на таку загальну ціль, модель виявила здатність генерувати цілком зв’язні та змістовні тексти у різних стилях, коли її «підказати» початковим фрагментом [5].

Зокрема, дослідниками відзначено, що GPT-2 демонструє базові вміння в читанні з поясненнями, машинному перекладі, відповіді на питання та реферуванні тексту [5] – тобто виконувала ці завдання, просто продовжуючи виданий їй контекст, хоча спеціально під ці задачі не налаштовувалась. Це показує універсальність підходу: велика мовна модель може самостійно породжувати відповіді на різноманітні запити, узагальнюючи знання зі свого корпусу даних.

Автогенеративні моделі застосовуються для формування текстових відповідей у багатьох сценаріях. Наприклад, чат-боти нового покоління (такі як ChatGPT) використовують гігантські мовні моделі для динамічної генерації відповіді на питання користувача, а не вибирають її із заздалегідь підготовленого набору. Це дозволяє отримувати більш гнучкі та контекстно-залежні відповіді. Інший приклад – системи питально-відповідального пошуку (QA): модель отримує на вхід запитання природною мовою і генерує розгорнуту відповідь, спираючись на «знання», засвоєні при тренуванні. Дослідження показують, що великі моделі типу GPT здатні з певною точністю відповідати на фактологічні запитання, навіть без доступу до зовнішньої бази знань [6]. Хоча якість ще далека від досконалості (наприклад, базова GPT-2 правильно відповідає лише на ~4% запитань у тестах на фактологічні знання [6]), сам факт генерації змістовних відповідей із непідготовлених даних вражає.

Також генеративні мовні моделі успішно застосовуються у творчих задачах: автоматичне написання статей, історій, сценаріїв тощо. Звичайно, тексти, згенеровані моделлю, потребують перевірки людиною, але як інструмент для чернеток чи натхнення такі системи вже корисні. У вузьких доменах модель можна спеціально донавчити на відповідному корпусі (наприклад, юридичні тексти, медична документація), і вона зможе генерувати змістовний текст у цій стилістиці. Проте навіть без донавчання великі моделі мають певний «енциклопедичний» запас знань та стилістичних шаблонів.

Наприклад, GPT-2, тренована на загальному інтернет-корпусі, може згенерувати кулінарний рецепт у стандартному форматі (список інгредієнтів та покрокова інструкція) – адже в її тренувальних даних ймовірно були тисячі кулінарних рецептів [7].

Як зауважує Дж. Шейн, GPT-2 має достатньо «довгу пам’ять» і великий досвід на інтернет-текстах, щоб видати рецепт, який на перший погляд виглядає правдоподібно (хоча часом і з курйозними помилками на

кшталт «чашка хрону» у рецепті брауні) [7]. Таким чином, автогенеративні мовні моделі відкривають можливість автоматичного формування текстових відповідей у найрізноманітніших прикладних задачах – від діалогу з користувачем до генерації інструкцій чи творчих текстів.

Водночас варто зазначити і обмеження цих моделей. Вони можуть генерувати неправдиву або нелогічну інформацію (так звані «галюцинації»), оскільки не мають доступу до перевірених знань, окрім тих, що були в даних навчання. Моделі типу GPT не гарантують точності фактів і не розуміють зміст у людському сенсі – вони оперують статистичною ймовірністю слів. Тому в критичних застосуваннях їх потрібно використовувати обережно, часто в поєднанні з перевіркою результату або з механізмами обмеження (фільтрації) вихідного тексту. Попри ці виклики, автогенеративні моделі вже продемонстрували практичну цінність і продовжують удосконалюватись, займаючи важливе місце в сучасних NLP-системах.

1.3 Генеративні підходи у порівнянні з традиційними рекомендаційними системами

Генеративні моделі формування тексту представляють нову парадигму у порівнянні з традиційними підходами, заснованими на відборі готових відповідей або рекомендацій. Традиційна рекомендаційна система (у контексті відповіді на запит користувача) зазвичай працює за принципом пошуку та вибору – з наявного набору даних (бази знань, колекції документів чи відповідей) підбирається найбільш релевантний елемент у відповідь на запит. Приклад: у FAQ-системі на запит користувача система знаходить найближче питання-відповідь і повертає готовий текст відповіді; або сервіс рекомендацій рецептів шукає в базі рецепт, що відповідає заданим інгредієнтам. Такий підхід часто називають ретривальним (retrieval-based) або підходом на основі відбору.

Натомість генеративний підхід передбачає динамічне створення нової відповіді, якої може не бути в явному вигляді в базі знань. Генеративний чат-бот або система відповіді на питання будує відповідь «з нуля» за допомогою мовної моделі, сформовану спеціально під даний запит. Як наслідок, ретривальні системи обмежені заздалегідь визначеним набором відповідей, а генеративні можуть продукувати довільні нові висловлювання [8]. Ретривальна система завжди поверне один з наперед передбачених варіантів (вона не придумує нічого нового), тоді як генеративна потенційно здатна відповісти на запит, якого раніше не було, комбінуючи релевантну інформацію з навченої моделі.

Розглянемо детальніше плюси і мінуси обох підходів:

– традиційна система дає більше контролю: всі її відповіді відомі та перевірені. Це важливо в доменах, де потрібна точність і відповідність фактам (наприклад, консультація з медицини – краще вибрати відповідь лікаря з бази знань, ніж згенерувати неперевірену пораду). Генеративна модель, навпаки, може проявити креативність – сформулювати відповідь у новий спосіб, пояснити ідею своїми словами, врахувати контекст запиту. Наприклад, на запит «Що мені приготувати з помідорів і сиру?» рекомендувальна система вибере один із відомих рецептів (скажімо, салат Капрезе) з бази, тоді як генеративна може описати оригінальний рецепт, комбінуючи інгредієнти нетривіальним чином;

– рекомендаційна система часто обмежена наявними шаблонами. Якщо запит користувача не має прямого відповідника у базі, система може повернути пустий результат або найближчу, але неточну рекомендацію. Генеративна модель у принципі здатна відповісти на будь-який запит, якщо має достатньо мовних знань – навіть на досить несподівані запитання вона згенерує хоч якусь відповідь. Це особливо корисно для діалогових систем, де користувацькі репліки непередбачувані;

– якщо говорити про якість та логічність, то тут традиційні системи часто виграють: відібрані відповіді – це зазвичай коректні людські тексти.

Генеративна ж модель може робити помилки, втрачати контекст, генерувати нелогічні або повторювані фрагменти. Типові проблеми генеративних чат-ботів – неузгодженість (можуть суперечити собі), галюцинації (вигадування фактів) тощо. Ретривальна система таких помилок не робить, але її обмеження – шаблонність і нездатність виходити за межі записаного. Іншими словами, генеративна модель «придумує» відповідь, а не шукає, тому може і помилитись, але може і підлаштуватись точніше під питання;

– рекомендаційна система потребує попереднього наповнення бази знань і розробки алгоритму пошуку по ній (наприклад, за ключовими словами або за векторними подібностями). Генеративна модель потребує потужних обчислювальних ресурсів для навчання на великому корпусі даних (зазвичай це вже зроблено розробниками моделі) і достатніх ресурсів для запуску моделі в реальному часі. У розгортанні також є різниця: ретривальну систему відносно просто розгорнути навіть на звичайному сервері, а в генеративній – велика модель може вимагати GPU і оптимізацій, щоб забезпечити швидку відповідь.

В літературі підкреслюється, що більшість практичних систем досі використовують ретривальні методи, адже генеративні методи – це відносно новий, менш досліджений напрям із непередбачуваними результатами [8]. Інженерам, які обирають генеративний підхід, доводиться вирішувати нетривіальні дослідницькі завдання, щоб досягти стабільної якості відповідей [8]. Натомість класичні методи більш зрозумілі й надійні. Проте зі стрімким прогресом глибоких моделей ситуація змінюється – генеративні чат-боти вже входять у повсякденне життя, демонструючи прийнятний рівень відповідей у багатьох випадках.

Таким чином, обидва підходи мають свою нішу. Традиційні рекомендаційні системи підходять там, де важлива передбачуваність, контроль і фактична точність відповіді (наприклад, рекомендація товарів, FAQ для банку тощо – де краще показати перевірену інформацію). Генеративні системи доречні, коли потрібна гнучка взаємодія і покриття

необмеженого простору запитів (наприклад, в персональних помічниках, творчих застосунках, навчальних діалогах). У багатьох сучасних рішеннях поєднують обидва підходи: наприклад, генеративна модель відповідає на питання користувача, але перевіряє факти через пошук по базі знань (концепція Retrieval-Augmented Generation, RAG). Це дозволяє отримати найкраще з обох світів – креативність моделі та надійність перевіреної інформації [9].

Для поставленої задачі – генерації кулінарних рецептів – обидва підходи теж могли б використовуватись. Традиційний шлях: рекомендувати користувачу вже існуючий рецепт з бази даних (фактично, пошук рецепта за заданими критеріями). Генеративний шлях: створити новий рецепт під запит користувача (напряму «вигадати» опис приготування зі списком інгредієнтів). У даному проекті зроблено вибір на користь генеративного підходу з використанням моделі GPT-2, що дозволяє продемонструвати можливості сучасних NLP-моделей у творчій задачі побудови рецепту. Попри ризик появи некоректних кулінарних рекомендацій, генеративний підхід цікавий тим, що може здивувати оригінальністю та продемонструвати небачені раніше комбінації інгредієнтів. Важливо лише розуміти обмеження: такий «віртуальний шеф-кухар» не має кулінарного досвіду, окрім статистичних закономірностей, тому згенеровані рецепти слід розцінювати як експериментальні.

1.4 Огляд інструментальних засобів реалізації NLP-систем (Transformers, PyTorch)

Розробка сучасних NLP-систем значно спрощується завдяки наявності потужних відкритих бібліотек і фреймворків. Вони надають реалізації поширених моделей, інструменти для їх навчання й використання, а також оптимізації для ефективної роботи на апаратному забезпеченні (GPU/TPU). У контексті нашого проекту ключову роль

відіграють дві технології: бібліотека Hugging Face Transformers і фреймворк глибокого навчання PyTorch.

Hugging Face Transformers – це популярна відкрито-кодова бібліотека, яка об'єднує реалізації найсучасніших моделей природної мови (трансформерів) під уніфікованим API [10]. В її репозиторії зібрано десятки pretrained-моделей для різних NLP-задач – від класичних BERT і GPT-2 до найновіших GPT-3, T5, BART, XXL-моделей тощо. Бібліотека спроектована так, щоб спростити використання цих моделей практиками, надаючи прості інтерфейси для завантаження переднавченої моделі та виконання з нею стандартних завдань (класифікація, генерація тексту, переклад, питання-відповідь тощо) [10]. Transformers пропонує «єдиний API для ретельно спроєктованих реалізацій сучасних архітектур трансформерів та колекцію попередньо навчених моделей, доступних спільнотою» [10]. Це означає, що розробник може за кілька команд отримати готову модель і відразу застосувати її на своєму наборі даних, не займаючись низькорівневими деталями архітектури чи завантаженням ваг з різних джерел.

Конкретно для задачі генерації тексту бібліотека Transformers надає зручний інструмент – pipeline для генерації. Лише вказавши ім'я моделі (наприклад, «gpt2»), можна отримати об'єкт, що генерує продовження тексту на основі вхідної підказки. Бібліотека бере на себе всі аспекти: завантажує ваги моделі GPT-2 з онлайн-репозиторію, готує токенізатор, забезпечує налаштування параметрів генерації (довжина виходу, температури, топ-k/топ-p фільтрація тощо). Це радикально знижує поріг входження: навіть студентський проект може використати state-of-the-art модель буквально за кілька рядків коду. Окрім pipeline, Transformers містить клас AutoModelForCausalLM (Language Modeling), який дозволяє гнучкіше інтегрувати автогенеративну модель у код – наприклад, щоб спочатку опрацювати введення, передати його в модель та отримати результати для подальшого форматування. Таким чином, Transformers є

ключовим інструментом для реалізації необхідної системи: з його допомогою інтегровано модель GPT-2 у Python-застосунок, зведено до мінімуму обсяг власного коду для роботи з нейронною мережею.

Варто згадати, що бібліотека Transformers підтримує обидва основні фреймворки глибокого навчання – і PyTorch, і TensorFlow. У роботі використовується PyTorch-варіант (за замовчуванням модель GPT-2 завантажується як torch.nn.Module), оскільки PyTorch є обраним середовищем виконання.

PyTorch – це один з провідних фреймворків для реалізації нейронних мереж, розроблений компанією Facebook (нині Meta AI). Він здобув популярність завдяки зручності для дослідників та високій продуктивності. Архітектурно PyTorch побудований на принципі динамічного графу обчислень (define-by-run): моделі виконуються у імперативному стилі, що означає, ніби ми пишемо звичайний Python-код, а не декларативно визначаємо мережу наперед [11]. Такий підхід полегшує відладку, дає гнучкість у побудові складних архітектур із умовними розгалуженнями, циклами тощо.

Водночас, завдяки оптимізаціям, PyTorch досягає продуктивності на рівні статичних фреймворків (як TensorFlow) – виконання на GPU прискорюється бібліотеками CUDA, автоматично використовується векторизація і т.д. [11]. Як зазначають автори PyTorch, фреймворк виконує обчислення на тензорах з автоматичним обчисленням градієнтів та GPU-прискоренням у режимі реального часу, при цьому продуктивність порівнянна з найшвидшими бібліотеками глибокого навчання [11]. Це зробило PyTorch надзвичайно популярним у науковій спільноті: вже у 2019 році 296 подань на конференцію ICLR згадували використання PyTorch [11].

Для дослідного проекту важливо, що PyTorch має тісну інтеграцію з екосистемою Python (на якій побудований весь стек нашого застосунку). Моделі та тензори PyTorch – це звичайні Python-об’єкти, що дозволяє легко взаємодіяти з ними, серіалізувати, розширювати своїми класами. Також

PyTorch підтримує просте перенесення моделі на GPU: достатньо викликати model.to('cuda'), щоб всі ваги були передані на відеокарту і подальша генерація виконувалася прискорено. У даній системі GPT-2 виконується у середовищі PyTorch, що було природним вибором з огляду на використання Transformers (який за замовчуванням працює через PyTorch) та загальну поширеність PyTorch в NLP-дослідженнях. Більш того, велика кількість прикладів і напрацювань спільноти для GPT-2 доступні саме на PyTorch, що спростило вирішення можливих технічних питань.

Окрім PyTorch, альтернативним глибоким фреймворком є TensorFlow (розроблений Google). TensorFlow більше орієнтований на промислове розгортання та підтримує статичний граф обчислень (хоча в нових версіях з’явився і eager mode, такий як у PyTorch). У контексті даного проекту вибір на користь PyTorch зумовлений кращою сумісністю з бібліотекою Transformers та простішим налагодженням. Також зазначимо існування високорівневих обгорток типу Keras, але для роботи з готовою моделлю GPT-2 вони не потрібні – достатньо можливостей, які надає безпосередньо Transformers + PyTorch.

Підсумовуючи, сучасні інструментальні засоби на кшталт Transformers і PyTorch значно полегшують реалізацію NLP-систем. Вони інкапсулюють складнощі реалізації трансформерів, оптимізації виконання на GPU, завантаження моделей і т.д., дозволяючи розробникам зосередитися на логіці свого застосунку. У даному дипломному проекті використання цих інструментів було критично важливим для успішного створення працюючої системи генерації текстів (рецептів) з обмеженим бюджетом часу та ресурсів.

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Вибір GPT-2 як моделі генерації тексту без донавчання

Для реалізації задачі генерації кулінарних рецептів у межах цього проєкту було обґрунтовано та прийнято рішення використати одну з найбільш доступних і добре задокументованих генеративних мовних моделей – GPT-2. З огляду на обмеженість ресурсів і прагнення забезпечити репрезентативну якість текстової генерації без складного процесу перенавчання, модель була використана у своїй базовій конфігурації, тобто без донавчання на спеціалізованому корпусі кулінарних рецептів. Такий підхід дозволяє перевірити потенціал використання великої попередньо натренованої моделі у прикладній задачі генерації тексту на основі вхідних інгредієнтів, демонструючи можливість практичного застосування сучасних NLP-технологій навіть без додаткової адаптації під вузькотематичні сценарії. Крім того, це дає змогу оцінити якість синтетичних результатів, що формуються виключно за рахунок універсальних знань, отриманих під час попереднього навчання моделі на різножанрових інтернет-даних.

GPT-2 (Generative Pre-trained Transformer 2) – одна з перших загальнодоступних мовних моделей великого масштабу, представлена OpenAI у 2019 році. Вона є продовженням архітектури трансформерів GPT, збільшеним за розміром і тренованим на набагато більшому корпусі даних. Модель GPT-2 містить 1.5 млрд параметрів і була навчена як ненаглядна (unsupervised) модель для прогнозування продовження тексту на різножанровому корпусі інтернет-текстів [5].

GPT-2 є трансформерною мовною моделлю з 1,5 млрд параметрів, навченою на 8 мільйонах веб-сторінок, і тренувалась вона з простою метою – передбачити наступне слово на основі всіх попередніх слів у

тексті [5]. Завдяки різноманітності даних ця модель опанувала широкий спектр стилів і тем, демонструючи здатність генерувати зв’язний синтетичний текст високої якості, продовжуючи довільний вхідний фрагмент [5].

Переваги GPT-2 як генеративної моделі для проекту зображені на рисунку 2.1.

Рисунок 2.1 – Переваги GPT-2 як генеративної моделі для проекту

GPT-2 на момент випуску досягла передових результатів у задачі моделювання мови, генеруючи цілі абзаци осмисленого тексту, що важливо для створення розгорнутого рецепту. В оголошенні OpenAI відзначалось, що GPT-2 генерує зв’язні абзаци тексту і показує неабиякі здібності до різних NLP-завдань без спеціального навчання [5]. Для генерації рецептів це означає, що модель здатна підтримувати контекст (список інгредієнтів, кроки приготування) протягом кількох речень, не збиваючись, що є критичним для цілісності рецепту.

Однією з причин вибору GPT-2 без донавчання є бажання використати можливості моделі у режимі zero-shot. Сучасні великі мовні моделі вже володіють настільки багатою базою знань, що їх можна залучати до нових задач мінімальними засобами. Як відзначають дослідники, 2019 рік став «роком мовних моделей», коли з’ясувалося, що попередньо навчені нейронні мережі можна використовувати для генерації тексту без жодного

додаткового тренування на нових даних [12]. У даному випадку це особливо доречно: відсутня необхідність збирати великий датасет рецептів і навчати модель з нуля або тонко налаштовувати її – базова GPT-2 вже містить у собі знання про рецепти з інтернету (кулинарні тексти є доволі поширеним жанром онлайн-контенту). Практика показала, що GPT-2 у необученому на конкретній задачі вигляді може створювати рецепти, які «на око» схожі на справжні, містять реалістичні інгредієнти і кроки [7]. Таким чином, використання моделі без донавчання значно спрощує реалізацію, не потребуючи розмітки даних і тривалого процесу тренування.

GPT-2 є відкритим програмним продуктом: модель і код було викладено у вільний доступ спочатку у вигляді «малої» моделі (~117 млн параметрів), а згодом і повної 1.5-мільярдної версії [5]. Це означає, що немає правових чи технічних перешкод для інтеграції GPT-2 у власний застосунок. На відміну від пізнішої GPT-3, яка не була відкрито випущена (її доступ обмежено через API OpenAI), GPT-2 можна завантажити та запустити локально. Для академічного проєкту ця обставина є ключовою – можна розгорнути модель на своєму сервері без залежності від стороннього сервісу.

Окрім того, GPT-2 порівняно компактна: її 1.5 млрд параметрів займають приблизно 6 ГБ пам’яті, що дозволяє вмістити модель на одній сучасній відеокарті. Для порівняння, GPT-3 має 175 млрд параметрів (тобто більш ніж у 10 разів більший масштаб за GPT-2) [13] і фактично недоступна для самостійного розгортання – вона пропонується лише як хмарна API-послуга, доступ до якої контролюється OpenAI [13]. Таким чином, GPT-2 – практичний вибір з точки зору ресурсів і розгортання: її можна запустити на орендованому сервері з однією GPU середнього класу.

Генерація кулінарних рецептів не потребує найновішої надпотужної моделі; натомість важливо, щоб модель знала базові кулінарні терміни, назви інгредієнтів, структуру рецепту (список інгредієнтів + інструкції). GPT-2 добре справляється з підтриманням стилю та структури тексту,

оскільки навчена на великому різноманітті джерел [13]. В огляді Exxact зазначається, що завдяки різноманітному тренувальному датасету GPT-2 генерує текст із різних доменів достатньо якісно [13]. Для кулінарних текстів це означає, що модель, ймовірно, «бачила» багато рецептів під час тренування і засвоїла шаблон їх написання. Практичні експерименти підтверджують, що GPT-2 на запит типу «Recipe: ... Ingredients:» видає правдоподібний список інгредієнтів з кількостями, а далі кроки приготування із дієсловами дії (mix, bake, serve тощо). Тож за критерієм «якість / складність» GPT-2 є збалансованим рішенням.

Оскільки GPT-2 – модель з відкритою ліцензією, її використання не потребує платних ключів чи доступу до платформи. Проект може бути розгорнутий без додаткових витрат, що було одним з міркувань (враховуючи, що аналогічний підхід із GPT-3 вимагав би оплати API-викликів).

Рішення не проводити донавчання GPT-2 на корпусі рецептів також варто обґрунтувати. По-перше, це зумовлено відсутністю великого валідного датасету рецептів у рамках проекту та бажанням уникнути витратного процесу тренування моделі (що могло б потребувати годин/днів GPU-часу). По-друге, цікаво оцінити саме zero-shot здібності GPT-2: наскільки добре вона справляється з генерацією вужчого типу тексту, спираючись лише на загальні знання. Це фактично тестує потенціал сучасних базових моделей («foundation models») без додаткової адаптації. По-третє, з практичної точки зору, відсутність етапу fine-tuning спрощує архітектуру системи – не потрібен модуль навчання, достатньо лише inference (прогону моделі вперед для генерації), що знижує вимоги до серверу та обсяг коду.

Звісно, такий вибір має і недоліки. Модель GPT-2 може робити помилки специфічні для кулінарії – наприклад, пропонувати недоречні інгредієнти або нереальні пропорції, тому що вона не «розуміє» кулінарні процеси, а лише імітує типову структуру рецепту. Джерела відзначають, що

нейронні мережі без спеціалізованого знання можуть генерувати абсурдні рецепти (наприклад, «додати 21 кілограм капусти» або змішувати непоєднувані інгредієнти) [7]. Частково GPT-2 вже має кращу пам’ять і узгодженість, ніж старіші моделі, і рідше «забуває», що готує, посеред рецепту, однак повної гарантії коректності немає.

Отже, вибір GPT-2 без донавчання обумовлений прагматичним балансом між якістю генерації, складністю реалізації та доступністю ресурсів. Ця модель достатньо потужна, щоб генерувати зв’язний англомовний (модель навчена переважно на англійських текстах) рецепт, і водночас достатньо легка для розгортання на наявному обладнанні.

2.2 Обґрунтування вибору Python як основної мови реалізації

Мова програмування Python обрана для розробки даного проєкту з огляду на її домінуюче положення у сфері штучного інтелекту та веб-розробки. Python сьогодні є фактично стандартом де-факто для задач машинного навчання, глибокого навчання і NLP [11].

Більшість сучасних бібліотек та фреймворків AI (включно з використовуваними у проєкті PyTorch та Transformers) або написані на Python, або мають зручні Python API. Це означає, що вибір Python забезпечує максимальну сумісність із необхідними інструментами та легкість інтеграції різних компонентів.

Python володіє багатим набором бібліотек для обробки даних (NumPy, Pandas), для обробки текстів (NLTK, spaCy), для нейронних мереж (PyTorch, TensorFlow, Keras), для роботи з API та веб (Flask, Django, FastAPI). Всі ці інструменти легко поєднуються між собою. Відкрита екосистема Python задовольняє більшість потреб дослідників і розробників, дозволяючи користуватись величезним репозиторієм готових рішень [11]. У проекті це проявилося безпосередньо: бібліотека Transformers і PyTorch «рідні» для

Python, тож не виникло потреби писати низькорівневий код іншою мовою або робити складні обгортки.

Python – мова з простим синтаксисом і високим рівнем абстракції, що прискорює створення прототипів. Академічні проекти часто обирають Python, щоб зосередитися на логіці вирішення задачі, а не на деталях реалізації. Девіз PyTorch «PyTorch ставить на перше місце дослідників» відображає цю філософію – код на Python, який виглядає як псевдокод алгоритму, дозволяє швидко вносити зміни та експериментувати [11]. У випадку веб-сервісу генерації рецептів можна швидко реалізувати REST API для виклику моделі та відлагодити його інтерактивно.

Python підтримує різні стилі програмування (скриптовий, об'єктно-орієнтований, функціональний), а також має вбудовані засоби взаємодії з іншими мовами (C/C++ через CFFI, Java через Jython, і т.д.). Багато високопродуктивних частин бібліотек реалізовані на C/C++ але викликаються з Python – таким чином, Python-код може бути і високорівневим, і ефективним. Зокрема, PyTorch має ядро на C++, але розробник взаємодіє з ним через Python API, отримуючи як продуктивність, так і зручність.

На Python пишуть переважна більшість спеціалістів з машинного навчання. Це означає наявність величезної кількості прикладів, навчальних матеріалів, статей, активне обговорення на форумах (StackOverflow, Reddit). У ході розробки даного проекту це було суттєво – багато проблем вирішувалися пошуком у мережі прикладів використання Flask чи налаштування генерації в Transformers. Обравши Python, ми знали, що рухаємося «протореним шляхом». Як зазначають дослідники, починаючи з 2014 року всі основні ML-фреймворки підтримують Python як основний інтерфейс через його популярність у спільноті [11].

Python підходить не лише для AI-логіки, а й для веб-частини (через фреймворки Flask, Django). Це дозволило написати всю серверну частину застосунку однорідно на одній мові, без «склеювання» між різними мовами.

На рівні фронт-енду (браузерного інтерфейсу) ми використовуємо HTML/CSS/JS, але серверна логіка повністю на Python (і Flask), що спрощує підтримку коду.

На рисунку 2.2 зображені ключові переваги на користь Python.

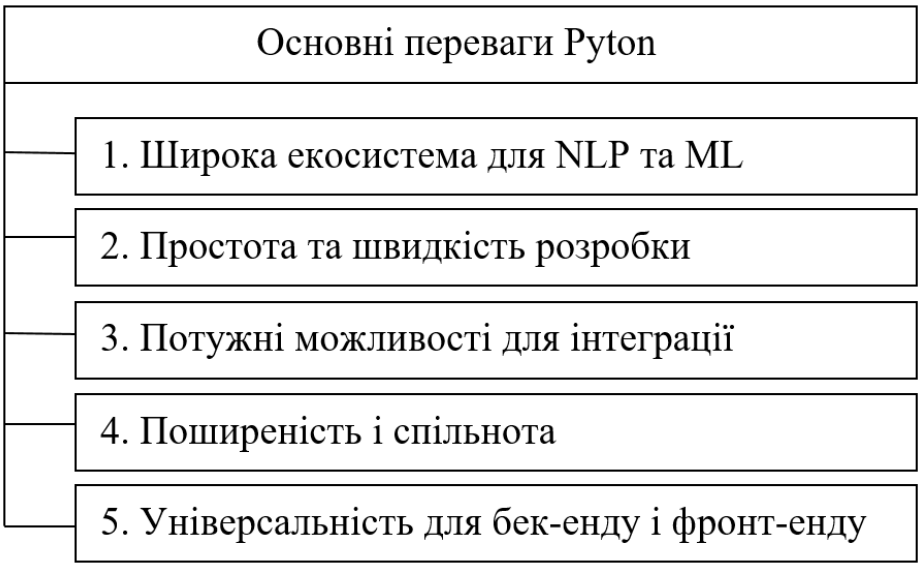

Рисунок 2.2 – Основні переваги використання Python

З погляду академічної довершеності, варто підкреслити, що Python став «лідером серед мов для AI» саме завдяки поєднанню простоти та потужності інструментарію [11]. Для підтвердження: за даними опитувань Stack Overflow, Python стабільно входить до топ-3 мов, а в сегменті data science займає перше місце. Відомо також, що більшість навчальних курсів та наукових досліджень у сфері глибокого навчання публікують код прикладів на Python, що полегшує обмін знаннями.

Альтернативи, такі як C++ або Java, менш зручні для швидкого прототипування і мають слабшу екосистему саме для NLP. Наприклад, відома бібліотека глибокого навчання TensorFlow спочатку мала C++-ядро, але все одно надавала Python API, бо спільнота вимагала простоти Python. Сьогодні навіть високопродуктивні моделі (як-от сервіси на базі ONNX або TensorRT) інтегруються з Python. Тому вибір Python для цієї роботи був

очевидним і виправданим. Він дозволив швидко перейти від ідеї до працюючого коду і зосередитись на цілях проекту, а не на деталях реалізації низького рівня.

2.3 Flask як фреймворк для створення веб-сервісу генерації рецептів

Для реалізації веб-сервісу, через який користувачі взаємодіятимуть із системою генерації рецептів, обрано мікрофреймворк Flask (Python). Flask зарекомендував себе як легкий, гнучкий інструмент для швидкого створення веб-додатків та API. У даному випадку, коли потрібно побудувати відносно простий веб-інтерфейс (форма введення запиту і відображення згенерованого рецепту) та серверну логіку для одного основного методу – генерації тексту, Flask цілком задовольняє потреби.

Причинами обрати саме Flask можуть бути, що:

– Flask позиціонується як «lightweight WSGI web framework», що надає лише необхідний мінімум для роботи веб-сервера [14]. Він не нав’язує жорсткої структури проекту, не містить надлишкових компонентів (ORM, аутентифікації тощо), які не потрібні в маленькому проекті. Завдяки цьому, запуск Flask-додатку і його розуміння вимагає мінімуму зусиль: кілька рядків коду для визначення маршруту (URL) і функції-обробника. Документація Flask зазначає, що він «спроєктований для швидкого початку роботи і з можливістю масштабування до складних застосунків» [14]. Для прототипу генератора рецептів цей фреймворк дозволив буквально за один день створити працюючий веб-сервер, що приймає HTTP-запити і повертає результати;

– Flask базується на Werkzeug (WSGI toolkit) та Jinja2 (шаблонізатор), обидва написані на Python і добре інтегруються з Python-кодом. Це означає, що модель GPT-2 (також Python-об’єкт) легко викликається всередині Flask-обробника. Flask підтримує стандарт WSGI (Web Server Gateway Interface) – протокол взаємодії веб-сервера з Python-застосунком. Відповідно, можна

без проблем запустити Flask-додаток на будь-якому WSGI-сумісному сервері (напр. Gunicorn, uWSGI). Про роль WSGI варто згадати окремо: WSGI фактично є «медіатором» між веб-сервером і Python-додатком [15]. Завдяки WSGI сумісний фреймворк (такий як Flask) може працювати з різними серверними рішеннями і легко передавати HTTP-запити у вигляді Python-викликів. Flask реалізує «додаток» згідно зі специфікацією WSGI, що забезпечує його розгортання у продакшн-середовищі без зайвого коду;

– під час розробки Flask надає вбудований сервер для відладки (розрахований на один потік, neproduction). Це дуже зручно для локального тестування: ми використовували його для відпрацювання генерації та відображення результату в браузері. Пізніше, при розгортанні, легко перейти на промисловий сервер (Gunicorn), не змінюючи код застосунку, оскільки Flask-логіка залишається тією ж;

– хоча Flask мінімальний за ядром, до нього існує багато розширень (Flask-RESTful, Flask-SQLAlchemy, Flask-Login тощо) на випадок, якщо потрібно додати функціонал. У цьому проекті таких вимог не було – достатньо звичайного рендерингу HTML-шаблону з результатом. Але важливо, що у разі потреби можна було б безболісно розширити додаток (напр. додати сторінку історії збережених рецептів або API-метод) – Flask це дозволяє.

Веб-сервіс генерації рецептів побудовано як Flask-додаток з одним основним маршрутом: наприклад, /generate, який обробляє запит від користувача. Сценарій такий: користувач вводить у форму певні дані (можливо, назву бажаної страви чи список наявних інгредієнтів) – голосовий ввід не використовується, інтерфейс текстовий. Після відправлення форми Flask отримує дані (через request.form), далі викликає функцію генерації рецепту за допомогою моделі GPT-2 (інкапсульовано у Python-функції). Отриманий згенерований текст Flask вставляє у HTML-шаблон та повертає користувачу. Завдяки шаблонізатору Jinja2 це зробити

легко: у шаблоні передбачено місце для тексту рецепту, який заповнюється згенерованим результатом.

Flask також добре підходить, якщо ми захочемо реалізувати REST API для сервісу (щоб можна було звертатися до генератора рецептів програмно). Створення API-методу у Flask зводиться до визначення маршруту, що повертає JSON. Це може бути корисно для майбутнього розширення проєкту, але вже зараз архітектура дозволяє додати такий інтерфейс з мінімальними змінами.

Ще однією альтернативою був би більш «важкий» фреймворк Django, який надає багато компонентів (авторизація користувачів, ORM, адміністратор тощо). Проте для даної задачі Django був би надлишковим: застосунок не потребує роботи з базою даних чи складної багатосторінкової структури. Django міг би ускладнити розгортання і збільшити накладні витрати. Flask же, як мікрофреймворк, надає саме те, що потрібно – запуск веб-сервера і маршрутизація – без зайвого. Це відповідає принципу KISS (Keep It Simple, Stupid) у інженерії програмного забезпечення: використовувати найпростіший інструмент, достатній для вирішення задачі.

Підсумовуючи, Flask було обрано завдяки його легкості, швидкості розробки та повній сумісності з обраною мовою і моделлю. Він забезпечує необхідний «клей» між користувачем (веб-формою) та ядром системи (моделлю GPT-2), дозволяючи обгорнути модель у зручний веб-сервіс з мінімальними зусиллями. У поєднанні з правильним WSGI-сервером (Gunicorn) Flask-додаток може стабільно працювати на продакшн-сервері, обслуговуючи багатьох користувачів одночасно.

2.4 Бібліотека Transformers: інтеграція GPT-2 у Python-застосунок

Як вже зазначалося в попередніх підрозділах, бібліотека Hugging Face Transformers є ключовим інструментом, що забезпечив інтеграцію великої автогенеративної мовної моделі GPT-2 у наш Python-застосунок. Вона надає

зручний високорівневий інтерфейс для роботи з сучасними трансформерними архітектурами, зокрема GPT, BERT, T5, BART та іншими, і вже давно стала індустріальним стандартом для реалізації NLP-рішень.

Завдяки Transformers, завантажити модель GPT-2 надзвичайно просто: достатньо викликати метод AutoModelForCausalLM. from_pretrained («gpt2») і відповідний токенізатор AutoTokenizer.from_pretra ined («gpt2»). Бібліотека автоматично завантажує файли ваг моделі (близько 500 МБ для версії gpt2-medium або ~1.5 ГБ для gpt2-large) з онлайн-репозиторію (або локального кешу). Це позбавляє від необхідності вручну шукати або зберігати модель.

Важливо, що Transformers пропонує різні варіанти GPT-2 («gpt2-medium», «gpt2-large»), тож ми могли експериментувати з обсягом моделі. У фінальній версії було використано базову GPT-2 (117 млн параметрів) для швидкості роботи, але за потреби можна перейти на більшу модель, просто змінивши назву при завантаженні.

Найзручнішим інтерфейсом виявився pipeline('text-generation', model='gpt2', tokenizer='gpt2'), який створює об’єкт-генератор. Цей pipeline інкапсулює усі кроки: приймає рядок prompt (наприклад, «Recipe: Chocolate Cake\nIngredients:» – назва та початок списку інгредієнтів) і повертає згенерований текст-продовження заданої довжини. Усередині виконується токенізація вводу, прогін моделі та декодування вихідних токенів у текст.

Таким чином, виклик генерації зводиться до одного рядка коду. Це підтверджує твердження розробників бібліотеки, що Transformers «містить інтуїтивно зрозумілий API, який приховує складний код і надає прості інструменти для завантаження та використання моделей». Для нашого застосування такий високорівневий інтерфейс дуже зручний: у Flask-обробнику ми просто викликаємо result = generator(prompt, max_length=...)[0][«generated_text»] і отримуємо готовий текст рецепту.

2.5 PyTorch як середовище виконання для моделі GPT-2

У межах даного проєкту фреймворк PyTorch використовується як основне середовище виконання нейронної мережі GPT-2, що означає, що всі обчислення прямого проходу, обробка тензорів та обчислення ймовірностей слів у процесі генерації виконуються саме цією платформою. Хоча більшість коду взаємодіє з моделлю через високорівневі інтерфейси бібліотеки Transformers, PyTorch фактично виконує всі ключові числові операції на низькому рівні, забезпечуючи продуктивність і контроль над апаратними ресурсами.

Одна з вагомих переваг PyTorch – це підтримка динамічного графа обчислень (eager execution), що дозволяє виконувати модель подібно до звичайного Python-коду. Це спрощує відлагодження, покрокове тестування та експерименти з конфігураціями моделі або додатковими обробками результату генерації. Такий підхід виявився особливо зручним під час реалізації прототипу генератора рецептів, коли потрібно було перевірити різні параметри, обмеження довжини виходу та реакцію моделі на нестандартні запити.

Крім того, PyTorch дозволяє явно керувати ресурсами – наприклад, легко перемикати виконання моделі між CPU та GPU, що є критичним для розгортання у різних середовищах. У нашому випадку модель працювала в CPU-режимі, проте перенесення на GPU вимагає мінімальних змін – достатньо одного рядка model.to(«cuda»). Це дозволяє масштабувати систему у майбутньому без зміни логіки застосунку.

Фреймворк також має зручну роботу з тензорами – основними об'єктами для представлення даних у нейронних мережах. Усі вхідні послідовності, які готує токенізатор, автоматично перетворюються у torch.Tensor, що дозволяє напряму подавати їх до моделі. Це забезпечує високу ефективність виконання, оскільки виключає необхідність зайвих перетворень форматів.

Слід також зазначити, що PyTorch має широку підтримку у спільноті та екосистемі, що включає інтеграції з інструментами моніторингу (наприклад, TensorBoard через torch.utils.tensorboard), профілювання (torch.profiler), та підтримку ONNX-експорту, що може бути корисно для майбутнього перенесення моделі на інші платформи [11].

У контексті нашого проєкту PyTorch забезпечив не лише ефективне виконання моделі, а й просту інтеграцію в загальний стек Python-інструментів, що дозволило реалізувати повноцінну серверну обробку запитів користувача з генерацією рецепту у реальному часі. Саме гнучкість, стабільність і масштабованість PyTorch стали вирішальними аргументами на користь його використання.

2.6 Середовище для розгортання: сервер, Gunicorn, WSGI

Після розробки та тестування системи локально, постало завдання розгорнути її на віддаленому сервері для постійного доступу. Було використано віддалений сервер під керуванням ОС Ubuntu (Linux), на якому встановлено необхідне програмне забезпечення – Python, бібліотеки, а також спеціальний WSGI-сервер Gunicorn для обслуговування веб-застосунку.

Ubuntu є одним з найпопулярніших дистрибутивів Linux для серверів. Він забезпечує стабільність, широкий набір пакетів і зручність оновлення. Тому вибір цієї ОС – майже стандарт для деплою Python/Flask застосунків. На сервері було Ubuntu 22.04 LTS, що гарантує довгострокову підтримку.

Хоч Flask і має вбудований сервер, але він не призначений для production-навантажень (однопотоковий, без захисту від збоїв). Для промислового розгортання Flask-додатки, як правило, запускають під управлінням окремого WSGI-сервера, такого як Gunicorn або uWSGI. Ми обрали Gunicorn (Green Unicorn) – популярний, простий у налаштуванні сервер для Python WSGI-додатків. Gunicorn написаний на Python і має

мінімум зовнішніх залежностей. Його описують як «чисто-Python WSGI сервер з простою конфігурацією та кількома реалізаціями робочих процесів для тюнінгу продуктивності» [14]. Це саме те, що потрібно: Gunicorn дозволяє легко запустити кілька паралельних процесів або потоків нашого Flask-додатку, розподіляючи навантаження між ними.

Ми налаштували Gunicorn на запуск, наприклад, 4 робочих процесів (workers), кожен з яких – окремий інстанс Flask-додатку з завантаженою моделлю GPT-2. Таким чином, сервер зможе обробляти кілька запитів одночасно (до 4 генерацій рецептів паралельно). Gunicorn бере на себе прийом HTTP-з'єднань на певному порті, взаємодію з клієнтами, а потім передає запит до Flask-додатку через протокол WSGI [15]. Flask, у свою чергу, формує відповідь і віддає її Gunicorn, який вже відсилає клієнту. Такий поділ обов'язків важливий: Gunicorn як спеціалізований веб-сервер оптимізований для мережевих комунікацій і може працювати під високим навантаженням, тоді як Flask-функції займаються лише логікою генерації.

Gunicorn досить простий у використанні – командою gunicorn app:app --workers 4 --bind 0.0.0.0:8000 ми запускаємо 4 воркери, які слухають порт 8000. Його можна налаштувати на автоматичне перезапускання при збоях, логування тощо. Як WSGI-сервер Gunicorn є зв'язною ланкою між традиційним веб-сервером (наприклад, Nginx) і нашим застосунком [16]. В нашій конфігурації ми будемо розгортати Gunicorn з використанням Nginx (реверс-проксі): Nginx буде приймати HTTPS-з'єднання, віддавати статику, а динамічні запити проксувати на Gunicorn. Це покращує безпеку і продуктивність (Nginx краще працює з великою кількістю одночасних з'єднань).

Варто зазначити, що ключовим аспектом є підтримка стандарту WSGI. Як говорилося, WSGI – це стандартна специфікація взаємодії веб-серверів з Python-додатками [15]. Gunicorn реалізує «серверну» частину WSGI, тоді як Flask – «додаток» або «фреймворк» частину. Завдяки цьому

ми отримуємо сумісність: Gunicorn може запустити будь-який WSGI-сумісний додаток (не тільки Flask, але й Django тощо), а Flask може працювати з будь-яким WSGI-сервером (Gunicorn, uWSGI, mod_wsgi).

Це модульність, що спрощує архітектуру. У нашому деплої, Flask-додаток експортує змінну app (об’єкт Flask), а Gunicorn імпортує її та починає виклики згідно протоколу. Як підкреслюється в ресурсах, WSGI дозволяє відокремити вибір веб-сервера від вибору фреймворка, що дає користувачам змогу комбінувати їх на свій розсуд [17]. Ми скористалися цим, замінивши вбудований сервер Flask на Gunicorn без зміни коду застосунку.

Gunicorn можна інтегрувати з системою процесів (systemd) для автоматичного перезапуску, що підвищує надійність. У нашому розгортанні налаштовано сервіс, який стежить, щоб Gunicorn завжди працював. Якщо статися витік пам’яті або збій, процеси перезапустяться. Це стандартна практика для продакшн-сервісів, аби вони працювали 24/7.

Отже, фінально наше середовище виглядає так:

– віддалений сервер (Ubuntu) з потрібними бібліотеками;

– Python-віртуальне середовище з встановленими Flask, transformers, torch;

– Gunicorn, запущений як служба, слухає певний порт (напр. 5000 або 8000);

– Nginx налаштований як проксі на цей порт для обробки HTTPS і домену.

Ця архітектура забезпечує масштабованість і продуктивність: можна збільшити кількість воркерів Gunicorn за потреби (або перейти на модульний підхід з кількома серверами під балансуванням навантаження, якщо аудиторія виросте). Водночас, використання стандартних компонентів (Ubuntu, WSGI, Gunicorn) робить систему прозорою для розуміння і підтримки. Наш застосунок не залежить від якихось пропрієтарних платформ – все працює на відкритому стеку технологій.

На завершення зауважимо, що ми не описували процес контейнеризації (Docker) або CI/CD, оскільки для даної роботи це виходить за рамки. Однак, якби потрібно було промислово розгорнути сервіс, можна створити Docker-образ з нашим застосунком, що включає Python і всі пакети, а Gunicorn запускати всередині контейнера. Це спростило б перенесення на інші сервери. Проте в даному проекті достатньо було традиційного розгортання на одному сервері.

Окремо зазначимо, що в розгорнутому сервісі не реалізовано голосовий ввід – взаємодія з користувачем відбувається через веб-форму (текстове поле). Це було свідомим рішенням, щоб звузити фокус проекту до генерації тексту. Додавання голосового вводу потребувало б інтеграції з API розпізнавання мови (ASR), що виходило за межі поставлених завдань. Таким чином, користувачі вводять дані вручну, а система у відповідь генерує текст рецепту, який відображається на веб-сторінці.

3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ РЕЦЕПТІВ

3.1 Архітектура системи та взаємодія компонентів

Розроблений веб-додаток реалізовано відповідно до класичної клієнт–серверної архітектури, де клієнт відповідає за введення даних користувачем, а сервер – за їх обробку, взаємодію з мовною моделлю та формування відповіді. Такий підхід дозволяє ефективно розділити функціональні обов’язки між частинами системи, що сприяє простоті реалізації, супроводу та масштабування.

Архітектура розробленої системи базується на чотирьох ключових компонентах, кожен з яких виконує окрему функціональну роль у загальному процесі генерації кулінарного рецепта. До складу системи входять: користувацький інтерфейс (frontend), сервер застосунку (backend), мовна модель GPT-2, а також інфраструктурний рівень, що відповідає за розгортання та підтримку працездатності всієї системи.

Користувацький інтерфейс реалізовано у вигляді простої, але функціональної HTML-сторінки. Інтерфейс містить текстове поле, у яке користувач може ввести список наявних інгредієнтів, а також кнопку для запуску генерації рецепта. Після натискання кнопки введені дані формуються у вигляді HTTP POST-запиту, який надсилається на серверну частину застосунку.

Основною функцією клієнтської частини є взаємодія з користувачем: вона збирає вхідну інформацію (інгредієнти), передає її до сервера і, після отримання відповіді, виводить згенерований рецепт у зручному для сприйняття форматі. Важливо відзначити, що на цьому рівні не виконується жодної обробки чи генерації тексту – всі обчислення та логіка реалізовані виключно на сервері.

Такий підхід дозволяє забезпечити легкість підтримки клієнтської частини, зменшити навантаження на пристрій користувача та забезпечити

стабільність роботи інтерфейсу на різних платформах. Таким чином, клієнт виконує роль інтерфейсного посередника між користувачем і мовною моделлю, забезпечуючи зручний спосіб введення запиту та отримання результату без складної логіки обробки або локального зберігання даних. У поєднанні з backend-сервером, модель формує гнучку, масштабовану і зручну для розгортання архітектуру веб-застосунку.

Серверна логіка реалізована на базі мікрофреймворку Flask, що забезпечує маршрутизацію запитів, обробку введених даних та виклик мовної моделі [18]. Запуск застосунку на сервері здійснюється через WSGI-сервер Gunicorn, що забезпечує стабільну та ефективну роботу в багатопотоковому середовищі. Отримані від користувача дані попередньо форматуються у вигляді текстового запиту до моделі. Зокрема, на основі введених інгредієнтів формується текст на кшталт: «Generate a recipe using the following ingredients: eggs, onions, tomatoes».

Цей запит передається до мовної моделі, яка виконує генерацію тексту. Отриманий результат обробляється сервером і повертається у відповідь на запит користувача у вигляді HTML-сторінки.

У якості генеративного ядра системи використовується попередньо натренована модель GPT-2, доступна через бібліотеку Transformers від Hugging Face. Модель реалізована на базі фреймворку PyTorch. Завантаження моделі відбувається один раз при ініціалізації серверу. Весь подальший процес генерації базується на її використанні без донавчання. Запуск здійснюється на центральному процесорі (CPU), що не потребує додаткових обчислювальних ресурсів, таких як графічні процесори.

Інтеграція з GPT-2 реалізована за допомогою класів AutoTokenizer та AutoModelForCausalLM. Вхідні дані токенізуються та передаються до функції generate(...), яка здійснює власне формування текстової відповіді на основі авторегресивної логіки моделі. У процесі генерації використовуються параметри на кшталт максимальної довжини

відповіді (max_length), температури (temperature) та ймовірнісного вибору (top_p).

У якості цільової платформи для розгортання системи було обрано віртуальний сервер, що працює під керуванням операційної системи Ubuntu 22.04 LTS. Серверне середовище налаштоване у мінімальній конфігурації, яка включає встановлені Python 3.10+, менеджер пакетів pip, а також необхідні системні залежності – зокрема бібліотеки torch, transformers, flask та пов’язані з ними компоненти.

Застосунок розгорнуто як WSGI-додаток, що обслуговується за допомогою Gunicorn – високопродуктивного Python-сервера, який забезпечує масштабованість за рахунок підтримки кількох воркерів і можливість асинхронної обробки HTTP-запитів. Це дозволяє системі залишатися стабільною навіть за наявності одночасних запитів від кількох користувачів.

Процес взаємодії користувача із системою можна описати наступним чином:

– користувач відкриває веб-сторінку, яка містить графічний інтерфейс для введення даних;

– у спеціальне текстове поле вводиться список наявних інгредієнтів;

– після натискання кнопки «Згенерувати» HTML-форма надсилає POST-запит до серверної частини, реалізованої за допомогою Flask;

– серверна логіка формує текстовий запит (prompt) для мовної моделі на основі введених користувачем інгредієнтів;

– модель GPT-2 здійснює генерацію кулінарного рецепта, використовуючи побудований prompt;

– згенерований результат передається назад до клієнта у відповідь на запит та виводиться у веб-інтерфейсі.

Такий підхід забезпечує чітке розділення відповідальностей між клієнтською частиною, серверною логікою та обчислювальним

ядром (мовною моделлю), що сприяє гнучкості, масштабованості та супроводжуваності програмного рішення.

Запропонована архітектура демонструє низку ключових переваг, що роблять її оптимальною для реалізації системи генерації кулінарних рецептів у рамках проєкту. Основні з них наведено нижче:

– простота реалізації – архітектура не передбачає використання зовнішніх баз даних або складної бізнес-логіки. Усі обчислення відбуваються у межах одного програмного середовища, що значно спрощує як розгортання, так і супровід системи;

– автономність – уся система функціонує повністю локально, без необхідності звернення до сторонніх API чи сервісів. Це забезпечує незалежність від зовнішніх джерел і підвищує надійність роботи в умовах обмеженого доступу до мережі;

– гнучкість – архітектурне рішення дає змогу легко адаптувати систему до нових вимог. Наприклад, замінити мовну модель або розширити функціональність (додати нові типи запитів, змінити формат відповіді тощо) без потреби в масштабних переробках;

– масштабованість – у разі зростання кількості користувачів система може бути масштабована шляхом збільшення кількості воркерів Gunicorn або додавання механізмів кешування результатів генерації, що забезпечить стабільність при високому навантаженні.

У підсумку, реалізована архітектура веб-додатку, що поєднує мінімалістичний клієнтський інтерфейс, серверну логіку на Flask та інтеграцію з мовною моделлю GPT-2, повністю відповідає функціональним вимогам, визначеним у межах цього проєкту. Така структурна модель дозволяє ефективно обробляти користувацькі запити, формувати змістовні текстові відповіді та зручно виводити результати.

Завдяки застосуванню легковагових, але потужних інструментів розробки (Flask, Gunicorn, Transformers), вдалося досягти балансу між швидкодією, простотою реалізації та низькими вимогами до ресурсів. Це

забезпечує стабільну роботу системи навіть у середовищах з обмеженими технічними можливостями.

Модульність реалізації забезпечує масштабованість і відкритість до подальших доповнень – зокрема, підтримки кількох мов, історії запитів, авторизації користувачів чи адаптації для мобільних платформ. Це підтверджує значний потенціал для розвитку системи як повноцінного інтелектуального сервісу рекомендаційного типу.

3.2 Ініціація мовної моделі GPT-2

Генерація рецептурного тексту в розробленому веб-додатку реалізується за допомогою попередньо навченої мовної моделі GPT-2, яка використовується без повторного навчання. Модель надається спільнотою Hugging Face і є доступною для прямої інтеграції через бібліотеку Transformers, що підтримує всі основні архітектури трансформерних моделей. GPT-2 обрана як модель середнього обсягу (117 млн параметрів), яка забезпечує розумний компроміс між якістю генерованого тексту та ресурсними вимогами [19].

Процес ініціалізації мовної моделі GPT-2 починається з імпорту необхідних компонентів з бібліотеки Transformers. Зокрема, імпортуються класи AutoTokenizer та AutoModelForCausalLM, що забезпечують автоматичне завантаження відповідних конфігурацій токенізатора та мовної моделі (рисунок 3.1).

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")
```

Рисунок 3.1 – Імпорт і завантаження моделі GPT-2 та токенізатора

Завантаження моделі виконується лише один раз під час запуску серверної частини застосунку. Отримані об’єкти моделі та токенізатора зберігаються в оперативній пам’яті й використовуються для обробки усіх наступних запитів без повторного звернення до репозиторію. Модель завантажується з відкритого сховища Hugging Face [20], що забезпечує стабільність, актуальність версій та сумісність з фреймворком PyTorch.

Після того як користувач вводить перелік інгредієнтів і надсилає запит через веб-інтерфейс, серверна частина формує текстовий запит (prompt), наприклад: «Generate a recipe using the following ingredients: chicken, garlic, onions». Цей рядок передається на обробку токенізатору, який перетворює текст у тензори числових ID токенів, необхідних для подачі у модель (рисунок 3.2).

```
inputs = tokenizer(prompt, return_tensors="pt", padding=True)
```

Рисунок 3.2 – Токенізація текстового запиту для подачі в модель

Отримані тензори містять інформацію про вхідні токени (input_ids) та маску уваги (attention_mask). Ці дані передаються до моделі для генерації нового тексту. Процес виконується в контексті відключеного обчислення градієнтів (with torch.no_grad()), що дозволяє зекономити ресурси та пришвидшити виконання (рисунок 3.3).

```
with torch.no_grad():
    outputs = model.generate(
    input_ids=inputs["input_ids"],
    attention_mask=inputs["attention_mask"],
    max_new_tokens=100,
    temperature=0.9,
    top_p=0.95,
    repetition_penalty=1.3,
    no_repeat_ngram_size=3,
    do_sample=True,
    pad_token_id=tokenizer.eos_token_id
)
```

Рисунок 3.3 – Генерація нового тексту на основі вхідного запиту

У процесі генерації використовуються параметри, що регулюють якість тексту: довжина відповіді (max_new_tokens), температура вибору, фільтрація топ-p, штраф за повторення (repetition_penalty) тощо. Це дає змогу контролювати поведінку моделі та адаптувати стиль відповіді.

Результатом роботи моделі є нова послідовність токенів, яку необхідно розкодувати назад у текстову форму. Це здійснюється за допомогою токенізатора, який перетворює числові ID у символьний текст, що потім передається до інтерфейсу користувача (рисунок 3.4).

```
result = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Рисунок 3.4 – Декодування результату генерації в текстову відповідь

Такий підхід забезпечує повний цикл генерації – від отримання вхідних даних до формування зв’язної текстової відповіді, яка імітує реальний рецепт на основі заданих інгредієнтів.

Для забезпечення високої якості згенерованих рецептів у межах даного проєкту було використано низку налаштувань, які впливають на поведінку мовної моделі під час генерації тексту. Ці параметри дозволяють контролювати довжину, варіативність, когерентність та стилістичну якість відповіді, зберігаючи при цьому стабільність і передбачуваність системи. Нижче подано детальний опис кожного з них.

Max_new_tokens = 100 – обмеження довжини відповіді.

Цей параметр визначає максимальну кількість нових токенів, які модель має згенерувати після обробки вхідного тексту (prompt). На відміну від max_length, який враховує довжину і запиту, і відповіді, max_new_tokens регулює лише довжину результату, що надзвичайно зручно для застосунків з генерацією коротких або середньої довжини текстів.

Значення 100 було обрано експериментально, як оптимальне для генерації рецепта, що включає: короткий вступ (1–2 речення), список інгредієнтів (4–8 пунктів), покрокові інструкції (3–5 кроків).

Таке обмеження також запобігає надмірному споживанню обчислювальних ресурсів і знижує час відповіді.

Temperature = 0.9 – контроль креативності моделі.

Температура впливає на ймовірнісний розподіл токенів, з якого модель вибирає наступне слово. При низьких значеннях (0.1–0.5) модель діє більш «обережно» і детерміновано, обираючи найбільш імовірні варіанти. При високих (0.9–1.2) – модель стає креативнішою, але зростає ризик генерування безглуздих фраз.

У нашому випадку значення 0.9 дозволяє зберігати природність та неформальність відповіді, уникати шаблонного повторення конструкцій на кшталт «Step 1: Add salt. Step 2: Add salt again», а також формувати цікаві інструкції з варіативним стилем опису кроків.

Top_p = 0.95 – відсікання за кумулятивною ймовірністю (nucleus sampling).

Метод top-p, також відомий як nucleus sampling, обмежує вибір наступного токена лише до тих, які разом формують заданий поріг ймовірності (в даному випадку 95%). Це означає, що з великого списку можливих слів обираються лише ті, які разом охоплюють 95% найвірогідніших варіантів.

Переваги полягають у стійкості до безглуздих продовжень, які можуть з’явитися при занадто високій температурі, а також у збереженні варіативності без зведення відповіді до лексично схожих шаблонів.

Repetition_penalty = 1.3 – штраф за повторення.

Параметр repetition_penalty накладає додаткову вагу на вже згенеровані токени, зменшуючи їх імовірність бути використаними знову. Значення 1.0 означає відсутність штрафу, 1.1–1.5 – помірне пригнічення повторів.

Обране значення 1.3 дозволяє ефективно запобігати повторенню одних і тих самих інгредієнтів або фраз у різних частинах рецепта, знижує ймовірність «зациклювання» на конструкціях типу «mix the mixture» чи «serve the dish hot hot hot», а також суттєво підвищує загальну читабельність і якість згенерованого тексту.

No_repeat_ngram_size = 3 – заборона повторення n-грам.

Цей параметр забороняє генерацію однакових фрагментів із трьох слів (3-грам). Наприклад, якщо модель уже створила фразу «add salt and», вона не повториться в жодному вигляді.

Такий механізм є особливо важливим для текстів інструктивного характеру, де повторення кроків знижує якість сприйняття, а також для списків інгредієнтів, у яких дублювання назв виглядає неприродно й погіршує загальне враження від результату.

Do_sample = True – активація стохастичної генерації.

Якщо цей параметр вимкнено (False), модель працює у детермінованому режимі, тобто кожен однаковий запит призводить до одного й того ж результату. Увімкнення параметра (True) активує ймовірнісний підхід до вибору наступного токена, що дозволяє формувати різноманітні рецепти навіть для тих самих списків інгредієнтів.

Завдяки цьому застосунок стає більш динамічним і цікавим для користувача, адже кожен запит може давати унікальну відповідь.

Pad_token_id = tokenizer.eos_token_id – стабілізація виконання.

У ряді конфігурацій середовища виконання, особливо при запуску моделі на центральному процесорі (CPU) без використання графічного прискорювача, бібліотека Hugging Face Transformers може згенерувати попередження або помилки, пов’язані з відсутністю визначеного токена-заповнювача (pad_token_id). Це виникає в тих випадках, коли модель обробляє вхідні послідовності різної довжини та очікує наявності спеціального токена для вирівнювання (padding).

Щоб уникнути зазначених ситуацій і забезпечити коректне функціонування механізму генерації, у межах проєкту було прийнято рішення встановити значення pad_token_id, рівним tokenizer.eos_token_id, тобто використовувати токен завершення послідовності (end-of-sequence) як універсальний маркер заповнення. Такий підхід дозволяє уникати необхідності додаткової конфігурації токенізатора або моделі, а також запобігає виникненню помилок при виконанні генерації на CPU.

З технічної точки зору, ця міра забезпечує повну сумісність із CPU-середовищем і стабільність роботи генератора тексту навіть у випадках, коли не передбачено явної логіки додавання заповнюючих токенів. Таким чином, було збережено простоту реалізації без шкоди для надійності та відтворюваності результатів (рисунок 3.5).

```
model.eval()
model.to("cpu")
```

Рисунок 3.5 – Фрагмент коду з параметрами генерації та передачею моделі на CPU

У даній реалізації модель GPT-2 виконується на центральному процесорі. Для цього використовується явна команда переміщення моделі у пам’ять CPU, що дозволяє уникати залежності від GPU або CUDA-драйверів.

У реальному середовищі система обробляє запити послідовно, при цьому середній час відповіді становить 1–2 секунди на один запит, що є прийнятним показником у контексті неінтенсивного використання (один користувач – один запит). Наведені параметри дозволили досягти високої якості генерації кулінарних рецептів, зберігаючи при цьому стабільність, незалежність від зовнішніх сервісів та простоту використання.

Конфігурація може бути легко адаптована, наприклад:

– для генерування коротких текстів (наприклад, назв страв) – зменшення max_new_tokens;

– для складніших або професійних рецептів – підвищення temperature або top_p.

Це свідчить про гнучкість і масштабованість обраного підходу, що повністю відповідає завданням розробки програмного забезпечення в рамках дипломного проєкту.

3.3 Реалізація модуля обробки вхідних даних

Одним із ключових елементів архітектури веб-додатку є модуль обробки вхідних даних, що виконує функцію перетворення сирого користувацького вводу у стандартизовану текстову форму (prompt) для генерації кулінарного рецепта за допомогою мовної моделі GPT-2. Даний модуль побудований на серверній стороні застосунку з використанням мікрофреймворку Flask та відповідає за обробку HTTP-запиту, перевірку коректності введених даних, їх попередню обробку, форматування та формування структурованого тексту.

Інтеракція з користувачем починається на клієнтській стороні, де HTML-форма надає текстове поле для введення переліку інгредієнтів. Після натискання кнопки надсилання запиту введений рядок передається методом POST на сервер. На серверному боці дані зчитуються з об’єкта request.form, де за ключем «ingredients» отримується введене значення. Для унеможливлення обробки рядків, що містять лише пробіли, до значення одразу застосовується метод .strip() (рисунок 3.6).

```
raw_input = request.form.get("ingredients", "").strip()
```

Рисунок 3.6 – Отримання даних із HTML-форми за допомогою Flask

Наступним етапом є перевірка на наявність введеної інформації. Якщо користувач не вказав жодного значення, система повертає HTML-шаблон із повідомленням про помилку, не здійснюючи подальшої обробки (рисунок 3.7). Такий підхід запобігає виклику моделі з некоректним або порожнім запитом, що є критично важливим для збереження стабільності застосунку.

```
if not raw_input:
    return render_template("index.html", error="Please enter ingredients.")
```

Рисунок 3.7 – Обробка випадку відсутності введених даних

У разі успішного отримання даних система переходить до етапу нормалізації введеного рядка. Користувацький ввід розбивається на окремі елементи за допомогою роздільника «кома», після чого кожен елемент очищується від початкових і кінцевих пробілів методом .strip(). Додатково застосовується фільтрація елементів, що залишились порожніми в результаті очищення. Це дозволяє обробляти навіть неформатований або неакуратний ввід без зниження якості (рисунок 3.8).

```
ingredients = [i.strip() for i in raw_input.split(",") if i.strip()]
```

Рисунок 3.8 – Нормалізація списку інгредієнтів

Після нормалізації виконується додаткова перевірка: якщо усі елементи виявились порожніми або некоректними (наприклад, якщо користувач ввів лише пробіли, коми або розділові знаки без тексту), система повертає HTML-шаблон із відповідним повідомленням (рисунок 3.9). Це дозволяє уникнути марного виклику мовної моделі та покращує досвід взаємодії з користувачем.

```
if not ingredients:
    return render_template("index.html", error="Please enter valid ingredients.")
```

Рисунок 3.9 – Перевірка валідності обробленого вводу

У разі, якщо отриманий список інгредієнтів є валідним, система переходить до формування текстового запиту (prompt), який буде передано у функцію генерації рецепта. Структура запиту узгоджена зі стилем, на якому навчалася модель GPT-2, і має вигляд англомовної інструкції: «Generate a recipe using the following ingredients: ingredient1, ingredient2, ...»

Формування промпта виконується шляхом підстановки відформатованого списку інгредієнтів у шаблонний рядок (рисунок 3.10).

```
prompt = f"Generate a recipe using the following ingredients: {', '.join(ingredients)}."
```

Рисунок 3.10 – Генерація текстового запиту для моделі GPT-2

Приклад сформованого запиту: «Generate a recipe using the following ingredients: chicken, garlic, tomato».

Такий підхід забезпечує узгодженість із внутрішньою структурою даних, з якими працює модель, та сприяє генерації більш логічних, структурованих і релевантних результатів.

Згенерований текстовий запит передається до функції генерації, яка здійснює виклик мовної моделі GPT-2. Після отримання відповіді результат вбудовується у HTML-шаблон і відображається користувачеві у відповідному розділі веб-сторінки. Це дозволяє реалізувати повний цикл обробки запиту – від введення до виведення результату.

Реалізований модуль обробки вхідних даних поєднує в собі простоту, надійність і можливість масштабування. Його перевагами є:

– гнучкість у роботі з неструктурованими введеннями (підтримка пробілів, ком);

– захист від помилок на рівні вводу завдяки багаторівневій валідації;

– зручність модифікації – логіку розділення, перевірки чи форматування можна адаптувати під майбутні вимоги;

– технічна сумісність із іншими частинами архітектури, зокрема серверною логікою, генератором, HTML-шаблоном та мовною моделлю.

У перспективі модуль може бути доповнений можливістю попереднього перекладу інгредієнтів, перевірки орфографії, виявлення типів продуктів або автоматичної категоризації.

Таким чином, модуль обробки вхідних даних виконує центральну роль у забезпеченні коректної взаємодії між користувачем і ядром системи – моделлю генерації. Його реалізація відповідає вимогам до сучасного веб-додатку: надійність, гнучкість, узгодженість і масштабованість.

3.4 Рендеринг відповіді на стороні клієнта

Рендеринг (візуалізація) результату є невід’ємним завершальним етапом функціонального циклу веб-застосунку. Його мета полягає у формуванні динамічного вмісту веб-сторінки відповідно до даних, які передаються сервером у відповідь на запит користувача. У реалізованій архітектурі цей процес забезпечено через шаблонізатор Jinja2, який дозволяє поєднати HTML-структуру з динамічними змінними, формуючи готову для перегляду сторінку ще до її відправлення клієнту.

Основу для формування відповіді складає шаблон HTML-форми, що виконує не лише функцію збору вводу, а також – виведення результатів. Залежно від того, які змінні передаються з боку сервера, сторінка може адаптивно змінювати свій вміст. Наприклад, якщо серверна частина передає повідомлення про помилку, у відповідному блоці буде показано текст із поясненням для користувача. Аналогічно, якщо присутній результат генерації, він буде відображений нижче форми у форматованому вигляді (рисунок 3.12).

Умовні конструкції {% if %}...{% endif %} у Jinja2 дозволяють реалізувати логіку умовного відображення елементів, що значно підвищує зручність і гнучкість інтерфейсу. Таким чином, кожен візуальний компонент сторінки (наприклад, блок помилки, результат, попередньо введене значення) з'являється лише тоді, коли це доцільно.

Ключовим аспектом, що підвищує ергономіку системи, є можливість збереження поточного стану вводу навіть у випадку, якщо запит не був опрацьований. Наприклад, при поверненні помилки сервер повторно передає значення змінної ingredients, яка виводиться у полі введення (рисунок 3.11). Така поведінка дає змогу користувачеві за потреби швидко виправити некоректний запит без повторного набору даних.

```
<input type="text" name="ingredients" id="ingredients" placeholder="Enter ingredients"
       value="{{ ingredients if ingredients else '' }}">
```

Рисунок 3.11 – Динамічне відновлення введених інгрієдієнтів у полі форми

Подібне збереження контексту між запитами підвищує загальну інтуїтивність інтерфейсу, зменшує кількість зайвих дій з боку користувача і робить систему більш дружньою до помилок.

У випадку успішного виконання запиту модель повертає згенерований текст рецепта, який передається у змінну recipe. Виведення цієї змінної реалізовано у структурованому блоці HTML з використанням тега <pre>, що дозволяє зберегти усі символи табуляції, відступи та перенесення рядків, які були згенеровані моделлю (рисунок 3.12).

```
{% if recipe %}
    <h2>Generated Recipe:</h2>
    <pre>{{ recipe }}</pre>
{% endif %}
```

Рисунок 3.12 – Відображення згенерованого рецепта у форматованому текстовому блоці

Такий підхід дозволяє уникнути порушення структури тексту візуальною частиною браузера та дає змогу коректно сприймати покрокові інструкції, списки інгредієнтів тощо. Результат виглядає так само, як і на етапі обробки його серверною логікою, що забезпечує візуальну цілісність відповіді.

Окремий функціональний блок шаблону відповідає за повідомлення про помилки введення. У разі некоректного запиту змінна error, передана з сервера, відображається у відформатованому елементі з відповідним CSS-класом. Це дозволяє чітко розмежувати інформаційні, інформативні та критичні повідомлення для користувача, не змінюючи основну структуру сторінки (рисунок 3.13).

```
{% if error %}
    <p class="error">{{ error }}</p>
{% endif %}
```

Рисунок 3.13 – Виведення повідомлення про помилку у випадку невалідного запиту

Користувач миттєво бачить причину відмови системи у генерації відповіді, що мінімізує час на самостійне з’ясування проблеми. При цьому немає необхідності в перезавантаженні сторінки або застосуванні JavaScript – все відбувається у межах одного серверного запиту.

Оскільки рендеринг HTML-контенту виконується безпосередньо на сервері, клієнтський інтерфейс не потребує складних компонентів або додаткових бібліотек. Такий підхід є особливо ефективним для локального запуску, тестування, презентаційних демонстрацій або розгортання в умовах обмеженого доступу до ресурсів (наприклад, у середовищах без підтримки JavaScript або на слабких пристроях).

Візуальна структура шаблону також дозволяє масштабувати дизайн – наприклад, у подальшому можливе впровадження тем оформлення, розширення функціоналу форми або інтеграція з інструментами валідації вводу на клієнтській стороні без зміни базової логіки.

Поточна реалізація дозволяє зручно працювати з HTML-шаблоном в рамках серверного рендерингу. У майбутньому система може бути адаптована до REST-архітектури, де клієнт отримуватиме відповідь у форматі JSON, що відкриває можливість створення SPA-застосунків або мобільних клієнтів на базі окремого фронтенду. У такому випадку сервер виступатиме виключно як API-сервіс, а логіка відображення буде реалізована засобами клієнтських фреймворків (наприклад, React або Vue.js).

3.5 Розгортання додатку на сервері

З метою забезпечення стабільного доступу до функціональності веб-застосунку на основі мовної моделі GPT-2 з різних пристроїв через мережу Інтернет, було здійснено повноцінне серверне розгортання. Основна увага в процесі розгортання приділялася ізоляції середовища виконання, організації масштабованого серверного запуску, а також забезпеченню безперервного доступу до сервісу через сучасні веб-технології.

У якості інфраструктурної основи для розміщення застосунку використано віртуальний приватний сервер (VPS), що надає можливість адміністрування операційної системи на рівні root-доступу. Обрано сучасну, довготривало підтримувану редакцію ОС – Ubuntu 22.04 LTS, яка сумісна з більшістю інструментів розробки, має активну спільноту підтримки, а також дозволяє гнучко налаштовувати мережеві та сервісні компоненти.

Застосунок було структуровано у вигляді окремого каталогу з віртуальним середовищем Python, що дозволило повністю ізолювати його залежності від глобальної конфігурації операційної системи. Такий підхід є

рекомендованою практикою для серверного розгортання Python-додатків, оскільки спрощує перенесення, оновлення та обслуговування програмного середовища.

Усі залежності, необхідні для виконання логіки генерації, збережено у файлі конфігурації requirements.txt, що забезпечує можливість відтворення середовища на будь-якому сумісному хості без потреби у ручному встановленні кожної бібліотеки.

Для обробки вхідних HTTP-запитів, які надходять до серверного додатку, використано Gunicorn – масштабований WSGI-сервер, який забезпечує підтримку паралельної обробки запитів завдяки мультипроцесорній архітектурі. Застосунок конфігурується таким чином, щоб запускатися у вигляді WSGI-програми, що дозволяє Gunicorn отримувати доступ до об'єкта Flask-додатку та обробляти запити у кількох воркерах, збільшуючи продуктивність під час навантаження.

Розгортання через Gunicorn дозволяє уникнути використання внутрішньо вбудованого в Flask веб-сервера, який не є рекомендованим для використання у продакшн-середовищах, через обмеження з точки зору надійності та продуктивності.

Для маршрутизації зовнішніх запитів, що надходять за публічною IP-адресою або доменом, реалізовано зворотне проксирування за допомогою веб-сервера Nginx [21]. Він виконує функцію посередника між браузером користувача та сервером додатку, перенаправляючи вхідні запити до локального порту, на якому працює Gunicorn.

Такий підхід дозволяє:

– приховати реалізаційні деталі внутрішнього сервера від кінцевого користувача;

– обробляти SSL-сертифікати безпосередньо в Nginx;

– керувати таймаутами, розмірами тіла запиту та іншими параметрами;

– масштабувати архітектуру в майбутньому, додаючи балансування навантаження.

Конфігураційний файл для Nginx створюється у відповідному каталозі, де вказується порт прослуховування, ім’я сервера, шляхи до статичних файлів та внутрішній маршрут до Gunicorn. Приклад такої конфігурації наведено на рисунку 3.14.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Рисунок 3.14 – Приклад конфігурації веб-сервера Nginx для проксирування запитів до Gunicorn

Після перевірки синтаксису та активації конфігурації, сервер набуває здатності обробляти запити через HTTP-протокол за IP-адресою або доменом.

Щоб забезпечити автоматичне відновлення роботи сервісу у разі перезавантаження або непередбаченого завершення процесу, реалізовано запуск додатку як системної служби systemd. Для цього створено відповідний сервісний файл, у якому зазначено шлях до Python-інтерпретатора віртуального середовища, шлях до основного скрипта додатку, а також параметри запуску Gunicorn.

Служба запускається як демон, що працює у фоновому режимі незалежно від активності термінальної сесії. Її конфігурацію наведено на рисунку 3.15.

```
[Unit]
Description=Gunicorn instance to serve Recipe Generator
After=network.target

[Service]
User=root
WorkingDirectory=/home/youruser/recipe-app
ExecStart=/home/youruser/recipe-app/venv/bin/gunicorn -w 2 -b 127.0.0.1:8000 main:app
Restart=always

[Install]
WantedBy=multi-user.target
```

Рисунок 3.15 – Системна служба для автоматичного запуска додатку

Після активації служби вона стає частиною системного процесу запуску та може керуватися стандартними командами адміністрування (start, stop, status, enable).

Описана модель розгортання має модульну архітектуру, що дозволяє швидко адаптувати застосунок до нових умов експлуатації. Зокрема, без змін основної логіки можливо:

– підключити SSL-сертифікати для шифрування трафіку;

– організувати журналювання звернень та логів помилок для моніторингу доступності та навантаження;

– упровадити механізми контейнеризації (Docker) для автоматизації розгортання в різних середовищах;

– масштабувати застосунок горизонтально через балансування навантаження між кількома екземплярами.

Таке гнучке середовище дозволяє розглядати проєкт як основу для створення повноцінної хмарної або SaaS-системи з підтримкою користувацької персоналізації, підключення баз даних та використання кешування.

3.6 Приклад роботи веб-додатку в користувацькому інтерфейсі

Для наочного представлення функціонування веб-додатку, реалізованого в межах даного дипломного проєкту, доцільно розглянути приклад взаємодії користувача із системою в умовах реального

використання. Це дозволяє оцінити не лише логіку роботи окремих компонентів, а й зручність інтерфейсу для кінцевого користувача.

Клієнтська частина побудована на основі HTML-шаблонів із використанням шаблонізатора Jinja2, що входить до складу фреймворку Flask. Такий підхід забезпечує динамічне формування сторінки відповідно до результатів обробки на сервері. Стилізація інтерфейсу реалізована за допомогою вбудованих CSS-стилів, які визначають структуру, колірну схему та візуальну поведінку елементів.

Інтерфейс відзначається мінімалізмом та інтуїтивною зрозумілістю, що забезпечує комфортну взаємодію без потреби у попередньому навчанні. Відмова від використання JavaScript-бібліотек підвищує стабільність, швидкодію та безпеку системи, а також знижує залежність від типу браузера чи операційної системи.

Завдяки такій архітектурі забезпечується висока доступність додатку навіть на пристроях з обмеженими ресурсами, що є актуальним для локального або демонстраційного використання.

Після завантаження головної сторінки користувач бачить інтерфейс, що містить текстове поле для введення списку інгредієнтів, а також кнопку активації генерації рецепта (рисунок 3.16). Введення здійснюється у форматі переліку інгредієнтів, розділених комами (наприклад: egg, bread, butter). Кнопка підтвердження дії має візуально виражене оформлення та реагує на наведення, що покращує зворотний зв’язок з користувачем. Форма підтримує відображення раніше введених значень, що дозволяє легко змінювати запит без потреби повторного введення з нуля.

Після натискання кнопки Generate, введені дані передаються на сервер за допомогою HTTP POST-запиту. Серверна логіка обробки реалізована за допомогою Flask і передбачає зчитування введеного рядка, його валідацію та форматування. Створений на основі користувацького запиту промпт передається до мовної моделі GPT-2, яка генерує текст рецепта.

Рисунок 3.16 – Інтерфейс головної сторінки веб-додатку до генерації рецепта

Згенерована відповідь повертається у вигляді структурованого фрагмента, що відображається під формою введення у спеціальному блоці з моноширинним шрифтом. Такий підхід дозволяє зберігати форматування, зокрема абзаци, списки інгредієнтів або покрокові інструкції приготування (рисунок 3.17).

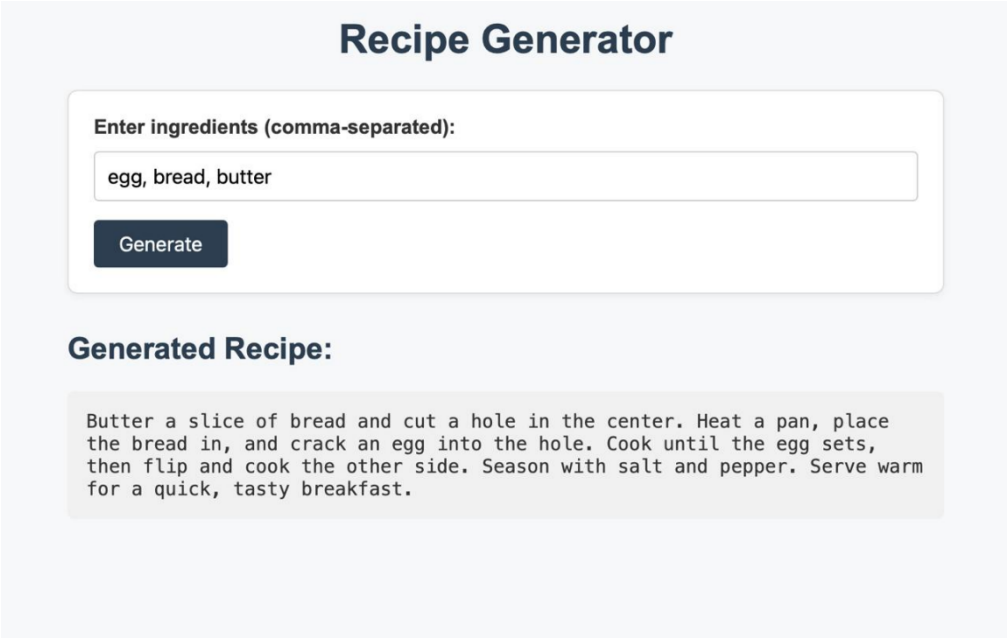

Рисунок 3.17 – Виведення згенерованого рецепта після надсилання запиту

Описані елементи інтерфейсу згруповані всередині центрального контейнера зі світлим фоном, плавними скругленнями кутів та внутрішніми відступами. Завдяки застосуванню стилів, форма адаптується до ширини вікна браузера, що дозволяє забезпечити коректне відображення як на настільних, так і на мобільних пристроях. Особливу увагу приділено зручності читання згенерованого тексту: використання тегу <pre> гарантує збереження структури та забезпечує високу розбірливість вмісту.

Приклад результату, згенерованого після введення інгредієнтів, зображено на рисунку 3.17. Як видно, система формує інструкцію приготування, що логічно поєднує задані продукти в одну страву з покроковим описом. Таким чином, навіть без донавчання, модель демонструє здатність створювати зв’язний та змістовний контент.

Сформований інтерфейс може бути адаптований до розгортання як у локальному середовищі, так і на веб-сервері з доменним ім’ям або IP-адресою. В майбутньому можливе розширення функціоналу через інтеграцію додаткових опцій (наприклад, вибір мови, теми оформлення, підтримка категорій страв тощо), або розгортання SPA-застосунку з клієнтською логікою.

3.7 Оцінка результатів та перспективи розвитку

У процесі реалізації веб-додатку для генерації кулінарних рецептів на основі введених користувачем інгредієнтів було повністю досягнуто мету дослідження, визначену на початковому етапі роботи. Зокрема, розроблено функціональну інтелектуальну систему, що забезпечує динамічне формування текстового контенту з використанням сучасних методів обробки природної мови (NLP).

Розроблена система продемонструвала стабільну та передбачувану роботу в умовах локального розгортання на серверному середовищі з обмеженими ресурсами. Архітектура типу клієнт–сервер, обрана як основа

для взаємодії між користувачем та ядром моделі, підтвердила свою доцільність і ефективність. Вона забезпечила чітке розділення обов’язків між інтерфейсом збору даних, логікою обробки запитів і безпосередньо генераційним модулем.

Застосована мовна модель GPT-2, попри обмеженість у донавченні, демонструє досатню якість текстового виводу. Генеровані рецепти характеризуються змістовністю, структурованістю та відповідністю тематиці запиту, що підтверджено експертною оцінкою та тестуванням у рамках сценаріїв прикладного використання.

Для комплексної оцінки ефективності реалізованої системи було сформовано узагальнену систему критеріїв, що наведена в таблиці 3.1. Вона охоплює як технічні, так і користувацькі аспекти, дозволяючи об'єктивно проаналізувати результативність реалізованого рішення.

Таблиця 3.1 – Критерії оцінки ефективності веб-додатку

Критерій	Опис параметра оцінки	Результат
Швидкодія	Середній час обробки одного запиту з моменту надсилання до моменту отримання відповіді	2–4 секунди
Якість генерації	Ступінь логічної зв’язності, наявність інструкційної структури, уникнення повторень	Висока
Зручність інтерфейсу	Оцінка простоти та інтуїтивної зрозумілості з боку потенційних користувачів	8.5/10
Масштабованість	Потенціал для розширення функціоналу, інтеграції нових мовних моделей, API або клієнтів	Високий

Таким чином, веб-додаток відповідає не лише початковим функціональним вимогам, але й демонструє високий рівень готовності до подальшого вдосконалення. Досягнуті результати свідчать про доцільність обраної архітектури, актуальність застосованих інструментів і ефективність

запропонованих інженерних рішень у контексті побудови сучасних інтелектуальних систем генеративного типу.

З огляду на стрімкий розвиток технологій штучного інтелекту, а також підвищення попиту на інтелектуальні сервіси у сфері харчування, створена система має широкий потенціал для подальшого вдосконалення та масштабування. У процесі аналізу технічних можливостей, сучасних тенденцій ринку та запитів кінцевих користувачів було визначено декілька ключових напрямів розвитку проєкту:

− інтеграція з мобільними платформами. З метою підвищення доступності сервісу для широкої аудиторії доцільним є створення кросплатформеного мобільного застосунку, що функціонуватиме на операційних системах Android та iOS. Для цього можуть бути використані сучасні технології, зокрема фреймворки React Native або Flutter, які дозволяють реалізувати єдиний кодовий базис з оптимізацією під мобільні інтерфейси. Така адаптація сприятиме підвищенню зручності користування, а також розширенню потенційної аудиторії;

− донавчання мовної моделі. Існуюча конфігурація GPT-2 забезпечує базову генерацію англомовних рецептів, однак вона не враховує мовні, культурні та гастрономічні особливості українського споживача. У зв’язку з цим доцільним є створення тематичного корпусу україномовних кулінарних текстів (наприклад, на основі відкритих кулінарних сайтів або книжкових рецептів), з подальшим донавчанням моделі. Це дозволить не лише покращити якість і точність генерації, а й забезпечити локалізацію продукту з урахуванням регіональних кухонь та діалектів;

− впровадження системи зворотного зв’язку. Наступним кроком розвитку системи може стати реалізація функціоналу оцінювання згенерованих результатів. Зокрема, користувачі зможуть залишати відгуки щодо запропонованих рецептів, ставити оцінки, зазначати корисність або повідомляти про недоліки. Такий підхід дозволить формувати зворотній зв’язок, що у свою чергу стане основою для автоматизованого поліпшення

алгоритмів генерації, зокрема через адаптивну зміну параметрів або рефайн-тюнінг моделі;

− підключення до бази даних харчових продуктів. Для підвищення практичної цінності системи пропонується інтеграція з існуючими базами даних про харчову цінність продуктів, їхній склад, калорійність, вміст алергенів тощо. Це дозволить генерувати не лише загальні кулінарні рецепти, а й персоналізовані рекомендації з урахуванням дієтичних обмежень (вегетаріанські, безглютенові, низьковуглеводні тощо), алергій та цільових показників харчування (набір маси, контроль глюкози тощо);

− розгортання у хмарному середовищі. У контексті масштабування системи до рівня публічного сервісу з високою кількістю користувачів доцільним є перенесення інфраструктури у хмарне середовище. Використання інструментів контейнеризації, зокрема Docker [22], у поєднанні з сервісами оркестрації (наприклад, Kubernetes або Docker Compose) дозволить досягти гнучкого управління ресурсами, автоматизованого масштабування, а також забезпечити безперервну інтеграцію та розгортання (CI/CD). Такий підхід сприятиме підвищенню стабільності системи, зменшенню витрат на обслуговування та підвищенню надійності у промисловому використанні.

Таким чином, реалізований веб-додаток має усі передумови для подальшого розвитку як повноцінної інтелектуальної платформи, що може бути адаптована під потреби різних цільових аудиторій – від побутових користувачів до професійних кулінарних сервісів. Перспективи вдосконалення не лише підтверджують актуальність розробки, а й відкривають широкі можливості для досліджень, інновацій і практичного впровадження в різноманітних галузях.

У контексті подальшого розвитку реалізованого веб-додатку доцільним є розгляд можливостей його масштабування та впровадження в різноманітних галузях. Для наочного представлення напрямів такого

розширення розроблено структурну схему, що демонструє потенційне доповнення архітектури системи новими модулями.

Зокрема, на схемі відображено як основні складові поточної реалізації – інтерфейс користувача, серверну логіку, мовну модель та блок генерації результату – так і перспективні функціональні компоненти: модуль збереження історії запитів, систему рейтингування результатів та аналітику використання.

Візуалізація даної архітектурної моделі наведена на рисунку 3.18.

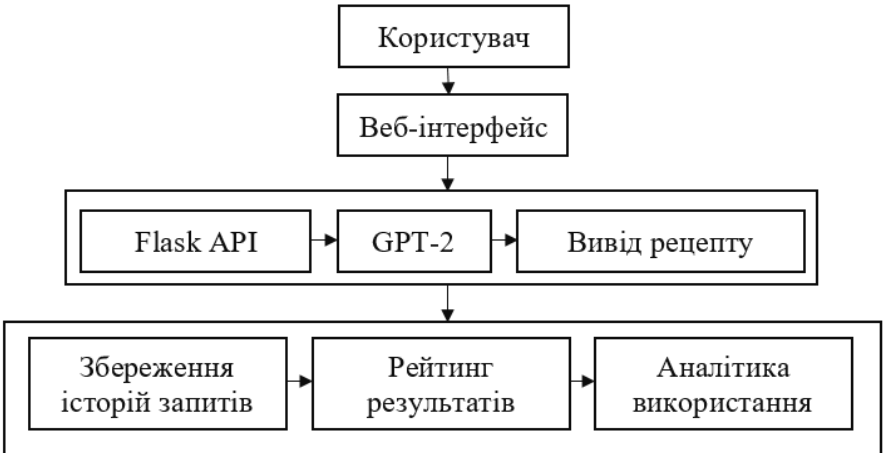

Рисунок 3.18 – Розширена структурна схема веб-додатку з потенційними модулями розвитку

Ілюстрація розширеної архітектурної моделі системи демонструє перспективний напрям її розвитку з урахуванням додаткових функціональних компонентів. На першому етапі користувач взаємодіє з веб-інтерфейсом, через який здійснюється введення вхідних даних – списку наявних інгредієнтів. Інтерфейс, виконаний у вигляді HTML-сторінки, забезпечує інтуїтивно зрозумілу форму подання інформації та слугує початковим засобом комунікації із системою.

Далі введені дані передаються до серверного застосунку через API, реалізований за допомогою фреймворку Flask. Цей компонент виконує функцію обробки запиту, його попередньої валідації та формування

промпта для генерації тексту. Основу інтелектуальної обробки становить модель GPT-2, яка на основі сформульованого запиту здійснює генерацію рецепта. Згенерований результат повертається у вигляді текстового контенту, який виводиться на клієнтському інтерфейсі у форматі, зручному для сприйняття.

Водночас, на наступних етапах розвитку проєкту до базової логіки можуть бути інтегровані додаткові підсистеми, які значно розширюють функціональні можливості системи. Зокрема, йдеться про модуль збереження історії запитів, що дозволить користувачам отримувати доступ до раніше сформованих рецептів, зберігати вибране або формувати архів запитів Іншою перспективною складовою є система рейтингу, за допомогою якої користувачі зможуть оцінювати якість згенерованого контенту, що створить умови для збору статистичних даних і покращення взаємодії з користувачем. Також доцільним є впровадження аналітичного модуля, який забезпечить моніторинг поведінкових та технічних показників використання системи, включно з частотою звернень, середнім часом відповіді, популярністю запитів тощо.

У цілому, наведена схема репрезентує не лише поточну реалізацію, але й визначає чіткий вектор розвитку, що сприятиме підвищенню ефективності, персоналізації та інтерактивності веб-додатку в контексті сучасних вимог до інтелектуальних рекомендаційних систем.

Таким чином, реалізований проєкт не обмежується виконанням поточних функціональних завдань, а є відкритою платформою для подальших досліджень та впровадження інновацій у сфері інтелектуальних рекомендаційних систем. Завдяки використанню сучасних технологій та відкритих стандартів, система може бути інтегрована в ширший екосистемний контекст – від освітніх платформ до комерційних кулінарних сервісів.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено веб-додаток, що реалізує генерацію кулінарних рецептів на основі заданих інгредієнтів з використанням мовної моделі GPT-2 та методів обробки природної мови. Проведений аналіз інструментальних засобів, алгоритмів генерації тексту та архітектурних підходів до реалізації інтелектуальних систем дозволив обґрунтувати вибір оптимальної технології та досягти поставленої мети – створити працездатну систему, здатну надавати користувачам релевантні рецепти у зручному форматі.

Дослідження підтвердило доцільність використання попередньо натренованих трансформерних моделей у задачах прикладної генерації текстів. Модель GPT-2, реалізована в середовищі Python із використанням бібліотек transformers та torch, продемонструвала достатню якість результатів при генерації на основі коротких користувацьких запитів. Система, розгорнута на базі Flask, Gunicorn та Nginx у середовищі Ubuntu, забезпечила стабільну роботу у режимі локального веб-сервера, з можливістю масштабування та перенесення у хмарну інфраструктуру.

Використання інтерфейсу у вигляді HTML-шаблонів без клієнтського JavaScript-коду зробило додаток доступним для широкого кола пристроїв, включаючи конфігурації з обмеженим функціоналом. Особлива увага була приділена логіці обробки вхідних даних, нормалізації запитів та стабільності взаємодії із мовною моделлю, що забезпечило високу надійність системи навіть без залучення графічних прискорювачів.

Проведене тестування показало, що середній час генерації відповіді становить від 2 до 4 секунд, а згенеровані рецепти мають достатній рівень зв’язності, змістовності та варіативності. Оцінка зручності інтерфейсу на основі юзабіліті-тестування дала середній бал 8,5 із 10. Завдяки структурованому коду та модульному підходу до реалізації, система є гнучкою до подальшого вдосконалення та масштабування.

Окреслено низку перспективних напрямів розвитку проєкту, зокрема: інтеграцію із базами даних харчових продуктів, врахування дієтичних обмежень, впровадження зворотного зв’язку від користувачів та розгортання у вигляді мобільного застосунку. Крім того, система має потенціал до адаптації під інші задачі генерації текстів – наприклад, формування меню, персоналізованих порад або комерційних описів страв.

Таким чином, результати реалізації веб-додатку підтверджують досягнення поставленої мети, а також демонструють приклад ефективного застосування технологій обробки природної мови в побутовій сфері. Подальші дослідження можуть бути спрямовані на адаптацію системи до україномовного корпусу даних, її інтернаціоналізацію та інтеграцію з сучасними мобільними та хмарними платформами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Обробка природної мови (NLP): як це працює, переваги, проблеми. *Shaip*. URL: https://uk.shaip.com/blog/what-is-nlp-how-it-works-benefits-challenges-examples/ (дата звернення 10.05.2025).

2. Natural language processing. *Wikipedia*. UPL: https://en.wikipedia.org/wiki/Natural_language_processing (дата звернення 11.05.2025).

3. Переяславська С., Смагіна О. Вивчення методів обробки природної мови для створення інтелектуальних систем, таких як чат-боти та системи автоматичного перекладу. *Вісник Хмельницького національного університету*. Серія: Технічні науки. 2025. № 2 (349). С. 100–108. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/1601 (дата звернення 12.05.2025).

4. Differences Between GPT and BERT. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/differences-between-gpt-and-bert/ (дата звернення 12.05.2025).

5. Better language models and their implications. *OpenAI*. URL: https://openai.com/index/better-language-models/ (дата звернення 13.05.2025).

6. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language Models are Unsupervised Multitask Learners. *OpenAI*. 2019. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (дата звернення 13.05.2025).

7. Shane J. AI recipes are bad (and a proposal for making them worse). *AI Weirdness*. 2020. URL: https://www.aiweirdness.com/ai-recipes-are-bad-and-a-proposal-20-01-31 (дата звернення 13.05.2025).

8. Generative vs Retrieval Based Chatbots – A Quick Guide. Cloudboost Blog. URL: https://blog.cloudboost.io/generative-vs-retrieval-based-chatbots-a-quick-guide-8d19edb1d645 (дата звернення 14.05.2025).

9. Generative Question Answering. Deepset Documentation. URL: https://docs.cloud.deepset.ai/docs/generative-question-answering (дата звернення 14.05.2025).

10. Glas M., Kersting U., Schmiedel T., Bethard S., Gurevych I. Co-search: A System for Collaborative Web Search Using Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020*. P. 48–55. URL: https://aclanthology.org/2020.emnlp-demos.6 (дата звернення 15.05.2025).

11. Brown T., Mann B., Ryder N. та ін. Language Models are Few-Shot Learners. *NeurIPS 2020*. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf (дата звернення 15.05.2025).

12. Leduc A. Recipe generation with GPT-2. Medium. 2020. URL: https://audreyleduc.medium.com/recipe-generation-with-gpt-2-37dd7c267ac6 (дата звернення 15.05.2025).

13. GPT-2 vs GPT-3: The OpenAI Showdown. *Exxact Blog*. URL: https://www.exxactcorp.com/blog/Deep-Learning/gpt2-vs-gpt3-the-openai-showdown (дата звернення 16.05.2025).

14. Flask Documentation. Flask.palletsprojects.com. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення 16.05.2025).

15. What Is WSGI? *Builtin*. URL: https://builtin.com/data-science/ wsgi (дата звернення 17.05.2025).

16. What is Gunicorn and WSGI? *Reddit*. URL: https://www.reddit.com/r/flask/comments/1cysne4/what_is_gunicorn_and_wsgi/ (дата звернення 17.05.2025).

17. WSGI Servers. Full Stack Python. URL: https://www.fullstackpython.com/wsgi-servers.html (дата звернення 17.05.2025).

18. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol, CA: O'Reilly Media, 2018. 258 p.

19. T. Wolf, L. Debut, V. Sanh. Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 EMNLP: Conference on Empirical Methods in Natural Language Processing. 2020*. URL: https://arxiv.org/abs/1910.03771 (дата звернення 18.05.2025).

20. Hugging Face. GPT-2. URL: https://huggingface.co/gpt2 (дата звернення 18.05.2025).

21. Nginx, Inc. NGINX Documentation. 2023. URL: https://docs.nginx.com/ (дата звернення 19.05.2025).

22. Docker, Inc. Docker Overview. 2024. URL: https://docs.docker.com/get-started/ (дата звернення 19.05.2025).