

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему КОМП'ЮТЕРНА СИСТЕМА ОПИТУВАННЯ ДОВІДНИКА
ТУРИСТА НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

Виконав: студент 2 курсу, групи КНТ-622м
спеціальності

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Спеціалізовані комп'ютерні системи

МОРОЗ А.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент МАЛИЙ О.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістерський

Спеціальність 123 Комп'ютерна інженерія

(код і найменування)

Освітня програма (спеціалізація) «Спеціалізовані комп'ютерні системи»

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“ 24 ” жовтня 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

МОРОЗА Артура Романовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комп'ютерна система опитування довідника туриста

на основі штучного інтелекту

керівник проєкту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “ 24 ” жовтня 2023 року № 400

2. Строк подання студентом проєкту (роботи) 10 грудня 2023 року

3. Вихідні дані до проєкту (роботи) існуючі методи обробки даних, сервіси нейронних мереж

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

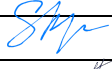
1) Аналіз технічного завдання, дослідження можливості та якості підвищення динамічності системи за допомогою інтеграції нейронних мереж;

2) Розробка структури системи;

3) Дослідження системи;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	СКРУПСЬКИЙ С.Ю., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів роботи існуючих туристичних систем	01.10.2023 р.	
2	Аналіз існуючих методів та алгоритмів роботи нейронних мереж	16.10.2023 р.	
3	Розробка структури системи	25.10.2023 р.	
4	Розробка алгоритму роботи системи	01.11.2023 р.	
5	Інтеграція нейронних мереж у проєкт	05.11.2023 р.	
6	Дослідження системи	10.11.2023 р.	
7	Оформлення отриманих результатів у ПЗ	15.11.2023 р.	
8	Оформлення графічного матеріалу	25.11.2023 р.	
9	Оформлення допоміжного матеріалу	30.11.2023 р.	

Студент  Артур МОРОЗ
(підпис) (ім'я, ПРИЗВИЩЕ)

Керівник проєкту (роботи)  Степан СКРУПСЬКИЙ
(підпис) (ім'я, ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 100 с., 30 рис., 1 табл., 43 джерел.

ДИНАМІЧНІ ЗАПИТИ, КОМП'ЮТЕРНА СИСТЕМА, НЕЙРОННІ МЕРЕЖІ, ОПТИМІЗАЦІЯ ВИТРАТ, ОПТИМІЗАЦІЯ РОБОЧОГО ЧАСУ, ШТУЧНИЙ ІНТЕЛЕКТ

Об'єкт дослідження – комп'ютерна система опитування довідника туриста на основі штучного інтелекту.

Предмет дослідження – використання штучного інтелекту для підвищення динамічності системи.

Мета роботи – проектування та аналіз комп'ютерної системи опитування довідника туриста на основі штучного інтелекту.

Матеріали, методи та технічні засоби: алгоритми обробки даних, сервіси нейронних мереж з існуючими базами знань, проектування та дослідження комп'ютерної системи опитування довідника туриста на основі штучного інтелекту. Програма дозволяє дослідити підходи використання динамічної логіки нейронних мереж замість стандартної програмованої логіки.

Результати – виведено та проаналізовано підходи використання нейронних мереж для підвищення ефективності системи. Обрано відповідні архітектуру та інструменти для реалізації додатку, та проведено аналіз з використанням розробленої комп'ютерної системи

Галузь використання – відкрите використання будь-якими користувачами.

ABSTRACT

Explanatory note to the master's work: 100 p., 30 figures, 1 table, 43 sources.

ARTIFICIAL INTELLIGENCE, COMPUTER SYSTEM, COST OPTIMIZATION, DYNAMIC QUERIES, NEURAL NETWORKS, WORK TIME OPTIMIZATION

Research Object: The study focuses on a computerized tourist guide inquiry system based on artificial intelligence.

Research Subject: The utilization of artificial intelligence to enhance the dynamism of the system.

Research Goal: To design and analyze a computerized tourist guide inquiry system leveraging artificial intelligence.

Materials, Methods, and Technical Means: Data processing algorithms, neural network services with existing knowledge bases, and the design and study of the AI-based computerized tourist guide inquiry system. The program explores the use of dynamic logic in neural networks as an alternative to standard programmed logic.

Results: The approaches of using neural networks to increase system efficiency have been derived and analyzed. Appropriate architectures and tools for the application's implementation were selected, followed by an analysis using the developed computer system.

Application Field: The system is intended for open use by all users.

ЗМІСТ

Перелік умовних позначень, скорочень, термінів	8
Вступ.....	9
1 Теоретичні відомості.....	10
1.1 Опис предметної області	10
1.2 Базові поняття нейронних мереж	12
1.2.1 Бінарна ступінчаста функція активації	17
1.2.2 Лінійна функція активації	18
1.2.3 Нелінійні функції активації.....	19
1.2.4 Функція активації сигмоїди/логістична функція (Sigmoid).....	20
1.2.5 Функція активації на основі гіперболічного тангенса (Tanh)	21
1.2.6 Функція активації виправленого лінійного елемента (ReLU).....	23
1.2.7 Функція активації нещільного виправленого лінійного елемента (Leaky ReLU).....	24
1.2.8 Нормована експоненційна функція активації (Softmax)	26
1.3 Висновки за розділом 1	28
2 Розробка системи	29
2.1 Вибір архітектури додатку	29
2.1.1 Монолітна архітектура.....	31
2.1.2 Клієнт-серверна архітектура	34
2.1.3 Мікросервісна архітектура	36
2.1.3 Висновок з вибору архітектури	38
2.2 Вибір нейронної мережі	39
2.2.1 Технічні умови для функціонування нейронної мережі	39

2.2.2 Вибір API сервісу нейронної мережі.....	42
2.2.3 Можливості використання OpenAI API у веб-системі.....	47
2.3 Вибір засобів розробки	50
2.3.1 Вибір серверного фреймворку	51
2.3.2 Вибір архітектури серверної частини	55
2.3.3 Архітектурний шаблон Porto	56
2.3.4 Вибір засобів розробки клієнтської частини.....	61
2.3.4 Вибір клієнтського фреймворку	62
2.4 Висновок з розробки системи	66
3 Дослідження системи.....	68
3.1 Актуальність впровадженого підходу.....	69
3.2 Порівняння використання нейронних мереж з традиційними підходами програмування	71
3.3 Підхід використання GPT-API.....	79
3.3.1 Надання системою відповіді користувачеві.....	79
3.3.2 Аналіз підходу використання GPT-API для надання відповіді користувачеві	84
3.3.3 Надання системою відповіді для обробки системою	89
3.3.4 Аналіз результатів підходу використання GPT-API	90
3.4 Висновки за розділом 3.....	94
Висновки	95
Перелік джерел посилання	97

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ШІ	–	Штучний інтелект;
НМ	–	Нейронна мережа;
LLM	–	Large Language Model;
AI	–	Artificial Intelligence;
API	–	Application Programming Interface
ELU	–	Exponential Linear Unit;
DOM	–	Document Object Model;
VDOM	–	Virtual Document Object Model;
JS	–	Javascript
TS	–	Typescript
JSON	–	JavaScript Object Notation

ВСТУП

Сучасний туристичний ринок зазнає значних трансформацій під впливом новітніх технологій, що неминуче впливає на структуру і функціонування туристичних систем. Головну роль у цьому процесі відіграють веб-системи, які стають все більш розповсюдженими та невід'ємними складовими туристичної індустрії. Важливість аналізу та розуміння основоположних аспектів функціонування туристичних систем, їх ключових компонентів та механізмів взаємодії визначає актуальність теми даної роботи.

Об'єктом дослідження стали саме веб-системи туристичного сектору, які є найбільш затребуваними у сучасному світі. Через широкий спектр послуг та велику кількість інформації, які вони надають, ці системи відіграють важливу роль у задоволенні потреб туристів. Однак, разом з тим, вони стикаються з рядом викликів, які потребують негайного вирішення. До таких викликів належить необхідність адаптації до змінюваних умов ринку та вимог споживачів, а також потреба в швидкому та ефективному прийнятті рішень.

Протягом проведеного аналізу буде зосереджено увагу на можливостях оптимізації веб-систем туристичного сектору за допомогою інноваційних технологій, таких як нейронні мережі. Ці технології відкривають перед туристичними системами нові можливості, дозволяючи підвищити ефективність їх роботи. У результаті аналізу були виявлені основні принципи роботи нейронних мереж, їх специфічні особливості та можливості застосування в контексті туристичної індустрії для пришвидшення швидкості створення системи, оптимізації часових витрат, підвищення динамічності системи та оптимізації експертного досвіду.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Опис предметної області

Сучасний світ несе в собі безмежні можливості для пізнання та дослідження нових країн, культур і традицій. Туризм, як важлива складова культурного та економічного обміну, набуває все більшого значення в житті сучасної людини. Подорожі стають доступнішими, а інтерес до вивчення світу зростає. У цьому контексті особливо актуальним стає питання якісної інформаційної підтримки туристів [1-2].

В сучасних умовах величезною популярністю користуються електронні засоби комунікації та інформаційні технології, які можуть бути використані для забезпечення потреб туристів. Це стосується не тільки зручних сервісів бронювання готелів чи квитків, але й інформаційних систем, здатних забезпечити туристів необхідною інформацією про об'єкти відвідування, культурні події, транспортну інфраструктуру та інше [3].

Комп'ютерна система опитування довідника туриста – це інтелектуальна інформаційна система, яка надає відповіді на запитання туристів, базуючись на базі даних з інформацією про різноманітні туристичні об'єкти, культурні заходи, інфраструктуру регіону тощо [3-6].

Основними складові комп'ютерної системи опитування довідника туриста є база даних туристичної інформації та інтерфейс для користувача [7-8]:

База даних зберігає інформацію про туристичні об'єкти, культурні заходи, інфраструктуру регіону тощо. Інформація має бути структурована та актуалізована.

Інтерфейс дозволяє користувачу задавати питання та отримувати відповіді. Інтерфейс може бути реалізований у вигляді веб-сайту, мобільного додатка або чат-бота у соціальних мережах [9-10].

Серед ключових викликів, які можуть виникати у комп'ютерних системах подібного роду, варто відзначити питання актуальності інформації. Для ефективності та корисності системи необхідно забезпечити постійне оновлення

інформації, що зберігається, включно з даними про туристичні об'єкти, події, транспортну інфраструктуру тощо. Така задача потребує значних ресурсів та зусиль з боку адміністраторів системи.

Іншим значним аспектом є обмежена можливість гнучкої взаємодії з користувачами. Система повинна мати здатність адекватно розуміти запити користувачів та надавати зрозумілі та вичерпні відповіді.

Не менш важливою є адаптивність системи до постійних змін у туристичній галузі, яка є динамічною і схильною до змін. Система має бути готовою до адаптації під нові умови, реагуючи на зміни в туристичному ринку, законодавстві, технологіях та інших аспектах, що впливають на якість обслуговування користувачів.

Проблема актуальності даних може бути вирішена за допомогою постійної ручної модифікації даних адміністраторами чи менеджерами, або ж застосуванням автоматизованих систем парсингу з місцевих веб-сайтів. Однак, вирішення питань гнучкості взаємодії та адаптивності до змін вимагає використання більш складних та нестандартних підходів.

Традиційно ці системи створені за допомогою програмованої логіки. Але такі системи можуть бути жорсткими і негнучкими, оскільки вони обмежені варіантами відповідей та взаємодій, які були задумані та програмовані заздалегідь. Це веде до того, що системи не можуть адекватно реагувати на непередбачувані відповіді або випадки, що не були заздалегідь обмежені логікою програмування.

Нейронні мережі пропонують потенційне рішення для цієї проблеми. Вони здатні вчитися на даних і адаптуватися до нових ситуацій, замість того щоб слідувати жорстко заданому набору правил. Це може зробити системи опитування електронного довідника туриста більш гнучкими і здатними краще задовольнити потреби користувачів.

1.2 Базові поняття нейронних мереж

Нейронні мережі - це один із видів систем штучного інтелекту, що використовуються для вирішення широкого спектру завдань, від розпізнавання образів і мови до прогнозування та класифікації даних. Ці системи наслідують принципи організації та функціонування біологічних нервових систем, що дозволяє їм моделювати складні взаємозв'язки в даних і здійснювати глибокий аналіз інформації [11].

Застосування нейронних мереж охоплює широкий спектр галузей, включаючи біомедицину, фінанси, промисловість, освіту та наукові дослідження. Завдяки здатності нейронних мереж адаптуватися до складної структури даних та вирішувати завдання, що вимагають аналізу великої кількості інформації, вони відіграють ключову роль у розвитку технологій штучного інтелекту [12].

Нейронні мережі використовуються для розробки інтелектуальних систем, здатних виконувати складні задачі, аналогічно людському розуму. Це може включати розпізнавання мови, аналіз тексту, визначення образів та відео, прогнозування та оптимізацію рішень. Завдяки здатності адаптуватися до різноманітних умов та навчатися на основі досвіду, нейронні мережі можуть виявляти закономірності в даних та робити обґрунтовані прогнози.

В контексті нейронних мереж, нейрон визначається як фундаментальна обчислювальна одиниця. Його дизайн заснований на принципах функціонування біологічного нейрона, проте він представлений у більш абстрактній та спрощеній формі. Структурно, нейрон приймає на вхід декілька сигналів, які подаються з інших нейронів чи зовнішніх джерел. Кожен із цих вхідних сигналів множиться на певний ваговий коефіцієнт, який відображає його релевантність для обчислювального процесу [13].

Ключовою особливістю роботи нейрону є його здатність до навчання та адаптації до нових даних. Процес навчання полягає у корекції вагових коефіцієнтів

та зміщення на основі різниці між очікуваним та фактичним виводом, що дозволяє оптимізувати роботу мережі.

Вагові коефіцієнти, які є числовими параметрами, вказують на ступінь важливості відповідного вхідного сигналу для нейрону. Під час навчання моделі, ці коефіцієнти постійно коригуються з метою оптимізації виводу нейронної мережі. Після того як всі вхідні сигнали оброблені з урахуванням їхніх ваг, їх сумарне значення підсумовується, а до отриманої суми додається додатковий параметр, відомий як "bias" або зміщення, як показано на рисунку 1.1 [14].

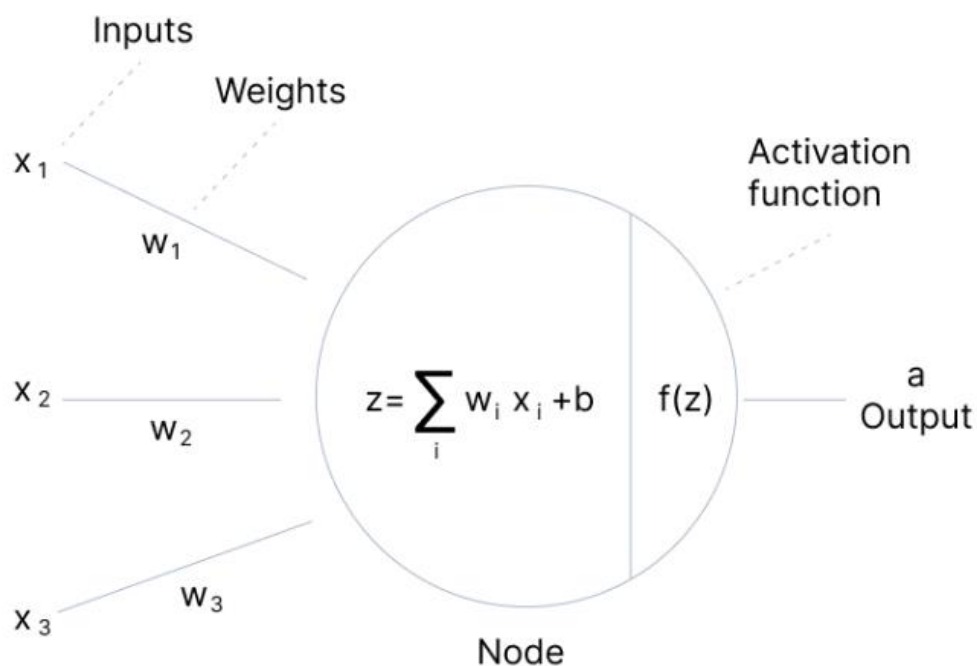


Рисунок 1.1 – Математична модель програмованого нейрону

Залежно від характеру та інтенсивності цих вхідних сигналів, мозок обробляє їх та вирішує, чи слід активувати нейрон (запустити його) чи ні.

На основі специфіки та сили вхідних сигналів мозок аналізує їх та визначає, чи має бути нейрон активованим.

У сфері глибокого навчання така ж роль призначена Функції активації. Через це її іноді називають Функцією переходу в рамках Штучних нейронних мереж.

Головне завдання Функції активації - це конвертація сумарного вхідного сигналу, зваженого вузлом, у вихідне значення, що направляється на прихований рівень або використовується як кінцевий вивід.

Структура штучної нейронної мережі - це ієрархічна організація взаємопов'язаних нейронів, що обробляють і передають інформацію. Вона включає вхідний, один або кілька прихованих та вихідний шари [15].

На рисунку 1.2 зображено структуру нейронної мережі, формованої через між'єднані нейрони. Кожний із цих нейронів характеризується власною вагою, параметром зміщення та відповідною функцією активації.

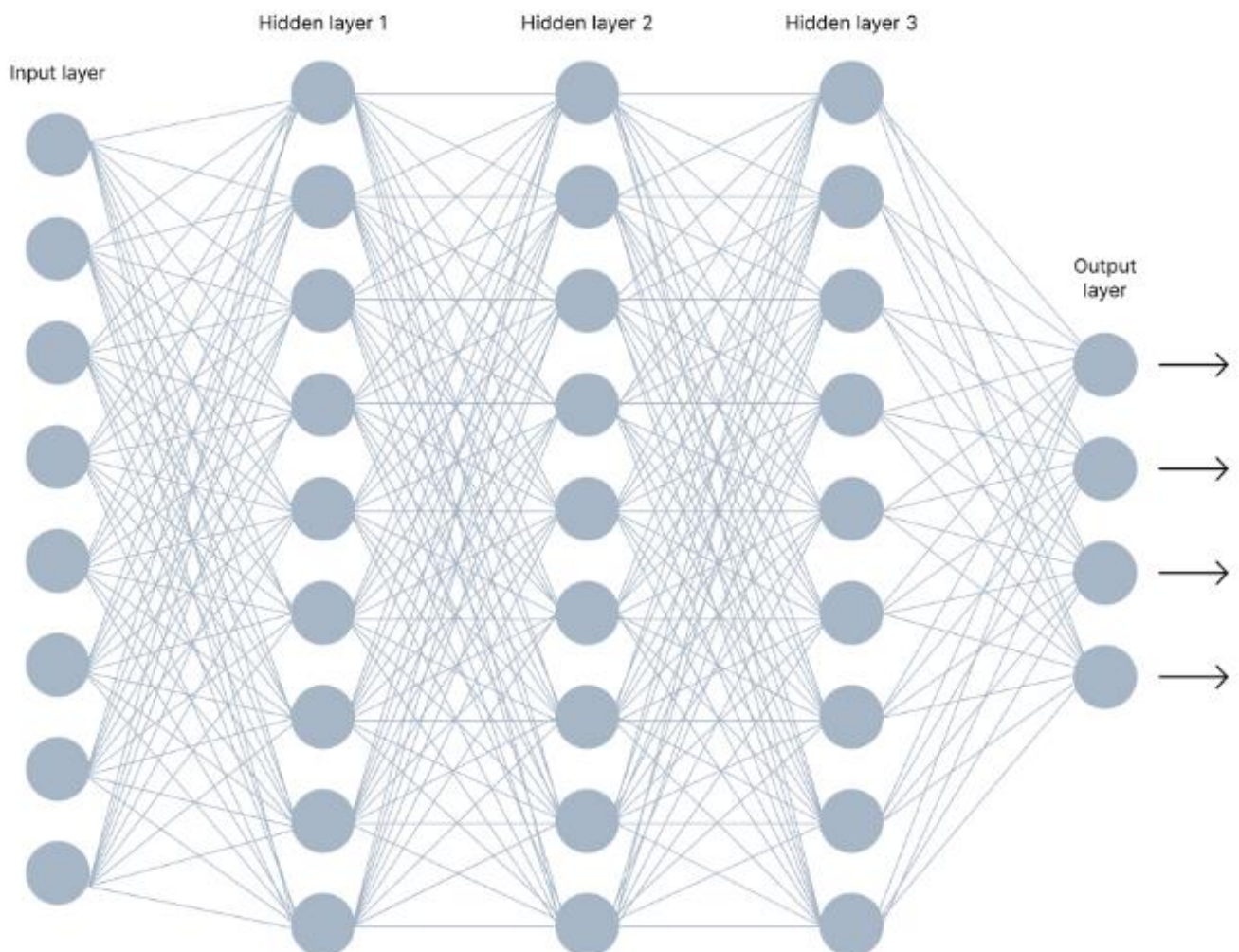


Рисунок 1.2 – Приклад структури нейронної мережі

Вхідний шар (Input Layer) - це перший рівень, який безпосередньо приймає вхідні дані або ознаки для подальшої обробки. Він складається з вузлів (або нейронів), кожен з яких відповідає за окрему вхідну ознаку чи параметр. На цьому рівні не виконуються активні обчислення або перетворення; основна функція вхідного шару полягає в передачі вхідних даних до наступних, прихованих шарів мережі для подальшого аналізу та обробки. Таким чином, величина або кількість вузлів у вхідному шарі зазвичай відображає кількість ознак або параметрів у вхідному наборі даних.

Прихований шар (Hidden Layer) - це шар, що знаходиться між вхідним та вихідним шарами і відіграє ключову роль у виявленні та абстрактному представленні складних залежностей у вхідних даних. Він складається з одного або кількох рівнів вузлів (нейронів), які виконують обчислення на основі даних, отриманих від попереднього шару, і передають результат до наступного шару.

Кожен нейрон у прихованому шарі обробляє вхідну інформацію за допомогою вагових коефіцієнтів, зміщення та функції активації, що дозволяє мережі виявляти неочевидні зв'язки та шаблони в даних. Приховані шари можуть адаптуватися під час процесу навчання, оптимізуючи свої ваги для кращого представлення даних.

Кількість прихованих шарів та нейронів у кожному шарі може суттєво варіюватися в залежності від складності задачі та структури мережі. Ці шари відповідають за "глибоке" у глибокому навчанні, дозволяючи моделі вивчати більш складні та ієрархічні залежності в даних.

Вихідний шар (Output Layer) - це шар, що є кінцевим рівнем, який підсумовує результати обчислень мережі та виводить кінцевий результат. Цей шар відіграє вирішальну роль у формуванні відповіді мережі на вхідні дані.

Складається вихідний шар з вузлів (нейронів), кількість яких залежить від конкретної задачі. Наприклад, для задач класифікації з двома класами може бути всього один вихідний нейрон, а для задачі класифікації з багатьма класами кількість вихідних нейронів може дорівнювати кількості класів.

Кожен нейрон у вихідному шарі обробляє інформацію, отриману від прихованих шарів, і за допомогою специфічної для задачі функції активації виводить кінцевий результат. Наприклад, для задачі бінарної класифікації часто використовується сигмоїдна активація, а для задач регресії - лінійна активація.

Кінцевий результат, отриманий від вихідного шару, може інтерпретуватися різними способами: як ймовірності приналежності до певного класу, як прогнозоване числове значення або як будь-яка інша інформація, що відповідає конкретній задачі.

Щоб нейронна мережа ефективно навчилася вирішувати конкретні задачі, вона використовує два основних методи обчислень: пряме поширення та зворотнє поширення помилки.

Пряме поширення (Feedforward) – це процес передачі інформації через мережу від вхідного шару до вихідного. Під час цього процесу вхідні дані проходять через всі приховані шари мережі, де кожен нейрон обробляє дані за допомогою своїх вагових коефіцієнтів та функції активації, і в кінцевому результаті формується вихідний сигнал.

Зворотнє поширення помилки (Backpropagation) — це метод, який використовується для оптимізації вагових коефіцієнтів мережі на основі помилок у її прогнозах. Після прямого поширення і отримання прогнозованого виводу, відбувається порівняння з реальним значенням, і на основі цього розраховується помилка. Ця помилка потім поширюється у зворотньому порядку від вихідного шару до вхідного, допомагаючи оновити ваги кожного нейрона таким чином, щоб зменшити загальну помилку мережі. Цей процес повторюється багато разів під час навчання мережі [16].

У процесі обчислення результат передається через конкретизовану функцію, яка відома як функція активації. Дана функція встановлює реакцію нейрона на отриманий вхідний стимул.

Функції активації вводять додатковий механізм на кожному етапі прямої передачі, проте такий підхід розглядається як оптимальний [17].

Припустимо, що наша нейронна мережа працює без застосування функцій активації. В такому контексті дія кожного нейрона зводиться до лінійного трансформування вхідних даних з використанням вагових коефіцієнтів та зміщень. Така ситуація виникає через те, що незалежно від кількості вбудованих прихованих шарів, їх діяльність є однорідною, адже комбінація двох лінійних функцій також має лінійний характер [18].

Хоча архітектура мережі стає менш заплутаною, вона не може ефективно вирішувати складні завдання, перетворюючись на модель лінійної регресії.

Існують три основних типи функцій активації: бінарна, лінійна та нелінійна.

1.2.1 Бінарна ступінчаста функція активації

Бінарна ступінчаста функція оперує пороговим значенням, що встановлює, чи має бути активований нейрон.

Коли вхідний сигнал подається на функцію активації, він зіставляється з визначеним пороговим параметром. Якщо вхідний сигнал вищий за цей параметр, нейрон стає активним; в протилежному випадку - він залишається неактивним, а його результат не надсилається до наступного прихованого слою. Графік бінарної ступінчатої функції активації приведено на рисунку 1.3.

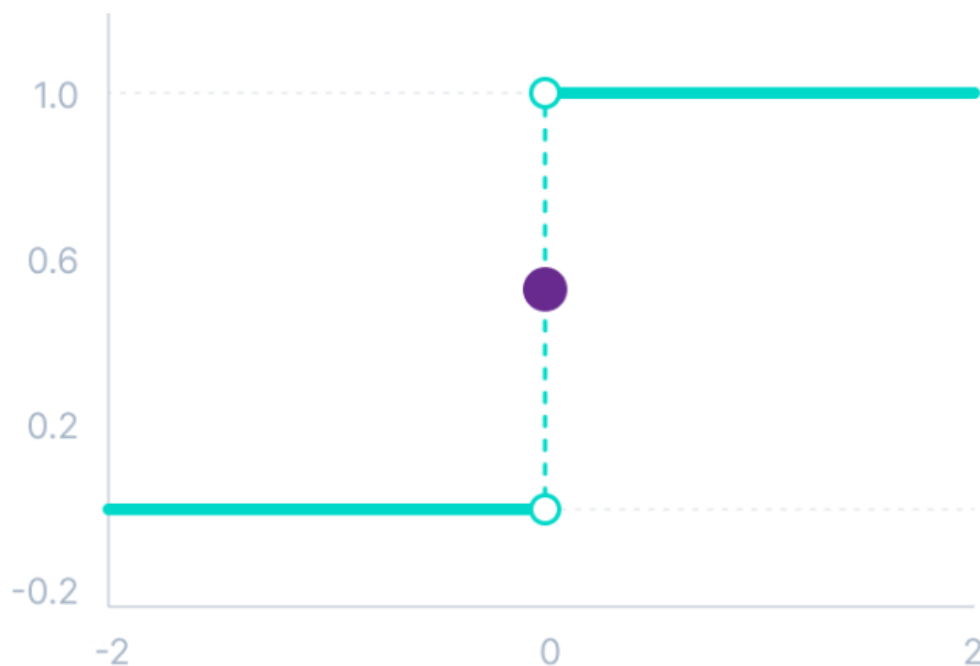


Рисунок 1.3 – Графік бінарної ступінчатої функції активації

Математично бінарну ступінчасту функцію можна представити формулою 1.1:

$$f(x) = \begin{cases} 0 & \text{для } x < 0 \\ 1 & \text{для } x \geq 0 \end{cases} \quad (1.1)$$

Проте існують певні обмеження у використанні бінарної ступінчастої функції.

По-перше, ця функція не здатна забезпечити вихідні дані з декількома значеннями, що обмежує її застосування у сфері багатокласової класифікації.

По-друге, градієнт ступінчастої функції дорівнює нулю, що створює труднощі в процедурі зворотнього поширення помилки в мережах. Ці фактори обмежують застосування бінарної ступінчастої функції в деяких областях нейронних мереж та машинного навчання.

1.2.2 Лінійна функція активації

Лінійна активаційна функція, яка іноді називається "тотожнім перетворенням" або просто "без активації" (з множником 1.0), відображає активацію, що корелює з вхідними даними [18]. Графік лінійної функції активації приведено на рисунку 1.4.

Дана функція лише передає вхідну вагову суму без будь-яких змін.

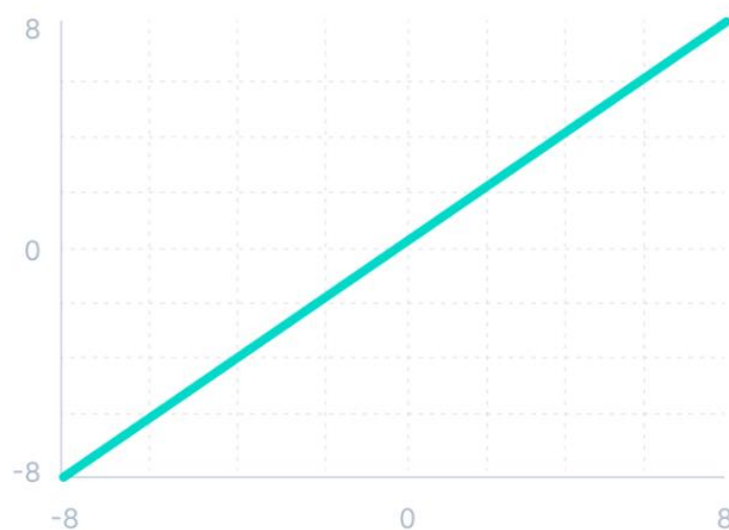


Рисунок 1.4 – Графік лінійної функції активації

В математичному виразі лінійна активаційна функцію можна представити формулою 1.2:

$$f(x) = x \quad (1.2)$$

Проте існують два істотних обмеження у використанні лінійної активаційної функції.

Перше обмеження полягає в тому, що процес зворотнього поширення стає неможливим, оскільки похідна цієї функції є константою та не залежить від вхідного сигналу x . Це ускладнює адаптацію ваг нейронної мережі на основі помилок у виході.

Друге обмеження полягає в тому, що застосування лінійної активаційної функції у мережі з декількома рівнями робить усі ці рівні еквівалентними одному рівню. Незалежно від кількості рівнів у мережі, кожен наступний рівень буде функціонально лінійно залежним від попереднього, що в результаті перетворює мережу з лінійною активаційною функцією на систему, аналогічну однорівневій.

1.2.3 Нелінійні функції активації

Раніше розглянута лінійна активаційна функція є еквівалентом моделі лінійної регресії.

Через свої обмежені можливості вона не в стані дозволити моделі встановлювати глибокі кореляції між вхідними та вихідними параметрами мережі [18].

Нелінійні функції активації здатні подолати обмеження, які існують у лінійних функціях. По-перше, вони роблять зворотнє поширення інформації можливим, оскільки похідна цих функцій корелює з вхідними даними. Це дозволяє аналізувати, як саме певні ваги вхідних нейронів впливають на результати прогнозування, забезпечуючи більш ефективний процес навчання.

По-друге, нелінійні функції активації дають змогу комбінувати кілька шарів нейронів, що забезпечує більшу гнучкість та потужність нейронної мережі. Вихідні значення у таких мережах є результатом нелінійної комбінації вхідних параметрів, які пройшли через декілька послідовних шарів. Це означає, що вихідне значення

може бути представлене як складна взаємодія параметрів у мережі, що розширює можливості моделювання та вирішення складних задач.

Існує 10 основних нелінійних функцій активації. Проте у якості основних чотирьох можна виділити: сигмоїдну функцію, гіперболічний тангенс, ReLU (Rectified Linear Unit) та Leaky ReLU. Кожна з них має свої особливості та сфери застосування в залежності від конкретних завдань та структури нейронної мережі.

1.2.4 Функція активації сигмоїди/логістична функція (Sigmoid)

Сигмоїдальна функція активації, також відома як логістична, є специфічною характеристикою у сфері нейронних мереж. Дана функція може обробляти дійсне число на вході, а на виході генерує значення, що знаходяться в діапазоні між 0 і 1 [19]. Графік сигмоїдної функції активації приведено на рисунку 1.5.

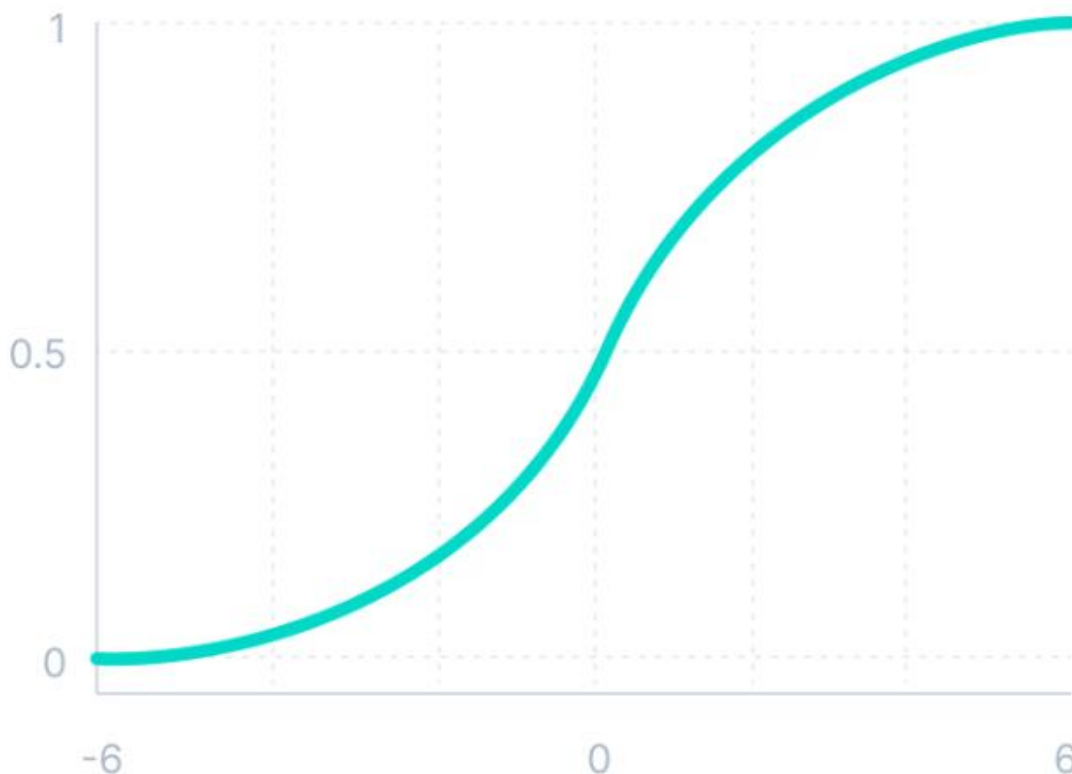


Рисунок 1.5 – Графік сигмоїдальної функції активації

Позитивніші вхідні значення спричиняють те, що вихідна величина наближається до 1,0. З іншого боку, при більш негативних вхідних значеннях вихід тендує до 0,0.

Математично сигмоїдальної функцію можна представити формулою 1.3:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.3)$$

Сигмоїдальна функція активації користується популярністю з кількох причин. По-перше, вона ідеально підходить для моделей, де основною метою є прогнозування ймовірності, оскільки ймовірність будь-якої події існує в межах від 0 до 1, і сигмоїдальна функція відповідає цьому вимогам. По-друге, сигмоїдальна функція є диференційованою, що забезпечує плавний перехід градієнта та виключає різкі стрибки вихідних значень, що відображається в її характерній "S"-подібній формі на графіку.

Тим не менш, сигмоїдальна функція має певні обмеження. Наприклад, значення градієнта є істотними лише у діапазоні від -3 до 3, за межами цього діапазону графік стає майже горизонтальним. Це означає, що для значень, які перевищують 3 або менші за -3, градієнти будуть дуже малі, наближаючись до нуля, що може призвести до зупинки процесу навчання мережі через проблему зниклого градієнта. Крім того, вихід сигмоїдальної функції не є симетричним відносно нуля, внаслідок чого вивід усіх нейронів матиме однаковий знак, що може ускладнити та зробити менш стабільним процес навчання нейронної мережі.

1.2.5 Функція активації на основі гіперболічного тангенса (Tanh)

Функція активації на основі гіперболічного тангенса (Tanh) демонструє значні аналогії з сигмоїдальною чи логістичною функцією активації. Її графічна репрезентація відображає характерний "S"-подібний контур, але, на відміну від сигмоїди, діапазон її значень розташовується в межах від -1 до 1.

Якщо вхідний параметр має підвищене (позитивне) значення, то результуюче вихідне значення стремиться до 1,0. Протилежно, за умови зниженого (негативного) вхідного параметру, вихідне значення приближається до -1,0. Графік функції активації на основі гіперболічного тангенса приведено на рисунку 1.6.

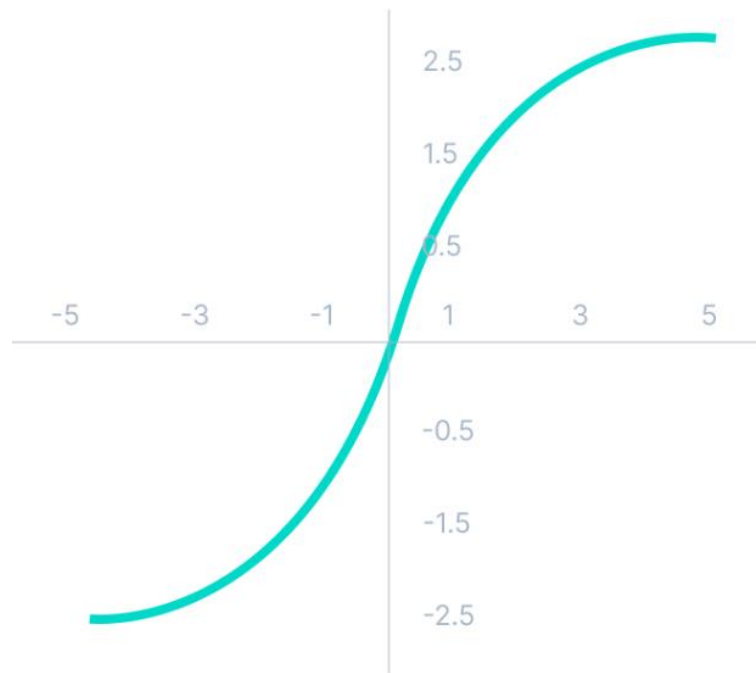


Рисунок 16 – Графік функції активації на основі гіперболічного тангенса

В контексті математичного представлення, функцію активації на основі гіперболічного тангенса можна описати формулою 1.4:

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad (1.4)$$

Перевагами застосування функції активації Tanh є те, що результати, отримані з її допомогою, центровані близько нуля. Це полегшує класифікацію вихідних даних як виражено негативних, нейтральних або виражено позитивних. Така центрованість результатів є особливо корисною у процесі класифікації.

Функція Tanh часто використовується у прихованих шарах нейронних мереж, оскільки її діапазон значень лежить у проміжку від -1 до 1. Внаслідок цього, середнє значення активації у прихованому шарі зазвичай дорівнює нулю або знаходиться поблизу нього. Це сприяє оптимальному центруванню даних, що, у свою чергу, поліпшує процес навчання наступних шарів мережі.

Однак, основним обмеженням функції Tanh є також проблема зниклого градієнта, що характерна й для сигмоїдальної функції активації. При цьому важливо відзначити, що градієнт функції Tanh має більшу крутизну порівняно з

сигмоїдальною, що може бути важливим при певних умовах застосування. Однак слід зазначити, що хоча як сигмоїдальна, так і Tanh функції мають тенденцію до зниклого градієнта, особливість Tanh полягає в тому, що її вивід центрований навколо нулевого значення, а градієнти не мають строгих обмежень щодо їхнього напрямку. Отже, в академічних дослідженнях та практичних застосуваннях нелінійність на основі Tanh часто визнається більш доцільною в порівнянні із сигмоїдальною.

1.2.6 Функція активації виправленого лінійного елементу (ReLU)

Функція активації на основі "Виправленого Лінійного Елементу", або ReLU, займає ключове місце у сфері нейронних мереж та глибокого навчання.

Дана функція, попри свій лінійний вигляд, відзначається особливою математичною структурою, яка дозволяє їй мати похідну. Ця особливість робить можливим ефективне зворотнє поширення помилки, що є критично важливим для оптимізації нейронних мереж. Графік функції активації виправленого лінійного елементу приведено на рисунку 1.7.

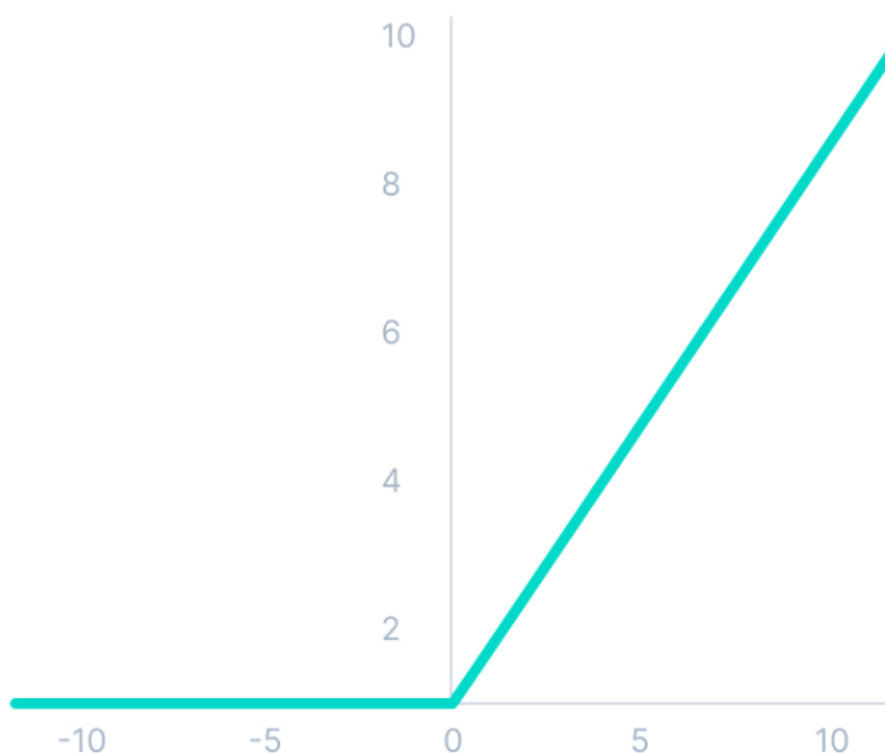


Рисунок 1.7 – Графік функції активації на основі виправленого лінійного елементу

Однією з відмінних характеристик ReLU є те, що не всі нейрони активуються одночасно. Тобто, нейрони стають неактивними, коли результат лінійного перетворення є меншим за нуль.

В контексті математичного представлення, функцію активації на основі виправленого лінійного елементу можна описати формулою 1.5:

$$f(x) = \max(0, x) \quad (1.5)$$

Функція активації на основі "Виправленого Лінійного Елементу", або ReLU, займає ключове місце у сфері нейронних мереж та глибокого навчання.

Ключові переваги використання ReLU (Rectified Linear Unit) як функції активації включають, по-перше, підвищену обчислювальну ефективність порівняно з такими функціями, як сигмоїд або гіперболічний тангенс. Це досягається завдяки активації обмеженої кількості нейронів. По-друге, ReLU сприяє швидшій збіжності процесу градієнтного спуску до глобального мінімуму функції втрат, що обумовлено її лінійним та ненасичувальним характером. Такі особливості роблять ReLU особливо привабливою для використання у багатьох застосуваннях у галузі глибокого навчання.

Однак, існує основне обмеження функції активації ReLU, відоме як проблема "Вмираючого ReLU". Ця проблема виникає через те, що від'ємна частина графіка функції призводить до нульового значення градієнта. Внаслідок цього, під час процесу зворотного поширення помилки, деякі ваги та зміщення для певних нейронів не оновлюються, що може призвести до втрати активності цих нейронів ("мертві нейрони"), які не активуються надалі. Також, через те, що всі від'ємні вхідні значення трансформуються у нуль, це може обмежити здатність моделі ефективно навчатися та адаптуватися до різноманітних даних.

1.2.7 Функція активації нещільного виправленого лінійного елементу (Leaky ReLU)

Функція активації нещільного виправленого лінійного елементу представляє собою оптимізовану варіацію класичної функції ReLU, яка була вдосконалена для

вирішення існуючої проблеми, відомої як "Вмираючий ReLU". Основним нововведенням в цій функції є додавання невеликого позитивного градієнта в області негативних значень, що сприяє підтримці активності нейронів. Графік функції активації нещільного виправленого лінійного елемента приведено на рисунку 1.8.

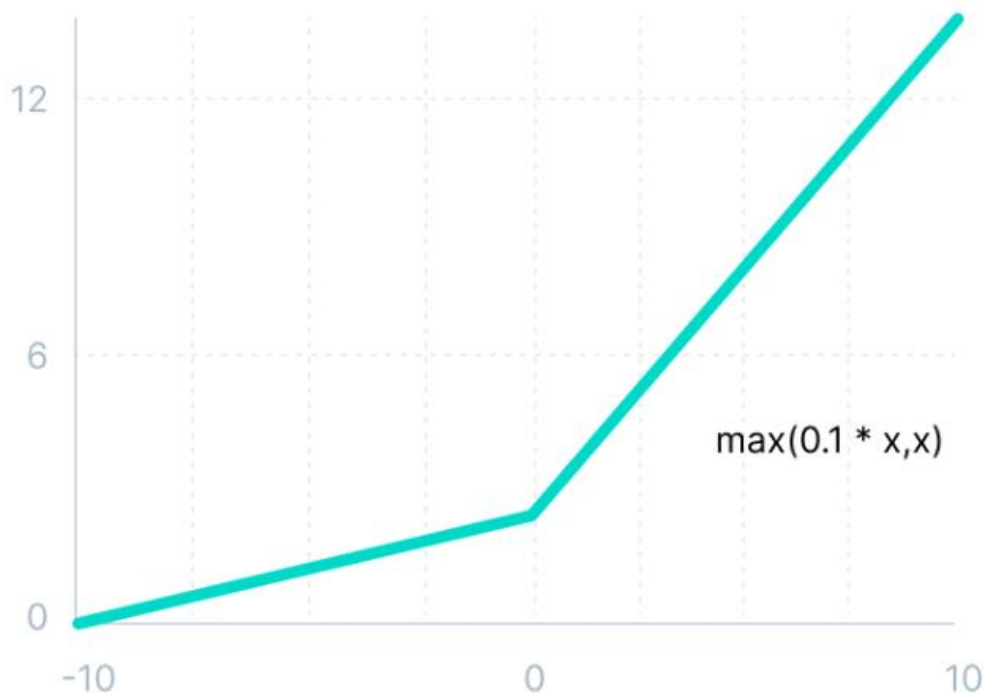


Рисунок 1.8 – Графік функції активації нещільного виправленого лінійного елемента

В контексті математичного виразу функції активації Leaky ReLU можна описати формулою 1.6:

$$f(x) = \max(0.1x, x) \quad (1.6)$$

Функція активації Leaky ReLU поєднує в собі всі ключові переваги функції ReLU. Однак відзначимо додатковий аспект: завдяки модифікації Leaky ReLU дозволяє процесу зворотного поширення враховувати навіть від'ємні вхідні значення.

Зміна у функції активації, що забезпечує наявність градієнта для від'ємних значень і виключає їх зведення до нуля, запобігає формуванню "мертвих" нейронів, які не активуються. Така характеристика сприяє підвищенню стабільності та ефективності роботи нейронних мереж.

Проте, існують обмеження у використанні функції активації нещільного виправленого лінійного елементу (Leaky ReLU). Одним з них є потенційна нестабільність прогнозів для від'ємних вхідних значень. Іншим обмеженням є те, що градієнт для від'ємних значень, хоча і не зводиться до нуля, залишається досить малим, що може призвести до уповільнення процесу навчання моделі. Ці аспекти потрібно враховувати при виборі функції активації для конкретних задач у галузі глибокого навчання.

1.2.8 Нормована експоненційна функція активації (Softmax)

Для глибокого розуміння властивостей функції активації Softmax слід почати з її концептуальної основи - сигмоїдальної (або логістичної) функції активації, яка є ключовою у визначенні ймовірностей, графік якої представлений на рисунку 1.9 [20].

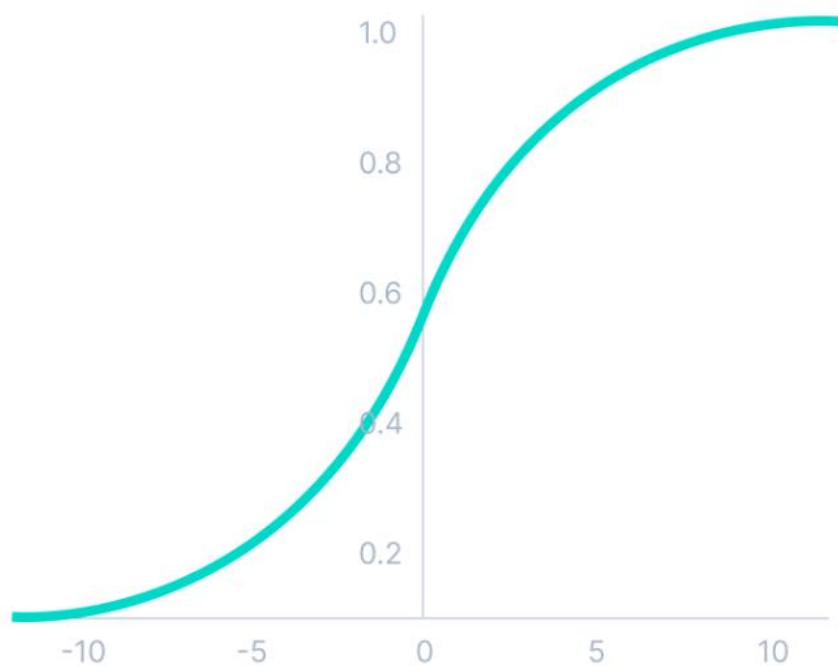


Рисунок 1.9 – Графік функції визначення ймовірності на основі сигмоїдальної функції активації

В контексті математичного виразу функції визначення ймовірності на основі сигмоїдальної функції активації можна описати формулою 1.7:

$$sig(t) = \frac{1}{1 + e^{-t}} \quad (1.7)$$

Розглянемо властивості сигмоїдальної функції активації в контексті задач класифікації. Значення, отримані від сигмоїдальної функції, розташовуються в діапазоні між 0 та 1, що дозволяє розглядати їх як ймовірності приналежності до певного класу. Однак, коли мова йде про багатокласову класифікацію, дана функція може проявляти певні недоліки.

Прикладом може служити ситуація, коли ми отримуємо п'ять відносних ймовірностей від сигмоїдальної функції, як-от: 0.8, 0.9, 0.7, 0.8 та 0.6. Очевидним питанням стає: як адекватно розглядати такий розподіл ймовірностей? Адже інтуїтивно зрозуміло, що сума ймовірностей усіх можливих класів має дорівнювати одиниці.

Для вирішення цієї проблеми та отримання більш виваженого розподілу ймовірностей вводиться функція Softmax. Ця функція, по суті, представляє собою генералізацію сигмоїдальної функції для множинних класів та дозволяє обчислити відносні ймовірності для кожного класу.

Важливо підкреслити, що функція Softmax зазвичай використовується як функція активації у завершальному шарі нейронних мереж при реалізації задач багатокласової класифікації, оскільки вона забезпечує коректний розподіл ймовірностей серед доступних класів. В контексті математичного виразу функцію Softmax можна описати формулою 1.8:

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \quad (1.8)$$

Для глибшого розуміння принципу роботи функції активації Softmax розглянемо специфічний приклад.

Припустимо, що у контексті задачі класифікації нам необхідно розглянути три потенційні класи. Відповідно, у завершальному шарі нашої нейронної мережі присутні три нейрони. За умовами нашого прикладу виходи з цих нейронів формують наступний вектор значень: $[1.8, 0.9, 0.68]$.

Застосувавши до цих виходів функцію активації Softmax з метою перетворення їх на ймовірнісну шкалу, ми отримаємо вектор ймовірностей: $[0.58, 0.23, 0.19]$.

На підставі отриманих результатів можемо зробити висновок, що функція Softmax вказує на найбільш імовірний клас, присвоюючи йому значення 1, у той час як іншим класам присвоюється значення 0. Таким чином, найбільш вірогідним класом в цьому прикладі є клас, що відповідає першому нейрону (з індексом 0) з трьох доступних нейронів.

1.3 Висновки за розділом 1

У даному розділі було проведено всебічний аналіз аспектів існування туристичних систем, переважна більшість з яких сьогодні представлена у формі веб-систем. В результаті розгляду ключових елементів функціонування туристичної системи, її основоположних компонентів та механізмів взаємодії було виявлено низку викликів та проблематичних питань, що стоять перед цією галуззю. До таких викликів відносяться, зокрема, велика кількість інформації, потреба в адаптації до змінних умов ринку та вимог споживачів, а також необхідність швидкого та ефективного прийняття рішень.

Проведений аналіз дав змогу виявити потенціал для покращення та оптимізації існуючих веб-систем туристичного сектору шляхом впровадження сучасних технологій, зокрема нейронних мереж. Нейронні мережі, як один з ключових напрямків розвитку сучасної інформаційної технології, демонструють високу ефективність як інструмент для оптимізації туристичних систем. У роботі

були досліджені основні принципи роботи нейронних мереж, їх характерні риси та можливості застосування у контексті туристичної індустрії. На підставі аналізу було встановлено, що нейронні мережі можуть бути використані для аналізу даних, прогнозування тенденцій та ідентифікації закономірностей. Це, в свою чергу, дає змогу забезпечити більш ефективне планування та управління туристичними ресурсами, а також підвищити якість послуг, що надаються туристам.

У перспективі подальшого розвитку технології нейронних мереж та її інтеграції у туристичні системи можна очікувати створення інноваційних рішень. Ці рішення дозволять адаптуватися до динамічних змін умов ринку, одночасно забезпечуючи високу якість обслуговування та задоволення потреб туристів.

2 РОЗРОБКА СИСТЕМИ

2.1 Вибір архітектури додатку

У процесі створення програмного забезпечення, архітектурне проектування становить фундамент, який визначає не лише майбутню стабільність та ефективність продукту, але й його здатність до розвитку та інтеграції з іншими системами. Важливість цього етапу неможливо переоцінити, адже правильно обрана структура стимулює інновації та забезпечує відповідність продукту сучасним вимогам ринку.

Архітектура не тільки визначає, як програма буде взаємодіяти з кінцевими користувачами та іншими системами, але й як вона зможе адаптуватися до майбутніх змін, що є критично важливим у швидкозмінному технологічному світі. Вибір архітектури повинен враховувати не тільки поточні бізнес-вимоги, але й передбачуване майбутнє, щоб забезпечити довгострокову життєздатність програмного продукту.

При виборі архітектури важливо враховувати такі аспекти, як спроможність системи до швидкого відновлення після збоїв, легкість внесення змін та додавання

нових функціональних можливостей, забезпечення високого рівня безпеки та здатність до масштабування згідно зі зростанням кількості користувачів чи обсягів даних. Також слід враховувати можливості інтеграції з іншими системами та платформами, а також вимоги до розгортання та підтримки програмного забезпечення.

Окрім технічних аспектів, необхідно звертати увагу й на бізнес-вимірювання якості архітектури, такі як вплив на загальну вартість володіння системою, швидкість виходу на ринок нових функцій, а також відповідність продукту правовим та регулятивним вимогам.

У контексті вибору архітектури, залежно від сектору ринку, в якому діє компанія, певні характеристики можуть бути визначальними. Для організацій, що працюють з чутливими даними, як-от фінансові установи чи медичні організації, першочерговою є безпека. Для технологічних стартапів чи сервісів, що оперують з великими даними, важливою стає здатність до швидкого аналізу та обробки інформації. У випадку медіа-платформ та ігрових сервісів, критичною є продуктивність та забезпечення безперервного доступу до контенту.

Отже, проектування архітектури, є комплексним завданням, яке вимагає глибокого розуміння як технічних, так і бізнесових аспектів. Це мистецтво знаходження ідеальної рівноваги між мінливими вимогами ринку, технологічними обмеженнями та стратегічними цілями компанії.

Відповідно до книги "Архітектура програмного забезпечення на практиці", архітектурне проектування відіграє значущу роль у досягненні успіху проекту, при цьому автори виділяють кілька ключових причин, які можуть бути релевантними для невеликих команд або окремих розробників:

Архітектура є каталізатором для властивостей якості програмного продукту. Як приклад, децентралізований характер рівнорядної архітектури природно забезпечує високий рівень доступності. З іншого боку, архітектура, заснована на мікросервісах, вирізняється легкістю модифікації завдяки чіткому розподілу функціоналу між компонентами.

Архітектура також сприяє ефективній взаємодії між усіма учасниками процесу, адже вона віддзеркалює структуру організації, забезпечуючи кожному чітке розуміння власної відповідальності у контексті програмного продукту.

Замість акценту на розробці компонентів, архітектура стимулює зосередитися на їх взаємодії, що змушує мислити про комунікацію між частинами системи, а не лише про спосіб написання коду.

Окрім того, архітектура накладає певні обмеження на процес проектування, які, в свою чергу, відкривають простір для креативності в інших аспектах. Задані межі дозволяють розуміти, де можна виявити творчий підхід, не наражаючи на небезпеку розвиток чи якість системи.

Автори підкреслюють, що рішення, ухвалені на початковому етапі, часто мають вирішальне значення та обмежують гнучкість змін у майбутньому, тому необхідно з початку проекту глибоко аналізувати вимоги та якомога краще планувати архітектуру.

У цьому контексті досвідчений архітектор володіє перевагою над новачком, оскільки він здатен прогнозувати майбутнє на значно довший термін та оцінити вплив різних проектних рішень на систему.

Серед усіх наявних шаблонів та архітектур було обрано три найпопулярніші та найпоширеніші, кожна з яких має чіткі переваги та недоліки: Монолітна архітектура, клієнт-серверна (Service-Oriented Architecture) та мікросервисна.

2.1.1 Монолітна архітектура

Монолітна архітектура, яка застосовується у створенні програмного забезпечення, представляє собою класичний, цілісний підхід до конструювання систем. Вона втілює інтегрований дизайн, де усі елементи програми — починаючи від користувацького інтерфейсу і закінчуючи механізмами доступу до даних — є частинами одного суцільного кодового комплексу. Це забезпечує єдиний простір для взаємодії компонентів, що сприяє злагодженому розвитку та експлуатації системи, особливо коли розмір команди чи обсяг проекту є обмеженими.

У такому типі архітектури будь-які модифікації програмного коду вимагають повторного запуску цілісної системи. Цей аспект може стати в нагоді, спрощуючи

процес розробки та забезпечуючи уніфікований контроль над проектом. Проте, ця ж характеристика може призвести до труднощів при масштабуванні та оновленні програми, оскільки навіть мінорні зміни потребують великого обсягу роботи для їх імплементації та перевірки [21].

Водночас, завдяки централізації ресурсів, монолітні системи можуть демонструвати високу швидкодію, оскільки компоненти обмінюються даними без зовнішнього міжпрограмного спілкування. Негативною стороною є те, що вся система стає вразливою до збоїв: неполадки в одному сегменті можуть вивести з ладу весь продукт.

Проектуючи монолітну систему, важливо уникати перетворення коду на складноструктурований "спагеті-код", який ускладнює подальшу підтримку та розвиток програми. Це передбачає дотримання високих стандартів кодування та обдумування архітектурних рішень на початкових етапах розробки.

Монолітна архітектура репрезентується сервером, що виконує роль основного вузла для оброблення даних, інтеграції та координації всіх елементів системи. В рамках такого сервера реалізований інтерфейс користувача (UI) формується через вбудовані шаблонізатори, що дозволяють впроваджувати динамічні дані безпосередньо в статичні HTML-структури.

Ілюстративним прикладом слугує веб-додаток, створений з використанням фреймворку Django, який призначений для конструювання єдиних і суцільних веб-сайтів та програм. Django пропонує шаблонізатор, що інтегрує Python-код в HTML, полегшуючи процедуру створення інтерактивних веб-сторінок. Такий підхід дозволяє розробникам ефективно керувати контентом, опрацьовувати запити до баз даних і впроваджувати бізнес-логіку, уникаючи залучення зайвих служб чи серверів. Програмний продукт, розроблений на монолітній основі, представляє собою єдину інтегровану систему, що працює на одному сервері. Структуру такого додатку показано на рисунку 2.1.

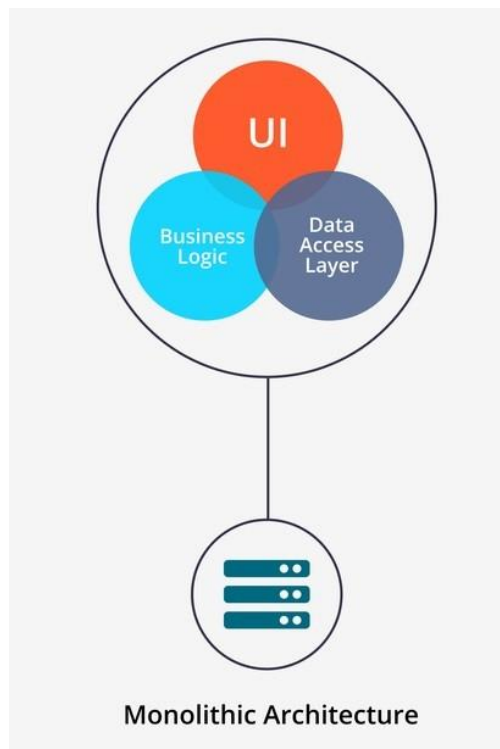


Рисунок 2.1 – Структура монолітної архітектури

Така архітектура виявляється доречною для проектів із стабільним набором вимог, де часті оновлення або розширення функціоналу не є критичними. Однак, коли з'являється потреба в масштабуванні чи модифікації окремих елементів, монолітна структура може становити серйозні виклики. Зміни зажадають повної переінсталяції програми, що займає чимало часу і може призвести до збоїв у функціонуванні системи.

Незважаючи на свої обмеження монолітна архітектура продовжує залишатися вибором для певних ситуацій, таких як розробка прототипів чи продуктів з обмеженою функціональністю, де швидкість виведення на ринок є ключовим фактором. Тим не менш, зі збільшенням складності та обсягу проекту, моноліт може обмежувати технічний прогрес, спонукаючи до переходу на більш модульні та гнучкі архітектурні рішення, такі як мікросервіси або модульні архітектури, які дозволяють ефективніше масштабувати та оновлювати систему, адаптуватися до змінних вимог ринку та технологічного середовища.

2.1.2 Клієнт-серверна архітектура

Клієнт-серверна архітектура є домінуючою моделлю для розробки сучасних додатків, особливо в розгалужених і розподілених мережових середовищах. У цій моделі, клієнтські програми взаємодіють із сервером, який виконує більшість обробки даних і зберігає централізовані дані. Це дозволяє користувачам доступатися до однакової інформації, незалежно від їхнього місцезнаходження чи використовуваного пристрою [22].

Значна перевага такої архітектури полягає в тому, що вона підтримує одночасний доступ багатьох клієнтів до спільних ресурсів, що дуже ефективно для бізнес-додатків, що потребують колаборації та спільної роботи з даними. Крім того, розподіл ролей та завдань між клієнтами та серверами може підвищити загальну продуктивність системи, оскільки сервери можуть бути оптимізовані для специфічних завдань, таких як обробка запитів баз даних, виконання складних обчислень або забезпечення безпеки.

На практиці, клієнт-серверна архітектура може бути реалізована в різних формах. Наприклад, у традиційній дворівневій архітектурі, клієнт безпосередньо взаємодіє з сервером бази даних. Трьохрівнева модель вводить додатковий рівень — сервер застосунків, що слугує проміжним шаром між клієнтами та сервером баз даних, що може виконувати бізнес-логіку та обробку даних.

Розглянемо клієнт-серверну архітектуру на прикладі веб-додатку, який використовує ExpressJS для бекенду та React для фронтенду. ExpressJS — це гнучкий Node.js веб-фреймворк, що надає ряд можливостей для розробки веб-додатків та API. React — це бібліотека JavaScript для створення користувацьких інтерфейсів, яка дозволяє розробникам конструювати інтерактивні UI компоненти, які ефективно оновлюються та відображаються при зміні даних. Структуру такого додатку показано на рисунку 2.2.

Client-Server Architecture

High-Level Diagram

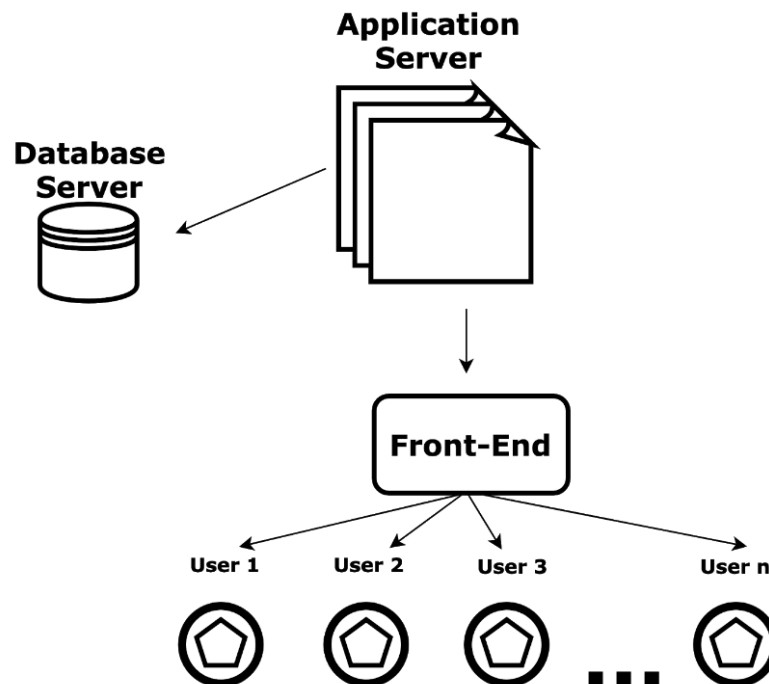


Рисунок 2.2 – Структура клієнт-серверної архітектури

У контексті веб-додатків для управління проектами, сервер, що використовує ExpressJS, бере на себе роль центрального вузла, що займається обробленням запитів протоколу HTTP, виконанням бізнес-процедур, як-от призначення завдань і моніторинг прогресу, а також взаємодією з базами даних для архівування інформації про проекти та їх учасників. Паралельно з цим, клієнтський інтерфейс, створений за допомогою React, забезпечує інтуїтивне відображення веб-сторінок, дозволяючи користувачам взаємодіяти з системою в режимі реального часу.

Розглянемо процес ініціації нового проекту як взірць взаємодії клієнта і сервера. Представник бізнесу заповнює електронну форму на сторінці та ініціює її подання, що активує React компоненти, які піклуються про збір даних та їх надсилання на сервер за допомогою асинхронних запитів. Сервер, на свою чергу, перевіряє коректність отриманої інформації, реєструє новий проект у системі управління базами даних і відсилає підтвердження клієнту, який відповідає оновленням інтерфейсу без потреби перезавантаження веб-сторінки.

Дана архітектура вирізняється можливістю масштабування та адаптації до змінних вимог, що забезпечується окремою оптимізацією серверних та клієнтських компонентів. Оновлення баз даних чи оптимізація серверних запитів можуть здійснюватись незалежно від клієнтської частини, що вносить зручність у процес розвитку продукту та його обслуговування.

Незважаючи на численні переваги, клієнт-серверна архітектура вимагає уважного підходу до проектування та управління. Важливо продумати систему таким чином, щоб уникнути перебоїв у роботі та забезпечити надійний захист даних. Централізоване зберігання інформації вимагає високого рівня безпеки, оскільки будь-які вразливості можуть призвести до серйозних наслідків. Потреба у постійному оновленні та масштабуванні може стати викликом, особливо при швидкому зростанні користувачів чи бізнес-процесів.

Навіть із можливими складнощами, клієнт-серверна модель залишається фундаментальною для багатьох сучасних веб-додатків, пропонуючи баланс між гнучкістю, масштабованістю та ефективністю для широкого спектру програмних рішень.

2.1.3 Мікросервісна архітектура

Монолітна архітектура представляє собою єдину інтегровану систему, де всі компоненти застосунку тісно взаємопов'язані. Натомість, мікросервісна архітектура полягає в декомпозиції застосунку на множину мініатюрних, самостійних модулів, кожен з яких виконує власні функції та має відокремлену від інших базу даних. Ці модулі розробляються як автономні сервіси, що взаємодіють один з одним за допомогою API або обміну повідомленнями [23].

Описуючи методологію мікросервісів, слід згадати, що вона передбачає створення застосунків як сукупності незалежних служб, висвітленої Мартіном Фаулером. Кожна служба функціонує автономно, власне обробляючи процеси та спілкуючись через зручні протоколи, зокрема HTTP. Структуру такого додатку показано на рисунку 2.3.

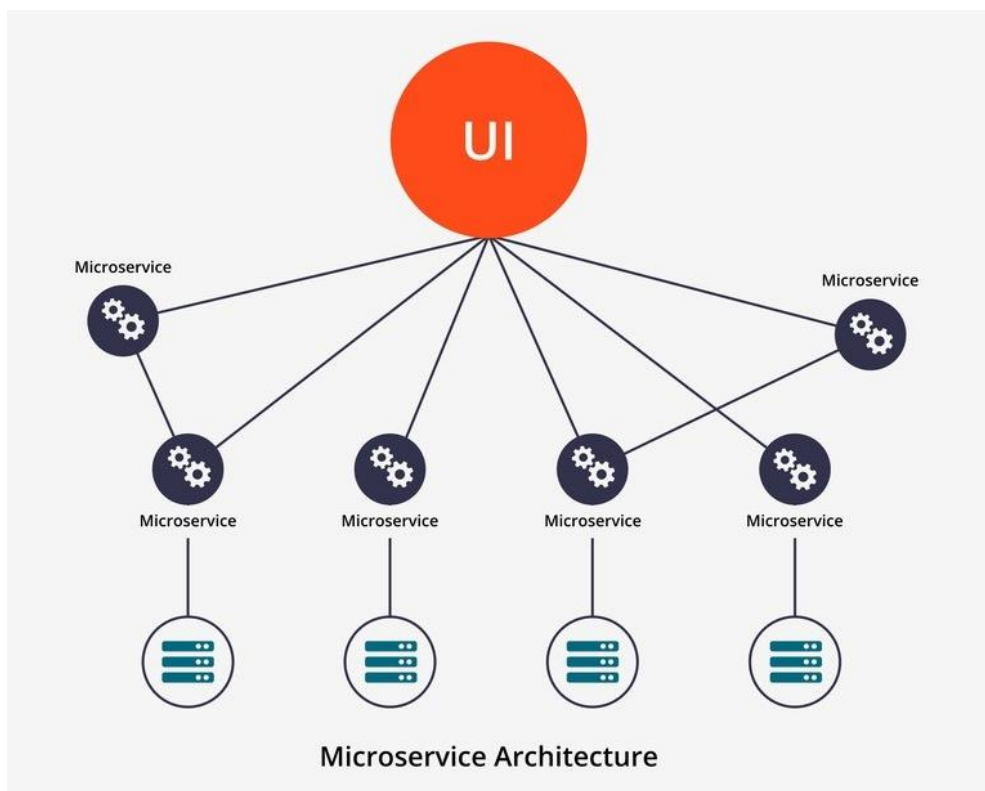


Рисунок 2.3 – Структура мікросервісної архітектури

У рамках мікросервісної архітектури значна увага приділяється незалежності компонентів, що дозволяє підвищити гнучкість розробки та експлуатації застосунків. Ця автономність уможливлює окреме розгортання та оновлення кожного сервісу, що сприяє більшій стійкості системи до помилок та полегшує впровадження нових можливостей. Окрім того, мікросервіси забезпечують спрощене обслуговування, адже командам розробників дозволяється зосереджуватися на окремих сервісах, що відповідають за конкретні бізнес-завдання.

Мікросервісна архітектура також надає гнучкість у виборі технологій, оскільки кожен сервіс може використовувати різні програмні стеки. Помилки в одному сервісі не поширюються на весь застосунок, що знижує загальний ризик при внесенні змін.

Проте, мікросервісна архітектура вносить певну складність у вибір та налаштування взаємозв'язків між модулями та базами даних. Ця складність вимагає виваженого управління та суворого контролю за інтеграцією сервісів. Розробка

такої системи супроводжується вирішенням комплексу поперечних завдань, таких як конфігурація, логування, моніторинг, що вимагає додаткових зусиль. Також варто зазначити, що тестування розподіленої системи з множинними компонентами є більш трудомістким, ніж тестування уніфікованої монолітної архітектури.

2.1.3 Висновок з вибору архітектури

Провівши детальний аналіз трьох основних архітектурних моделей — монолітної, клієнт-серверної та мікросервісної — було встановлено, що для даного проекту найбільш придатною є клієнт-серверна архітектура. Зважаючи на масштаб та вимоги проекту, вона демонструє значну гнучкість, стабільність та ефективність.

Клієнт-серверна архітектура, яка передбачає чітке розділення обов'язків між сервером, що забезпечує бізнес-логіку, та клієнтами, які відповідають за презентаційний шар, дозволяє нам виконувати масштабування та оновлення системи з мінімальними витратами часу та ресурсів. Цей підхід не тільки забезпечує ефективну взаємодію між користувачем та системою, але й спрощує процес розробки, адже розробники можуть зосередитися на певних компонентах системи, не порушуючи роботу інших.

Додатково, клієнт-серверна архітектура дозволяє з легкістю перейти від прототипу до повноцінної комерційної системи. На відміну від монолітної структури, що може виявитися негнучкою при зміні вимог, та мікросервісної архітектури, яка вимагає значних витрат на початкову розробку та координацію сервісів, клієнт-серверна модель дозволяє уникнути цих труднощів.

Завдяки своїй ефективності та гнучкості, клієнт-серверна архітектура забезпечує надійну основу для майбутнього росту та розвитку даного проекту, дозволяючи безперервно переходити від стадії тестування та валідації до впровадження системи на ринку. У підсумку, вибір цієї архітектури зумовлений стратегічними перевагами, які вона пропонує для досягнення довготривалих цілей.

2.2 Вибір нейронної мережі

У попередньому розділі 1.3 було докладно висвітлено теоретичні аспекти нейронних мереж, викладено основи їх структури та принципи функціонування. Тепер увага буде зосереджена на практичній стороні застосування нейронних мереж.

У даному розділі буде проаналізовано технічні умови, необхідні для ефективної роботи нейронних мереж, виявити потенційні шляхи оптимізації їх використання, а також розглянути додаткові можливості, які ця технологія пропонує.

2.2.1 Технічні умови для функціонування нейронної мережі

Технічні умови для функціонування нейронних мереж охоплюють цілу низку параметрів, які варто розглянути для забезпечення оптимальної роботи системи. Початковим етапом є визначення апаратної підтримки, яка має забезпечувати достатню обчислювальну потужність. Це особливо важливо, оскільки нейронні мережі потребують інтенсивних обчислень, особливо у фазі навчання, де використовуються методи глибокого навчання. Обчислювальні потужності повинні відповідати обсягам даних та складності моделі, що розробляється.

Далі важливу роль відіграє програмне забезпечення, яке має бути сумісне із вибраною обчислювальною платформою та мовами програмування, які застосовуються в розробці. Системи повинні мати здатність до швидкого обміну даними, а також ефективної взаємодії з користувачем, що вимагає відповідних інтерфейсів і інструментів для візуалізації.

Не менш важливим є забезпечення високої пропускної здатності мережевих з'єднань, оскільки великі обсяги даних повинні передаватися без затримок та втрат. Це стосується як внутрішніх, так і зовнішніх комунікацій, які пов'язані з обміном даними між різними компонентами системи або зовнішніми джерелами.

Крім того, необхідно звернути увагу на системи зберігання даних. Вони мають бути надійними, масштабованими та здатними до швидкого доступу до

великих обсягів інформації. Для нейронних мереж, що працюють з великими датасетами, потрібно забезпечити можливість ефективної роботи з базами даних, які можуть містити терабайти інформації.

Також варто зазначити, що безпека є ключовим аспектом технічних умов. Системи повинні бути захищені від несанкціонованого доступу та забезпечувати конфіденційність оброблюваних даних. Це включає в себе як фізичний захист обладнання, так і захист програмного забезпечення.

Наостанок, необхідно враховувати здатність системи до масштабування, оскільки потреби у обчислювальних ресурсах можуть змінюватися у залежності від складності задач, що вирішуються, та обсягів даних для обробки. Така здатність дозволяє системі адаптуватися до зростаючих потреб без необхідності перебудови всієї архітектури.

У сукупності ці технічні умови створюють основу, на якій можлива побудова ефективних та надійних нейронних мереж, здатних вирішувати широкий спектр завдань від простого аналізу даних до складного машинного навчання та штучного інтелекту.

У якості прикладу технічних умов для функціонування нейронних мереж використаємо відкриті дані типових датацентрів, які зазвичай використовуються для великих обчислювальних систем, таких як нейронні мережі.

Для задоволення високих вимог до обчислювальних потужностей, які ставлять перед дата-центрами нейронні мережі, такі центри оснащуються спеціалізованим обладнанням. В основі їх технічного арсеналу лежать графічні процесори високої продуктивності, призначені для паралельних обчислень — невід'ємної частини тренування та застосування нейронних мереж. Чіпи від NVIDIA, як-от Tesla V100 чи новітні A100, є яскравими прикладами таких пристроїв, що значно прискорюють роботу з нейронними мережами.

Не менш важливою є наявність потужних систем зберігання даних, адже для тренування нейронних мереж потрібні об'ємні набори даних. Швидкісне мережеве обладнання забезпечує ефективний обмін даними в межах дата-центру, знижуючи час доступу до даних до мінімуму.

Системи охолодження займають ключове місце в інфраструктурі таких центрів через значну кількість тепла, що виробляється обчислювальними компонентами. Тому часто використовуються передові рішення, зокрема рідкісне охолодження, для підтримання оптимальної температури апаратури.

Електропостачання — це ще один критичний елемент, адже робота нейронних мереж вимагає значних електроенергетичних ресурсів, які рахуються мегаватами. Дата-центри забезпечені надійним джерелом живлення, яке підтримується системами безперебійного електропостачання та генераторами для забезпечення безперервної роботи під час відключень.

Архітектура самого дата-центру, включно з розміщенням стійок та загальним дизайном системи, розроблена з урахуванням оптимізації простору, забезпечуючи при цьому достатній потік повітря та доступність для технічного обслуговування та модернізації.

Ці загальні принципи можуть варіюватися в залежності від конкретних потреб, масштабу операцій та технологічного прогресу на момент проектування та впровадження дата-центру.

Вартість оренди потужностей дата-центрів для обчислень нейронних мереж може істотно варіюватися в залежності від багатьох чинників, включно з масштабами операцій, специфікою використовуваного обладнання та додаткових послуг, що надаються. Наприклад, оренда хмарного комп'ютера NVIDIA DGX починається з 36,999 доларів США на місяць за один інстанс. Така оренда забезпечує доступ до хмарного комп'ютера з восьмима GPU NVIDIA H100 або A100 та 640 ГБ GPU-пам'яті [24].

Для тих, кому потрібні більш значні обчислювальні потужності, NVIDIA пропонує AI Superpod, доступ до якого можна отримати через підписку на послуги штучного інтелекту за 90,000 доларів США на місяць, що задовольняє потреби високопродуктивних досліджень та обчислень [25].

Наведені приклади належать до вищого цінового сегменту, адаптовані для інтенсивних обчислювальних завдань і високих вимог до продуктивності. Важливо зазначити, що ціни для менш вимогливих завдань або операцій меншого масштабу

можуть бути значно нижчими, а також вартість може зменшуватися з часом внаслідок зростання конкуренції та технологічного прогресу. Проте, навіть якщо вартість оренди обчислювальних потужностей знизиться в декілька разів, витрати на запуск та експлуатацію власної нейронної мережі все ще будуть високими.

З огляду на значні фінансові та часові витрати, пов'язані з самостійним розгортанням та навчанням нейронних мереж, що вимагають значних обчислювальних потужностей та спеціалізованих знань, для багатьох організацій цей шлях залишається непрохідним. Процес навчання може затягнутися на роки, поглинаючи ресурси та вимагаючи постійної уваги від висококваліфікованих фахівців. Враховуючи, що це може призвести до сотень тисяч годин кваліфікованої праці, вартість такого проекту може виявитися непомірною для більшості дослідницьких груп або стартапів.

У контексті пошуку оптимальних шляхів впровадження технологій штучного інтелекту та машинного навчання, особливої уваги заслуговує можливість використання API вже навчених нейронних мереж. Цей підхід відкриває перспективи для широкого спектру застосувань без необхідності вкладення значних ресурсів у розробку та навчання власних моделей.

2.2.2 Вибір API сервісу нейронної мережі

Використання готових рішень, API нейронних мереж, дозволяє швидко інтегрувати передові можливості штучного інтелекту у різноманітні процеси та продукти. Такі API надають доступ до розширених функцій обробки даних, природномовного розуміння, аналізу зображень та інших складних завдань, які зазвичай вимагають глибоких знань у галузі машинного навчання.

Однією з ключових переваг цього підходу є зменшення часу та витрат на розвиток проекту. Замість багатомісячних циклів розробки та оптимізації власних нейронних мереж, організації можуть відразу приступити до впровадження готових рішень, значно прискорюючи процес розробки та запуску нових продуктів та послуг.

Ще однією важливою перевагою є гнучкість та масштабованість, які надаються такими рішеннями. API вже навчених мереж дозволяють легко

адаптувати рішення під специфічні потреби проекту, забезпечуючи при цьому здатність до масштабування відповідно до зростаючих потреб та вимог.

Водночас, існують певні обмеження, пов'язані з використанням готових API. Наприклад, може відсутній контроль над внутрішньою логікою моделі та обмежені можливості для спеціалізованої настройки. Однак у багатьох сценаріях ці обмеження компенсуються широким спектром можливостей, які надають готові нейронні мережі.

Таким чином, використання API вже існуючих нейронних мереж є вагомим рішенням для організацій, які прагнуть інтегрувати технології штучного інтелекту в свої процеси, забезпечуючи при цьому ефективність, швидкість та інноваційність своїх рішень.

При виборі API для нейронних мереж, важливо звернути увагу на ряд критичних аспектів, включаючи функціональність, масштабованість, цінову політику та специфікації, пропоновані різними відкритими API. Оцінюючи відомі API, слід розглянути їх особливості, потенційні області застосування та відповідність конкретним вимогам проекту.

API від OpenAI, особливо серія GPT, займає важливе місце у сучасному світі обробки природної мови. Наприклад, GPT-3 встановив новий стандарт у генерації тексту, автоматизації відповідей та інших завданнях пов'язаних з обробкою природної мови, завдяки своїй високій точності та гнучкості. Цей API забезпечує передові можливості, такі як глибоке навчання на основі контексту, що дозволяє створювати текст, що максимально наближений до людського письма. Його застосування варіюється від створення вмісту для веб-сайтів, блогів, до розробки інтерактивних чатботів та автоматизації клієнтського обслуговування [26].

Крім того, OpenAI пропонує інноваційні моделі, такі як GPT-3.5 і недавно запуснений GPT-4, які посилюють можливості AI у генерації тексту, збагачуючи їх більшою точністю та розширеними можливостями з розуміння та відтворення людського мовлення. Ці моделі, завдяки їх великому об'єму навчальних даних та складності алгоритмів, можуть виконувати більш складні задачі, такі як переклад,

вирішення запитань з багатьма кроками, та навіть програмування, надаючи значні переваги у багатьох галузях, від академічних досліджень до розробки додатків.

API від OpenAI також надає високий рівень інтеграції з іншими сервісами та платформами, що дозволяє розробникам легко вбудовувати ці потужні можливості AI в їхні додатки. Це включає інтеграцію з веб-додатками, мобільними програмами, ігровими додатками та навіть складнішими системами, які використовують AI для аналізу великих даних. Використання API від OpenAI дозволяє компаніям та розробникам розширити можливості своїх продуктів, забезпечуючи більшу автоматизацію, персоналізацію та інноваційність у своїх послугах.

TensorFlow, створений Google, є відкритою платформою для машинного навчання, яка пропонує багатий набір інструментів для розробки та навчання нейронних мереж. Його Python API є популярним через простоту використання та універсальність, хоча TensorFlow також сумісний з іншими мовами програмування. TensorFlow Core API, зосереджений на роботі низького рівня, дозволяє розробляти складні моделі з високим ступенем налаштування [27].

TensorFlow активно розвивається, зосереджуючись на прискоренні навчання та виводу моделей, розподіленому обчисленню через DTensor API та легкості впровадження у машинне навчання. Ці особливості роблять TensorFlow ідеальним для широкого діапазону застосувань, від академічних досліджень до промислових проектів.

Google Cloud AI та Microsoft Azure AI, обидві платформи, пропонують комплексні рішення для роботи з нейронними мережами в хмарному середовищі, але кожна має свої унікальні характеристики. Вони надають широкий спектр інтегрованих сервісів та інструментів, адаптованих до різноманітних бізнес-потреб, проте вартість їх використання може збільшуватися з розширенням масштабів проекту [28-29].

Google Cloud AI, розроблений Google, відомий своєю високою гнучкістю та широким спектром послуг. Ця платформа забезпечує інструменти для розробки, навчання та реалізації моделей машинного навчання, використовуючи інноваційні технології Google в області штучного інтелекту. Її інтуїтивно зрозумілі інтерфейси

та API, доступні для різних мов програмування, роблять платформу доступною для широкого спектра розробників.

Однією з особливостей Google Cloud AI є її здатність ефективно обробляти великі обсяги даних, пропонуючи рішення для розподіленого обчислення та автоматизованого машинного навчання. Це сприяє швидкому створенню та оптимізації моделей, мінімізуючи час та зусилля, потрібні для їх розвитку.

Ключові компоненти платформи включають AutoML, що автоматизує генерацію моделей машинного навчання, та AI Platform, яка надає комплексні інструменти для розробки, навчання та запуску моделей у хмарі. Вона також пропонує спеціалізовані сервіси, такі як Vision AI для обробки зображень та Natural Language API для аналізу тексту.

Google Cloud AI відзначається масштабованістю та ефективністю, надаючи рішення, які підходять для проектів різного масштабу. Це робить її ідеальним варіантом для компаній, які шукають надійні, швидкі та інноваційні рішення в галузі штучного інтелекту.

Ціни на послуги Google Cloud AI варіюються залежно від обраного продукту та обсягу використання. Платформа пропонує конкурентні тарифи, особливо при виборі довготривалих планів, але технічна підтримка може бути дороговартісною, особливо для новачків у галузі.

Використання готових оптимізованих моделей часто є більш ефективним, ніж розробка власних мереж, оскільки вони вже пройшли тривалий процес навчання і можуть забезпечувати високу якість результатів з мінімальними зусиллями з боку розробника.

Поза цим, важливо звернути увагу на підтримку спільноти та наявність документації. Сильна спільнота користувачів та розробників може надавати цінну підтримку, навчальні матеріали та покращення. Наявність глибокої та зрозумілої документації є ключовою для швидкого вивчення та ефективного використання API.

Також слід врахувати масштабованість та гнучкість API. Масштабованість важлива для проектів, які можуть з часом розростатися, тоді як гнучкість дозволяє адаптувати API під різні потреби проекту.

У сфері інтеграції нейронних мереж OpenAI займає провідне місце завдяки унікальному поєднанню переваг, що робить його оптимальним вибором для реалізації пілотних проектів з впровадження штучних нейронних мереж. Платформа відзначається легкістю використання, мінімізуючи потребу в складних налаштуваннях, що сприяє ефективній інтеграції продуктів OpenAI в різноманітні проекти. Це, в свою чергу, сприяє оцінці переваг та обмежень цього підходу.

Особливістю OpenAI є можливість глибокої кастомізації. Різнорічність налаштувань та опцій дозволяє задовольнити специфічні вимоги проектів, що веде до створення адаптивних рішень у сфері штучного інтелекту. Велика навчальна база сприяє простій інтеграції платформи в проекти різного спрямування, розширюючи спектр можливих застосувань.

API OpenAI пропонує різноманітні мовні моделі, спеціалізовані на певних завданнях, кожна з яких має індивідуальні цінові параметри. Наприклад, GPT-4 пропонується за ціною 0.03 долара США за введення 1K токенів та 0.06 долара США за виведення, тоді як варіант з розширеним 32K контекстом коштує 0.06 долара США за введення та 0.12 долара США за виведення. Модель GPT-4 Turbo з підтримкою 128K токенів контексту відрізняється оновленням знань та розширеними можливостями, коштуючи 0.01 долара США за введення та 0.03 долара США за виведення. Ця модель також доступна з можливістю візуального сприйняття, що дозволяє її застосування у широкому діапазоні сценаріїв. Для діалогових систем, GPT-3.5 Turbo з 16K контекстом пропонується за доступною ціною 0.0010 долара США за введення та 0.0020 долара США за виведення, тоді як варіант gpt-3.5-turbo-instruct з 4K контекстом коштує 0.0015 долара США за введення та 0.0020 долара США за виведення [30].

OpenAI постійно розвивається, інтегруючи новітні технологічні інновації, що надає користувачам доступ до передових інструментів у сфері штучного інтелекту. Незважаючи на обмежену базу знань, яка може вважатися недоліком, зростаюча

популярність рішень від OpenAI сприяє швидкому оновленню бази знань, забезпечуючи її актуальність та релевантність.

Таким чином, вибір OpenAI як платформи для інтеграції нейронних мереж відображає баланс між зручністю використання, можливістю адаптації під конкретні проекти та доступом до інноваційних технологій. Це робить OpenAI оптимальним рішенням для розробки та аналізу нейронних мереж, враховуючи його можливість швидкого створення ефективних та адаптивних рішень для широкого спектру застосувань.

2.2.3 Можливості використання OpenAI API у веб-системі

Простота використання API від OpenAI є однією з його основних переваг, що відкриває широкі можливості для розробників. Завдяки простому механізму взаємодії, що базується на створенні та відправці POST-запитів з відповідними параметрами та заголовками, використання цього API не потребує спеціалізованих пакетів чи глибоких технічних знань у цій сфері. Це робить платформу доступною для широкого кола розробників, незалежно від їх досвіду роботи зі штучним інтелектом.

Простота інтеграції з OpenAI API має значний вплив на процес розробки. Розробники, які працюють з різними мовами програмування, такими як JavaScript, Python, PHP та інші, а також різними фреймворками як для клієнтської, так і для серверної частини, мають змогу легко інтегрувати функціонал OpenAI у свої проекти. Це особливо важливо в контексті тестової розробки, де гнучкість та швидкість впровадження нових технологій є ключовими. Завдяки цій простоті використання OpenAI API стає ідеальним інструментом для швидкого прототипування та експериментування з різними аспектами штучного інтелекту [31].

Взаємодія з OpenAI API може бути здійснена за допомогою різних HTTP клієнтів, таких як Axios, що є популярним варіантом для браузерів та Node.js. Розробник має спочатку визначити URL для API, який відповідає обраній моделі OpenAI, такій як GPT-3, з відповідним URL ендпоінту.

Для формування запити створюється об'єкт з параметрами, які включають текст для аналізу або генерації відповіді. Серед основних параметрів, які варто включити до об'єкту, є "prompt" - текстовий запит до моделі, та "max_tokens" - максимальна кількість токенів у відповіді.

Подальший крок включає виконання POST-запиту за допомогою Axios, передавши вказаний URL та параметри запиту. Особливу увагу слід приділити заголовкам запиту, які мають містити ключ доступу до API для забезпечення автентифікації та авторизації. Ключ доступу зазвичай розташовується у заголовку 'Authorization' у форматі 'Bearer (Токен доступу)', який є суттєвим для забезпечення безпеки доступу до можливостей API.

Після відправки запиту через Axios, розробник отримує відповідь від API, яка зазвичай надається у форматі JSON. Це дозволяє з легкістю обробляти отримані дані для подальших потреб проекту. Приклад взаємодії з OpenAI API з використанням мови програмування TypeScript та пакету Axios наведено на рисунку 2.4.

```
const params : {model: string, messages: stri... = {
  model: 'gpt-3.5-turbo',
  messages: [templateMessage],
  temperature: 0.2
};

const openAiClient: AxiosInstance = axios.create({
  headers: {
    Authorization: 'Bearer ' + apiKey,
    'OpenAI-Organization': organization
  }
});

no usages
async function fetchGpt() : Promise<void> {
  try {
    const response: AxiosResponse = await openAiClient
      .post(url: 'https://api.openai.com/v1/chat/completions', data: params);
    console.log(response.data);
  } catch (e) {
    console.log(e);
  }
}
```

Рисунок 2.4 – Приклад взаємодії з OpenAi Api з використанням мови програмування TypeScript та пакету Axios

Хоча працювати з OPENAI API надзвичайно просто, однак значно краще використовувати офіційний пакет `openai`. Цей пакет спрощує процес налаштування та може навіть виключити деякі параметри, які роблять конфігурацію складнішою для розуміння.

Використання офіційного пакету `openai` дозволяє автоматизувати багато аспектів взаємодії з API, зменшуючи необхідність вручну налаштовувати кожен параметр запиту. Це забезпечує більш чистий та зрозумілий код, у якому зосереджена увага лише на важливих аспектах взаємодії з API. Приклади використання пакету `openai` Node.js у мові програмування Typescript для конфігурації з'єднання показано на рисунку 2.5, а для виконання запиту та обробки відповіді – на рисунку 2.6.

```

1  import { Configuration, OpenAIApi } from 'openai';
2  import { ChatCompletionRequestMessage } from 'openai/api';
3
4  const configuration: Configuration = new Configuration( param: {
5    organization: 'ORG-Key',
6    apiKey: 'API-Key'
7  });
8  const openai: OpenAIApi = new OpenAIApi( configuration: configuration);
9
10 const templateMessage: string = 'Your Template Message';
11
12 const message: ChatCompletionRequestMessage = {
13   role: 'user',
14   content: templateMessage
15 };
16
17 export { openai, message, templateMessage };

```

Рисунок 2.5 – Приклад використання пакету `openai` Node.js у мові програмування Typescript для конфігурації з'єднання

```

37 try {
38   const response = await openai.createChatCompletion(
39     createChatCompletionRequest: {
40       model: 'gpt-3.5-turbo',
41       messages: [message],
42       temperature: 0.2
43     }
44   );
45   const array = JSON.parse(text: response.data.choices[0].message.content);
46   const countries = array.map((item) => CountriesProvider.getCountryByCode2( code: item)).filter((item) => !!item);
47   console.log(countries);
48 } catch (e) {
49   console.log(e);
50 }

```

Рисунок 2.6 – Приклад використання пакету openai Node.js у мові програмування Typescript для виконання запиту та обробки відповіді

Окрім того, пакет openai надає додаткові функції та утиліти, які можуть значно поліпшити процес розробки. Наприклад, він може автоматично обробляти аутентифікацію та інші необхідні налаштування, забезпечуючи безпечне та ефективно з'єднання з API.

Використання офіційного пакету також сприяє кращому дотриманню практик та стандартів розробки, оскільки пакет регулярно оновлюється та підтримується розробниками OpenAI. Це забезпечує сумісність із останніми оновленнями API та введення нових функцій та вдосконалень.

Таким чином, хоча пряма взаємодія з OpenAI API є простою, використання офіційного пакету openai є більш рекомендованим підходом. Це спрощує процес налаштування, робить код більш читабельним та забезпечує використання найкращих практик розробки, забезпечуючи ефективність та безпеку взаємодії з штучним інтелектом.

2.3 Вибір засобів розробки

Сучасний світ програмування пропонує великий вибір інструментів та технологій, які полегшують та оптимізують процес створення програмного забезпечення. Існує можливість писати код на різних мовах програмування, однак важливо розуміти, що створення всього "з нуля" часто не є найбільш раціональним

рішенням. Натомість, використання готових фреймворків дозволяє значно спростити та прискорити розробку, використовуючи вже наявні рішення та підходи.

У контексті клієнт-серверної архітектури, важливим аспектом є вибір відповідних фреймворків для серверної та клієнтської частини проекту. Вибір правильного фреймворку може значно вплинути на ефективність та якість кінцевого продукту. У цьому розділі ми зосередимося на аналізі та виборі найбільш підходящих фреймворків для реалізації обраної архітектури, враховуючи такі аспекти, як масштабованість, гнучкість, підтримка спільноти та можливості інтеграції з іншими інструментами та технологіями.

Цей підхід не лише спрощує процес розробки, але й гарантує, що програмне забезпечення буде побудоване на перевірених та надійних технологічних рішеннях, що відповідають сучасним стандартам та вимогам ринку.

2.3.1 Вибір серверного фреймворку

У даному підрозділі розглядається ключове питання вибору технологічного стеку для серверної частини веб-додатку. Сучасний світ веб-розробки пропонує широкий спектр серверних фреймворків, кожен з яких має свої унікальні особливості, переваги та обмеження. Вибір правильного серверного фреймворку є критично важливим, оскільки він впливає не лише на процес розробки, але й на продуктивність, масштабованість та безпеку кінцевого додатку.

Також буде проаналізовано різні аспекти, що слід врахувати при виборі серверного фреймворку. Це включає розгляд таких критеріїв, як мова програмування, на якій базується фреймворк, його зрілість та підтримка спільнотою, можливості розширення та інтеграції з іншими інструментами та сервісами, а також вимоги до продуктивності та безпеки проекту.

У рамках аналізу буде розглянуто три найпопулярніших серверних фреймворках, кожен з яких пов'язаний з певною мовою програмування: Spring для Java, Laravel для PHP та Express для JavaScript. Розгляд цих фреймворків дасть змогу оцінити їхні особливості в контексті різних технологічних підходів та мов програмування.

Spring, відомий як один з провідних фреймворків для розробки на Java, характеризується значною гнучкістю та широким спектром можливостей. Розроблений з метою спрощення роботи з Java, Spring надає розробникам комплексний інструментарій для створення ефективних веб-додатків та мікросервісів. Його легка, модульна структура дозволяє вибирати лише ті компоненти, які потрібні для конкретного проекту, сприяючи точному відповіданню проектним вимогам.

Однією з найважливіших переваг Spring є реалізація патерну "Inversion of Control", який забезпечує спрощення управління залежностями та процесу розробки. Це відкриває шлях для створення чистого, добре структурованого коду. Підтримка об'єктно-орієнтованого та аспектно-орієнтованого програмування полегшує управління перехресними аспектами програми, включаючи безпеку та управління транзакціями. Широка інтеграція з різними зовнішніми технологіями та сервісами надає Spring універсальність.

Тим не менш, незважаючи на ці переваги, Spring має і суттєві недоліки. Його складність може становити виклик для новачків, а велика кількість компонентів та залежність від сторонніх бібліотек може ускладнити управління та оновлення проєктів. Крім того, Spring може виявитися вимогливим до ресурсів, що може бути обтяжливим для обмежених систем. Особливо актуальними є потенційні проблеми зі швидкістю виконання у великих додатках та складнощі зі збереженням високої продуктивності та масштабованості. Це особливо важливо для стартапів, які можуть швидко перерости у великі комерційні проєкти, де обмежена масштабованість Spring може стати значною перешкодою [32].

Попри вказані недоліки, Spring продовжує користуватися популярністю завдяки своїй потужності та багатофункціональності, підкріпленої активною спільнотою розробників та всебічною документацією, яка сприяє ефективному освоєнню фреймворку. Отже, Spring, незважаючи на свої недоліки, залишається вагомим вибором для розробників, які прагнуть відповідати високим вимогам сучасної веб-розробки.

У свою чергу Express.js, який є мінімалістичним та гнучким фреймворком для Node.js, заслуговує на детальний аналіз, виходячи з його значної популярності у сфері розробки веб-додатків. Цей фреймворк є ключовою складовою екосистеми Node.js та сприяє спрощенню процесу створення веб-додатків, забезпечуючи розробників потрібними інструментами для реалізації серверної логіки.

Основними перевагами Express.js є його легкість та гнучкість, що надають розробникам значну свободу в архітектурних рішеннях та використанні додаткових модулів або бібліотек. Ці властивості роблять Express ідеальним варіантом для швидкої розробки веб-додатків та API. Він значно спрощує маршрутизацію запитів та обробку відповідей, які є ключовими аспектами ефективної роботи веб-сервера. Розробники можуть без зусиль визначати маршрути та методи обробки запитів, використовуючи різноманітні middleware для аутентифікації, логування та інших функціональних потреб.

Висока продуктивність Express.js, зумовлена ефективністю Node.js та легкістю самого фреймворку, визначає його як вдалий вибір для проектів, що потребують швидкої обробки великої кількості одночасних запитів. Підтримка широкого спектру плагінів і додаткових модулів надає розробникам можливість гнучкого розширення функціоналу своїх додатків.

Проте, наявність певних недоліків у Express.js не може бути ігнорована. Відсутність строгих правил структури або архітектури може призвести до складнощів у управлінні великими та комплексними проектами. Висока залежність від зовнішніх пакетів і модулів може створювати потенційні ризики у питаннях безпеки та сумісності. Крім того, Express.js може бути не найкращим рішенням для великих, складних систем, де необхідна масштабованість та інтеграція з широким спектром сервісів [33].

Laravel, відомий як передовий PHP-фреймворк, заслужив визнання у сфері веб-розробки завдяки своїй гнучкості та комплексному функціоналу. Цей фреймворк ефективно вирішує низку стандартних завдань, пов'язаних із розробкою веб-додатків, таких як маршрутизація, аутентифікація, робота з базами даних та кешування.

Однією з ключових особливостей Laravel є високоякісна система автоматизованого тестування. Це значно спрощує процес перевірки коду, підвищує якість продукту, забезпечує надійність та безпеку веб-додатків. Тестування в Laravel охоплює широкий спектр від юніт-тестування до функціонального та приймального тестування, дозволяючи розробникам забезпечувати високий рівень якості коду.

Система шаблонів Blade, яка є інтегральною частиною Laravel, відрізняється своєю інтуїтивністю та гнучкістю. Одночасно з цим, Laravel забезпечує ефективну інтеграцію з клієнтською частиною на JavaScript за допомогою вбудованих пакетів, розширюючи можливості для створення сучасних веб-інтерфейсів.

Однією з важливих переваг Laravel є його сучасна та вичерпна документація, яка постійно оновлюється та надає детальні та зрозумілі вказівки з використання фреймворку. Ця документація є незамінним ресурсом для як нових, так і досвідчених розробників, сприяючи швидкому засвоєнню функціоналу Laravel.

Активна спільнота розробників та регулярні оновлення забезпечують не лише постійне вдосконалення фреймворку, але й швидку підтримку нових технологій та підходів у веб-розробці. Це сприяє збереженню актуальності Laravel у динамічному світі технологій.

Водночас, Laravel може бути викликом для новачків через його складність та обширність. Необхідність залучення зовнішніх бібліотек може ускладнити процес управління проектами. Крім того, потреба у значних системних ресурсах та питання масштабування у великих проектах можуть стати обмеженнями при використанні Laravel [34].

Таким чином, Laravel проявляє себе як потужний та гнучкий фреймворк для розробки веб-додатків, надаючи розробникам широкі можливості для реалізації різноманітних проектів. Його використання сприяє підвищенню ефективності та якості розробки в сучасному веб-просторі.

На підставі аналізу трьох розглянутих фреймворків можна зробити висновок, що Spring виявляється надто складним у використанні, тоді як Express.js надмірно розширює свободу дій розробника у написанні коду. Обидва ці фреймворки, Spring

та Express.js, можуть зіткнутися з проблемами масштабування, особливо якщо тестовий проект перетвориться на комерційний. З огляду на ці аспекти, вибір на користь Laravel здається найбільш виправданим. Laravel забезпечує збалансоване співвідношення гнучкості, структурованості та масштабованості, що робить його оптимальним вибором для розробки тестових та комерційних проектів.

2.3.2 Вибір архітектури серверної частини

Під час аналізу різних фреймворків для розробки веб-додатків стає зрозумілим, що однією з ключових проблем, з якими стикаються розробники, є питання масштабованості проектів. Проте, в багатьох випадках ця проблема може бути компенсована через вибір певного типу архітектури.

Існує безліч підходів до архітектури веб-додатків, які вирізняються своїми перевагами та недоліками. Деякі архітектурні підходи зосереджуються на чіткому розмежуванні логіки роботи з даними та бізнес-логіки, що сприяє легкості управління та розширення функціональності. Інші архітектури наголошують на швидкості розробки, забезпечуючи простоту та гнучкість процесів реалізації.

Серед існуючих архітектурних рішень багато сприяють покращенню масштабованості проектів. Однак, в цьому контексті особливо вирізняється архітектура Porto, яка була спеціально створена для вирішення як проблем роботи з даними, так і питань масштабованості. Porto - це архітектурний підхід, що базується на концепції модульності та розділення відповідальності, що забезпечує високий рівень гнучкості та адаптивності у розробці великих та складних систем.

Пристосування Porto до потреб конкретного проекту дозволяє ефективно управляти різними аспектами розробки, від бізнес-логіки до роботи з даними, тим самим оптимізуючи процес розвитку та забезпечуючи необхідну масштабованість. Це робить Porto особливо привабливим вибором для проектів, які вимагають гнучкості у розвитку та здатності адаптуватися до змінюваних умов та вимог ринку.

Porto є сучасним архітектурним шаблоном програмного забезпечення, який надає розробникам всебічний набір керівництв, принципів та шаблонів для організації їх коду у високо підтримуваний та повторно використовуваний спосіб.

Основною метою Porto є допомога розробникам у створенні програмного забезпечення, яке є масштабованим, гнучким та легким у підтримці з часом.

Porto особливо підходить для веб-проектів середнього та великого розміру, які вимагають високого ступеня гнучкості та масштабованості. За допомогою Porto розробники можуть створювати високо масштабовані моноліти, які легко можуть бути розділені на декілька мікросервісів у разі потреби. Такий підхід дозволяє повторно використовувати бізнес-логіку або функції додатків у кількох проектах, роблячи його ідеальним вибором для команд, які працюють над різноманітними проектами.

2.3.3 Архітектурний шаблон Porto

Архітектурний шаблон Porto базується на кількох встановлених архітектурних концепціях, включаючи дизайн орієнтований на домен (DDD), модульну, мікроядерну, модель-представлення-контролер (MVC), шарувату та архітектуру дії-домен-респондент (ADR). Використовуючи ці встановлені концепції, Porto надає розробникам надійний та добре випробуваний фреймворк для створення програмного забезпечення.

Додатково до цих основоположних концепцій, Porto також дотримується низки визнаних принципів дизайну, таких як SOLID, ООП, LIFT, DRY, CoC, GRASP, Генералізація, Висока Когезія та Низьке Зв'язування. Ці принципи призначені для забезпечення того, що програмне забезпечення, побудоване за допомогою Porto, є підтримуваним, масштабованим та легким для розуміння.

Загалом, Porto представляє собою потужний та гнучкий архітектурний шаблон програмного забезпечення, який пропонує розробникам всебічний набір інструментів для створення масштабованого, підтримуваного та повторно використовуваного програмного забезпечення.

Архітектурний шаблон Porto, сучасне рішення в галузі програмної архітектури, включає два основних рівні: "Контейнери" та "Корабель". "Контейнери" вміщують усі аспекти бізнес-логіки застосунку, тоді як "Корабель" включає код інфраструктури, який є загальним для всіх "Контейнерів". Ці рівні

можуть бути реалізовані в будь-якому обраному фреймворку, наприклад, у Laravel чи Rails, і розміщені в директоріях `app/` або `src/` в кореневій частині проекту.

Поділ коду застосунку на ці два рівні дозволяє Porto забезпечити чітке відділення бізнес-логіки від інфраструктурного коду, спрощуючи тим самим підтримку та оновлення застосунку. У "Контейнерах" знаходиться код, специфічний для конкретного застосунку, тоді як "Корабель" містить спільний код, який можна використовувати у декількох "Контейнерах".

Такий підхід дозволяє легко масштабувати застосунок, додаючи чи видаляючи "Контейнери" за потреби, не впливаючи на основний інфраструктурний код. Porto забезпечує можливість створення масштабованих та легко підтримуваних архітектур програмного забезпечення, які можуть адаптуватися до змінюваних вимог.

Перед поглибленням у деталі, важливо зрозуміти різні рівні коду, що присутні у кодовій базі. Рівень низького рівня включає в себе код фреймворку, який забезпечує базові операції, як от читання файлів з диску чи взаємодію з базою даних, та зазвичай знаходиться у директорії `Vendor`. Рівень середнього рівня, розташований у "Корабелі", є загальним кодом застосунку, який обслуговує вищий рівень та покладається на нижчий рівень для своєї роботи. Вищий рівень, розміщений у "Контейнерах", містить бізнес-логіку та охоплює складну логіку застосунку.

Поділ коду застосунку на ці рівні допомагає забезпечити, що кожен рівень виконує відповідні завдання. Код низького рівня забезпечує базову функціональність, код середнього рівня служить зв'язуючою ланкою між низьким та вищим рівнями, а код вищого рівня містить специфічну бізнес-логіку застосунку.

Завдяки Porto можна легко розділити код на ці рівні, забезпечуючи, що кожен з них відповідає за відповідні завдання. Такий поділ сприяє підтримці коду більш організованим та масштабованим з часом.

Щоб краще уявити взаємозв'язок між рівнями "Контейнери", "Корабель" та базовим фреймворком, можна порівняти рівень "Контейнери" з вантажними

контейнерами, які покладаються на "Корабель" - вантажний корабель, який, в свою чергу, покладається на фреймворк - море. Для того, щоб зобразити взаємодію між різними рівнями, аналогія показана на рисунку 2.7, а схематична діаграма шарів – на рисунку 2.8 [35].

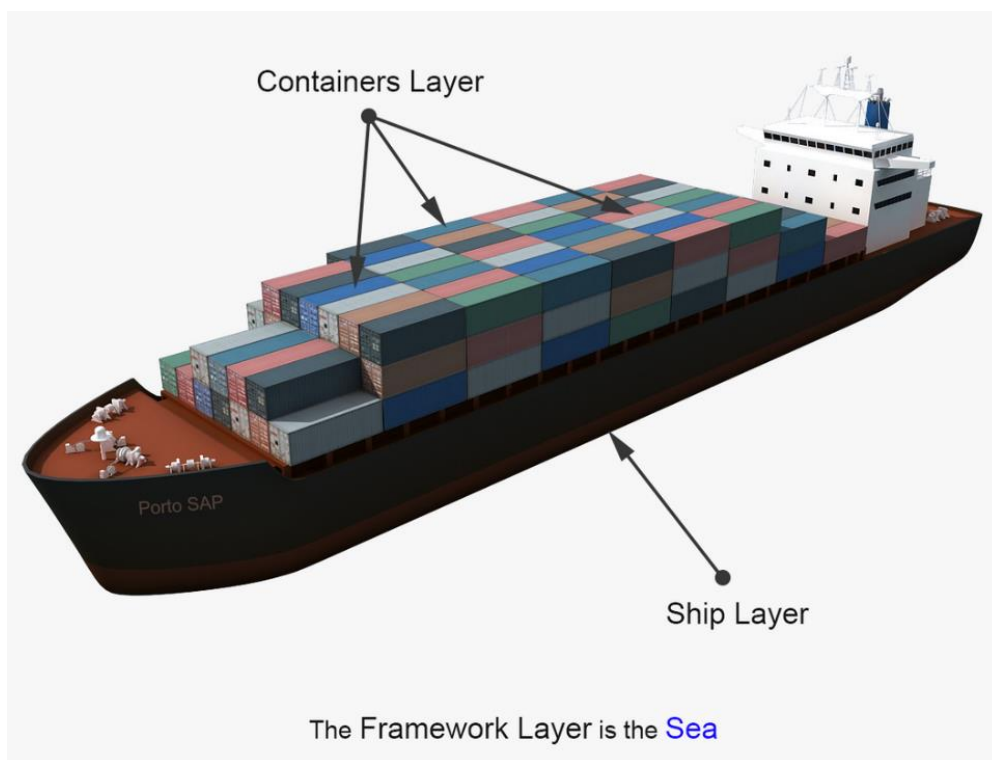


Рисунок 2.7 – Візуальне представлення архітектурного шаблону Porto

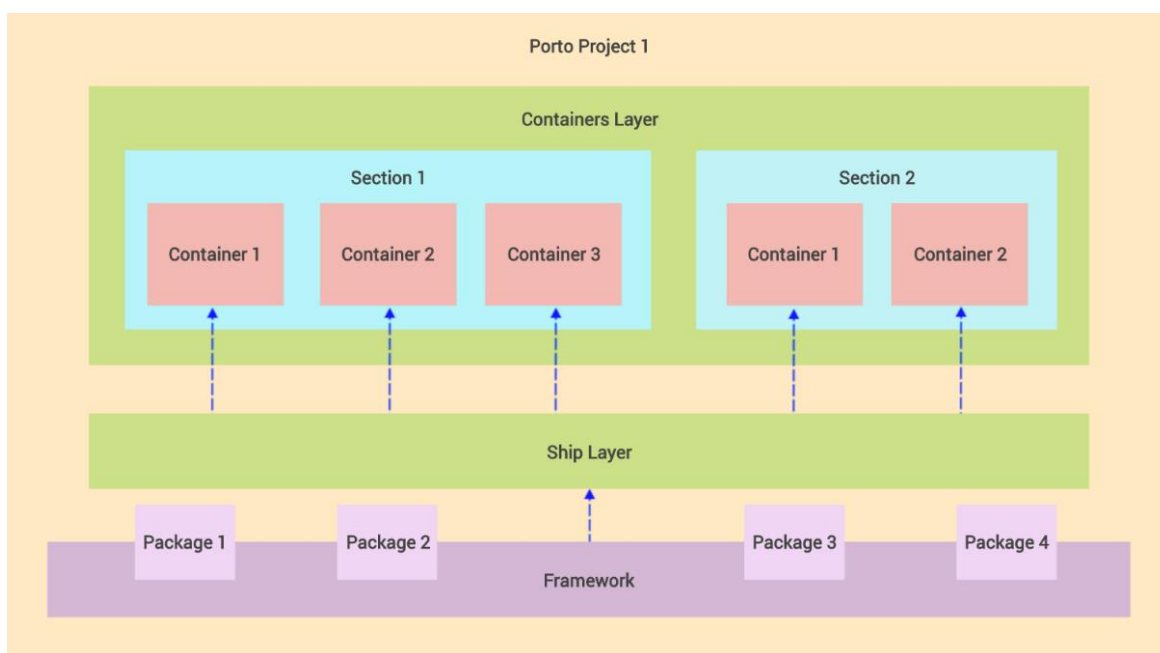


Рисунок 2.8 – Схематичне зображення шарів архітектурного шаблону Porto

У архітектурній моделі Porto використовуються кілька ключових компонентів, які відіграють важливу роль у розробці веб-додатків. Серед цих компонентів можна виділити Routes, Controllers, Requests, Actions, Tasks, Models, Views, Transformers, Exceptions та Sub-Actions.

Routes служать для визначення шляхів запитів у застосунку, в той час як Controllers забезпечують логіку обробки цих запитів. Requests відповідають за обробку вхідних даних, які надходять до застосунку. Actions виконують конкретні бізнес-операції, тоді як Tasks дозволяють реалізовувати повторно використовувані елементи бізнес-логіки. Models використовуються для роботи з даними, а Views відповідають за представлення даних користувачу.

Transformers служать для перетворення даних у потрібний формат, наприклад, JSON або XML. Exceptions використовуються для обробки помилок, які можуть виникати під час виконання застосунку. Sub-Actions дозволяють деталізувати бізнес-логіку, розбиваючи її на менші, легко керуємі частини.

Усі ці компоненти спільно формують структуру застосунку, діаграма якої зображено на рисунку 2.9, що дозволяє розробникам створювати масштабовані, гнучкі та легко підтримувані веб-додатки. Porto забезпечує чіткий поділ функцій і відповідальності між різними елементами системи, сприяючи таким чином підвищенню ефективності розробки та забезпеченню високої якості кінцевого продукту.

Ще однією з істотних переваг є те, що архітектурний шаблон Porto уже реалізовано як окреме рішення або надбудову для фреймворку Laravel під назвою Ariato. Ця надбудова забезпечує розробникам можливість використовувати перевірену структуру Porto в межах фреймворку Laravel, розширюючи його функціональність та спрощуючи процес розробки високоякісних веб-додатків [36].

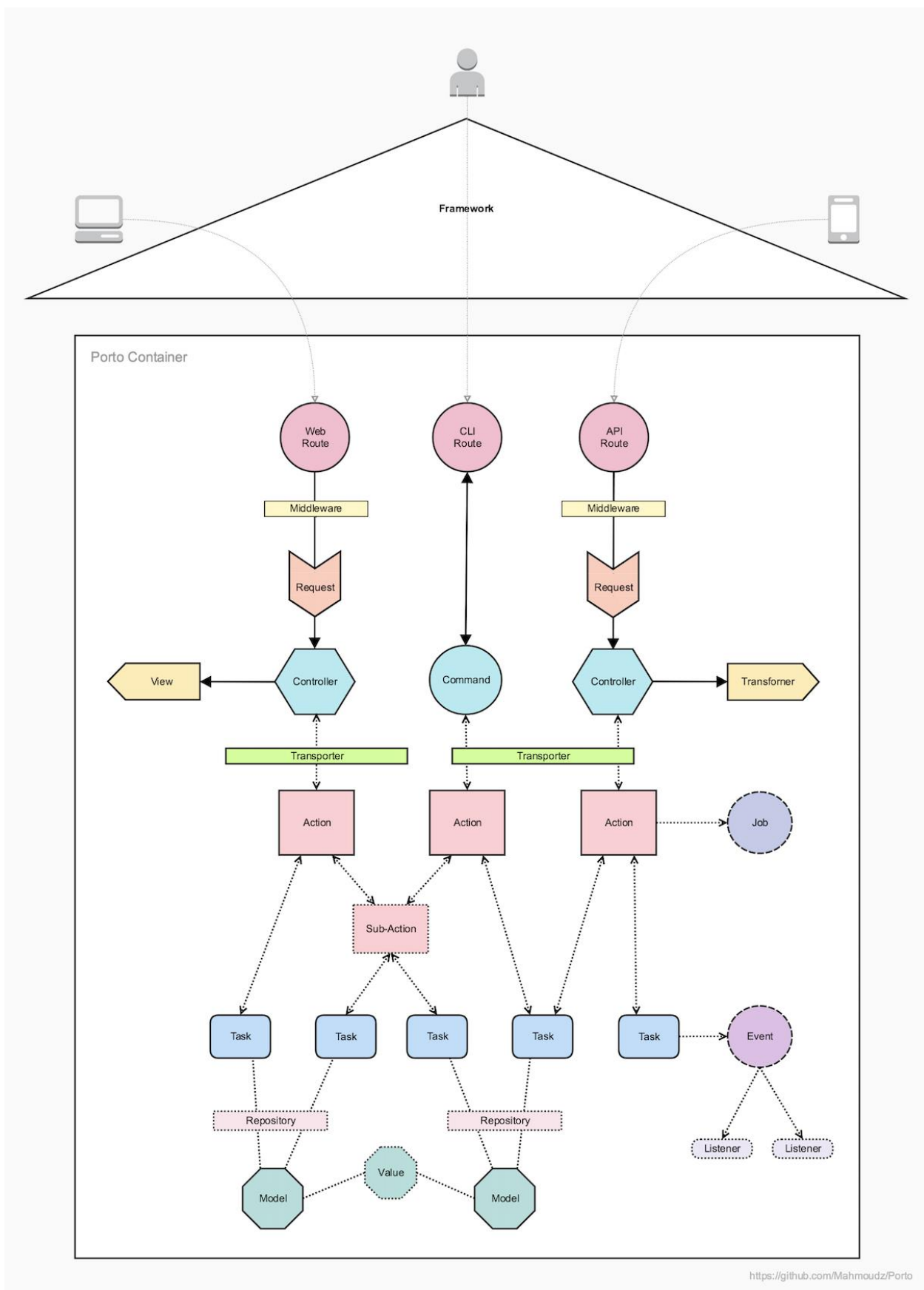


Рисунок 2.9 – Діаграма залежності основних елементів архітектурного шаблону Porto

2.3.4 Вибір засобів розробки клієнтської частини

При розробці клієнтської частини веб-додатків вибір інструментів завжди базується на використанні HTML як основи для структурування веб-сторінок. Водночас, для оформлення та стилізації цих сторінок застосовуються такі технології, як CSS, SCSS, SASS, Tailwind і інші, що дозволяють створювати привабливий та функціональний візуальний інтерфейс. Ці засоби стилізації допомагають розробникам надати унікальний характер та індивідуальність кожному веб-додатку, забезпечуючи йому відповідність до сучасних стандартів дизайну.

JavaScript, як основна мова програмування для розробки клієнтської частини веб-додатків, посідає провідне місце у світі веб-розробки. Її популярність та гнучкість, яка дозволяє втілювати складні функції взаємодії з сервером і управління даними, відіграють ключову роль у виборі JavaScript розробниками. У поєднанні з HTML та стилізаційними інструментами, JavaScript створює фундамент для високоякісних та зручних для користувача веб-додатків.

Однак, серед основних недоліків JavaScript слід виділити відсутність строгої типізації. Це може призводити до помилок у роботі програм, ускладнює відстеження помилок та може становити ризик для надійності коду. Динамічна типізація в JavaScript з одного боку є гнучкою, але з іншого - може стати джерелом неочікуваних помилок та ускладнювати процес відладки коду [37].

Важливо відзначити, що для вирішення проблеми динамічної типізації існує мова TypeScript. Ця мова є надбудовою до JavaScript і вносить строгую типізацію, що дозволяє підвищити надійність та передбачуваність коду. Тим не менш, TypeScript має свій недолік - потребу в перекомпіляції в нативний JavaScript для роботи у браузері. Проте цей недолік не є значним завдяки наявності численних рішень типу пакувальники, такі як Vite, Webpack та багато інших, які автоматизують цей процес. До того ж, будь-які клієнтські фреймворки і без того потребують переводу кодової бази як на JavaScript, так і на TypeScript на нативний JavaScript для роботи у браузері, тому використання пакувальника є необхідним і неминучим.

В цілому, TypeScript виправляє ключовий недолік JavaScript, пов'язаний із динамічною типізацією, одночасно зберігаючи всі переваги JavaScript. Це робить TypeScript привабливим вибором для розробників, які прагнуть до більшої стабільності та надійності свого коду. Таким чином, TypeScript та JavaScript разом формують міцну основу для сучасної веб-розробки, пропонуючи розробникам широкі можливості для створення якісних і функціональних веб-додатків [38].

2.3.4 Вибір клієнтського фреймворку

У даному підрозділі ми зосередимося на ключовому питанні вибору технологічного стеку для клієнтської частини веб-додатку. На сьогоднішній день світ веб-розробки пропонує значний вибір клієнтських фреймворків, кожен з яких вирізняється своїми унікальними особливостями, перевагами та потенційними обмеженнями. Це створює великі можливості для розробників, але водночас ставить перед ними складне завдання вибору найбільш оптимальних інструментів для конкретного проекту.

В рамках аналізу буде розглянуто три з найпопулярніших клієнтських фреймворків, кожен з яких базується на мові програмування JavaScript або її суперсеті – TypeScript, а саме Angular.js, React.js та Vue.js. Ці фреймворки були обрані через їх широке використання у сучасній веб-розробці та здатність забезпечити високий рівень продуктивності, масштабованості та гнучкості.

Angular.js, як один із провідних фреймворків для розробки веб-додатків, заслужено займає своє місце серед популярних інструментів в сучасному світі веб-розробки. Він пропонує ряд значущих переваг, але й має деякі недоліки, які важливо враховувати при виборі технології для проекту.

Однією з основних переваг Angular.js є його комплексність та інтегрованість. Фреймворк надає розробникам великий набір інструментів та функцій з коробки, таких як двостороннє зв'язування даних, шаблонізація, залежності введення, а також вбудована підтримка AJAX. Це забезпечує високу продуктивність розробки та спрощує реалізацію складних веб-додатків.

Angular.js, який розроблений корпорацією Google, вважається одним з найбільш масштабованих та потужних фреймворків у сфері веб-розробки. Його

модульний підхід до розробки дозволяє створювати перевикористовувані компоненти, що можуть бути легко інтегровані у різні частини додатку. Ця гнучкість та модульність значно спрощують управління кодом, роблячи процес розробки більш організованим та ефективним.

Однією з ключових переваг Angular.js є його інтеграція з іншими технологіями та інструментами, яка відкриває широкі можливості для розробки різноманітних додатків. Angular.js використовує TypeScript як основну мову програмування, що додає строгу типізацію та вищий рівень абстракції, сприяючи написанню більш чистого та структурованого коду.

Проте, Angular.js має ряд недоліків, які варто враховувати. Основним з них є складність у навчанні та освоєнні, оскільки Angular.js включає велику кількість концептів та шаблонів проектування. Це може стати викликом для новачків або розробників, які не мають досвіду роботи з подібними фреймворками [39].

Щодо продуктивності, Angular.js іноді може бути повільнішим у порівнянні з іншими фреймворками, особливо у великих та складних додатках. Механізм двостороннього зв'язування даних, хоча й є потужним інструментом, може вимагати додаткових ресурсів для обробки, що може призвести до зниження продуктивності.

Іншим недоліком є складність інтеграції з деякими зовнішніми бібліотеками та інструментами. Angular.js пропонує багато вбудованих функцій, але інтеграція зі сторонніми інструментами може потребувати додаткових налаштувань та розуміння того, як найкраще їх використовувати в контексті Angular.js.

Загалом, Angular.js є потужним інструментом для розробки веб-додатків, який пропонує розробникам багато функціональних можливостей та гнучкість у розробці. Однак, це також фреймворк, який вимагає значного часу для навчання та глибокого розуміння його архітектури, щоб ефективно його використовувати.

React.js, розроблений командою Facebook, є одним з найважливіших фреймворків у сучасному світі веб-розробки. Він пропонує сучасні підходи до створення інтерактивних користувацьких інтерфейсів, що значно спрощує процес розробки веб-додатків. Основою успіху React.js є його компонентний підхід, який

дозволяє розробникам створювати інкапсульовані компоненти, що управляють своїм станом, а потім складати їх у складні інтерфейси.

Однією з ключових переваг React.js є його декларативність. Коли розробники створюють інтерфейси за допомогою React, вони просто описують, як компоненти виглядають у різних станах. React потім ефективно оновлює та рендерить лише ті компоненти, які дійсно змінилися. Це відрізняється від традиційного підходу, де необхідно вручну вносити зміни в DOM, що часто може бути витратним і помилковим.

Ще однією важливою перевагою є використання віртуального DOM, який дозволяє React бути дуже швидким. Віртуальний DOM – це легка копія реального DOM, і оновлення в ньому відбуваються набагато швидше, ніж у реальному DOM. Це знижує кількість взаємодій з реальним DOM, які є дорогавартісними з точки зору продуктивності [40].

Проте, React.js має деякі недоліки. Наприклад, управління станом у великих застосунках може стати складним. У міру зростання додатку, кількість компонентів і взаємодій між ними збільшується, що може ускладнити управління станом. Це змушує розробників вдаватися до додаткових бібліотек, таких як Redux, для більш ефективного управління станом на рівні додатку.

Ще одним важливим недоліком є той факт, що React є лише бібліотекою, а не повноцінним фреймворком. Це означає, що для побудови повнофункціонального додатку з React, розробникам часто потрібно інтегрувати інші бібліотеки та інструменти, що може призводити до складностей із сумісністю та потребує більш глибокого розуміння екосистеми.

Незважаючи на ці виклики, React.js продовжує бути одним з найбільш популярних інструментів для розробки веб-додатків. Його здатність до швидкого рендерингу, модульність компонентів та гнучкість інтеграції з іншими бібліотеками роблять його ідеальним вибором для багатьох проектів веб-розробки.

Звертаючи увагу на ці аспекти, можна сказати, що React.js - це потужний інструмент для створення сучасних веб-додатків, який пропонує розробникам гнучкість, ефективність і широкі можливості для реалізації різноманітних проектів.

Vue.js, створений Еваном Ю, є одним із провідних фреймворків у світі веб-розробки, здобувши значну популярність серед розробників завдяки своїй легкості та гнучкості. Цей фреймворк відомий своєю простотою у навчанні та ефективністю у використанні, що робить його привабливим вибором для розробників, особливо для тих, хто лише починає свій шлях у веб-розробці.

Однією з основних переваг Vue.js є його легка інтеграція та швидкий старт. Цей фреймворк можна легко додати до існуючого проекту, і він не вимагає складних налаштувань, щоб почати роботу. Крім того, Vue.js пропонує чистий та інтуїтивно зрозумілий синтаксис, який сприяє швидкому розвитку продуктивності розробників.

Іншою перевагою Vue.js є його реактивність та гнучкість. Фреймворк використовує систему реактивних даних та віртуальний DOM, що автоматично оновлює UI відповідно до змін у даних, тим самим зменшуючи необхідність в ручному оновленні DOM. Така реактивність робить Vue.js ідеальним для створення інтерактивних користувацьких інтерфейсів.

Однак, Vue.js має деякі обмеження та недоліки. Наприклад, через його легкість і гнучкість, Vue може не завжди бути найкращим вибором для великих та складних додатків, де потрібна більш строга структура та керування станом. У таких випадках, використання Vue у поєднанні зі складнішими інструментами управління станом може стати необхідністю [41].

Іншим недоліком є відносно мала спільнота порівняно з такими фреймворками, як React чи Angular. Це може впливати на кількість доступних ресурсів для навчання та підтримки, хоча спільнота Vue.js швидко розвивається.

У підсумку, Vue.js є відмінним вибором для тих, хто шукає легкий та гнучкий фреймворк для швидкої розробки веб-додатків. Його простота у використанні та висока продуктивність роблять його привабливим для малих та середніх проектів, хоча для великих проектів може знадобитися більш комплексний підхід.

На підставі проведеного аналізу трьох вивчених фреймворків - Angular, React та Vue - можна зробити висновок, що кожен із них має свої унікальні переваги та недоліки. Angular, як повнофункціональний фреймворк, може виявитися надто

складним для тестового проекту з огляду на його масштабність та високий поріг входження. React, хоча й є гнучким та популярним вибором, потребує від розробників залучення додаткових бібліотек для повноцінної функціональності, що може додати складності у процесі розробки.

Vue, з іншого боку, виокремлюється своєю простотою, легкістю використання та гнучкістю, забезпечуючи достатній рівень структурованості без зайвої складності. Ці характеристики роблять Vue особливо привабливим для тестових проектів, які з часом можуть перетворитися на комерційні. Vue також пропонує збалансований підхід до масштабування, що є важливим аспектом при переході від тестової стадії до комерційного використання.

Враховуючи ці аспекти, вибір на користь Vue виглядає найбільш виправданим для розробки тестових та потенційно комерційних проектів. Його легкість, гнучкість та можливості масштабування забезпечують оптимальне середовище для швидкої розробки, ефективного тестування та успішного комерційного запуску проекту.

2.4 Висновок з розробки системи

У другому розділі пояснювальної записки було здійснено комплексний аналіз, присвячений вибору системи для розробки проекту. Основна увага була приділена вибору архітектури додатку та системи нейронних мереж.

Спершу було зосереджено увагу на виборі загальної архітектури додатку. Враховуючи динаміку розвитку програмного забезпечення та різноманітність сучасних технологій, були розглянуті такі архітектурні шаблони, як монолітна, клієнт-серверна (Service-Oriented Architecture) та мікросервісна архітектура. Ці моделі були вибрані через їх поширеність та ефективність. Кожна з них має свої специфічні переваги та обмеження, які були детально проаналізовані. За результатами аналізу, клієнт-серверна архітектура була визначена як найбільш

відповідна для даного проекту. Вона забезпечує високий рівень гнучкості, стабільності та ефективності, дозволяючи ефективно масштабувати та оновлювати систему. Чітке розділення відповідальностей між сервером та клієнтською частиною оптимізує процес розробки та підтримку системи, дозволяючи розробникам фокусуватися на конкретних компонентах.

Далі було здійснено вибір нейронної мережі для проекту. З огляду на технічні умови та необхідність ресурсів для створення власної нейронної мережі, було прийнято рішення використовувати сторонні сервіси. Такий підхід дозволяє використовувати переваги готових навчених мереж без потреби у самостійному розгортанні складних систем. У ході аналізу були розглянуті такі сервіси, як API від OpenAI, TensorFlow, Google Cloud AI та Microsoft Azure AI. За результатами цього аналізу, API від OpenAI було обрано як найбільш підходяще через його простоту використання, економічну ефективність та високу адаптивність до різних вимог проекту.

Після визначення архітектури системи, було проведено вибір засобів розробки. Були проаналізовані такі серверні фреймворки, як Spring для Java, Laravel для PHP та Express для JavaScript. Кожен з цих фреймворків має свої унікальні характеристики та переваги. Однак, у результаті було вибрано Laravel, оскільки він забезпечує оптимальне співвідношення гнучкості, структурованості та масштабованості, що є важливими для розробки як тестових, так і комерційних проектів.

Для серверної частини було обрано архітектурний шаблон Porto, спеціально розроблений для вирішення проблем обробки даних та масштабування системи. Porto забезпечує високий рівень модульності та гнучкості, що є ключовими для розробки великих та складних систем.

На завершення, для клієнтської частини було обрано Vue.js. Цей фреймворк відзначається простотою, легкістю використання та гнучкістю, забезпечуючи достатній рівень структурованості без надмірної складності. Vue.js виявився ідеальним для тестових проектів, що можуть трансформуватися у комерційні, забезпечуючи збалансований підхід до масштабування.

Загалом, вибір Laravel як серверного фреймворку, шаблону Porto для архітектури серверної частини та Vue.js як клієнтського фреймворку став збалансованим рішенням, що відповідає вимогам проекту у плані гнучкості, структурованості та масштабованості. Ці інструменти забезпечують ефективну та гнучку платформу для розробки як тестових, так і комерційних проектів, сприяючи плавному переходу від тестової стадії до повноцінної комерційної системи.

3 ДОСЛІДЖЕННЯ СИСТЕМИ

Розробка веб-систем є важким завданням, яке зажадає значних ресурсів, особливо коли в процесі створення потрібно враховувати складну бізнес-логіку або використовувати знання експертів у певній галузі. Кожен компонент системи вимагає унікального підходу, детального аналізу та ретельного вивчення, щоб задовольнити конкретні потреби сфери її застосування. Однак, шляхом впровадження нейронних мереж, навчених на великих обсягах даних, можливо значно спростити цей процес. Ці мережі мають здатність ефективно здійснювати аналіз даних, їх класифікацію, а також створення контенту. Вони пропонують більш гнучке рішення порівняно з традиційними підходами, що здійснюються вручну, і можуть бути адаптовані до потреб конкретного проекту.

Використання нейронних мереж у веб-розробці відкрило нові перспективи у багатьох галузях. Системи, засновані на цих технологіях, спроможні виконувати різноманітні ролі: від розробника до керівника проекту, надаючи високу гнучкість та мультифункціональність. Це стало можливим завдяки широкому спектру знань та навичок, якими володіють такі системи. Однак, важливо розуміти, що, незважаючи на свої переваги, вони можуть не завжди точно відповідати вимогам користувача, але є досить гнучкими, щоб адаптуватися та вдосконалюватися.

В цьому контексті, GPT відіграє ключову роль, оскільки ця система вирізняється унікальними можливостями у сфері створення та обробки контенту.

Вона не тільки може генерувати літературні твори та композиції, але й писати та оптимізувати алгоритми, роблячи її незамінним інструментом у різних областях. Її доступність у публічному секторі відкриває широкі можливості для використання штучного інтелекту, наближаючи нас до нової ери технологічних інновацій.

Оприлюднення специфікацій та API для використання мовної моделі OpenAI є важливим кроком у цьому напрямку. Це дозволило інтегрувати мовну модель у різноманітні системи, уникаючи необхідності створення власних нейронних мереж, що є часомістким та складним процесом. Така інтеграція значно підвищує ефективність використання штучного інтелекту, відкриваючи нові шляхи для вирішення завдань, які раніше вимагали багаторічних досліджень та великих людських ресурсів.

3.1 Актуальність впровадженого підходу

Актуальність впровадження підходу використання GPT-API в різноманітних сферах діяльності є високою, оскільки цей інструмент відкриває нові можливості для обробки інформації, оптимізації процесів та інновацій у рішеннях. З огляду на сучасні тенденції в області штучного інтелекту та автоматизації, GPT-API представляє собою значний крок уперед у розумінні та обробці природної мови, що має потенціал революціонізувати багато аспектів роботи з даними.

Використання GPT-API для прямого отримання даних дозволяє користувачам ефективно використовувати можливості обробки нейронною мережею. Це включає аналіз великих обсягів тексту, виявлення закономірностей, генерацію відповідей на запити та навіть створення нового контенту. Цей підхід значно прискорює обробку інформації, знижує витрати часу та ресурсів, порівняно з традиційними підходами обробки даних.

Альтернативно, використання GPT-API для поповнення баз даних на основі відповідей нейронної мережі відкриває шлях для заміни дорогих та часомістких

процесів співпраці з експертами. Це дозволяє збирати, структурувати та аналізувати знання в специфічних областях без необхідності безпосереднього залучення фахівців. Такий підхід має потенціал суттєво знизити витрати та спростити доступ до високоякісної інформації.

Крім того, прямий доступ до навченої нейронної мережі через GPT-API надає можливість для вдосконалення існуючих алгоритмів та розробки нових оптимізованих рішень. Використання передових технологій у цьому контексті може сприяти розвитку інновацій, підвищенню продуктивності та впровадженню нових підходів у різноманітних галузях, від бізнесу до освіти та наукових досліджень.

Враховуючи ці аспекти, актуальність впровадження GPT-API у сучасні системи обробки даних та прийняття рішень є значною, оскільки цей підхід забезпечує значні переваги в швидкості, гнучкості та ефективності в порівнянні з традиційними підходами.

Додатково до зазначених переваг використання GPT-API, варто підкреслити ще кілька аспектів, які збільшують його актуальність та цінність.

По-перше, використання GPT-API сприяє збільшенню доступності та демократизації штучного інтелекту. Малий та середній бізнес, стартапи, освітні установи та навіть індивідуальні користувачі отримують можливість ефективно використовувати передові технології, що раніше були доступні лише великим корпораціям.

По-друге, GPT-API може слугувати потужним інструментом для підвищення якості та точності аналізу даних. Це особливо важливо у сферах, де потрібно швидко та точно рішення, засноване на великих масивах інформації, як-от в медицині, фінансах або наукових дослідженнях [42].

По-третє, інтеграція GPT-API в системи автоматизації та роботизації відкриває нові горизонти для розвитку смарт-технологій. Нейронні мережі можуть значно підвищити інтелектуальні здібності роботизованих систем, забезпечуючи більш гнучке та адаптивне рішення завдань.

Крім того, GPT-API надає можливості для персоналізації та глибокого індивідуалізованого підходу у спілкуванні з клієнтами, що відкриває нові перспективи для маркетингу та CRM-систем. Це може підвищити рівень задоволення клієнтів та ефективності комунікації.

GPT-API може істотно сприяти розвитку освітніх технологій, пропонуючи адаптивні навчальні системи, що зорієнтовані на індивідуальні особливості та потреби студентів. Його впровадження відкриває нові горизонти для інновацій, оптимізації та підвищення ефективності у різних галузях, роблячи його важливим елементом у світі, де технології швидко змінюються.

Використання нейронних мереж у розробці веб-систем надає розробникам нові можливості для підвищення продуктивності та ефективності використання ресурсів, відкриваючи нову еру у веб-технологіях, де інновації та оптимальність стають ключовими. Це підкреслює актуальність дослідження переваг нейронних мереж порівняно з традиційними підходами програмування, особливо у контексті розвитку веб-систем та комплексних інформаційних систем, які вимагають неперервного удосконалення в підходах до обробки даних та рішення задач.

3.2 Порівняння використання нейронних мереж з традиційними підходами програмування

У контексті сучасного програмування, розробка систем на основі нейронних мереж відіграє ключову роль, пропонуючи значні переваги порівняно з традиційними підходами.

В контексті швидкості створення систем, нейронні мережі мають вагомі переваги порівняно з традиційними підходами програмування. Значна економія часу та ресурсів, необхідних для створення функціональної системи, є одним з ключових аспектів нейронних мереж. Це зумовлено здатністю нейронних мереж адаптуватися до нових даних, що сприяє оптимізації процесу розробки.

Автоматизоване виявлення закономірностей та самонавчання на основі вхідних даних забезпечують зменшення залежності від втручання людини-програміста у процесі кодування.

В той же час, використання нейронних мереж дозволяє зменшити потребу у детальному моделюванні проблеми, оскільки мережа здатна самостійно вивчати характеристики та зв'язки між даними. Це контрастує з традиційними підходами програмування, де необхідно точно визначити всі правила та логіку системи перед її розробкою, що вимагає значних часових витрат та глибокого розуміння проблеми.

Здатність нейронних мереж до обробки великої кількості можливих комбінацій параметрів є однією з їхніх важливих переваг у порівнянні з традиційними підходами програмування. Традиційне логічне програмування базується на жорстко визначених правилах, що обмежує здатність системи адекватно реагувати на різноманітні вхідні дані. Навпаки, нейронні мережі формують відповіді, ґрунтуючись на вхідних даних, що дозволяє їм працювати з значно ширшим і більш гнучким набором комбінацій параметрів.

Ця здатність особливо цінна у комплексних системах, де вхідні параметри можуть мати багатовимірний характер і змінюватися в широких межах. Нейронні мережі здатні виявляти складні взаємозв'язки між параметрами, що є майже неможливим для систем, що функціонують на основі фіксованих правил. Окрім цього, глибоке навчання, яке є частиною роботи нейронних мереж, забезпечує їм можливість адаптуватися та вдосконалюватися в процесі обробки нових даних.

У ситуаціях, де потрібна висока ступінь адаптивності та здатність до генерування прогнозів на основі неочевидних зв'язків, нейронні мережі значно перевершують традиційні підходи. Їхня гнучкість у визначенні взаємозв'язків дозволяє знаходити рішення у випадках, коли точне програмування всіх можливих сценаріїв є непрактичним або неможливим [43].

Перевага нейронних мереж у сфері масштабованості та адаптації є одним з ключових аспектів, що відрізняють їх від традиційних підходів програмування. Нейронні мережі здатні ефективно масштабуватися для обробки збільшуючихся

об'ємів даних, а також адаптуватися до нових даних чи змінних обставин. Така адаптивність знижує необхідність постійного втручання розробників, роблячи системи на базі нейронних мереж більш надійними та економічно ефективними в довгостроковій перспективі.

У контексті масштабованості, нейронні мережі демонструють здатність до обробки великих масивів даних без значної втрати продуктивності. Це стає особливо важливим у сучасному світі, де об'єми даних постійно зростають. Використання нейронних мереж дозволяє системам адекватно реагувати на різноманітні вхідні дані та швидко адаптуватися до змін у них.

Нейронні мережі мають особливі переваги в навчанні та моделюванні складних, нелінійних залежностей у даних. Ця здатність дозволяє їм розуміти та реагувати на складні взаємозв'язки у вхідних даних, що в кінцевому результаті призводить до поліпшення відповідей системи. У ситуаціях, де традиційні підходи виявляються неспроможними вловлювати ці складні залежності, нейронні мережі можуть надавати більш точні та відтінені оцінки.

Здатність до виявлення та аналізу складних залежностей у даних є критично важливою для різноманітних застосувань, від обробки природної мови до прогнозування фінансових тенденцій. Нейронні мережі, завдяки своїй структурі та принципам навчання, здатні виявляти зв'язки між даними, які можуть бути непомітними для традиційних алгоритмів.

Наприклад, у сфері прогнозування погодних умов, нейронні мережі можуть виявляти складні патерни, залежні від безлічі факторів, що змінюються в часі. Традиційні підходи, які спираються на чітко визначені математичні моделі, часто не можуть адекватно враховувати таку багатовимірність та динаміку змін.

З іншого боку, у сфері фінансового аналізу, нейронні мережі здатні виявляти тонкі закономірності на ринках, які можуть втрачатися у традиційних статистичних моделях. Це дає можливість генерувати більш точні прогнози та стратегії інвестування.

Нейронні мережі є особливо придатними для обробки непередбачуваних ситуацій завдяки їхній адаптивності та здібностям до навчання. На відміну від

жорстких, заснованих на правилах систем, нейронні мережі можуть коригувати свою поведінку та відповіді залежно від розвитку або неочікуваних вхідних даних. Ця адаптивність робить їх цінними в застосуваннях, де вхідні дані можуть змінюватися або містити непередбачені елементи.

Важливою особливістю нейронних мереж є їх здатність до самонавчання, що дозволяє системам ефективно адаптуватися до змінних або непередбачуваних умов. Це означає, що системи можуть вдосконалюватися та адаптуватися до нових умов без безпосереднього програмування кожної можливої зміни. Такий підхід не тільки збільшує ефективність обробки даних, але й зменшує витрати на розробку та підтримку системи.

Таким чином, нейронні мережі виявляються незамінними в умовах необхідності роботи з комплексними даними, що містять багатовимірні та нелінійні взаємозв'язки. Це досягається за рахунок постійного аналізу та вивчення вхідних даних, що дозволяє системі виявляти нові патерни та відповідно до цього коригувати свої відповіді.

Такий підхід суттєво відрізняється від традиційних, правилкових систем, які працюють на основі попередньо визначеного набору інструкцій і не можуть адаптуватися до нових або змінних умов без втручання розробника. Нейронні мережі, навпаки, здатні самостійно "вчитися" з досвіду, тобто з кожною новою вхідною даниною, вони можуть удосконалювати своє розуміння задачі та відповідно підлаштовувати свої відповіді.

Ця гнучкість і здатність до навчання роблять нейронні мережі надзвичайно корисними в різноманітних областях, таких як розпізнавання образів, автоматизоване водіння, прогнозування фінансових ринків або навіть управління роботами, де вхідні дані можуть бути непостійними або непередбачуваними.

Нейронні мережі використовують своє розуміння закономірностей та залежностей у даних для прогнозування майбутньої поведінки користувачів. Ця здатність до прогнозування є вкрай цінною для надання рекомендацій або підтримки прийняття рішень у різних сферах, таких як електронна комерція, рекомендація контенту та персоналізований користувацький досвід. Аналізуючи

історичні дані, нейронні мережі можуть надавати прозоріння, що підвищують залученість та задоволеність користувачів.

Прогнозування поведінки користувачів стає можливим завдяки здатності нейронних мереж розпізнавати складні закономірності та взаємозв'язки в масивах даних. Наприклад, в електронній комерції нейронні мережі можуть аналізувати покупкову поведінку користувачів, їхні попередні вибори та відгуки, щоб пропонувати персоналізовані рекомендації продуктів, які найімовірніше їх зацікавлять.

У сфері рекомендації контенту, такі системи можуть аналізувати переглядову історію та вподобання користувачів, щоб пропонувати фільми, музику або книги, які відповідають їхнім інтересам та перевагам. Це не тільки підвищує задоволеність користувачів, але й сприяє збільшенню часу, проведеного ними на платформі.

У контексті персоналізованого користувацького досвіду, нейронні мережі можуть використовувати інформацію про минулу поведінку користувачів для створення більш релевантного та цікавого інтерактивного досвіду, що знову ж таки підвищує залученість та задоволеність.

Таким чином, нейронні мережі відіграють ключову роль у покращенні користувацького досвіду шляхом надання персоналізованих рекомендацій та підтримки прийняття рішень, використовуючи свої унікальні здібності до аналізу великих об'ємів даних та виявлення складних закономірностей.

Нейронні мережі, такі як ChatGPT, також виявляють високий рівень майстерності в роботі з природною мовою. Вони демонструють здатність до розуміння та генерації тексту, схожого на людський, що робить їх цінними для завдань, таких як чат-боти, переклад мови та генерація контенту.

Використання нейронних мереж у роботі з природною мовою базується на складних алгоритмах машинного навчання, які аналізують великі об'єми текстових даних. Це дозволяє їм розпізнавати мовні структури, вчитися з контексту та використовувати отримані знання для створення відповідей, які відображають природність та релевантність людського спілкування.

У сфері чат-ботів, нейронні мережі використовуються для створення систем, здатних вести діалог з користувачами, відповідаючи на запитання, надаючи інформацію та навіть імітуючи емоційні реакції. Це покращує взаємодію між людьми та цифровими сервісами, роблячи її більш природною та зручною.

У сфері перекладу мови, нейронні мережі дозволяють створювати системи, які можуть швидко та точно перекладати тексти між різними мовами, що є важливим для глобалізації та міжкультурного спілкування.

Крім того, у генерації контенту, такі системи можуть автоматизувати створення текстів різних форматів, від новинних статей до літературних творів, що забезпечує широкі можливості для залучення аудиторії та економії часу та ресурсів.

Таким чином, нейронні мережі забезпечують значні переваги у роботі з природною мовою, розширюючи можливості автоматизації та інтерактивності в цифрових сервісах і створюючи нові горизонти для взаємодії між людиною та машиною.

Нейронні мережі мають здатність до безперервного навчання та вдосконалення зі збільшенням обробки даних. Зіткнувшись із додатковими даними та проаналізувавши їх, вони можуть удосконалювати свої моделі, що призводить до збільшення ефективності та точності з часом. Ця адаптивність є особливо корисною в динамічних середовищах, де ландшафт даних постійно еволюціонує.

Процес безперервного навчання нейронних мереж базується на їх здатності аналізувати вхідні дані, виявляти закономірності та коригувати свої внутрішні параметри відповідно до нової інформації. Це означає, що чим більше даних обробляється, тим точнішими та ефективнішими стають ці мережі.

У сферах, де динаміка даних є високою, наприклад у фінансовому прогнозуванні, управлінні ризиками або в системах моніторингу здоров'я, здатність нейронних мереж до самонавчання та адаптації є особливо цінною. Вони можуть вчасно реагувати на зміни, вносячи корективи до своїх прогнозів або відповідей, що забезпечує більшу точність та надійність в роботі.

Крім того, здатність до безперервного навчання дозволяє нейронним мережам розвиватися та адаптуватися до нових умов без необхідності їх повного

перепрограмування чи втручання з боку розробників. Це робить їх ідеальними для використання в умовах швидкозмінного технологічного світу, де здатність до швидкого адаптування та оновлення є ключовою для успіху.

Таким чином, безперервне навчання та вдосконалення нейронних мереж є основою їх ефективності у різноманітних застосуваннях, надаючи їм перевагу в адаптації до постійно змінюваних умов та вимог.

Залежність нейронних мереж від тренувальних даних є однією з їхніх значних обмежень. Вони є ефективними лише настільки, наскільки якісні та репрезентативні дані, на яких вони навчалися. При стиканні з новими або неповними даними, що істотно відрізняються від їхнього навчального набору, нейронні мережі можуть виробляти менш точні результати. Це вимагає періодичних оновлень та повторного навчання для підтримання їхньої актуальності та продуктивності. Варто відзначити, що подібні обмеження застосовуються і до традиційних логічних систем при включенні нових знань чи даних, оскільки вони також потребують ручних коригувань та оновлень для пристосування до змін.

Залежність від даних в нейронних мережах виявляється у двох аспектах: обмеження знань, на основі яких вони можуть навчатися, та зміщення, що виникають через нерепрезентативність або недостатність даних. Якщо набір даних, на яких нейронна мережа була навчена, не відображає різноманітності реальних умов, модель може демонструвати низьку ефективність або виробляти помилкові висновки у нових ситуаціях.

У той же час, традиційні логічні системи також стикаються з проблемами пристосування до нових даних або змінених умов. Їхні алгоритми зазвичай базуються на чітко визначених правилах та інструкціях, які потребують ручного оновлення або коригування для врахування нових даних чи інсайтів.

Ці обмеження підкреслюють важливість якісного збору даних та процесів їх аналізу, що є критичними як для нейронних мереж, так і для традиційних систем. Забезпечення неперервного оновлення та вдосконалення цих моделей є ключовим для підтримки їхньої точності та надійності у швидко змінюваних умовах сучасного світу.

Ризик перенавчання також є важливою проблемою в контексті нейронних мереж, хоча при використанні вже належно навчених моделей, таких як GPT, ймовірність такого явища значно зменшується. Перенавчання відбувається, коли нейронна мережа надто точно адаптується до тренувальних даних, втрачаючи здатність ефективно працювати з новими, не баченими раніше даними. Це може призвести до зниження універсальності та загальної корисності моделі.

Однак, у випадку з GPT та іншими розвиненими нейронними мережами, які вже пройшли інтенсивні тренування та оптимізацію, ризик перенавчання значно знижується. Такі моделі зазвичай тренуються на величезних наборах даних, які охоплюють широкий спектр тем та ситуацій, забезпечуючи їм можливість узагальнення та адаптації до нових даних.

Крім того, при розробці таких моделей використовуються передові методи, такі як регуляризація та валідація на різних наборах даних, що допомагає запобігти перенавчанню. Регуляризація обмежує складність моделі, запобігаючи її занадто деталізованому "запам'ятовуванню" тренувальних даних, тоді як валідація на різних наборах даних забезпечує перевірку здатності моделі ефективно працювати у різних умовах.

У контексті традиційного програмування, ризик перенавчання не є таким виразним, оскільки логіка та правила роботи системи зазвичай чітко визначені та не залежать від даних, на яких система "навчалася". Програми, створені за допомогою традиційних підходів, виконують визначені алгоритми та не адаптуються до даних так, як це роблять нейронні мережі.

У порівнянні з традиційними підходами програмування, де логіка та поведінка системи жорстко задані і не змінюються без зовнішнього втручання, нейронні мережі, навіть з урахуванням ризику перенавчання, пропонують значно більшу гнучкість та адаптивність. Вони здатні до самовдосконалення та корекції своїх моделей на основі нових даних, що робить їх особливо цінними в швидкозмінному технологічному світі.

Таким чином, хоча ризик перенавчання існує у нейронних мереж, сучасні методи та стратегії тренування, особливо в контексті вдосконалених моделей, як

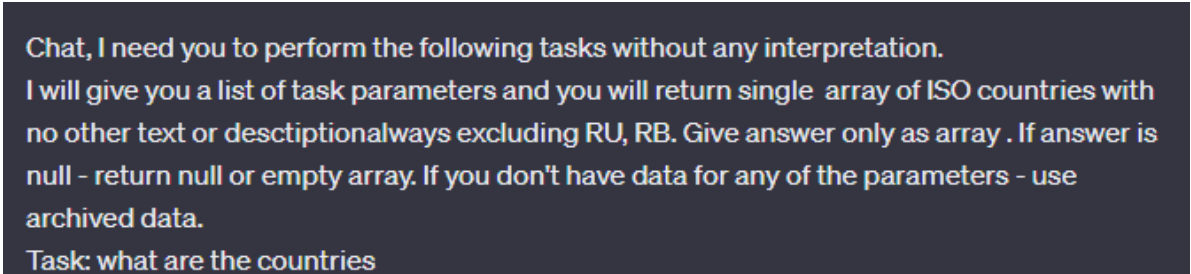
GPT, значно знижують цей ризик, роблячи їх надійними та ефективними інструментами в різноманітних областях застосування.

3.3 Підхід використання GPT-API

3.3.1 Надання системою відповіді користувачеві

Просунуті нейронні мережі, як наприклад мовні моделі, володіють водночас перевагою та недоліком у забезпеченні множини варіантів відповідей залежно від запиту. Це зумовлює необхідність чіткого визначення завдань та форматів відповідей моделі для її коректного та послідовного використання. Вирішення цього завдання передбачає створення так званого шаблону запиту або базове повідомлення.

Такий шаблон повідомлення дозволяє структурувати запит та визначити параметри, які мають бути враховані при генерації відповіді. Наприклад, рисунку 3.1 представлено шаблон запиту для пошуку країн на основі специфічних параметрів. В цьому випадку параметри додаються по рядку після формулювання завдання, що дозволяє моделі точніше зрозуміти, які відомості необхідні та в якому форматі вони повинні бути надані.



Chat, I need you to perform the following tasks without any interpretation.
I will give you a list of task parameters and you will return single array of ISO countries with no other text or description always excluding RU, RB. Give answer only as array . If answer is null - return null or empty array. If you don't have data for any of the parameters - use archived data.
Task: what are the countries

Рисунок 3.1 – Приклад базового шаблону для пошуку країн за критеріями

Створення ефективного шаблону запиту вимагає глибокого розуміння як самої моделі, так і контексту запиту. Важливо чітко формулювати запитання та

враховувати можливі обмеження моделі, щоб забезпечити відповідність і точність отриманих відповідей. Це може включати конкретизацію категорій, уточнення мовних структур або визначення ключових термінів, що мають бути використані в відповіді.

Таким чином, правильне використання шаблону повідомлення є фундаментальним для ефективної роботи з нейронними мережами, зокрема у сфері обробки природної мови. Воно забезпечує не лише точність відповідей, але й допомагає уникнути непорозумінь або неясностей, що можуть виникнути внаслідок багатоваріантності відповідей, яку пропонують сучасні нейронні мережі.

На рисунку 3.2 представлено запит для визначення країн, що знаходяться близько до вказаних координат із температурою понад 50 градусів Цельсія. Серед відповідних країн знаходимо такі, як Казахстан, Азербайджан, Іран, Саудівська Аравія, Ірак, Кувейт та Об'єднані Арабські Емірати, кожна з яких задовольняє поставлені умови.

```
Task: what are the countries
nearest to latitude: 47.901 ,longitude: 35.143 .
and temperature is higher 50 Celsius

["KZ", "AZ", "IR", "SA", "IQ", "KW", "AE"]
```

Рисунок 3.2 – Приклад запиту для визначення країн за двома параметрами

На рисунку 3.3 запит аналогічний до першого, але з використанням іншого діапазону температур — від 20 до 50 градусів Цельсія. Відповідно до цього запиту, визначено країни, більшість з яких розташовані в Європі, на Близькому Сході та в Африці.

```
Task: what countries are
nearest to latitude: 47.901    longitude: 35.14
and temperature is between 20 and 50 Celsius

["UA", "BY", "MD", "RO", "TR", "GR", "AL", "MK", "RS", "BG", "HR", "SI", "HU", "SK", "CZ", "PL", "LI",
"AT", "CH", "DE", "LU", "BE", "NL", "FR", "MC", "ES", "PT", "MA", "DZ", "TN", "LY"]
```

Рисунок 3.3 – Приклад запиту для визначення країн за двома параметрами

На рисунку 3.4 проілюстровано запит, розширений додатковою умовою про використання євро як місцевої валюти. У результаті цього запиту більшість країн з другого списку, що знаходяться в Європі, відповідають новому критерію, водночас країни з Близького Сходу, Північної Африки та Україна були виключені.

```
Task: what countries are
nearest to latitude: 47.901    ,longitude: 35.143
and temperature is between 20 and 50 Celsius
and currency is Euro

["DE", "FR", "BE", "LU", "NL", "ES", "PT", "IT", "GR", "AT", "SK", "SI", "ME", "EE", "LV", "LT", "IE",
"CY", "MT", "FI"]
```

Рисунок 3.4 – Приклад запиту для визначення країн за трьома параметрами

Ці приклади ілюструють важливість точного визначення параметрів запиту при роботі з нейронними мережами, адже навіть невеликі зміни можуть істотно вплинути на кінцевий результат. Вони також підкреслюють переваги адаптивності нейронних мереж, які дозволяють гнучко реагувати на різні запити, відповідаючи на конкретні потреби користувачів.

На рисунку 3.5 зображено базовий запит із єдиним критерієм – температурою між 20 та 50 градусами Цельсія. Відсутність умови для визначення країн, розташованих поряд із поточним місцем, призводить до значного розширення списку, що включає країни з усього світу. Це показує, як пропуск або модифікація

окремих критеріїв може розширити обсяг і різноманітність видачі даних за запитом.

```
Task: what countries are
temperature is between 20 and 50 Celsius

["AF", "DZ", "AO", "AR", "AU", "BD", "BF", "BJ", "BO", "BR", "BT", "BW", "CF", "CG", "CH", "CI", "CL",
"CM", "CN", "CO", "CR", "CU", "DJ", "DO", "DZ", "EC", "EG", "ER", "ET", "GA", "GH", "GM", "GN",
"GQ", "GT", "GW", "GY", "HN", "ID", "IN", "IQ", "IR", "JM", "JO", "KE", "KH", "KM", "KW", "LA", "LB",
"LR", "LY", "MA", "MG", "ML", "MM", "MR", "MT", "MX", "MY", "MZ", "NA", "NE", "NG", "NI", "NP",
"OM", "PA", "PE", "PH", "PK", "PR", "PS", "PY", "QA", "SA", "SD", "SG", "SL", "SN", "SO", "SS", "SV",
"SY", "TD", "TG", "TH", "TN", "TT", "TW", "TZ", "UG", "US", "VE", "VN", "YE", "ZA", "ZM", "ZW"]
```

Рисунок 3.5 – Приклад запиту для визначення країн за одним параметром

На рисунку 3.6 показано впровадження нового параметра – типу клімату. Інтеграція такого параметра вже існуючої системи потребуватиме додавання цього нового параметра в базу даних та актуалізацію кожного запису, шляхом отримання інформації з інших джерел або вручну. Зазвичай у ситуаціях програмування, програміст міг би створити не просто булевий тип поля для тропічного клімату, але й поле клімату для введення конкретних текстових значень, що може призвести до помилок через відсутність строгої типізації.

```
Task: what countries are
temperature is between 20 and 50 Celsius
and climate is tropical

["BR", "ID", "CD", "PH", "CO", "MY", "VE", "PE", "MM", "NG", "TH", "VN", "MG", "EC", "GH", "CI",
"PG", "CM", "UG", "TM", "LA", "PY", "SN", "MZ", "LK", "KH", "BD", "LR", "DO", "HT", "CR", "BS", "FJ",
"GY", "PA", "TT", "JM", "MU", "BW", "SL", "GW", "GM", "ET", "ML", "GA", "NE", "MR", "BF", "DJ",
"GN", "SZ", "SO", "GW", "TL", "CV", "SC", "ST", "BB", "AG", "LC", "VC", "GD", "DM", "BN", "SG",
"MU", "GY", "BB", "GD", "VC", "DM", "LC", "AG", "KN", "TT", "SR", "BZ", "JM"]
```

Рисунок 3.6 – Приклад запиту для визначення країн з новим параметром

Застосування GPT надає можливість додавати нові параметри у будь-якому порядку, формуючи легко модифіковану та адаптивну систему, яка не потребує

тривалих і складних змін, удосконалень чи виправлень помилок. Це спрощує процес роботи з даними, забезпечуючи гнучкість та ефективність відповідно до змінних умов та потреб. Таким чином, GPT відкриває нові можливості для розвитку і удосконалення інформаційних систем, що легко адаптуються до нових вимог та обставин.

На рисунках 3.7 та 3.8 проілюстровано приклад введення параметра, який не міг бути передбачений на етапі проектування системи. У даному випадку мова йде про діапазон років, протягом яких відбувалися або не відбувалися терористичні атаки.

```
Task: what countries are
temperature is between 20 and 50 Celsius
and currency is Euro
and terrorist attacks were carried out between 2001 and 2016

["FR", "DE", "ES", "BE", "IT", "NL", "GR", "PT", "IE", "AT", "FI", "LU", "EE", "LV", "LT", "SK", "SI", "CY",
"MT"]
```

Рисунок 3.7 – Приклад запиту для визначення країн із непередбачуваними параметрами

```
Task: what countries are
temperature is between 20 and 50 Celsius
and currency is Euro
and no terrorist attacks were carried out in 2013

["MT", "CY", "EE", "LV", "LT"]
```

Рисунок 3.8 – Приклад запиту для визначення країн із непередбачуваними параметрами

Аналізуючи відповіді моделі, можна побачити, що вона адекватно реагує на введення такого параметра та повертає список країн, які справді відповідають цим

критеріям. Це демонструє гнучкість та адаптивність систем, заснованих на нейронних мережах, у відповіді на непередбачені вимоги або параметри.

Застосування такого підходу може мати значні переваги у різних сферах, зокрема у сфері безпеки, аналітики даних, соціальних досліджень та багатьох інших. Воно дозволяє системам швидко адаптуватися до нових викликів і змін у вимогах, не вимагаючи глибокого перепроєктування або повного переналаштування. Така можливість робить нейронні мережі, як-от GPT, незамінними інструментами для сучасних динамічних і змінних умов, де швидкість відповіді та гнучкість системи є ключовими факторами успіху.

Це також відкриває шлях для більш глибокого аналізу та розуміння складних питань, оскільки можливість включення різноманітних та специфічних параметрів може допомогти виявити нові зв'язки та інсайти, які були недоступні при більш традиційних підходах до обробки даних. Таким чином, використання нейронних мереж, таких як GPT, забезпечує не лише адаптивність та гнучкість, але й глибину та багатство аналітичних можливостей.

3.3.2 Аналіз підходу використання GPT-API для надання відповіді користувачеві

У попередніх прикладах було продемонстровано підхід до взаємодії кінцевого користувача з нейронною мережею через добре сформульований автоматичний запит. Особливість цієї взаємодії полягає у використанні унікальних параметрів у кожному запиті, що вказує на неможливість передбачення та запису всіх потенційних відповідей у базу даних. Ця особливість підкреслює, що запити були спрямовані саме на задоволення потреб кінцевих користувачів, а не на збір та аналіз даних для внутрішнього використання.

Це призводить до висновку, що припустима система може функціонувати без великої бази даних для зберігання контенту, обмежуючись лише системною інформацією про користувачів. Такий підхід дозволяє значно спростити та прискорити процес розгортання системи, оскільки всі необхідні дані забезпечуються зовнішнім сервісом, який в даному випадку виступає як серверна частина з усіма потрібними даними.

Така модель роботи нейронної мережі є важливою для розробки та впровадження сучасних інформаційних систем, які орієнтовані на задоволення конкретних запитів користувачів. Вона забезпечує високий рівень гнучкості та адаптивності, дозволяючи системі швидко реагувати на змінні потреби користувачів без необхідності постійного оновлення великої бази даних. Таким чином, використання нейронних мереж у цьому контексті відкриває нові можливості для ефективного управління даними та взаємодії з користувачами, пропонуючи рішення, які можуть бути швидко адаптовані до різних сценаріїв використання.

Для оцінки кількості людино-годин, необхідних для створення веб-системи туристичного довідника з описаними функціями стандартними підходами, слід врахувати комплексний підхід до розробки, що охоплює аналіз вимог, проектування системи, розробку, тестування, впровадження та набір бази даних.

У процесі аналізу вимог здійснюється визначення функціональних та нефункціональних вимог системи, включаючи збір вимог до кожного запиту та аналіз можливих джерел даних, що вимагає приблизно 40-60 людино-годин.

Проектування системи передбачає розробку архітектури, вибір технологій та проектування інтерфейсу користувача, на що може бути витрачено від 80 до 120 людино-годин.

Розробка, яка включає створення бази даних, серверної логіки та клієнтської частини, потребує значних зусиль, оцінюваних у 400-600 людино-годин.

Тестування забезпечує надійність та коректність роботи системи, вимагаючи приблизно 100-150 людино-годин.

Впровадження та підтримка системи, включаючи розгортання на сервері та моніторинг, займають близько 50-100 людино-годин.

Окремо слід виділити набір бази даних, який може включати вручну введення даних або автоматизований парсинг з інших ресурсів. Цей процес може варіюватися в широких межах в залежності від обсягу та доступності даних, вимагаючи додаткових 200-400 людино-годин.

Таким чином, загальна оцінка для створення такої системи становить приблизно 1000-1400 людино-годин, умовно, залежно від різних факторів, таких як складність вимог, досвід команди та використані технології.

Використання нейронних мереж у якості основи для веб-системи туристичного довідника, як наведено вище, може істотно трансформувати підхід до розробки та експлуатації такої системи. Перевагою такого підходу є зменшення вимог до структури та обсягу роботи, які традиційно накладаються на систему.

У випадку використання нейронної мережі, система отримує можливість динамічно обробляти запити користувачів, що значно спрощує процес аналізу вимог. Зокрема, умови, за якими користувачі формулюють свої запити, стають більш гнучкими. Така автоматизація може скоротити час, витрачений на аналіз вимог, майже наполовину, оскільки більшість запитів та їх обробка будуть динамічно адаптовані системою.

Процес проектування системи також зазнає значних змін. Необхідність розробки лише "системної" частини системи може знизити час проектування в 3-4 рази, оскільки більшість ресурсів та зусиль може бути зосереджено на реалізації основних функціональних можливостей, не витрачаючи час на деталізацію обробки контенту.

Розробка повноцінної системи при цьому може бути значно прискорена. Оскільки основна частина роботи з контентом перекладається на нейронну мережу, час на розробку може скоротитися в 12-15 разів порівняно з класичним підходом. Це означає, що найбільш трудомісткі аспекти роботи з контентом, такі як його аналіз, сортування та презентація, виконуються за допомогою автоматизованих систем на базі штучного інтелекту.

Тестування системи, хоча й залишається важливим компонентом розробки, також може бути спрощено. Оскільки 80-95% функціоналу забезпечується зовнішнім сервісом, основна увага при тестуванні зміщується на інтеграцію системи з нейронною мережею та взаємодію з користувачем.

Що стосується впровадження та підтримки, хоча загальні зусилля не можуть бути значно зменшені, використання динамічних параметрів та бази знань

нейронної мережі замість традиційної бази даних знижує потребу в постійному оновленні та модифікації контенту.

Особлива економія часу досягається за рахунок відсутності необхідності наповнення власної бази даних. У цьому випадку база даних контенту є частиною бази знань нейронної мережі, яка вже містить велику кількість інформації. Усі ці показники зібрано у таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика етапів проектування двох підходів

	Стандартний підхід	Використання GPT-API
Аналіз вимог	40-60	20-30
Проектування системи	80-120	20-40
Розробка системи	400-600	50-100
Тестування	100-150	30-60
Впровадження та підтримка	50-100	30-50
Набір бази даних	200-400	0
Σ	900-1400	150-300

Проаналізувавши таблицю 3.1 можна екстраполювати графіки порівняння часових затрат від часу для обох підходів за допомогою TableCurve2D. Ці графіки зображено на рисунку 3.9.

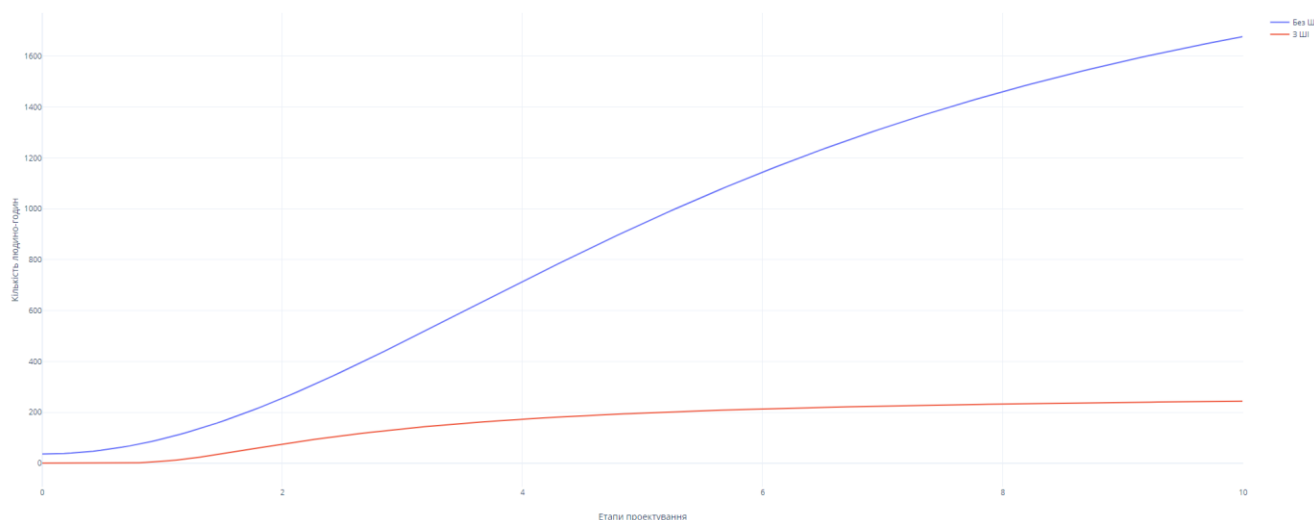


Рисунок 3.9 – Екстрапольовані графіки порівняння часових затрат від часу для обох підходів

Маючи екстрапольовані графіки можна також екстрапольовати графік вивільнення часу в залежності від етапу проектування. Цей графік зображено на рисунку 3.10.

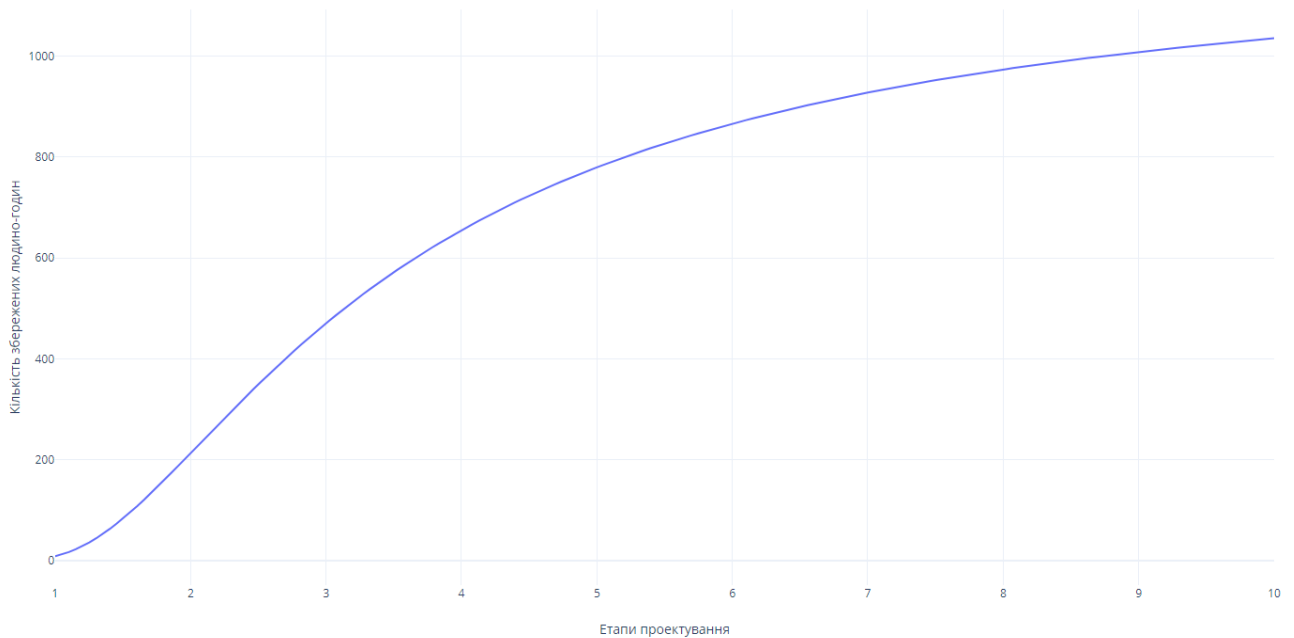


Рисунок 3.10 – Графік вивільнення часу в залежності від етапу проектування

Таким чином, загальний час, необхідний для розробки повноцінної системи, може зменшитися з 900-1400 людино-годин, які зазвичай необхідні при використанні стандартної програмованої логіки, до 150-300 людино-годин, завдяки використанню можливостей нейронних мереж, що відповідає прискоренню розробки системи у 5-7 разів.

Важливо зазначити, що витрати часу на розробку залежать не лише від обсягу задач, але й від їх складності. Створення більш складної системи, що включає новий набір параметрів та функціоналів, може вимагати непропорційно більше часу. Це обумовлено складністю інтеграції нових компонентів, необхідністю додаткових тестувань та адаптації системи до змінених умов.

Також слід врахувати, що застосування GPT-API для генерації відповідей несе в собі постійні витрати. Використання, наприклад, gpt-3.5-turbo має певну вартість, яка становить приблизно 0.00150 долара США за кожні 1000 токенів, що

еквівалентно приблизно 750 словам на запит. Це означає, що обробка 100 000 запитів вимагатиме витрат у розмірі 150 доларів США. Ці витрати стають значущими при великому обсязі оброблюваних запитів, що вимагає фінансових ресурсів.

Одним із способів вирішення цього питання є запровадження системи платних підписок для користувачів веб-системи. За такою моделлю, користувач отримує певну кількість безкоштовних запитів – наприклад, 50 – після чого для отримання додаткових послуг необхідно оформити підписку, вартість якої може становити від 5 до 10 доларів США на місяць. Такий підхід не лише забезпечує покриття витрат на GPT-API за рахунок коштів, отриманих від підписок, але й може приносити прибуток. Адже навіть за мінімальною підпискою у 5 доларів США, користувачу потрібно здійснити понад три тисячі запитів, щоб використати всю вартість підписки, що є доволі значною кількістю.

3.3.3 Надання системою відповіді для обробки системою

У процесі створення веб-системи туристичного довідника, використання GPT-API може відігравати ключову роль, яка виходить за рамки прямого взаємодії з кінцевими користувачами. Ця передова технологія нейронних мереж відкриває широкі можливості для генерації контенту людською мовою, що в свою чергу може бути збережено в базі даних веб-системи. Особливістю цього підходу є те, що після створення контенту за допомогою GPT-API, він проходить етап модерації. Цей етап включає перевірку, коригування та верифікацію інформації, що дозволяє забезпечити її точність та відповідність стандартам якості та моралі, перш ніж вона буде представлена кінцевому користувачеві. Незважаючи на потребу участі людини в процесі перевірки, ця робота не вимагає спеціалізованих знань, а тому є більш економічно вигідною порівняно з задіянням висококваліфікованих фахівців.

Цей підхід може стати ефективним рішенням для однієї з ключових проблем сучасної інформаційної діяльності – потреби у залученні експертів для генерації спеціалізованого контенту. Це стає особливо важливим у сферах, де необхідно створювати глибокі та деталізовані бази знань, часто потребуючи тривалого та витратного процесу наповнення від кваліфікованих фахівців. Особливо це

актуально для туристичних довідників, де інформація про культурні та історичні об'єкти, місцеві традиції, маршрути та інші нюанси має велике значення.

Розглядаючи приклад системи туристичного гіда, можна побачити, як традиційно бази знань таких систем вимагають значних часових витрат та експертизи для їх наповнення. Кожен запис, який зазвичай створюється експертами – гідами, має високу цінність завдяки деталізованому опису конкретних туристичних об'єктів. Використання ж GPT-API може істотно трансформувати цей процес, пропонуючи швидке генерування об'ємного контенту на основі вже наявних даних. Такий синтезований контент може бути ефективно оптимізовано та доповнено, знижуючи час та витрати на його створення.

Впровадження GPT-API у створення контенту для туристичного довідника не лише сприяє швидкому та ефективному наповненню бази даних, але й відкриває нові можливості для зниження залежності від ручної праці експертів. Це може вести до значного зниження витрат, а також підвищення продуктивності та доступності інформації для широкого кола користувачів. Таким чином, інтеграція передових технологій у сфері штучного інтелекту з відкриває нові горизонти для розвитку інформаційних систем, роблячи їх більш гнучкими, адаптивними та ефективними.

3.3.4 Аналіз результатів підходу використання GPT-API

Розглянемо, наприклад, використання цієї технології для детального опису підводного човна U-20, яке експонується у віденському музеї військової історії "Арсенал". GPT-API тут може виступати не лише як засіб інформування, але й як інструмент залучення відвідувачів до глибокого занурення в історичний контекст.

GPT-API здатний надати вичерпну інформацію, використовуючи свої обширні знання та аналітичні можливості. Система може описати, як U-20 був залучений до Першої світової війни, його роль у військових стратегіях того часу, а також особливості його конструкції та озброєння. Крім того, GPT-API може розповісти про повсякденне життя моряків на борту та виклики, з якими вони стикалися під час місій. Приклад такої взаємодії показано на рисунку 3.11.

Також на відміну від звичайного гіда, GPT може виявити значну адаптивність до особливих вимог різних груп відвідувачів. Наприклад, праця з дітьми вимагає не лише знань, але й здатності перетворити ці знання на щось більш цікаве, просте та захоплююче для молодого віку. Технічні характеристики та специфікації підводного човна U-20 можуть бути занадто складними та нецікавими для дітей у віці від 10 до 14 років. Окрім того, сама концепція підводного човна може виявитися для них незрозумілою.

GPT, як інтерактивний гід, має потенціал подолати ці виклики, пропонуючи адаптовану презентацію експонату, яка буде цікавою та зрозумілою для дітей. Наприклад, замість технічних деталей, система може розповісти захоплюючі історії про пригоди моряків, легенди, пов'язані з підводними човнами, чи цікаві факти про морське життя. Такий підхід допоможе дітям краще зрозуміти тему, стимулюючи їх уяву та інтерес до історії та зображений на рисунку 3.12.

Після процесу генерації відповідь, яка була створена, підлягала б збереженню в базі даних, де вона б пройшла етап модерації. Такий підхід забезпечує контроль якості генерованого контенту та відповідність його встановленим критеріям. Модерація важлива для перевірки точності, релевантності та відповідності інформації, що надається системою, до очікувань та потреб користувачів.

Таким чином, використання GPT як інтерактивного гіда в музеї відкриває нові можливості для адаптації інформації та подачі її в залежності від віку, інтересів та рівня розуміння відвідувачів. Це робить GPT не тільки ефективним засобом передачі знань, але й інструментом, здатним забезпечити кожному відвідувачу унікальний та пам'ятний досвід.

You are a knowledgeable guide in the Vienna Arsenal military history museum, with expertise in World War I and II naval history.

Response only with plain text with no lists.

Provide detailed information about the U-20 submarine, including its historical significance and features

The U-20 submarine holds a significant place in naval history, particularly in the context of World War I. Launched by Germany in 1912, U-20 was a type U 19 submarine and became notorious for its role in the sinking of the RMS Lusitania in 1915, an event that dramatically shifted public opinion and had far-reaching implications in the escalation of World War I.

The U-20 was about 64 meters long and 6 meters wide, powered by two diesel engines for surface travel and two electric motors for underwater movement. This design was quite advanced for its time, allowing it to operate both at sea and under the water with reasonable efficiency. Its armament typically included torpedoes and a deck gun, making it a formidable threat to merchant and military ships alike.

The most historically significant action of U-20 was the sinking of the RMS Lusitania, a British ocean liner. On May 7, 1915, under the command of Kapitänleutnant Walther Schwieger, U-20 torpedoed the Lusitania off the coast of Ireland. The ship sank in just 18 minutes, and almost 1,200 people lost their lives, including 128 Americans. This event caused international outrage, particularly in the United States, and is often cited as a key factor in turning public opinion in several countries against Germany, eventually contributing to the US entry into World War I in 1917.

The sinking of the Lusitania by U-20 showcased the deadly effectiveness of submarine warfare and marked a pivotal moment in military and naval strategy. It highlighted the vulnerability of large ships to submarine attacks and signaled a shift in naval tactics and international warfare rules. The event raised questions about the legality and morality of targeting civilian vessels, a debate that continued throughout both World Wars.

U-20's career ended in 1916 when it ran aground on the Danish coast. The crew destroyed the submarine to prevent its capture, marking the end of one of the most historically significant vessels in naval warfare. The legacy of U-20, particularly its role in the Lusitania sinking, remains a key subject of study for historians examining the naval warfare of World War I and its broader implications on international conflict.

Рисунок 3.11 – Приклад запиту імітації гідру для опису підводного човна U-20

You are a knowledgeable guide in the Vienna Arsenal military history museum, with expertise in World War I and II naval history.

Response only with plain text with no lists.

Provide detailed information about the U-20 submarine, including its historical significance and features imagine you are telling this information to a group of kids age 10-14 Years old

Alright, imagine a time over a hundred years ago, during World War I. This was a time when submarines were a relatively new kind of warship. One of these submarines was the U-20, and it played a really big role in history.

The U-20 was like a giant underwater metal fish, about as long as a blue whale. It was built by Germany and could travel both on top of the water and underneath it, which was pretty cool back then. It had powerful engines to move on the surface and electric motors for sneaking around underwater.

Now, the most famous thing the U-20 did was something quite dramatic and sad. In 1915, the U-20, under the command of a captain named Walther Schwieger, fired a torpedo and sank a huge ship called the RMS Lusitania. This ship was like a floating hotel, really big and luxurious. But, it wasn't just any ship; it had lots of people on board, including women and kids. When the U-20 sank it, nearly 1,200 people lost their lives, and that included 128 Americans.

This event made people all around the world very angry and sad, especially in the United States. It was one of the reasons why the United States decided to join the war against Germany two years later. So, you see, the action of this one submarine had a huge effect on the whole world.

The U-20 showed everyone how dangerous submarines could be to even the biggest ships. It changed the way people thought about fighting wars at sea and raised a lot of questions about what was right or wrong in warfare, especially when it came to attacking ships with civilians.

The story of the U-20 ended in 1916, not long after the Lusitania incident. It got stuck near Denmark, and the crew, not wanting it to get captured, decided to destroy it themselves. So, the U-20's journey ended there, but the story of what it did and how it affected the world war is something that historians still talk about today. It's a reminder of how a single action, like sinking a ship, can change the course of history.

Рисунок 3.12 – Приклад запиту імітації гіду для опису підводного човна U-20 для дітей віком 10-14 років

3.4 Висновки за розділом 3

У розділі дослідження системи було проведено глибокий аналіз актуальності впровадження нового підходу та здійснено детальне порівняння використання нейронних мереж з традиційними підходами програмування. Особлива увага приділялася аналізу можливостей та переваг, які нейронні мережі пропонують порівняно з класичними підходами розробки та програмування. Це охоплювало аспекти гнучкості, швидкості розвитку, а також здатності адаптуватися до складних та динамічних умов.

Ретельно було розглянуто підходи використання GPT-API як для надання системою відповідей користувачеві, так і для обробки системою запитів. В контексті відповідей користувачеві, GPT-API відкриває нові перспективи в інтерактивності та персоналізації, забезпечуючи високу точність та релевантність відповідей. Такий підхід дозволяє скорочувати часові витрати на етапах проектування та розробки, що, як було показано у дослідженні, призводить до прискорення процесу розробки системи приблизно у 5-7 разів.

Окрім цього, була проведена оцінка економічної складової, включаючи розробку бізнес-плану з урахуванням критерію прибутковості. Таке дослідження дозволило виявити потенційні фінансові переваги використання нейронних мереж, зокрема GPT-API, у порівнянні з більш традиційними підходами. Економічний аналіз включав оцінку витрат на розробку, впровадження, а також потенційну віддачу від використання системи в реальних умовах.

Результати дослідження чітко вказують на значні переваги, які нейронні мережі можуть принести в процеси розробки та впровадження сучасних інформаційних систем. Вони не лише сприяють ефективному використанню ресурсів і прискоренню процесу розробки, але й забезпечують нові можливості для створення більш адаптивних та інтелектуальних систем, здатних задовольнити широкий спектр потреб користувачів.

ВИСНОВКИ

У рамках дослідження було проведено детальний аналіз обраної сфери, де основною виявленою проблемою стала недостатня динамічність системи. Як потенційне рішення цього питання та для підвищення користувацького досвіду було запропоновано використання нейронних мереж.

Далі у роботі було здійснено глибоке дослідження основ нейронних мереж, включаючи їх архітектуру, принципи роботи, види функцій активації та механізми навчання. Аналіз підтвердив, що нейронні мережі мають великий потенціал у вирішенні проблеми недостатньої динамічності систем.

На етапі вибору нейронної мережі, з огляду на ресурсні обмеження та необхідність власної розробки, було вирішено скористатися зовнішніми сервісами. Це виявилось ефективним рішенням, яке дозволяє використовувати переваги готових рішень. Після аналізу різноманітних сервісів було обрано API від OpenAI за його ефективність та адаптивність.

Для розробки системи обрано клієнт-серверну за її збалансованість і відповідність проекту, Laravel як серверний фреймворк, який оптимально поєднує гнучкість, структурованість та масштабованість, та Vue.js для клієнтської частини, що відзначається своєю простотою і ефективністю використання. Використання архітектурного шаблону Porto для серверної частини забезпечило необхідну модульність та адаптивність.

Акцентування уваги на специфіці нейронних мереж, зокрема GPT-API, є визначальним у контексті розробки проєктів, де конфіденційність даних не є пріоритетною. Це забезпечує ефективність обробки запитів та швидкість надання актуальних відповідей, підвищуючи продуктивність системи та надаючи користувачам важливу інформацію оперативно.

Застосування нейронних мереж у розробці інформаційних систем покращує їх динамічність та адаптивність, забезпечуючи високий рівень ефективності відповідей. Глибокий аналіз підходів використання GPT-API відкриває нові

перспективи у сфері інтерактивності та персоналізації, що дозволяє не лише точно та релевантно відповідати на запити, але й значно скорочувати час на проектування та розробку.

Проведений детальний аналіз економічної складової, включно з розробкою бізнес-плану та оцінкою прибутковості, виявив фінансові переваги використання нейронних мереж, особливо GPT-API, порівняно з класичними підходами. Це охоплює оцінку витрат на розробку та імплементацію, а також потенційну віддачу від їх застосування у реальних умовах.

Завдяки дослідженню стає очевидним, що нейронні мережі мають значний потенціал у розробці та впровадженні передових інформаційних систем. Вони сприяють не тільки оптимізації використання ресурсів і прискоренню процесів розробки, але й відкривають шлях для створення більш гнучких та інтелектуальних систем, здатних відповідати на різноманітні потреби користувачів.

Наукова новизна полягає у запропонованому підході, заснованому на використанні нейронних мереж замість стандартної програмованої логіки у системах з відкритим набором даних для підвищення швидкості розробки, простоти підтримки та динамічності системи, а також оптимізації досвіду експертів.

Практична цінність отриманих результатів – розроблено підхід, використання якого підвищило ефективність розробки та роботи Web-системи в порівнянні з аналогами у 5-7 разів при рішенні задачі розробки та роботи системи з динамічними параметрами в умовах роботи з відкритими даними, а також дозволило замінити експертний досвід у певних галузях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 What is tourism services example?. URL: <https://lovethemaldives.com/faq/what-is-tourism-services-example> (дата звернення: 04.10.2023).

2 What is A tourist service and why does it matter?. URL: https://www.travlaw.co.uk/blog_posts/what-is-a-tourist-service-and-why-does-it-matter/ (дата звернення: 05.10.2023).

3 Tourism and travel-related services. URL: https://www.wto.org/english/tratop_e/serv_e/tourism_e/tourism_e.htm (дата звернення: 05.10.2023).

4 Web system development process: the complete guide. URL: <https://riseuplabs.com/web-system-development-process/> (дата звернення: 01.10.2023).

5 What is a web client and how it works with web server applications. URL: <https://cloudinfrastructureservices.co.uk/what-is-a-web-client-and-how-it-works-with-web-server-applications/> (дата звернення: 02.10.2023).

6 What is web server. URL: <https://www.techtarget.com/whatis/definition/Web-server> (дата звернення: 01.10.2023).

7 What is a web server?. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server (дата звернення: 01.10.2023).

8 What is a database?. URL: <https://www.oracle.com/database/what-is-database/> (дата звернення: 03.10.2023).

9 Meine Landausflüge. URL: <https://meine-landausfluege.de/> (дата звернення: 05.10.2023).

10 Components of travel services. URL: <https://opentextbc.ca/introtourism2e/chapter/components-of-travel-services/> (дата звернення: 06.10.2023).

11 What are neural networks?. URL: <https://www.ibm.com/topics/neural-networks> (дата звернення: 07.10.2023).

12 Neural network. URL: https://en.wikipedia.org/wiki/Neural_network (дата звернення: 07.10.2023).

13 What is a neuron?. URL: <https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron> (дата звернення: 08.10.2023).

14 What are Weights and Biases?. URL: <https://h2o.ai/wiki/weights-and-biases/> (дата звернення: 08.10.2023).

15 Basic structure of artificial neural networks. URL: <https://medium.com/datascienceearth/basic-structure-of-artificial-neural-networks-9aef29df9d> (дата звернення: 09.10.2023).

16 Everything you need to know about Neural Networks and Backpropagation. URL: <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a> (дата звернення: 09.10.2023).

17 Activation functions in neural networks [12 types & use cases]. URL: <https://www.v7labs.com/blog/neural-networks-activation-functions> (дата звернення: 11.10.2023).

18 Activation functions in neural networks. URL: <https://www.geeksforgeeks.org/activation-functions-neural-networks/> (дата звернення: 11.10.2023).

19 Why is sigmoid function important in artificial neural networks?. URL: <https://www.analyticsvidhya.com/blog/2023/01/why-is-sigmoid-function-important-in-artificial-neural-networks/> (дата звернення: 12.10.2023).

20 Softmax activation function: everything you need to know. URL: <https://www.pinecone.io/learn/softmax-activation/> (дата звернення: 14.10.2023).

21 Mertins D., El-Khoury R., Machado R. Monolithic architecture (architecture & design). Prestel Pub, 1995. 168 с.

22 Berson A. Client/server architecture. 2-ге вид. New York : McGraw-Hill, 1996. 569 с.

23 Borysenko V., Borysenko T. Modern approaches of design software applications based on microservice architecture. Computer and information systems and technologies. Kharkiv, Ukraine, 2020.

URL: <https://doi.org/10.30837/ivcsitic2020201441> (дата звернення: 12.11.2023).

24 DGX cloud is here: nvidia's AI factory services start at \$37,000. URL: <https://www.hpcwire.com/2023/03/21/nvidias-ai-factory-services-start-at-37000/> (дата звернення: 13.11.2023).

25 Nvidia is renting out its AI Superpod platform for \$90K a month. DigitalTrends. URL: <https://www.digitaltrends.com/computing/nvidia-ai-dgx-supercomputer-subscription-service/> (дата звернення: 13.11.2023).

26 OpenAI documentation. URL: <https://platform.openai.com/docs/introduction> (дата звернення: 14.11.2023).

27 TensorFlow core. URL: <https://www.tensorflow.org/guide> (дата звернення: 14.11.2023).

28 Google cloud AI. URL: <https://research.google/teams/cloud-ai/> (дата звернення: 15.11.2023).

29 Microsoft azure AI. URL: <https://azure.microsoft.com/en-us/solutions/ai> (date of access: 15.11.2023).

30 OpenAI API Pricing calculator. URL: <https://gptforwork.com/tools/openai-chatgpt-api-pricing-calculator> (дата звернення: 16.11.2023).

31 Open ai api referense. URL: <https://platform.openai.com/docs/api-reference/introduction> (дата звернення: 16.11.2023).

32 Spring framework. URL: <https://spring.io/projects/spring-framework> (дата звернення: 17.11.2023).

33 What is express.js? Everything you should know. URL: <https://kinsta.com/knowledgebase/what-is-express-js/> (дата звернення: 17.11.2023).

34 The laravel PHP framework – web app construction for everyone. URL: <https://kinsta.com/knowledgebase/what-is-laravel/> (дата звернення: 17.11.2023).

- 35 Porto (software architectural pattern).
URL: <https://github.com/Mahmoudz/Porto> (дата звернення: 18.11.2023).
- 36 Software architectural patterns. URL: <https://apiato.io/docs/9.x/getting-started/software-architectural-patterns/> (дата звернення: 18.11.2023).
- 37 The advantages and disadvantages of javascript.
URL: <https://softjourn.com/insights/the-advantages-and-disadvantages-of-javascript> (дата звернення: 19.11.2023).
- 38 TypeScript vs JavaScript – the state of the struggle in 2021.
URL: <https://tsh.io/blog/typescript-vs-javascript-comparison/> (дата звернення: 20.11.2023).
- 39 What is the Angular framework?. URL: <https://mdevelopers.com/blog/what-is-the-angular-framework> (дата звернення: 21.11.2023).
- 40 React documentation. URL: <https://react.dev/learn> (дата звернення: 21.11.2023).
- 41 What is vue js? The pros and cons of vue.js.
URL: <https://www.tatvasoft.com/outsourcing/2021/10/what-is-vue-js-and-its-benefits.html> (дата звернення: 21.11.2023).
- 42 Illia Solohubov, Artur Moroz, Mariia Tiahunova, Halyna Kyrychek and Stepan Skrupsky. Research of the Influence of Modern Artificial Intelligence Tools on Program Code Development. - The 6th International Workshop on Methods, Resources and Technologies for Open Learning and Research (MROL 2023) - Ivano-Frankivsk, September 18, 2023— у друці
- 43 Artur Moroz, Illia Solohubov, Mariia Tiahunova, Halyna Kyrychek and Stepan Skrupsky. Application of Neural Networks for an Electronic Tourist Guide. - The 6th International Workshop on Methods, Resources and Technologies for Open Learning and Research (MROL 2023) - Ivano-Frankivsk, September 18, 2023 – у друці