Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

РОЗРОБКА ЧАТ-БОТУ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ НАВЧАННЯ

Текстова частина до кваліфікаційної роботи

за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи

ст.в. Салата К. В.

_____

(підпис)

Виконала студентка 4 курсу

Сидорова Є.О.

«____»_____2024 р.

Київ 2024

Зміст

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

к.ф.-м.н., доц. Гороховський С.С

_____

(підпис)

«___» _____ 2024 р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студентці 4-го курсу, факультету інформатики Сидоровій Єлізавєті

Тема: Розробка чат-боту на основі великих мовних моделей для навчання

Вихідні дані:

Зміст ТЧ до дипломної роботи:

Зміст

Вступ

1.Загальна інформація про мовні моделі та чат-боти

2.Дослідження

3.Реалізація на основі дослідження

Висновок

Список використаної літератури

Дата видачі «____»_____ 2024 р. Керівник _____

(підпис)

Завдання отримала _____

(підпис)

Календарний план виконання курсової роботи

Тема: Розробка чат-боту на основі великих мовних моделей для навчання:

№	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	21.09.2023	
2.	Аналіз матеріалів за темою дипломної роботи	30.09.2023	
3.	Розробка та реалізація алгоритму створення чат-боту	10.10.2023	
4.	Написання текстової частини дипломної роботи	01.12.2023	
5.	Надання дипломної роботи на перевірку керівником	21.02.2024	
6.	Коригування на основі результатів перевірки	30.03.2024	
7.	Остаточне оформлення роботи та презентації	09.05.2024	
8.	Захист дипломної роботи	31.05.2024	

Сидорова Є.О. _____

Салата К.В. _____

«____» _____ _____ р.

Вступ

Світ довкола нас постійно стикається з інноваційними оновленнями в різних галузях. Застосування автоматизованих програм у вигляді «розумних» помічників для будь-яких підприємств є слушною нагодою для підтримки зв’язків між клієнтами та компанією. Цей процес охоплює аналітичні дослідження представленого взаємовідношення, забезпечує економію часових і грошових ресурсів та заохочує нових користувачів до використання наданих брендом послуг.

Чат-боти є надійними у сфері комерційному підході для різноманітних цільових аудиторій і послуг через цілодобове забезпечення підтримки, простоту у використанні й наявністю безлічі потрібного функціоналу.

Великі мовні моделі слугують архітектурним покращенням програмної системи текстових агентів. Їхнє впровадження до чат-ботів часто слугує не лише забезпеченням вибору серед мовних локалізацій, а ще й надають ширші можливості персоналізації під час інтерактивного використання.

Згідно з інтернет дослідженням компанії PSFK: 74% користувачів надають перевагу взаємодії з програмними помічниками під час пошуку відповіді на певні актуальні питання. Чат-боти з інтегрованим штучним інтелектом подобаються 50% користувачів за даними Dashly, тоді як Business Insider наголошує на тому, що за 2023 рік нейронні мережі змогли автоматизувати близько 73% адміністративних завдань. До того ж, відповідно до даних від Grand View Research світовий ринок чат-ботів до 2030 року має розширитись на 23,3%. [23]

Навчальні напрямки завчасно передбачають розлогий спектр функціональних особливостей. Згідно з визначенням концепту майбутнього помічника розробникам легше створювати власну велику мовну модель або ж обирати вже сформовану, щоб вносити корективи за допомогою алгоритмів трансферного й машинного навчання.

Актуальність використання нейронних мереж у чат-ботах для навчання полягає в полегшенні отримання інформації та перевірки власних знань користувачів. Функціональні особливості чат-ботів з інтегрованими розумними помічниками обираються на основі визначеної заздалегідь ідеї та цільової аудиторії. Проводячи основні й додаткові дослідження роботи різних мовних моделей, тестуючи їх на задачах із чітко визначеними вимогами та вираховуючи оцінку ефективності, розробники створюють найбільш оптимальну й коректну програму для подальшої публікації з постійним використанням. Важливо при цьому дотримуватися етичних користувацьких вимог, враховувати ризики упередженості в текстовому спілкуванні, забезпечення конфіденційності даних споживачів продукту та гарантувати прозорість під час роботи.

Основною метою цього дослідження є вивчення теоретичних відомостей про великі мовні моделі, розширення знань про чат-боти, практичне застосування навичок на базі вивчення інформації та аналіз усіх наданих матеріалів. Предметами експерименту є великі мовні моделі, способи їхньої інтеграції та аналітичні алгоритми для використання. Об’єктом розробки дипломної роботи є безпосередньо чат-бот для використання в навчальних цілях, що має слугувати зручним онлайн помічником для перевірки власних знань і підготовки за визначеною користувачем тематикою.

1. Загальна інформація про великі мовні моделі та чат-боти

1.1. Поняття чат-ботів та їхнє використання

Чат-боти – це програми, які імітують письмовий чи усний діалог між користувачем і помічником. В основі їхньої роботи зазвичай використовують штучний інтелект з його функціоналом: нейронні мережі, обробка природньої мови, глибинне та машинне навчання.

У залежності від типу чат-боту та його цільового використання помічників можна класифікувати. Наприклад, декларативні програми забезпечують структуроване автоматизоване вирішення питань користувача за одним загальним напрямком, використовуючи завчасно прописаний алгоритм. Зазвичай до цього типу відносять програми підтримки користувачів, трансакційні та ж програми для бронювання. Більш складним типом є цифрові помічники, які забезпечують персоналізовану інтерактивну взаємодію між користувачем і чат-ботом. Представниками віртуальних помічників у більшості випадків є голосові асистенти: Siri, Cortana, Google Assistant та інші. [1]

Комерційне впровадження чат-ботів для роботи компаній наразі є поширеною практикою в світі, оскільки це допомагає скоротити час очікування відповіді фізичних осіб, сприяє кращим аналітичним показникам у відношенні клієнт-підприємство й зменшує кількість витрат власникам. [2]

1.2. Поняття великих мовних моделей

Велика мовна модель – це програма, в основі яких знаходиться алгоритм глибинного навчання з використанням NLP (обробки природньої мови користувача). Це передбачає наявність трансформаторів моделі – нейронні мережі, які навчалися на дуже великому обсягу даних для майбутнього розуміння мови користувача програмою та можливостями перекладу й відповіді. Блокова структура основної архітектури подібна до людського мозку, тому часто великі мовні надають неабиякий внесок у розвиток

«розумних помічників». Трансформаторні моделі відносяться до типу генеративного штучного інтелекту, оскільки надають можливість автоматизованого створення відповіді у вигляді текстового контенту. Серед популярних представників є Bard, Bing, ChatGPT та інші.

Архітектура у вигляді моделі трансформатора складається з шифрувальника та дешифрувальника. Обробка даних базується на маркуванні даних, як вхідних та вихідних: під час їхнього отримання відбувається застосування математичних алгоритмів для виявлення можливих внутрішніх нейронних зв'язків, за цією структурою обробки й генерується відповідь. [3]

Основними шарами є самоувага (self-attention), шар прямих взаємозв'язків (FFN) та нормалізації. Від кількості їхній представників залежить потужність роботи великої мовної моделі. Важливо зазначити про позиційне кодування у трансформаторах: адаптивність програми має надавати розуміння змісту тексту навіть при непослідовній подачі слів за допомогою семантичного й синтаксичного значення вхідних даних.

Самоувага передбачує можливість зосереджуватися на окремих частинах вхідних даних, надаючи кожній свою вагу для визначення важливості в контексті. Такий підхід забезпечує економію часових та програмних ресурсів через те, що увага в першу чергу приділяється найбільш вагомим представникам. Здатність використання позиційного кодування в цьому контексті позиціонується ще й на багатопоточному підході, тобто доцільному розподіленні задач і нормалізації. У свою чергу шар прямих зв'язків перетворює вхідний текст для забезпечення абстракції. [4]

Загальний принцип роботи великих мовних моделей включає в себе комплексний підхід:

1. Навчання на великому обсязі даних, яке називають корпусом великої мовної моделі. Воно базується на контрольованій або ж неконтрольованій основі. Перший тип передбачає те, що всі завдання для аналізу мають завчасну відповідь, яку програма має знайти сама методами порівняльних навчальних прикладів. Проблематика цього методу працює на завдатках дискретного

значення кожного об’єкту та регресії, тому цей метод підходить лише для алгоритмів із завчасно визначеними контрольними й базовими точками, які, на жаль, не завжди є коректними та універсальними. Другий тип базується на великому наборі даних без конкретних інструкцій для їхнього аналізу та результату, тобто нейронна мережа намагається сама ідентифікувати структуру за допомогою базових принципів кластеризації, виявлення аномалій та асоціацій. Також присутній й напівконтрольований спосіб навчання з позначеними та непозначеними даними. Працюючи з аналізом природньої мови, розробники найчастіше використовують неконтрольоване навчання через контекстні проблеми, які й слугують тренуванням моделі з нуля. [5]

2. Використання принципу самоконтролю, використовуючи маркування для ідентифікації. Глибинне навчання з використання трансформатора допомагає створювати взаємовідносини між токенами й призначати оцінки параметрам. [6]

   На цьому етапі доречно вказати підказки для тонкого налаштування подальших завдань у вказаній цільовій діяльності. [3]

3. Впровадження моделі в постійне використання під час практичних ідей. Наприклад, аналіз даних за алгоритмом визначення настроїв. [6]

   На рисунку 1.2.1 зображене використання алгоритму аналізу даних настрою, використовуючи базові підказки. Велика мовна модель має специфікувати слова «красива» та «огидна» за допомогою семантичних значень на категорії позитивного й негативного настрою.

```
Відгук споживача: Ця рослина така красива!
Відгуки клієнтів: позитивні

Відгуки клієнтів: Ця рослина така огидна!
Відгуки клієнтів: негативні
```

Рисунок 1.2.1 Вправа аналіз настроїв

[3]

1.3. Історія створення великих мовних моделей

Початком в історії створення великих мовних моделей є дослідження, зроблені французьким вченим Мішелем Бреалем в 1883 році у галузі філології. Основною концепцією є вивчення способів взаємодії мов між собою й створення класифікаційних систем для цього. Надалі завдяки дослідженням Фердинанда де Соссюр було створено основу для високо функціональних мовних моделей, яка згодом внесла неабиякий внесок в NLP.

На відміну від звичайних мов, машинні алгоритми передбачають інтерактивну можливість для застосування математичних функцій. У 1958 році Френк Розенблатт використав алгоритмічну модель нейронних мереж з дослідженнями про машинне навчання, створивши нейронну мережу Mark 1 Perception.

Неабиякий внесок був спровокований першим чат-ботом ELIZA, який винайшов Джозеф Вейзенбаум у 1966 році. В основах структури закладено початки розвитку обробки природньої мови, тобто виокремлення ключових слів у вхідному запиті, їхньою обробкою та надання коректної відповіді, яка заздалегідь була визначена. [7]

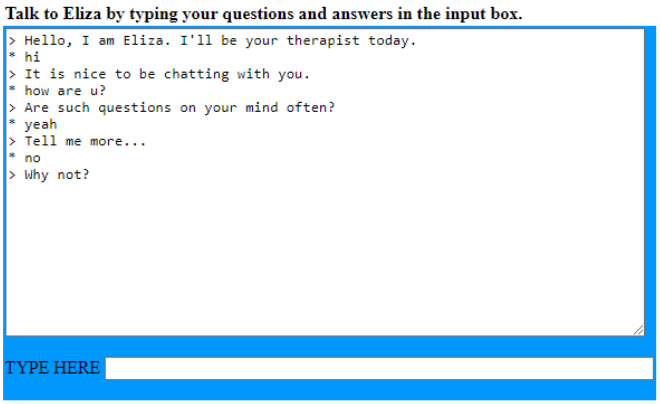


Рисунок 1.3.1 Чат-бот ELIZA

[8]

Впродовж розвитку машинного навчання як окремої галузі розробники стикалися з проблемами оптимізації: розмір даних для навчання й швидкісні характеристики роботи. На початку 1980-тих років компанія IBM

започаткували поняття малих мовних моделей, структура яких базувалася на статистично вирахуваній частоті вживання слів у реченнях і передбаченням наступних текстових об’єктів.

Завдяки появі Інтернету у 1990-тих роках, дослідники отримали змогу отримувати великі обсяги даних для навчання з електронних бібліотек. Згідно з цим, з’явилася необхідність у використанні придатних за потужністю систем, тому було створено електронні схеми для швидкої роботи з обробкою (GPU). Мовні моделі в цей період почали активно розвиватися, одним з найважливіших внесків була презентація генеративної конкурентної нейронної мережі в 2014 році. Ідейним змістом була «гра» між двома нейронними мережами, під час якої відбувалася генерація фотографії, щоб в результаті дебатів отримувалось найбільш реалістичне зображення. [7]

Генеративна конкурента модель працює на основі моделювання з використанням глибинного навчання. За схемою неконтрольованого навчання з двома підмоделями (генератор прикладів і дискримінатор для класифікації між правдоподібними й хибними) відбувається тренування, допоки розрізняльний процес не буде ідентифікувати приклади як істинні. [9]

Найбільш популярними сучасними представниками є:

- ChatGPT, який вперше був представлений у 2020 році з набором у 175 мільярдів параметрів. Модель GPT-3 базується на перекладі, текстовій генерації відповідей та завданнях з оперативним міркуванням. [10]
- PaLM 2, розроблена компанією Google, працює за алгоритмом використання оптимального масштабування та покращеною роботою корпусу. У травні 2023 року тестування надало високу оцінку за багатомовністю, швидкістю й можливістю розпізнавання й генерації. [11]
- XLNet – модель, яка оцінює шаблон закодованих токенів та робить прогноз за випадковим, непослідовним порядком. В основі лежить алгоритм авторегресивного моделювання, який описується багато напрямленим контекстом та автоматизованою регресією формування. За дослідженням її робота перевершує аналогічної мережі Bart від розробників Google. [12]

1.4. Типізація великих мовних моделей

Типізація великих мовних моделей залежить від їхньої архітектури та способу навчання, описаних у пункті 1.3. Застосування більш глибого дослідження допомагає виокремити такі види:

1. *Language Representation Model* – моделі, основна задача яких закладається в розумінні та генерації людської мови. В основі роботи лежить вже згаданий алгоритм обробки природної мови з аналізом тексту за допомогою виокремлення вагомих частин і трансформаторного підходу. Наприклад, BERT та RoBERTa. [13]
2. *Zero-shot Model* – моделі з присутнім узагальненим підходом та влучним прогнозування для запитів, з якими вони раніше не стикалися. В основі лежить Zero-shot класифікація, яка характеризується абстрактним глибинним навчанням. Попередньо навчена модель застосовується для вирішення завдань у іншому середовищі, тобто вона отримує підказки для роботи з новими завданнями і намагається їх виконати, використовуючи вже набуті вміння. Допомога закладається в наданні прикладів виконання бажаного запиту, які одночасно можуть стосуватися декількох параметрів класифікатора. Гарним прикладом є визначення емоційного забарвлення тексту через наявні тригерні слова. Даючи підказку «Класифікуй вхідний текст за однією з категорій емоцій (позитивна, негативна, нейтральна)» та надавши вхідний текст, користувач має отримати коректний результат визначення емоційного забарвлення від великої мовної моделі. Гарним прикладом може бути модель GPT-3. [14]
3. *Multimodal Model* – моделі, що здатні працювати не лише з текстовим контекстом, а й з графічними. Сфера комп’ютерної візуалізації на основі роботи потужних графічних процесорів розкриває поняття мультимодальної нейронної мережі, яке було презентоване у 2022 році. Типовими функціональними особливостями є розпізнавання об’єктів у текстовому й медіа поданні. Процес обробки сенсорних даних є одночасним, подібним до сприйняття людиною. На відміну від попередніх типів, мультимодальні моделі інтегрують різні джерела декількох напрямків (текст, зображення, відео, аудіо)

при своєму аналізі. Навчання передбачає їхнє об'єднання з використанням датчиків і токенів для динамічної роботи. Шифрувальник поєднує інформації з різних сенсорних вхідних даних для повної картини представлення в системі, а декодер будує прогнозований результат на основі введення. Технічна складова, звісно ж, вимагає більш складної архітектури. Прикладами є CLIP, DALL-E, METER.

На рисунку 1.4.1 представлена схема для представлення відмінностей роботи мультимодальної моделі від інших типів. [15]

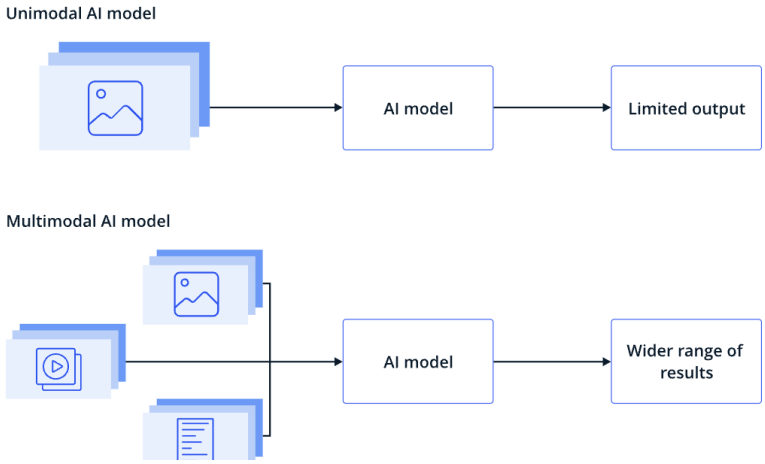


Рисунок 1.4.1 Відмінності між мультимодальною моделлю та іншими [15]

4. *Fine-Tuned or Domain-Specific Model* – моделі з тонкими налаштуваннями та додатковим навчанням на предметно-специфічних даних, наприклад, медичних. Така додаткова параметризація також має містити приклади-підказки для специфікації конкретної цільової сфери роботи. Процес тонких налаштуванні будується на виборі унікального набору даних, їхнього розподілу для навчання й тестування, тестуванні, коригуванні коефіцієнтних параметрів й оцінюванні продуктивності. Згідно із специфічними доповненнями має бути врегульована оптимізація. Серед стратегій тонкого налаштування визначають оперативну інженерію – метод з чітким розумінням мови моделі, ретельно продуманими підказками й відсутністю змін у загальній

архітектурі. Цей спосіб характеризується гнучкістю налаштування, проте потребує великої кількості уніфікованих даних. Наступною стратегією є повне тонке налаштування, яке працює індивідуально з кожним типом базової моделі для майбутньої адаптації. Хоча це і є одним з найбільш чітких методів щодо результатів, проте є доволі не оптимальним через ресурсні витрати. Інструкційна стратегія передбачає контрольоване навчання й тестування моделі не через загальний контекст, а з використанням детального алгоритму дій. При такому методі потрібно врегульовувати усі попередні й нові ваги параметрів, щоб не стикатися з проблемою «забування». Ще однією стратегією є ефективне тонке налаштування з параметрами (PEFT) – спосіб, який мінімізує проблему «забування» даних через тимчасове «замороження» попередніх навчених параметрів і виконання роботи настроювання лише на відносно малій підмножині. Перевагами є мінімізація використання пам’яті та абстракція для використання між різними середовищами. Остання стратегія тонкого налаштування називається адаптація низького рангу (LoRA). Вона є варіацією PEFT, при якій додається один додатковий набір даних, який є змінним і займає мало пам’яті. Структура роботи працює на модульному рівні, тобто передбачає впровадження в різні моделі. [16]

1.5. Переваги й недоліки використання чат-ботів з мовними моделями

Винахід чат-ботів з інтеграцією великих мовних моделей продовжує надавати популярність цифрових продуктів серед їхню модальність. Через різноманітні функціональні особливості додатки з використанням нейронних мереж стають доволі поширеними.

Спочатку розглянемо переваги чат-ботів з інтегрованими великими мовними моделями: зрозумілість, адаптивність і можливість покращення. Генеративна універсальність полягає в можливості простого розуміння й додавання нейронних мереж до будь-якої програми. Використання трансформаторів в архітектурі системи доволі сильно впливає на роботу: якщо з їхньою допомогою покращується контекстуальне розуміння, обробка різноманітних

виразів і коректність результату, то за їхньої відсутності – необхідно визначати чіткий доменний специфічний підхід з унікальними наборами даних та слідкувати за обмеженням узгодженості. [17]

Завжди потрібно зауважувати робоче навантаження: від капіталу для розробки до потужності обчислювальної інфраструктури. Навчання великої моделі потребує значного обсягу даних і кількості графічних процесорів для їхньої обробки. Ризики, які виникають під час неправильно підібраних початкових вимог, провокуватимусь проблеми експлуатації, складність пошуку можливих помилок і визначення ключових точок. При доречному підборі всіх параметрів велика мовна модель гарантуватиме точність, легкість навчання, гнучкість впровадження в різні проекти й високу продуктивність з мінімальною затримкою. [4]

2. Дослідження

2.2 Методи розробки чат-ботів з великими мовними моделями

Розробка чат-боту з використанням великих мовних моделей у першу чергу передбачає використання нейронної мережі, тому важливим кроком є вибір відповідної моделі для визначеної предметної області.

Одним зі способів є трансферне навчання, тобто використання попередньо навченої нейронної мережі з широким спектром даних на окремій задачі, яка не містить багато відомостей і якось пов’язана з першоджерелом. Такий спосіб є показовим представником методології проектування для уніфікованого пошуку вирішення проблеми та узагальнення роботи в цілому. Замість того, щоб проводити навчання з нуля, проводиться процес роботи з шаблонами, які отримуються під час роботи з завданнями. Основні процеси трансферного навчання залежать від початкового та середнього прошарку, де позначені навчальні дані. Для отримання нового кінцевого класифікатора слід переносити більшу частину попередньо набутих знань: наприклад, початкова модель вміє розпізнавати об’єкти, а після застосування трансферного навчання кінцевим етапом буде перевірка на присутність конкретного представника. Через перенесення готових навчальних даних цей спосіб є оптимальнішим, ніж створення нової моделі, проте важливими вимогами до моделі є доступність до цих відомостей, в іншому випадку – зміна роботи є майже неможливою. Також потрібно мати однакових тип вхідних даних й ідентифікувати найважливіші функції для абстрактного комбінування згідно з додатковими вимогами. [18]

Створення власної великої мовної моделі є більш важким процесом, що вимагає багато інформаційних, часових і функціональних ресурсів. Першим кроком є збір даних та їхня попередня обробка: текстові джерела мають бути релевантними та розділеними на мовні одиниці. Проектування архітектури трансформатора й ініціалізація моделі з певними вагами передбачає визначення мети попереднього навчання та оптимізації на невеликих завданнях. Мітки можуть змінюватися при обробці, оскільки вони залежать від

конкретної загальної ідеї та майбутніх запитів. Такі задачі призводять до початкового масштабування мережі, щоб знати приблизний обсяг обчислювальних ресурсів і складність шаблонів. Наступним кроком вирівнювання моделі згідно з цільовими вимогами: відбувається збір та об'єднання шаблонів. За допомогою визначеної мети корегуються мітки та відбувається тонке налаштування, що зберігає вагові коефіцієнти для розуміння мови. [19]

Оцінка ефективності за ітеративним підходом дає змогу зміни параметризації для досягнення найбільш чіткої мети. Важливими деталями для розгляду є обсяг ресурсів і їхнє зберігання, кореляція етичних вимог і наявність прозорості. Інкапсуляцію інформації й обчислень можна забезпечити через хмарні платформи, а економію часу для розподіленого навчання часто надають прикладні бібліотеки. Ризик упередженості керується за допомогою обсягу даних і обмежень за шаблонами, а наявність прозорості передбачає довгий процес аналізу поведінки моделі. [19]

Створення чат-боту залежить від його подальшого використання: наприклад, однією з найпопулярніших інтеграцій є чат-бот у месенджерах. Їхнє впровадження є доволі легким у підключенні та використанні користувачами. Представниками є Facebook Messenger, Telegram, Viber та інші. До того ж, компанія Microsoft пропонує власний інтегрований інструмент Microsoft Bot Framework для роботи з різними службами та платформами. Серед переваг є наявність відкритого коду, зручна екосистема та додавання функціоналу штучного інтелекту на природньої мови за допомогою Azure Cognitive Services. Цей сервіс дає змогу створювати чат-боти з розумними помічниками на корпоративному рівні з дотриманням прав автора та належним контролем всіх даних. Комплексний набір засобів передбачає інструменти для текстового й аудіо відтворення. Життєвий цикл чат-боту починається з проектування, дизайну й ознайомленням із засобами розробки: Azure, .NET SDK і NodeJS. Наступним кроком є, власне, процес розробки згідно з функціями, показаними на зразках у документації, визначенням майбутніх когнітивних послуг для

взаємодії з користувачами та вирішенням варіантів інтеграції. Далі відбувається процес ручного тестування та за допомогою емулятора. Кроки з публікацією та підключенням надають можливість використання серверів для безперервної безпечної роботи. Додатковою перевагою є аналітична оцінка, яку надає сервіс. Схема зображена на рисунку 2.1.1. [20]

Рисунок 2.1.1 Життєвий цикл чат-боту з використанням Microsoft Bot Framework

[20]

Підключення будь-якої великої мовної моделі до прикладної частини може бути локальним або серверним. Перший спосіб має більш конфіденційний підхід, тому що будь-які сторонні компанії не матимуть доступ до всіх запитів користувачів і розробників. Локальне розгортання великої мовної моделі починається із завантаження на власний девайс чи хмарне сховище. Надалі відбувається процес підключення до прикладної частини. Допоміжними інструментами для цього є програми з завчасно встановленими моделями для попереднього тестування, однією з них є Hugging Face. Зручністю використання є надання рейтингу доступних нейронних мереж і легкою інтеграцією до коду: наприклад, за допомогою бібліотеки transformers у мові програмування Python. Ще одним способом для запуску локальної мовної моделі є LangChain. Ця платформа підтримує власний фреймворк, який складається з декількох частин: бібліотека та шаблони LangChain з колекціями різноманітності архітектури, бібліотека LangServe для роботою REST API з та платформа LangSmith для оцінювання ефективності, тестування й контролю

роботи великої мовної моделі. Можна також розглянути продукти Llama, які адаптовані під різні внутрішні комплектації девайсів і доволі легкі в налаштуванні й встановлені, проте мають обмежений вибір серед бібліотеки. Аналогічно до них працює й продукт GPT4ALL. Серед мінусів локального розгортання великих мовних моделей є значне використання ресурсів пам'яті та відсутність доступу деяких з них для комерційного використання. [21]

Якщо притримуватися серверного типу підключення, то зручним та швидким способом є під'єднання за допомогою API. Наприклад, компанія OpenAI надає користувачам можливість підключення за допомогою власного секретного ключа та наявності спеціалізованої бібліотеки для мови програмування Python. Після коректного підключення згідно з документації до функцій можна створювати генеративну схему відправлення запитів й отримання відповідей. Бібліотека від OpenAI дає можливість додаткового коригування консервативної чи креативної відповіді, а також налаштування кумулятивної ймовірності можливих слів за допомогою параметризації. Цей спосіб сприяє зменшенню використання ресурсів пам'яті, проте є менш конфіденційним згідно з політикою розробників інтегрованих великих мовних моделей за допомогою ключа API. [22]

2.3 Дослідження роботи різних великих мовних моделей

Розробка чат-боту з використанням великої мовної моделі передбачає багато вимог до його функціональних особливостей. Під час цього дослідження було розглянуто декілька найбільш популярних нейронних мереж і визначено їхні переваги та недоліки.

1. GPT використовує велику мовну модель, яка була навчена на великих обсягах даних задля контекстуального розуміння слів. Розробниками є компанія OpenAI, яка надає 2 можливих варіанти використання – безкоштовне та платне. Різницею між ними слугує актуальність навчальної інформації моделі, оскільки більш стара версія корелюється даними, що стосуються лише періоду часу до 2021 року. Також серед плюсів використання підписки є те, що

користувачі мають доступ до усіх оновлень, плагінів і здатності отримувати швидші відповіді. Генеративна мовна модель є навченою на текстових даних з Інтернету, включаючи документи, статті, дослідження тощо. Алгоритмом навчання є безконтрольний спосіб, а трансформатор, що закладений в архітектуру, реалізовує механізми прогнозування з оцінкою достовірності, що залежить від запиту користувача. [24]

Коли розглядається мета створення чат-боту для навчальних цілей, слід зауважити перевагу в підтримці багатьох мов, наданні персоналізованих відповідей і універсальності вибору серед різноманітних тем, а також через масштабованість і адаптивність можна надати багато поточне використання майбутнього чат-боту. Серед наявних недоліків слід пам’ятати про розуміння контексту, що не завжди може бути коректним. Під час дослідження роботи мовних моделей в цілому було розглянуто поняття ризику упередженої інформації та конфіденційності даних при підключені через API ключ, у цьому випадку етичні вимоги залежать у більшості випадків від запитів користувачів, а підключення до прикладної частини може бути реалізовано як локальним способом, так і серверним. [25]

2. Llama 2 – велика мовна модель, що створена на основі алгоритму контрольованих тонких налаштувань і попереднього навчання. На відміну від її попередньої версії характеризується більшою продуктивністю через кількість токенів, групування запитів для масштабованості результативних даних і подвійного розширення довжини контексту. Розробниками є компанія Meta, які надають використання великої мовної моделі з відкритим кодом і безкоштовним доступом до користування. Способом підключення є локалізований: за допомогою реєстрації на офіційному сайті Meta Llama 2 користувачі отримають підтвердження на пошту для скачування моделі з GitHub. [26]

   Дотримуючись мети створення чат-боту для навчальних цілей, розробники мають зауважити про 3 основні варіації розміру моделі – 7, 13 чи 70 більйонів параметрів. Такі показники впливають на часові показники роботи з Llama 2,

проте всі з них здатні до генерацій текстових сценаріїв. Через наявні варіації розміру великої мовної моделі легко можна знайти оптимальний вибір для продуктивного використання власного обладнання, а безпека гарантується самим локалізованим підходом. Проте порівнюючи Llama 2, наприклад, з вже розглянутою моделлю GPT, розробники скоріше обиратимуть попередню. Оскільки продукт від OpenAI має вдвічі більше параметрів, що впливає на різноманітність підтримуваних мов, розуміння контексту й загальні генеративні здібності. Тестування Llama 2 з використання мов, відмінних від англійської, демонструє обмеження в розумінні або ж відмовою у відповіді на питання, вважаючи його недоречним. До того ж, здатність до кодування цієї моделі значно поступається іншим. [27]

3. Copilot – велика мовна модель від компанії Microsoft, також відома завдяки Bing Chat. В її основі лежить робота нейронних мереж і постійне навчання на основі запитів користувачів в онлайн режимі використання. Модель здатна генерувати текст, надавати відповіді у вигляді веб-пошукових запитів і створювати картинки. Така різноманітність можливостей створена використанням 3 різних моделей у своїй архітектурі ChatGPT-4, DALLE-3 і Prometheus. Цей приклад є представником трансферного навчання й інтеграційних можливостей різних моделей для свого проекту. Підключення створюється через серверний підхід. [28] Використання Copilot у майбутньому чат-боті для навчання створює аналіз таких деталей: підтримка багатомовності, пришвидшений процес генерації та конфіденційність. Через постійне навчання за допомогою запитів користувачів є доволі актуальним продуктом, проте наявність веб-пошукових відповідей може спровокувати питання кібербезпеки. [29]
4. Bard працює на основі полегшеної та оптимізованої версії великої мовної моделі LaMDA. Компанія Google описує роботу, як потужний механізм прогнозування та генерації текстових відповідей. Оскільки на момент написання цієї цей проект ще є тестовим, то варіантів його інтеграції до власних програм можливий лише за допомогою певних бібліотек. Для його

онлайн використання обов’язковим кроком є реєстрація в акаунті Google. Самі ж розробники зазначають, що ризик упередженості у відповідях ще проходить стадію налаштування для найбільш коректних відповідей. Проте тестування з багатомовністю показує вправні результати. [30]

Орієнтуючись на актуальність даних, що є доволі важливим поняттям для чат-боту з навчальними цілями, розробники гарантують найновіші й оновлені результати. Це передбачається тим, що пошукова компанія Google добуває інформацію в реальному часі та використовує її під час обробки запиту користувача. Як вже було зазначено, через тестовий режим публікації наразі відповіді можуть нести в собі неточну чи некоректну інформацію, а також непослідовність у відповідях. [31]

2.4 Дослідження інтеграційної платформи та функціоналу майбутнього чат-боту

Як вже було зазначено, одним з найпопулярніших способів використання чат-ботів є їхня інтеграція в месенджери. Результати опитування від AIN.UA зазначають, що найбільш популярним представником серед українців є Telegram, де для спілкування його обирають 50,6% респондентів. [32]

Зручністю використання цього месенджера є багатофункціональні особливості для розробки чат-ботів і їхнього розповсюдження. Безкоштовне використання, наявність персоналізації, легкість у розробці за допомогою безлічі бібліотек та конфіденційність. Безпека гарантується доступом до обмеженої збірки даних користувача, що вже є публічними, а будь-які запити захищаються зі сторони розробників та приватного API ключа до доступу. Гарним доповненням є мультиплатформеність месенджеру, оскільки він доступний на всіх основних мобільних і настільних операційних системах. Легкість пошуку чат-боту спрощує доступність до користувачів соціальної мережі, тому для створення чат-боту є доволі оптимальним варіантом для аудиторії майбутніх учасників. [33]

Підключення та налаштування чат-боту відбувається початково з використання безпосереднього месенджеру Telegram, який надає можливість

інтеграції власних помічників за допомогою офіційного боту BotFather. Він надає перелік усіх команд для настроювання майбутньої програми. Процес починається з базової вказівки /newbot, яка задає маркування чат-боту, тобто його ім’я та тег для публічного доступу серед усіх користувачів месенджеру. Надалі надається HTTP API токен для інтеграції управлінських можливостей до прикладної частини. Надалі вже з обраною мовою програмування чи можливими іншими інструментами створення функціоналу чат-ботів відбувається підключення за допомогою вже згаданого наданого ключа. Більш того за допомогою BotFather можна налаштовувати перелік доступних команд у власному чат-боті, його опис, аватар, опис та інше. [34]

Було створено власне опитування, під час якого було проведено аналітику впровадження чат-боту з використання великої мовної моделі. Серед 100 опитуваних респондентів більшістю є люди віком від 18 до 25 років, тому великий відсоток учасників наразі навчаються. Це створює більш сприятливі умови для розуміння актуальності розробки майбутнього чат-боту.

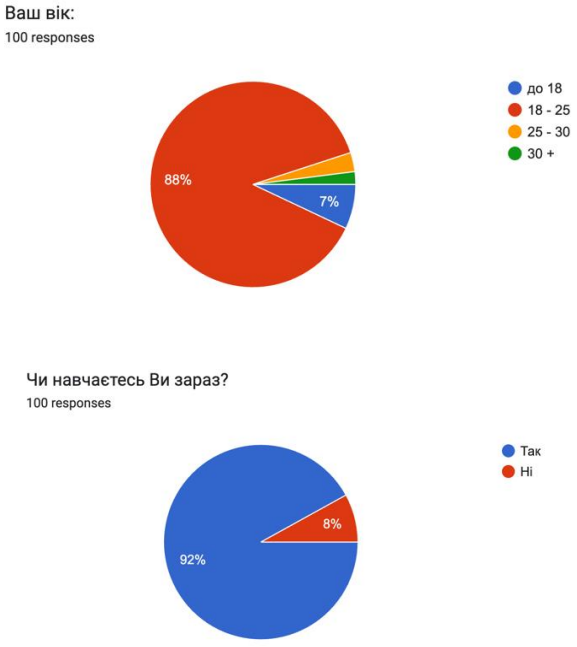


Рисунок 2.3.1-2.3.2 Початкова інформація про респондентів

Метою розробки є програма для спрощення підготовки до навчальних предметів і перевірка власних знань за допомогою тестів. Оскільки більшість з

респондентів використовує інтернет джерела для підготовки, то інтеграція популярних великих мовних моделей має сприяти відповідності інформації до сучасного використання. До того ж більш ніж 95% опитуваних користуються розумними помічниками у власному житті.

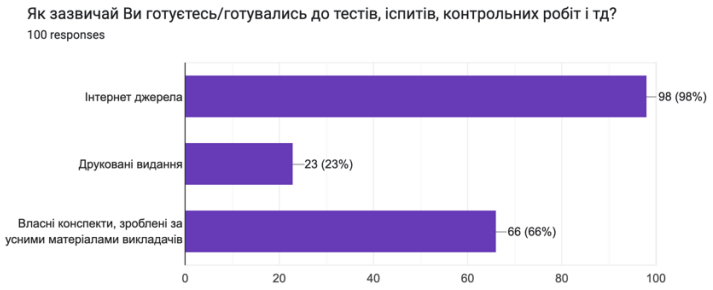


Рисунок 2.3.3 Способи підготовки респондентів до навчальних перевірок

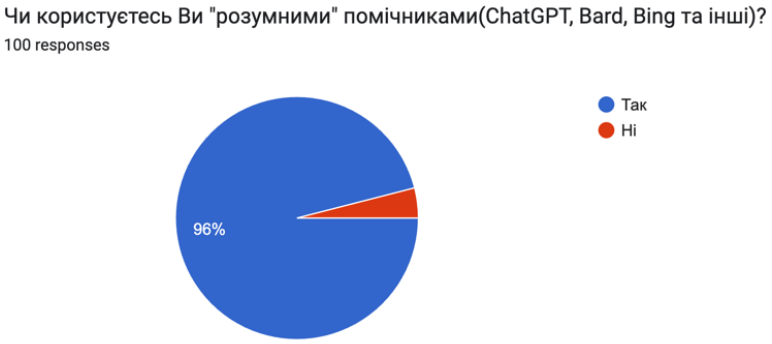


Рисунок 2.3.4 Використання «розумних помічників» серед респондентів

Загальні враження респондентів від роботи з розумними помічниками є вище середнього, проте майже 90% вважає, що їхнє впровадження для підготовки слугуватиме слушною ідеєю для спрощення процесу підготовки.

Рисунок 2.3.5 Оцінка використання «розумних помічників» серед респондентів

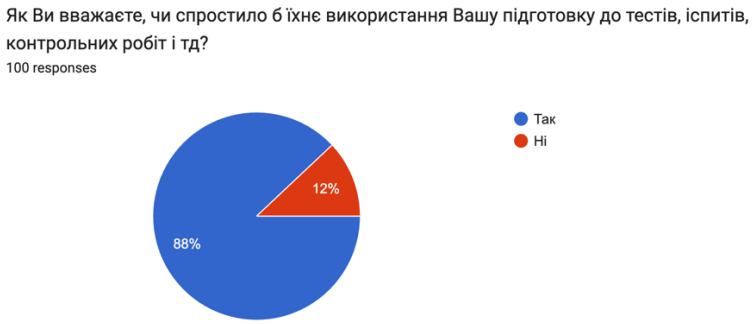


Рисунок 2.3.5 Оцінка щодо слушності впровадження «розумних помічників» для навчального процесу серед респондентів

Отже, створення чат-боту в месенджері Telegram з використанням великої мовної моделі має сприяти автоматизації навчальних процесів і бути актуальним інструментом серед студентів й учнів.

3. Реалізація на основі дослідження

3.1 Опис майбутнього чат-боту

Опис майбутнього чат-боту: програма, що допомагає студентам у підготовці до будь-яких оцінок їхніх знань за визначеною тематикою. У його основі має закладатися робота великої мовної моделі для генерації тестів та для перевірки за наданими користувачем відповідями.

За допомогою кнопкових навігаційних меню студенти обираються потрібні їм дисципліни, мову для тесту, його тему та нейронну мережу, яка слугуватиме помічником у роботі. У відповідях програми закладені основи фатичного діалогу та певні аспекти з обробки природньої мови.

Перевагами використання майбутнього чат-боту є доступність через сервіси месенджеру Telegram, універсальність мов серед особливостей генерації тестів, легкість під час пошуку дисциплін, а також можливість вибору серед представників нейронних мереж зі списку для отримання найбільш бажаного результату.

3.2 Вибір засобів для розробки

Python – об’єктно-орієнтована інтерпретована високорівнева мова програмування, яка базується на суворій динамічній типізації. Вона гарантує простоту під час написання та загальну читабельність коду. Також серед плюсів є широкий вибір середовищ для роботи й безліч бібліотек з відкритим кодом, які орієнтуються на різні вимоги розробників. [35]

Середовищем розробки для написання чат-боту є PyCharm – кросплатформена IDE для визначеної мови програмування. Серед переваг використання є автоматизована перевірка синтаксису Python з можливими зручними підказками, наявністю додаткових можливостей налаштування інтерфейсу й встановленням бібліотек. Розробники надають безкоштовний інструментарій зі всіма інструкціями для початківців. [36]

Як було зазначено в минулому розділі, платформою для інтеграції чат-боту є

месенджер Telegram через його актуальність серед користувачів і зручності в інструментах для налаштувань.

PostgreSQL – спеціальний інструмент для системи керування базами даних з можливістю розгортання їхньої роботи на сервері та відкритим кодом. Він є зручним для створення реляційних моделей з широким вибором для типізації параметрів, високим рівнем безпеки та можливістю розширювання. [37]

Під час розробки програми за допомогою мови програмування Python було використано такі бібліотеки:

1. aiogram – інструмент для більш зручної та пришвидшеної роботи з Telegram чат-ботами. У структурі лежить принцип асинхронності за допомогою бібліотеки asyncio та aiohttp, основи яких також використовувалися в коді. [38] Додатково, для гнучкості реєстрації та відслідковування подій було підключено модуль logging. [39]
2. psycopg2 – інструмент для підключення баз даних PostgreSQL, він є одним із найзручніших та найпопулярніших адаптерів через безпеку потоків, ефективність та гнучкість. [40]
3. Модулі docx, spire.doc, pandas та difflib використовуються в проекті для обробки текстових та табличних документів у форматі від MS Office для отримання інформації про предмети в розкладах різних предметів університету. Останній працює за механізмом обробки елементів у послідовності для відсіювання повторювальних і непотрібних представників. [41]- [44]
4. g4f – бібліотека для підключення до нейронних мереж з механізмами API та провайдерами. Розробники надають перелік доступних мовних моделей та способи їхнього доступу. Через підтримку асинхронності доволі добре підходить для розробки майбутнього чат-боту. [45]

Серед наданих нейронних мереж було обрано таких представників:

GPT-3.5, GPT-4 та Llama-2 основуються на мовних моделях “gpt-3.5-turbo”, “gpt-4”, “gpt-4-turbo” від компанії OpenAI й “Llama-2-70b-chat-hf” від Meta, а за допомогою провайдерів You, Liaobots і Blackbox надається доступ на

підключення до однойменних веб-сайтів для використання їхніх можливостей. Ці представники були обрані серед інших через швидкісні показники, підтримку різномовного контенту та правильності текстової генерації. Детальніше про їхню роботу буде розглянуто в наступному розділі.

3.3 Огляд та тестування

Перш за все слід розглянути створення бази даних за допомогою PostgreSQL:

Для збереження інформації про всі наявні дисципліни та їхню приналежність до визначених факультетів, років навчання та спеціальностей було створено таблицю “Lessons”, яка має атрибути lesson_id(Primary key), lesson_name, faculty, specialization і year.

Для збереження інформації про всіх користувачів, які кооперують з чат-ботом, було створено таблицю “Users”. Основною метою цієї таблиці є доступ до потрібних даних при взаємодії з програмою та відслідковування відповідних запитів. Таблиця має атрибути: user_id(Primary key), user_name, chat_id та current_request. Типи даних для колонок з обох таблиць зображені на рисунку 3.3.1.

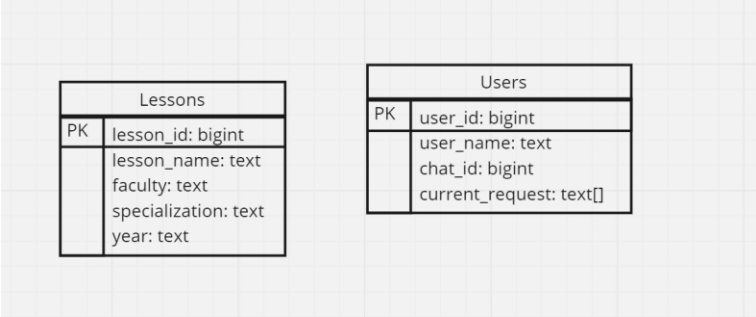


Рисунок 3.3.1 – Таблиці у базі даних

Надалі через спеціальний чат-бот @BotFather у Telegram відбувається базове налаштування майбутньої програми: команда /newbot визначає назву та надає посилання для майбутнього чат-боту, а за допомогою кнопкового меню встановлюється опис та фото його профіля(рис. 3.3.2-3.3.3).

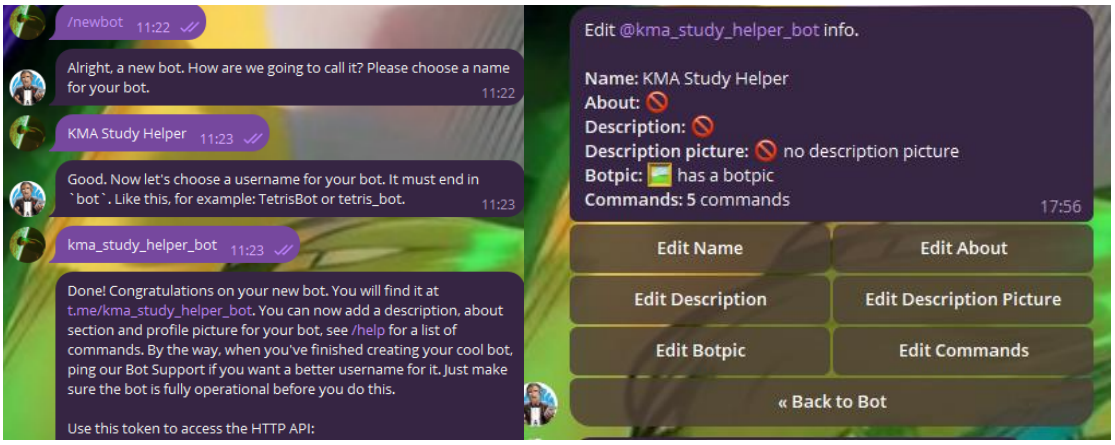


Рисунок 3.3.2-3.3.3 – Базове налаштування чат-боту у Telegram

Для цілісного збереження конфіденційних параметрів для підключення до бази даних, керування чат-ботом й доступом до локальних файлів у проекті створено окрему групу модулів під назвою Configurations. Файл botToken.py відповідає за підключення до Telegram за наданим до цього токеном, використовуючи бібліотеку aiogram. Файл databaseConfig.py зберігає в собі всі дані для підключення у вигляді змінних, а filesConfig.py тримає в собі посилання на розташування потрібних локальних файлів:

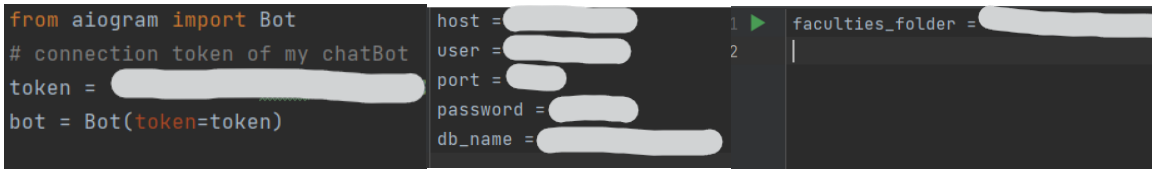


Рисунок 3.3.4-3.3.6 – Файли конфігурації програми

Локальні файли потрібні для збереження розкладів дисциплін під час певного навчального триместру. За допомогою парсингу цих файлів можливо дістати всі наявні предмети в певний учбовий період і їхню співзалежність із факультетом, роком навчання та спеціальністю. Отримання таких документів було можливим за допомогою спеціального сайту для студентів НаУКМА [46]. Завдяки цьому було створено спеціальну структуру збереження локальних файлів у головній папці FilesToParse: від папок з назвами факультетів до документів із назвами спеціальностей, в який збережено розклад:

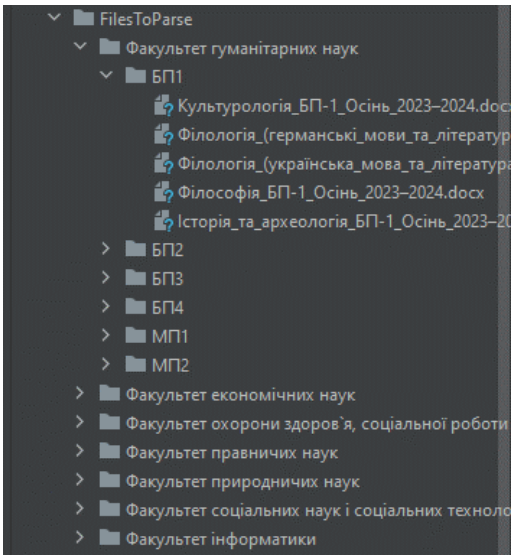


Рисунок 3.3.7 – Структура збереження локальних файлів FilesToParse для отримання інформації про дисципліни

Для роботи з парсингом розкладів у структурі групи DataBase існує модуль GetLessons.py. Оскільки в локальних документів існують різні розширення від MS Office, то для обробки було створено такий алгоритм: відповідно до типу файлу слід викликати визначені методи, щоб отримати результат у вигляді словнику в змінній dictionary_database для зручного додавання в базу даних.

Функція fill() працює як основна обгортка процесу: спочатку створюється локальна змінна lesson_id для майбутнього впровадження ключів уроків, отримуються назви факультетів у вигляді списку за допомогою локального посилання на папку FilesToParse, надалі за допомогою циклів працює механізм обробки структури вкладених файлів для отримання року навчання й спеціалізації. Отримання значень їхніх назв у форматі списків допомагає для роботи вкладених циклів і доступу до файлів розкладу за посиланнями. Далі відбувається перевірка згідно з розширенням остаточних документів:

1. Файли типу .docx обробляються за допомогою методу parseDocx(path_doc), де вхідним параметром є локальне посилання.
2. Файли типу .doc попередньо конвертуються до типу .docx за допомогою методу convertToDocx(path1, path2), де вхідними параметрами слугують попереднє й нове посилання. Після конвертування старий документ

видаляється, а обробка здійснюється аналогічно до попереднього пункту.

3. Файли типу .xlsx обробляються за допомогою методу parseExcel(path_doc), де вхідним параметром є локальне посилання.

Результуючим завданням для всіх є перевірка того, що знайдені заняття присутні в розкладі, за таких обставин відбувається виклик методу getLessonsNames(lessons) та підготовка до додавання інформації в базу даних за допомогою функції prepareForDB(lesson_id, lessons, faculty, specialization, year), в кінці змінений словник dictionary_database повертається:

```
dictionary_database = {}


def fill():
    lesson_id = 0
    # get names of the faculties in the begging folder
    faculties = os.listdir(faculties_folder)
    for faculty in faculties:
        # get years for each faculty
        years_folder = faculties_folder + f"//{faculty}"
        years = os.listdir(years_folder)
        for year in years:
            # get specializations for each faculty year
            specializations_folder = years_folder + f"//{year}"
            specializations = os.listdir(specializations_folder)

            for specialization in specializations:
                path1 = specializations_folder + f"//{specialization}"
                if specialization.endswith('.docx'):
                    # find lessons
                    lessons = parseDocx(path1)
                    if len(lessons) > 0:
                        names = getLessonsNames(lessons)
                        lesson_id = prepareForDB(lesson_id, names, faculty, getSpecializationNames(specialization), year)
                elif specialization.endswith('.doc'):
                    # convert to docx and remove .doc file
                    present_path = path1 + "x"
                    convertToDocx(path1, present_path)
                    os.remove(path1)
                    # find lessons
                    lessons = parseDocx(present_path)
                    if len(lessons) > 0:
                        names = getLessonsNames(lessons)
                        lesson_id = prepareForDB(lesson_id, names, faculty, getSpecializationNames(specialization), year)
                elif specialization.endswith('.xlsx'):
                    # find lessons
                    lessons = parseExcel(path1)
                    if len(lessons) > 0:
                        names = getLessonsNames(lessons)
                        lesson_id = prepareForDB(lesson_id, names, faculty, getSpecializationNames(specialization), year)
    return dictionary_database
```

Рисунок 3.3.8 – Метод fill() в модулі GetLessons.py

Детальний розгляд допоміжних методів, що викликаються у функції fill(): Методи для парсингу приймають на вхід посилання на локальний файл, за потреби конвертують його до іншого типу, за допомогою бібліотек отримують внутрішні дані й згідно з визначеними списками triggers проводять обробку

інформації для одержання результуючого списку.

Метод getLessonsNames(lessons) приймає на вхід список можливих дисциплін, визначає результуючий список lessons_result і список triggers для розділення. Метою функції є отримання назв предметів без дублікатів і зайвих символів, тому під час обробки циклу серед вхідного списку відбувається перевірка наявності назви предмету в кожному представнику за допомогою тригерних слів. Завдяки визначенню заборонених символів продовжується процес видозмінення поточного слова. Допоміжна змінна correct передбачена для можливості аналізу щодо доречного додавання теперішнього елементу в результуючий список. За умови того, що вона має правдиве значення, тобто поточне слово є предметом, відбувається перевірка на дублікати за допомогою функції similarityWords(word1, word2). Її робота основана на поверненні коефіцієнту схожості між двома словами за допомогою SequenceMatcher від бібліотеки difflib. Оцінювання двох параметрів і застосування методу ratio() повертає число на проміжку від 0 до 1, де 0 – не схожі, 1 – ідентичні. Отже, за допомогою цієї функції та перевірок на приналежність початкового й обробленого слова одне до одного випливає результат доречності додавання елементу до кінцевого списку:

```
def similarityWords(word1, word2):
    return SequenceMatcher(None, word1, word2).ratio()


def getLessonsNames(lessons):
    lessons_result = []
    # triggers to split
    triggers = ["ст.", "доц", "асн", "проф"]
    # getting only the names of lessons
    for lesson in lessons:
        correct = True
        for trigger in triggers:
            if lesson != 'Дисципліна, викладач' and trigger in lesson and len(lesson) > 1:
                les = lesson.split(trigger)[0]
                les = les.split('(')[0]
                if len(les) > 1:
                    # check if ends with letter
                    symbols = [' ', ',', '-', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
                    banned_words = ['лекція', "практика", "до уваги"]

                    last = les[-1]
                    while les and (last in symbols or les.endswith('\n')):
                        les = les[:-1]
                        last = les[-1]

                    first = les[0]
                    while les and (first in symbols or les.endswith('\n')):
                        les = les[1:]
                        first = les[0]

                    for w in banned_words:
                        if w.lower() in les.lower():
                            correct = False

                    # add only without banned word to res and unique
                    if correct:
                        for res in lessons_result:
                            if les in res or res in les or similarityWords(res, les) > 0.7:
                                correct = False

                    if correct:
                        lessons_result.append(les)
    return lessons_result
```

Рисунок 3.3.9 – Методи getLessonsNames(lessons) і similarityWords(word1, word2) в модулі GetLessons.py

Останнім методом цього модулю є prepareForDB(lesson_id, lessons, faculty, specialization, year), приймає на вхід інформацію про предмет і за визначеними аргументами заповнює глобальний словник dictionary_database, де ключем є lesson_id, а значенням є список з назви й приналежності до факультету, спеціальності й року навчання.

Наступним модулем в папці DataBase є DBManager.py, який слугує допоміжним інструментом для управлінням базою даних. Бібліотека psycopg2 та модуль Configurations.databaseConfig, який був розглянутий раніше, відповідають за підключення до бази даних і створення курсору. Метод для початкового налаштування бази даних і її заповнення створений для одноразового виклику за необхідності оновлення даних таблиць або їхнього створення за допомогою додаткових методів з використанням запитів мовою SQL та каскадного безпечного очищення даних з відомостей про дисципліни. Заповнення відбувається згідно з імпортованого методу fill() з модуля GetLessons.py й виклику методу додавання у базу даних add_lesson(lesson_id, lesson_name, faculty, specialization, year). Вхідні параметри слугують значеннями відповідних атрибутів у базі даних. За допомогою запиту мовою SQL та курсору відбувається підключення й виконання з можливістю відстеження й логування можливих помилок:

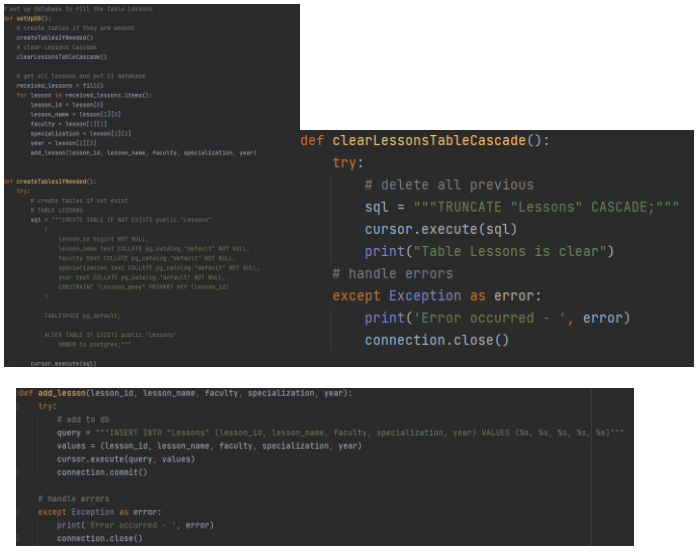


Рисунок 3.3.10-3.3.12 – Методи в модулі DBManager.py

Для кожної з таблиць реалізовано додаткові методи для управління й отримання даних. Першою є таблиця Users з такими функціями:

- add_user(user_id, user_name, chat_id) відповідає за додавання нового екземпляру користувача до бази даних. За умови того, що такий представник є новим, тобто перевірка від методу check_user(user_id) повернула хибне значення, відбувається запит типу INSERT мовою SQL з використанням наданих вхідних параметрів й встановленням атрибуту current_request значення None.
- check_user(user_id) відповідає за перевірку наявності екземпляру користувача з наданим ключем user_id у таблиці. При запиті типу SELECT і використанню методу fetchone() від створеного курсору повертається значення при поверненні одного екземпляру, та нічого в інших випадках.
- set_request(user_id, current_request) встановлює значення для атрибуту current_request в екземпляра з наданим ключем user_id за допомогою запиту типу UPDATE.
- get_request(user_id) повертає значення атрибуту current_request в екземпляра з наданим ключем user_id за допомогою запиту типу SELECT і методу fetchone().

У всіх методах передбачено відслідковування помилок та логування за потреби, а також виконання запитів за допомогою курсора:

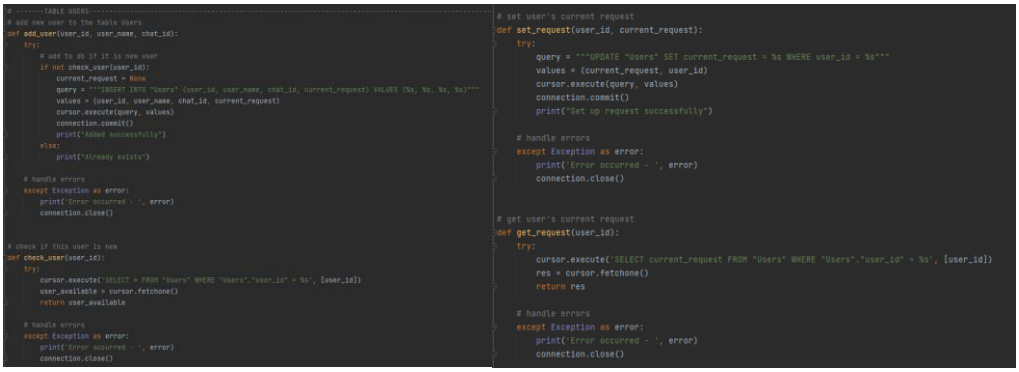


Рисунок 3.3.13-3.3.14 – Методи для таблиці Users в модулі DBManager.py

Для таблиці Lessons, окрім вже згаданих методів, існують ще такі:

- get_all(name) відповідає за отримання всіх уніфікованих значень атрибуту на

ім’я name з таблиці Lessons та їхню обробку до зручного вигляду в форматі списку.

- get_years_exact(current_request), get_specialization_exact(current_request) і get_lessons_exact(current_request) відповідають за отримання всіх наявних років навчання, спеціальностей та назв предметів відповідно, впорядкованих за алфавітом. За допомогою вхідного параметру current_request отримуються потрібні для запитів значення. Процес обробки до зручного вигляду у форматі списку відбувається аналогічно до попередньої функції.
- get_lesson_Id(current_request) відповідає за отримання уніфікованого ключа lesson_id згідно з наданими значеннями факультету, року навчання, спеціальності та назви предмету, що зберігається у вхідному параметрі current_request. Результатом є повернення значення цього ключа в єдиному варіанті за допомогою методу fetchone().

Процес обробки запитів всіх методів відбувається за допомогою курсора та типу запиту SELECT DISTINCT, у всіх функціях передбачено відслідковування помилок та логування за потреби:

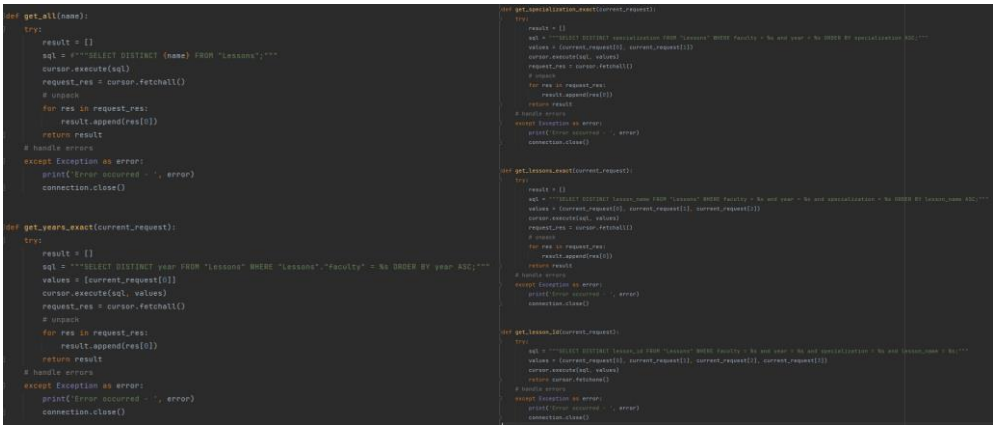


Рисунок 3.3.15-3.3.16– Методи для таблиці Lessons в модулі DBManager.py

Для управління чат-ботом створено групу BotFunctionality з головним файлом TelegramBot.py. У ньому відбувається запуск програми та її базові налаштування: перелік доступних команд set_up_commands() за допомогою types з бібліотеки aiogram, боту зі створеної конфігурації та підключенню роутера для обробки й оновлення подій до головного диспетчеру.

```python
# list of available commands in bot
async def set_up_commands():
    bot_commands = [
        types.BotCommand(command="/start", description="Розпочати діалог"),
        types.BotCommand(command="/help", description="Допомога"),
        types.BotCommand(command="/search", description="Пошук дисципліни"),
        types.BotCommand(command="/generate_test", description="Створити тест"),
        types.BotCommand(command="/check_my_answers", description="Перевірити відповіді на тест")

    ]
    await bot.set_my_commands(bot_commands)


# set up bot with commands and logging
async def main():
    await set_up_commands()
    dp = Dispatcher(storage=MemoryStorage())
    dp.include_router(router)
    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot, allowed_updates=dp.resolve_used_update_types())


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)
    asyncio.run(main())
```

Рисунок 3.3.17 – Модуль TelegramBot.py

Для абстракції методів інтерфейсу існує модуль UI_Elements.py, основною метою якого є створення кнопкових меню типів ReplyKeyboard(меню біля клавіатури без callback) та InlineKeyboard(меню біля повідомлення з callback) в залежності від запиту. Перший тип використовується для навігації під час вибору дисципліни, створюючи кнопки з назвами факультетів, років навчання, спеціальностей і предметів й надаючи підказки у полі для вводу повідомлень. Другий тип використовується для надання меню вибору та перевірок для конкретних тестів, їхнім управлінням, а також доступу до посилань зовнішнього браузера до веб-сторінок наявних нейронних мереж. Під час створення таких кнопок використовується механізм callback для надійної швидкої обробки подій при натисках.

Для отримання текстових відповідей чат-боту існує модуль Answers.py з методом get_answer(topic, input_text), який в залежності від отриманої тематики та можливого параметру для вставлення віддає текстову відповідь за принципом фатичного діалогу.

Модуль States.py за допомогою класу станів з бібліотеки aiogram створений для відслідковування подій 2 класів: Search(StatesGroup) для обробки того, що користувач обрав все необхідне для генерації предмету і

UserRequests(StatesGroup) для майбутнього очікування вхідного тексту від користувача.

Модуль Handlers.py створено для інкапсуляції логіки роботи команд чат-боту та його функціональних особливостей. Визначається роутер, який підключається до головного модулю, дістається загальна повна інформація з бази даних(списки всіх факультетів, років навчання, спеціальностей, дисциплін) і вказується список доступних нейронних мереж.

Далі визнається логіка обробки команд за допомогою роутера. Оскільки бібліотека aiogram працює асинхронно, то кожен запит користувача буде прив’язано лише до нього. Для того, щоб зберегти інформацію в базі даних під час першої обов’язкової команди start з очищенням можливих станів, створено функцію get_user_info(message) для отримання користувацьких відомостей і викликано метод add_user(user_id, user_name, chat_id) з модулю управління DBManager.py. Текстову відповідь чат-боту отримано з розглянутого модулю Answers.py й надіслано до персонального листування.

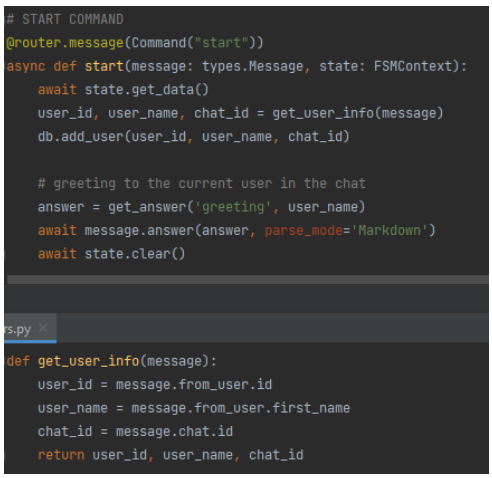


Рисунок 3.3.18 – Огляд команд у модулі Handlers.py

Логіка команди search передбачає ще навігаційне кнопкове меню типу ReplyKeyboard для пошуку дисципліни й поступове оновлення запитів користувача в базі даних за допомогою методу set_request(user_id, current_request) відповідного модулю. Спочатку попередній запит потрібно очистити, й з кожним кроком вибору додавати нове значення, видаляти й

модифікувати при поверненні для заміни відповідно. Для цього використовується механізм станів, а саме клас Search і його параметри. Створено асинхронні функції для модифікації запитів, контролю цілісності і надання потрібних меню: choose_year(message, text, state: FSMContext), choose_specialization(message, text, state: FSMContext), async def choose_lesson(message, text, state: FSMContext). На момент навігації років записується обраний факультет, під час вибору спеціальності або відбувається перехід для переобрання факультету, або записується вже вказаний рік, аналогічно й на вкладці про з предметами. Проте за умови того, що користувач обрав все, й перевірка наявності спрацювала, до запиту додається назва дисципліни за допомогою методу add_to_request(current_request, value, user_id) й показується меню типу InlineKeyboard для повторної перевірки користувачем в інтерфейсі чат боту. За відсутності будь-якої потрібної інформації надсилається текстове повідомлення й прохання обрати заново, стани завжди вимикаються. Допоміжна функція працює на основі розширення списку методом вставки елементу з модифікацією запиту згідно з розглянутим модулем.

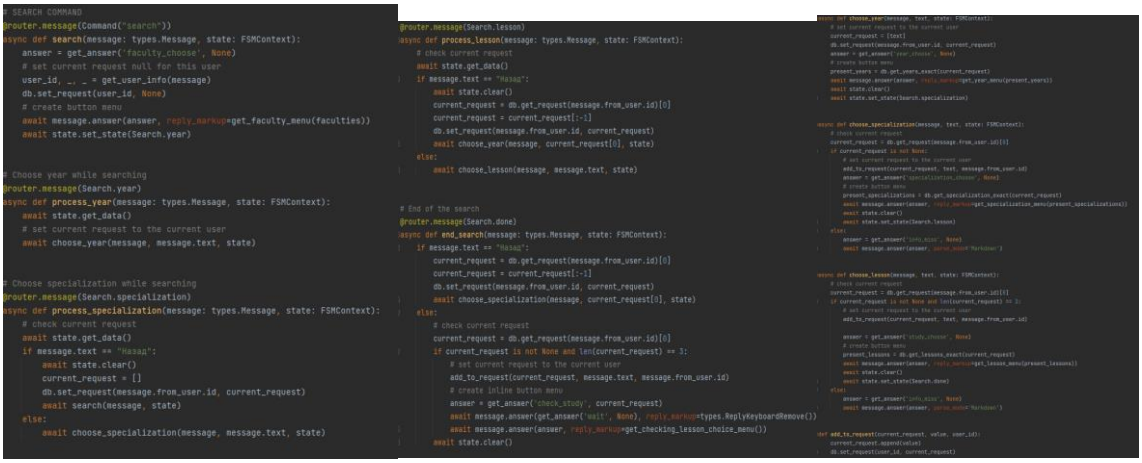


Рисунок 3.3.19-3.3.21– Огляд команд у модулі Handlers.py

Перевірка правильності запиту дисципліни відбувається через callback, влаштований до InlineKeyboard. У разі спрацювання натиску на саме ці кнопки запит даних буде починатися з ключового рядку “lesson_” і автоматично

оброблятиметься за допомогою асинхронного методу handle_lesson_choice(callback: types.CallbackQuery, state: FSMContext). Логіка роботи базується на виокремленні чіткої вказівки(правильний запит чи ні) за допомогою розділення стрічки й отриманню ідентифікатора чату. При некоректному варіанті користувачу буде надіслано повідомлення про зміну дисципліни, в іншому випадку – отримується теперішній запит, перевіряється на наявність потрібної інформації(за відсутності надсилається повідомлення про це), модифікується за потреби й звертається до методу create_topic(message: types.Message, state: FSMContext). Допоміжна функція надсилає повідомлення про очікування теми майбутнього тесту й встановлює стан UserRequests.topic_waiting для наступної обробки.

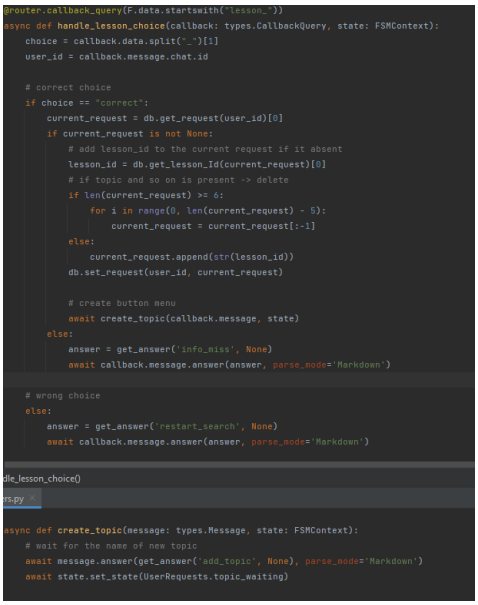
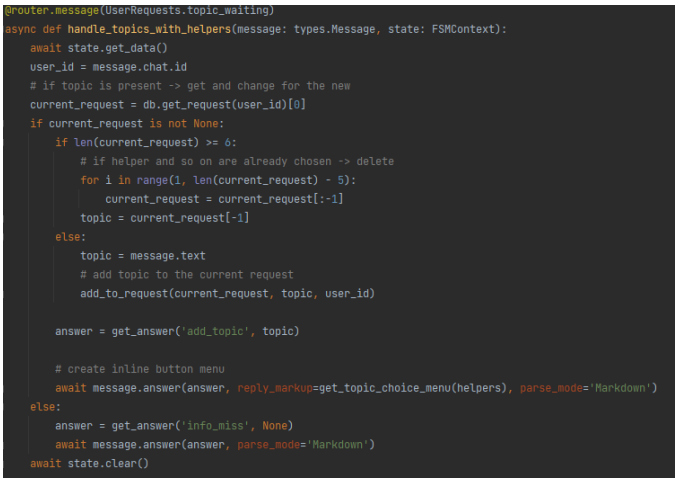


Рисунок 3.3.22 – Огляд роботи callback у модулі Handlers.py

За наявності потрібного стану спрацьовує метод для обробки вхідної теми: отримується інформація про користувача, перевіряється теперішній запит, за відсутністю потрібної інформації відправляється повідомлення користувачу, інакше – оновлюється запит й надсилається повідомлення з InlineKeyboard для вибору нейронної мережі чи переобрання теми, стани очищуються.



```python
@router.message(UserRequests.topic_waiting)
async def handle_topics_with_helpers(message: types.Message, state: FSMContext):
    await state.get_data()
    user_id = message.chat.id
    # if topic is present -> get and change for the new
    current_request = db.get_request(user_id)[0]
    if current_request is not None:
        if len(current_request) >= 6:
            # if helper and so on are already chosen -> delete
            for i in range(1, len(current_request) - 5):
                current_request = current_request[:-1]
            topic = current_request[-1]
        else:
            topic = message.text
            # add topic to the current request
            add_to_request(current_request, topic, user_id)

        answer = get_answer('add_topic', topic)

        # create inline button menu
        await message.answer(answer, reply_markup=get_topic_choice_menu(helpers), parse_mode='Markdown')
    else:
        answer = get_answer('info_miss', None)
        await message.answer(answer, parse_mode='Markdown')
    await state.clear()
```

Рисунок 3.3.23 – Огляд обробки теми у модулі Handlers.py

Надалі за допомогою callback, встановленого в попередньому меню, викликається метод handle_helpers(callback: types.CallbackQuery, state: FSMContext), де отримується інформація про користувача й уніфікується теперішній запит. За умови зміни теперішньої мовної моделі викликається попередня функція, в іншому випадку згідно з перевірками теперішнього запиту користувача надається або відповідь про недостатню інформацію, або вже встановлені дані й меню для вибору мови(українська, англійська, інша) чи поверненню назад.

У залежності від мови, що обрав користувач(програмно створена чи своя), відбувається перевірка запиту і його оновлення, або ж залучення роботи мовної моделі для контролю правильності вхідної назви за стану UserRequests.lang_waiting. Перевірка введеного тексту обробляється з додатковими налаштуваннями запиту користувача в базі даних і надає повідомлення про опрацювання його запиту в методі handle_language_input(message: types.Message, state: FSMContext). Відповідно до перевірки мовної моделі результат має містити в собі значення true або false згідно з дійсним існуванням мови відповідно. У будь-якому випадку коректності викликається метод completion_request(current_request, language, message: types.Message) для остаточного упакування запиту, інакше – навігація назад і повідомлення користувачу. Допоміжна функція завершення виконує контроль цілісності й надсилає інформаційне меню для перевірки запиту

користувачу в разі коректності й можливості змін, інакше - повідомлення про відсутність потрібних даних.

За допомогою пакунку LLM й модулю LLMManager.py працюють запити до наявних мовних моделей. Основа роботи закладається в бібліотеці g4f й вказанням потрібної моделі чи її провайдера. Всі методи працюють з урахуванням обробки помилок. Для перевірки коректності мови створена асинхронна функція check_language(request) з вхідним параметром назви мови. Контроль надається мовною моделлю gpt-3.5-turbo від OpenAI, запитом є штучно створена стрічка для перевірки істини твердження, яке за ручним тестуванням показало найкращі результати. За результатами обробки надається текст, який має містити в собі слово true чи false у випадку, якщо вхідна мова існує чи ні.

```
async def check_language(request):
    try:
        res = ""
        response_chat = await g4f.ChatCompletion.create_async(
            model="gpt-3.5-turbo",
            messages=[{"role": "user",
                       "content": f"Чи існує реально мова під назвою {request}? Відповідй надай у вигляді одного слова true(якщо існує) і false(якщо ні"}],
        )
        for message in response_chat:
            res += message
        print(res)
        return res
    except Exception as e:
        print(f"Error occurred {e} 39")
        return "Something went wrong\n._. Try again ._."
```

Рисунок 3.3.24 – Огляд роботи мовної моделі для перевірки мови у модулі LLMManager.py

У випадку того, що обрані для тесту дані коректні, спрацьовує callback для генерації тесту, інакше – повернення до вибору мови. Для створення контролю знань існує також окрема команда generate_test, яка як і обробка переадресування ґрунтується на виклику допоміжного методу test_generating(message: types.Message). Отримуючи й перевіряючи дані про користувача, він надає повідомлення про очікування впродовж генерації згідно з цілісністю, викликає метод test_maker(current_request, user_id, message) для передачі сформованого запиту до обраної мовної моделі, зберігає тест до екземпляру поточного користувача у базі даних й за функцією test_showing(user_id, message) виводить результат у месенджер з можливістю

повторної генерації, перевірки введених відповідей і навігації.

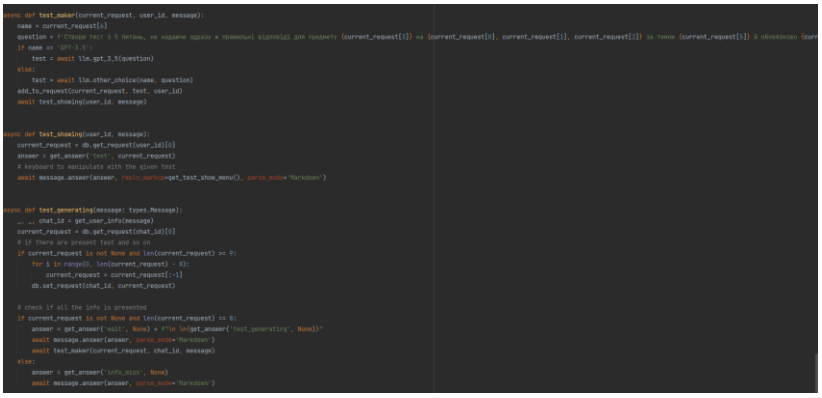


Рисунок 3.3.25 – Огляд процесу генерації тесту у модулі Handlers.py

Згідно з обраною нейронною мережею відбувається виклик потрібних функцій. Модель gpt-3.5-turbo від OpenAI працює аналогічно з методом перевірки мови, змінний лише вхідний параметр, який слугує повним запитом. Для інших нейронних мереж існує перевірка для встановлення потрібного провайдера чи виклику за допомогою назви мовної моделі(для gpt-4-turbo від OpenAI). Після цього за допомогою функцій other_provider(provider, question) і model_name_req(model, question) передається запит і потрібний тип підключення. Результат модифікується й повертається.

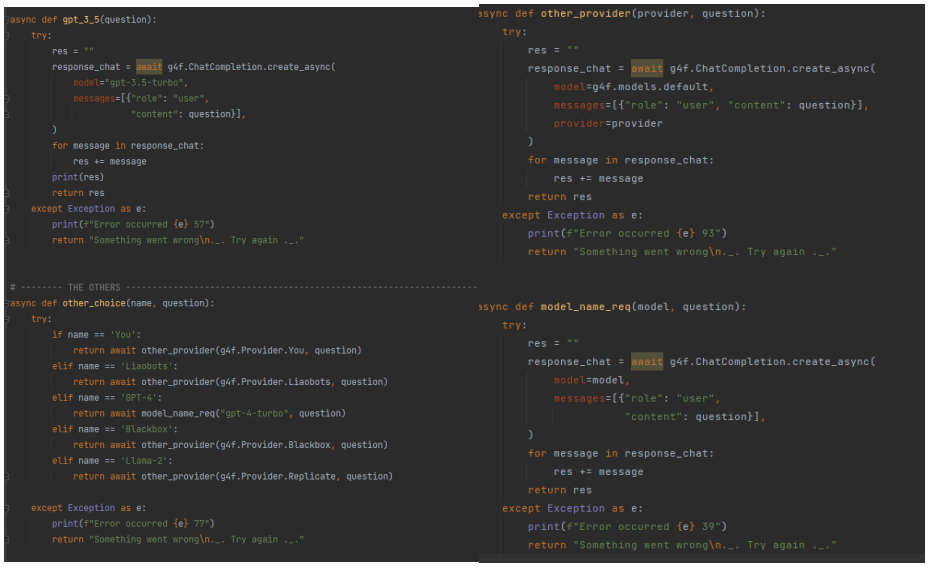


Рисунок 3.3.26-3.3.27 – Огляд роботи мовних моделей для генерації тестів у модулі LLMManager.py

Для контролю введених користувачем відповідей до згенерованого тесту існує обробка callback з можливістю додаткової перевірки й спеціальним повідомленням про це, а також команда check_my_answers. Робота обох функцій базується на виклику метода check_answers_test(message: types.Message, state: FSMContext) зі встановленням стану UserRequests.answers_waiting для очікування вхідного тексту. За таких обставин надані відповіді передаються до методу check_answers_bot(message: types.Message, answers), який перевіряючи цілісність всіх потрібних даних, надсилає запит на перевірку відповідей обраною мовною моделлю в модулі LLMManager.py. За результатами цього контролю надсилається повідомлення-відповідь нейронної мережі з можливістю повторної перевірки.

```python
# Handle checking result
@router.callback_query(F.data.startswith("check_results"))
async def check_result(callback: types.CallbackQuery, state: FSMContext):
    if len(callback.data.split("_")) == 3:
        await callback.message.answer(get_answer('additional_checking_answers_process', None), parse_mode='Markdown')
    await check_answers_test(callback.message, state)


# Handle input answers
@router.message(UserRequests.answers_waiting)
async def handle_test_answers(message: types.Message, state: FSMContext):
    await state.get_data()
    ans = message.text
    # if topic is present -> get and change for the new
    answer = get_answer('wait', None) + f"\n{get_answer('checking_answers_process', None)}"
    await message.answer(answer, parse_mode='Markdown')
    await check_answers_bot(message, ans)
    await state.clear()
```

```python
async def check_answers_test(message: types.Message, state: FSMContext):
    current_request = db.get_request(message.chat.id)[0]
    if current_request is not None and len(current_request) == 9:
        await state.get_data()
        answer = get_answer('check_answers', None)
        await message.answer(answer, parse_mode='Markdown')
        # handle answers
        await state.set_state(UserRequests.answers_waiting)
    else:
        answer = get_answer('info_miss', None)
        await message.answer(answer, parse_mode='Markdown')


async def check_answers_bot(message: types.Message, answers):
    current_request = db.get_request(message.from_user.id)[0]
    if current_request is not None and len(current_request) == 9:
        name = current_request[6]
        test = current_request[8]
        question = f'Перевір тест {test}, маючи такі відповіді {answers}. Надай правильні відповіді до нього та вкажи на помилки, які допустив користувач'
        if name == 'GPT-3.5':
            ans = await llm.gpt_3_5(question)
        else:
            ans = await llm.other_choice(name, question)
        # keyboard to manipulate with the given test
        answer = get_answer('checking_answers_done', ans)
        await message.answer(answer, reply_markup=get_test_check_menu())
    else:
        answer = get_answer('info_miss', None)
        await message.answer(answer, parse_mode='Markdown')
```

Рисунок 3.3.28-3.3.29 – Огляд процесу перевірки тесту у модулі Handlers.py

Також у чат-боті наявна команда help для надання інформації про

користування й кнопкового меню для переходу до веб-сторінок наявних мовних моделей, за якої очищуються можливі стани.

```
@router.message(Command("help"))
async def help_command(message: types.Message, state: FSMContext):
    await state.get_data()
    answer = get_answer('help_command', None)
    await message.answer(answer, reply_markup=get_help_menu())
    await state.clear()
```

Рисунок 3.3.30 – Огляд команди help у модулі Handlers.py

Отже, перейдемо до тестування роботи чат-боту:

- Запуск чат-боту. Команди start і help, меню всіх команд
  Бот має працювати, показуючи свій опис і фото профілю, а також надаючи список усіх команд. При першому використанні цих команд має надаватися потрібне повідомлення про початок роботи й надання інформаційного повідомлення відповідно. Виклик help повинен за натиском на кнопки відкривати посилання веб-сторінки наявних нейронних мереж:

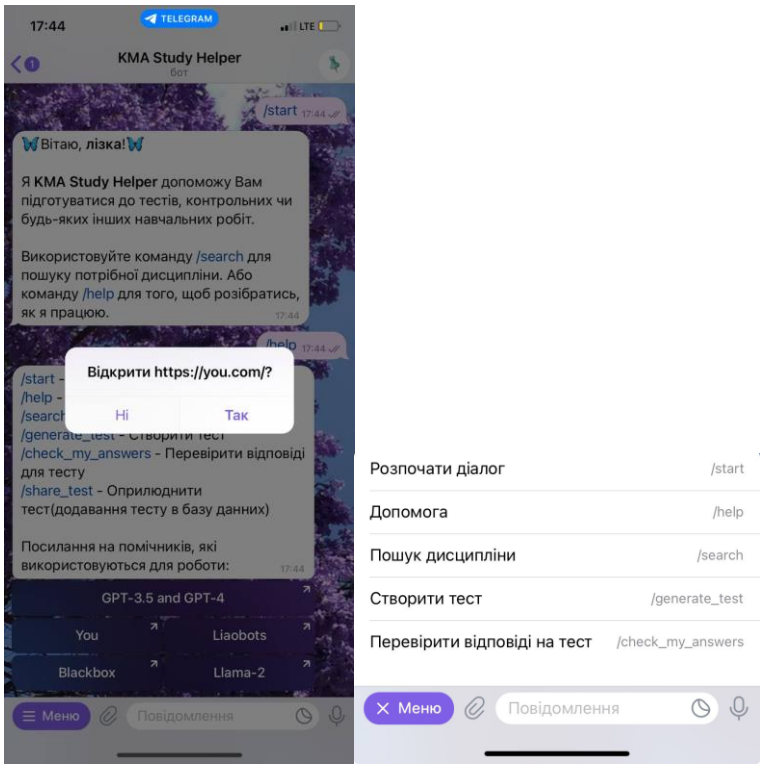
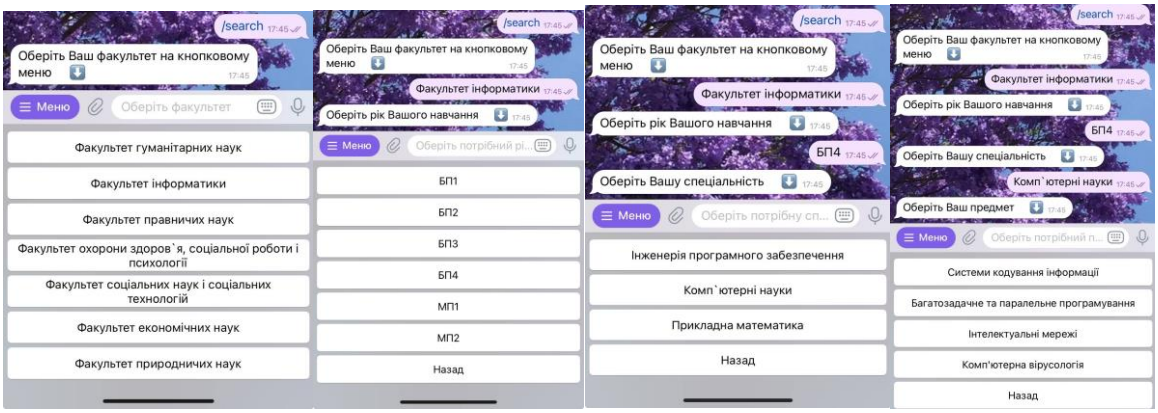


Рисунок 3.3.31-3.3.32 – Тестування команд початкового запуску

- Пошук за командою search
  Має надаватися навігаційне меню для пошуку й текстові підказки в

повідомленнях і полях вводу. Далі перевірка запиту користувачем за допомогою кнопок:



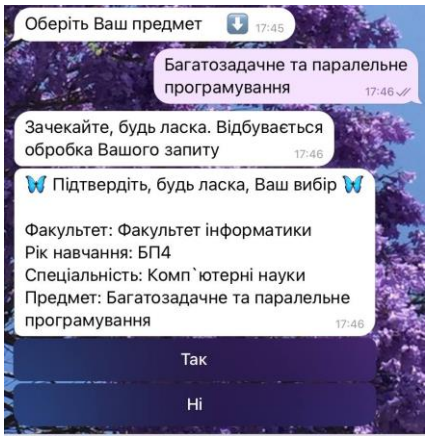


Рисунок 3.3.33-3.3.37 – Тестування вибору дисципліни з командою search

Користувач має вказати тему, обрати помічника й мову:

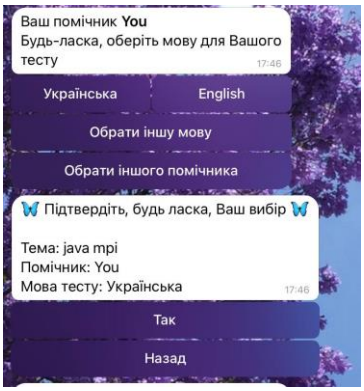


Рисунок 3.3.38 – Тестування вибору мовної моделі й мови

Зміни мають підтримуватися кнопковим меню, запити для вказання власної мови мають проходити перевірку моделлю:

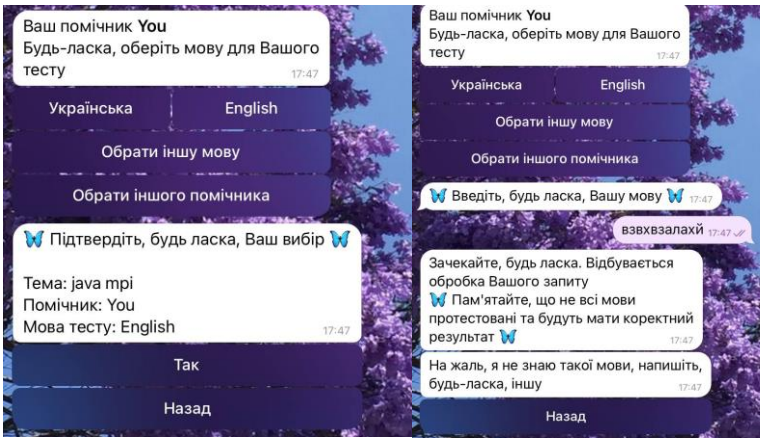
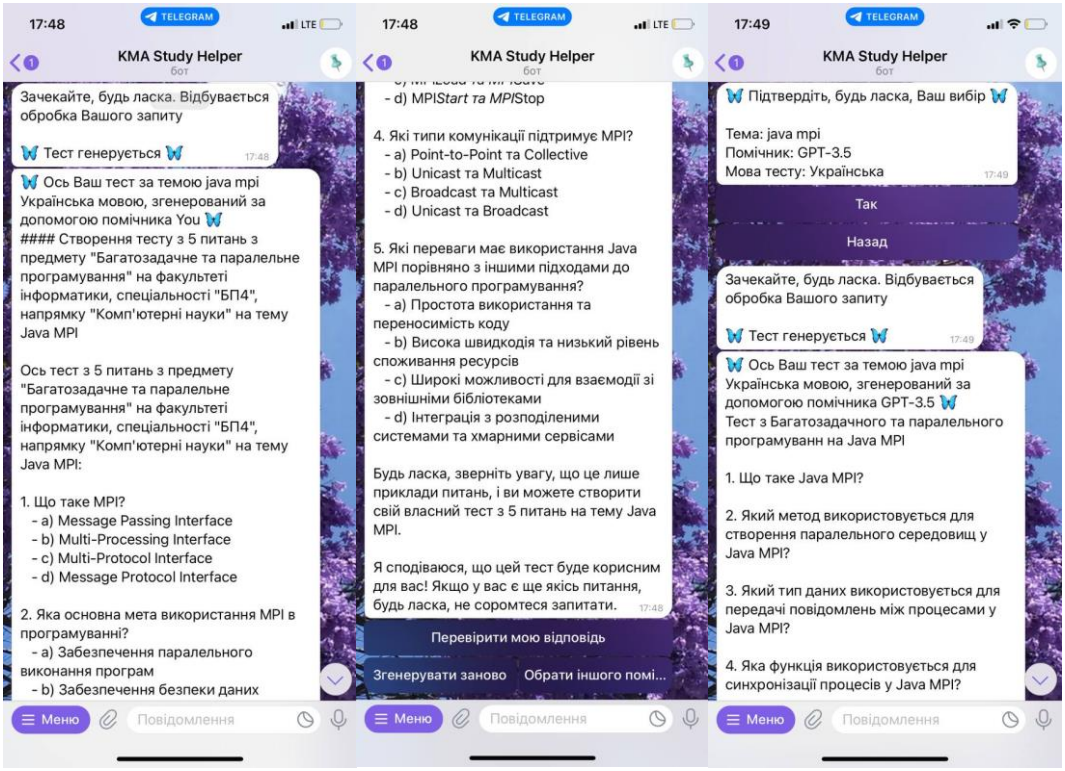


Рисунок 3.3.39-3.3.40 – Тестування зміни мови тесту

- Генерація за допомогою різних мовних моделей

  За вказанням всіх правильних даних, тест має генеруватися з можливими подальшими змінами параметрів вибору. Усі моделі мають працювати також за командою generate_test:



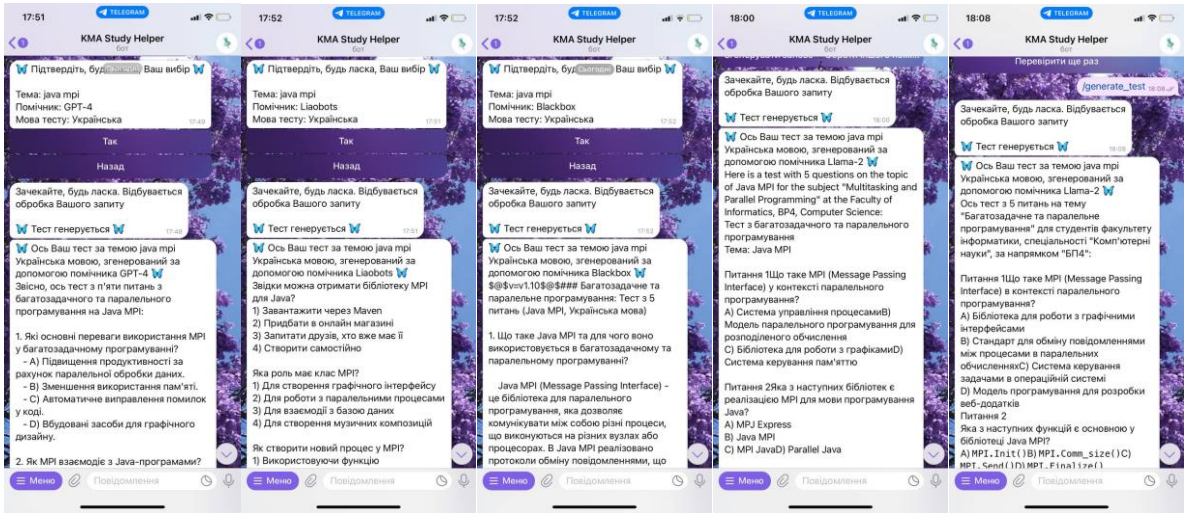


Рисунок 3.3.41-3.3.48 – Тестування генерації тесту різними мовними моделями

- Перевірка введених відповідей

  Надаючи текстові відповіді для тесту в зазначеному форматі після натиску на спеціальну кнопку або після виклику функції check_my_answers, обрана мовна модель має їх перевіряти з подальшим наданням додаткових спроб:

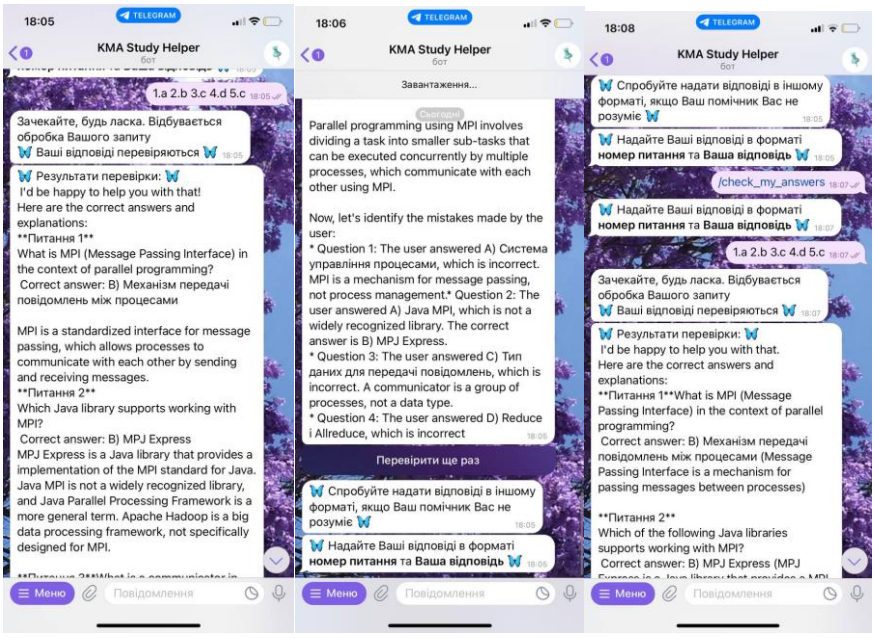


Рисунок 3.3.49-3.3.51 – Тестування перевірки відповідей користувача до його тесту

Отже, тестування пройдено успішно!

Висновок

Завдяки цій кваліфікаційній роботі було вивчено й розглянуто теоретичні відомості про чат-боти й мовні моделі. Також було опановано матеріали про роботу нейронних мереж, їхню типізацію, способи розробки й інтеграції.

Відповідно до дослідження і його мети було створено чат-бот у месенджері Telegram для зручної підготовки до оцінок знань з дисциплін, що викладаються в НаУКМА, за допомогою «розумних помічників». Оцінка роботи мовних моделей за допомогою тестування поглибила розуміння основ трансферного навчання нейронних мереж і найбільш оптимальних способів для їхнього підключення.

Програма, яку було розроблено, написана мовою програмування Python, включає в собі низку модулів для роботи з аналізом й обробкою розкладу, управлінням базою даних, асинхронної роботи запитів до наявного списку мовних моделей, а також керуванням і налаштуванням самого Telegram чат-боту. У документі детально й обґрунтовано описано вибір усіх використаних ресурсів.

В останньому розділі подано докладно описану архітектуру програми, ретельно розглянуто алгоритм її роботи з прикладами тестування для наглядної демонстрації коректного опрацювання запитів користувача.

У майбутніх перспективах, чат-бот можна впровадити для постійного онлайн використання студентами для спрощення процесу оцінки їхніх знань. Також програму можна вдосконалювати за допомогою розробки власних мовних моделей.

Список використаної літератури

1. [Електронний ресурс] What is a chatbot? https://www.oracle.com/chatbots/what-is-a-chatbot/
   (Перевірка на доступ до ресурсу - 07.01.2024)
2. [Електронний ресурс] What is a chatbot? https://www.ibm.com/topics/chatbots
   (Перевірка на доступ до ресурсу - 07.01.2024)
3. [Електронний ресурс] What is LLM? https://www.elastic.co/what-is/large-language-models
   (Перевірка на доступ до ресурсу - 07.01.2024)
4. [Електронний ресурс] What is LLM? https://www.nvidia.com/en-us/glossary/large-language-models/
   (Перевірка на доступ до ресурсу - 07.01.2024)
5. [Електронний ресурс] SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning? https://blogs.nvidia.com/blog/supervised-unsupervised-learning/
   (Перевірка на доступ до ресурсу - 07.01.2024)
6. [Електронний ресурс] LLM
   https://www.techtarget.com/whatis/definition/large-language-model-LLM
   (Перевірка на доступ до ресурсу - 07.01.2024)
7. [Електронний ресурс] A Brief History of Large Language Models
   https://www.dataversity.net/a-brief-history-of-large-language-models/
   (Перевірка на доступ до ресурсу - 07.01.2024)
8. [Електронний ресурс] ELIZA: a very basic Rogerian psychotherapist chatbot
   https://web.njit.edu/~ronkowit/eliza.html
   (Перевірка на доступ до ресурсу - 07.01.2024)
9. [Електронний ресурс] A Gentle Introduction to Generative Adversarial Networks (GANs)
   https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
   (Перевірка на доступ до ресурсу - 07.01.2024)

10. [Електронний ресурс] Language Models are Few-Shot Learners
https://arxiv.org/abs/2005.14165
(Перевірка на доступ до ресурсу - 07.01.2024)
11. [Електронний ресурс] PaLM 2
https://ai.google/discover/palm2/
(Перевірка на доступ до ресурсу - 10.01.2024)
12. [Електронний ресурс] XLNet: Generalized Autoregressive Pretraining for Language Understanding
https://arxiv.org/abs/1906.08237
(Перевірка на доступ до ресурсу - 10.01.2024)
13. [Електронний ресурс] What are large language models?
https://www.eweek.com/artificial-intelligence/large-language-model/#types
(Перевірка на доступ до ресурсу - 07.01.2024)
14. [Електронний ресурс] Zero-Shot Classification
https://huggingface.co/tasks/zero-shot-classification
(Перевірка на доступ до ресурсу - 15.01.2024)
15. [Електронний ресурс] THE FUTURE OF AI: HOW MULTIMODAL MODELS ARE LEADING THE WAY
https://www.leewayhertz.com/multimodal-model/
(Перевірка на доступ до ресурсу - 15.01.2024)
16. [Електронний ресурс] LLM Fine-Tuning Strategies for Domain-Specific Applications
https://dzone.com/articles/llm-fine-tuning-strategies-for-domain-specific-app
(Перевірка на доступ до ресурсу - 15.01.2024)
17. [Електронний ресурс] Advantages and Disadvantages of LLM Machine Translation versus Neural Machine Translation
https://blog.pangeanic.com/advantages-and-disadvantages-of-llm-machine-translation-versus-neural-machine-translation
(Перевірка на доступ до ресурсу - 15.01.2024)
18. [Електронний ресурс] What Is Transfer Learning? Exploring the Popular Deep

Learning Approach.
https://builtin.com/data-science/transfer-learning
(Перевірка на доступ до ресурсу - 30.01.2024)

19. [Електронний ресурс] How Do You Build a Large Language Model?
https://truera.com/ai-quality-education/generative-ai-overview/how-do-you-build-a-large-language-model/
(Перевірка на доступ до ресурсу - 30.01.2024)
20. [Електронний ресурс] Microsoft Bot Framework
https://dev.botframework.com/
(Перевірка на доступ до ресурсу - 30.01.2024)
21. [Електронний ресурс] 6 Ways For Running A Local LLM
https://semaphoreci.com/blog/local-llm
(Перевірка на доступ до ресурсу - 30.01.2024)
22. [Електронний ресурс] Beginner’s Guide to OpenAI API
https://medium.com/data-professor/beginners-guide-to-openai-api-a0420bc58ee5
(Перевірка на доступ до ресурсу - 30.01.2024)
23. [Електронний ресурс] Key Chatbot Statistics You Should Follow in 2024
https://www.chatbot.com/blog/chatbot-statistics/
(Перевірка на доступ до ресурсу - 30.01.2024)
24. [Електронний ресурс] ChatGPT large language model: Everything you need to know
https://indatalabs.com/blog/chatgpt-large-language-model
(Перевірка на доступ до ресурсу - 30.01.2024)
25. [Електронний ресурс] The Pros and Cons of ChatGPT: Unveiling the Power of AI Language Models
https://gleematic.com/the-pros-and-cons-of-chatgpt-unveiling-the-power-of-ai-language-models/
(Перевірка на доступ до ресурсу - 30.01.2024)
26. [Електронний ресурс] The Pros and Cons of ChatGPT: Unveiling the Power of AI Language Models

https://www.unite.ai/uk/llama-2-a-deep-dive-into-the-open-source-challenger-to-chatgpt/
(Перевірка на доступ до ресурсу - 30.01.2024)

27. [Електронний ресурс] Llama 2: A Comprehensive Guide https://www.simform.com/blog/llama-2-comprehensive-guide/
(Перевірка на доступ до ресурсу - 30.01.2024)
28. [Електронний ресурс] How Copilot works, technically speaking https://www.microsoft.com/en-us/bing/do-more-with-ai/how-bing-chat-works?form=MA13KP
(Перевірка на доступ до ресурсу - 30.01.2024)
29. [Електронний ресурс] Weighing the Pros and Cons: Is Microsoft Copilot Worth It? https://victorycto.com/microsoft-copilot-pros-and-cons/
(Перевірка на доступ до ресурсу - 30.01.2024)
30. [Електронний ресурс] Try Bard and share your feedback https://blog.google/technology/ai/try-bard/
(Перевірка на доступ до ресурсу - 30.01.2024)
31. [Електронний ресурс] Advantages and Disadvantages of Google Bard https://www.careerera.com/blog/advantages-and-disadvantages-of-google-bard
(Перевірка на доступ до ресурсу - 30.01.2024)
32. [Електронний ресурс] Для 50,6% читачів основним месенджером є Telegram. Результати опитування AIN.UA https://ain.ua/
(Перевірка на доступ до ресурсу - 30.01.2024)
33. [Електронний ресурс] Pros, cons and use cases of telegram chatbots https://botpenguin.com/blogs/pros-cons-and-use-cases-of-telegram-chatbots
(Перевірка на доступ до ресурсу - 17.02.2024)
34. [Електронний ресурс] How to Create a Telegram Bot using Python https://www.freecodecamp.org/news/how-to-create-a-telegram-bot-using-python/
(Перевірка на доступ до ресурсу - 17.02.2024)

35. [Електронний ресурс] Python Pros and Cons in 2024
https://www.netguru.com/blog/python-pros-and-cons
(Перевірка на доступ до ресурсу - 20.03.2024)
36. [Електронний ресурс] Get started with PyCharm
https://www.jetbrains.com/help/pycharm/feature-trainer.html
(Перевірка на доступ до ресурсу - 20.03.2024)
37. [Електронний ресурс] PostgreSQL Advantages and Disadvantages
https://www.aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/
(Перевірка на доступ до ресурсу - 20.03.2024)
38. [Електронний ресурс] aiogram
https://docs.aiogram.dev/uk-ua/dev-3.x/
(Перевірка на доступ до ресурсу - 20.03.2024)
39. [Електронний ресурс] logging — Можливість журналювання для Python
https://docs.python.org/uk/3/library/logging.html
(Перевірка на доступ до ресурсу - 20.03.2024)
40. [Електронний ресурс] psycopg2 2.9.9
https://pypi.org/project/psycopg2/
(Перевірка на доступ до ресурсу - 20.03.2024)
41. [Електронний ресурс] python-docx 1.1.2
https://pypi.org/project/python-docx/
(Перевірка на доступ до ресурсу - 20.03.2024)
42. [Електронний ресурс] Spire.Doc 12.4.0
https://pypi.org/project/Spire.Doc/
(Перевірка на доступ до ресурсу - 20.03.2024)
43. [Електронний ресурс] pandas 2.2.2
https://pypi.org/project/pandas/
(Перевірка на доступ до ресурсу - 20.03.2024)
44.[Електронний ресурс] difflib — Допоміжні засоби для обчислення дельта
https://docs.python.org/uk/3/library/difflib.html

(Перевірка на доступ до ресурсу - 20.03.2024)

45. [Електронний ресурс] g4f
https://pypi.org/project/g4f/
(Перевірка на доступ до ресурсу - 20.03.2024)

46. [Електронний ресурс] M.CA3
https://my.ukma.edu.ua/profile
(Перевірка на доступ до ресурсу - 27.04.2024)