Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Інструмент автоматизованого створення навчальних тестів з використанням LLM Gemini та хмарних сервісів GCP
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-3
Владислав Волокітін
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ст. викл. В'ячеслав Гребенюк
(посада, власне ім'я, прізвище)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Волокітіну Владиславу Генріховичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інструмент автоматизованого створення навчальних тестів з використанням LLM Gemini та хмарних сервісів GCP

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 червня 20 25 р.

3. Вихідні дані до роботи Дані з Інтернет джерел та наукових проєктів, науково-технічні публікації, документація до бібліотеки Google GenerativeAI, документація до API Google Forms, документація до фреймворку Flet, згенерований API-ключ до моделі Gemini

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз систем управління навчанням та засобів тестування
2) LLM для автоматизації створення тестів
3) Практична реалізація системи
4) Результати та демонстрація роботи застосунку

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз систем управління навчанням	20.05.2025	виконано
3	Порівняння існуючих платформ тестування	21.05.2025	виконано
4	Аналіз архітектури сучасних великих мовних	22.05.2025	виконано
	моделей		
5	Визначення вимог до системи створення тестів	23.05.2025	виконано
6	Практична реалізація застосунку	28.05.2025	виконано
7	Тестування та експериментальні дослідження	30.05.2025	виконано
8	Написання пояснювальної записки	04.06.2025	виконано
9	Перевірка на академічний плагіат	05.06.2025	виконано
10	Нормоконтроль	07.06.2025	виконано
11	Підготовка презентації та доповіді	09.06.2025	виконано
12	Попередній захист	12.06.2025	виконано
13	Рецензування	14.06.2025	виконано
14	Захист перед ЕК	18.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. В’ячеслав Гребенюк
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 65 с., 26 рис., 1 табл., 1 дод., 20 джерел.

АВТОМАТИЗАЦІЯ, ГЕНЕРАЦІЯ ТЕСТІВ, ІНЖЕНЕРІЯ ПРОМПТІВ, ІНТЕРФЕЙС КОРИСТУВАЧА, ОСВІТНІЙ ПРОЦЕС, ТЕСТУВАННЯ, API, GEMINI, GOOGLE FORMS, LLM.

Об’єкт дослідження – процес автоматичного створення тестових завдань на основі вхідної текстової інформації.

Мета роботи – дослідження можливостей автоматичної генерації тестів та розробка архітектури системи для її реалізації з використанням великої мовної моделі (LLM) Gemini і програмного інтерфейсу (API) Google Forms.

Методи дослідження – аналіз та порівняння функціоналу інструментів тестування в системах управління навчанням (LMS); дослідження можливостей LLM щодо генерації текстового контенту; обґрунтування вибору моделі Gemini та Google Forms API як компонентів системи на основі критеріїв функціональності, інтеграції та доступності API.

У результаті виконаної роботи проведено аналіз та порівняльну оцінку інструментів тестування в різних LMS. Досліджено потенціал LLM для генерації освітнього контенту, виявлено переваги моделі Gemini для створення тестових завдань, та обґрунтовано вибір саме Gemini і Google Forms API як технологічної основи системи.

Розроблено архітектуру системи автоматичної генерації тестів та сформульовано принципи її функціонування. Архітектура передбачає взаємодію з Gemini API для обробки вхідного тексту та генерації структурованих тестових завдань і подальшу взаємодію з Google Forms API для програмного створення відповідної форми тесту.

ABSTRACT

Bachelor's thesis contains: 65 pp., 26 fig., 1 tabl., 1 ann., 20 references.

API, AUTOMATION, EDUCATIONAL PROCESS, GEMINI, GOOGLE FORMS, LLM (LARGE LANGUAGE MODELS), PROMPT ENGINEERING, TEST GENERATION, TESTING, USER INTERFACE.

The object of study is the process of automatic creation of test tasks based on input textual information.

Purpose – to study the possibilities of automatic test generation and develop a system architecture for its implementation using the Gemini Large Language Model (LLM) and the Google Forms Application Programming Interface (API).

Research methods – analysis and comparison of the functionality of testing tools in learning management systems (LMS); study of the LLM capabilities for generating text content; justification of the choice of the Gemini model and Google Forms API as system components based on the criteria of functionality, integration and accessibility of the API.

As a result of the work performed, the analysis and comparative evaluation of testing tools in different LMSs was carried out. The potential of LLMs for generating educational content was investigated, the advantages of the Gemini model for creating test tasks were identified, and the choice of Gemini and Google Forms API as the technological basis of the system was substantiated.

The architecture of the automatic test generation system is developed and the principles of its functioning are formulated. The architecture provides for interaction with the Gemini API for processing the input text and generating structured test tasks and further interaction with the Google Forms API for programmatic creation of the corresponding test form.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

BERT – Bidirectional Encoder Representations from Transformers – двоспрямовані кодувальні представлення з трансформерів;

BPE – Byte Pair Encoding – кодування парами байтів;

CNN – Convolutional Neural Networks – згорткові нейронні мережі;

GCP – Google Cloud Platform – хмарна платформа Google;

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

GUI – Graphical User Interface – графічний інтерфейс користувача;

JSON – JavaScript Object Notation – нотація об’єктів JavaScript;

LLM – Large Language Model – великі мовна модель;

LMS – Learning Management System – система управління навчанням;

MCQ – Multiple Choice Questions – питання з кількома правильними відповідями;

PDF – Portable Document Format – портативний формат документів;

REST API – Representational State Transfer Application Programming Interface – програмний інтерфейс передачі репрезентативного стану;

RLHF – Reinforcement Learning from Human Feedback – навчання з підкріпленням на основі зворотного зв’язку від людини;

RNN – Recurrent Neural Networks – рекурентні нейронні мережі;

SCQ – Single Choice Question – питання з однією правильною відповіддю;

SFT – Supervised Fine-Tuning – контрольоване тонке налаштування;

UI – User Interface – інтерфейс користувача;

URL – Uniform Resource Locator – уніфікований локатор ресурсів.

ВСТУП

В умовах сучасного освітнього процесу викладачі стикаються з необхідністю регулярного створення різноманітних оцінювальних матеріалів, серед яких тести є одним з найпоширеніших інструментів перевірки знань студентів. Розробка якісних тестових завдань, що відповідають змісту навчального матеріалу та педагогічним вимогам, є процесом, який вимагає значних витрат часу та зусиль. Ця рутинна складова педагогічної діяльності знижує ефективність підготовки до занять та обмежує можливості для індивідуалізації контрольних заходів.

Одночасно з цим, останні роки відзначилися стрімким прогресом у розвитку технологій штучного інтелекту, зокрема великих мовних моделей (LLM), які демонструють вражаючі можливості в аналізі та генерації тексту. Це відкриває нові перспективи для автоматизації інтелектуальних завдань й процесів створення навчального контенту.

Актуальність даної роботи полягає в обґрунтуванні та дослідженні потенціалу використання сучасних LLM для автоматизації процесу створення тестових завдань, що дозволить викладачам ефективніше використовувати свій час.

Метою даної роботи є дослідження можливостей та розробка програмної системи, здатної автоматично генерувати тестові завдання на основі наданого текстового матеріалу. Як ключові технологічні компоненти системи обрано велику мовну модель Gemini для аналізу тексту та генерації питань і програмний інтерфейс Google Forms API для формування фінального тесту у відповідному форматі.

1 АНАЛІЗ СИСТЕМ УПРАВЛІННЯ НАВЧАННЯМ ТА ЗАСОБІВ ТЕСТУВАННЯ

Система управління навчанням (LMS) – це універсальна платформа, яка дозволяє зручно організовувати процес онлайн навчання. LMS є середовищем, яке дозволяє об’єднати навчальні матеріали, оцінювання та адміністративні функції, щоб зробити процес навчання більш доступним та зрозумілим [1].

Такі системи знаходять широке застосування як в академічних закладах (школах, університетах) для підтримки формальної освіти, так і в корпоративному секторі для організації професійного навчання, підвищення кваліфікації та контролю відповідності стандартам.

Окрім забезпечення доступу до навчального контенту, сучасні LMS автоматизують багато аспектів освітнього процесу: керують обліковими записами користувачів (студентів, викладачів, адміністраторів), дозволяють створювати та призначати курси, відстежують індивідуальний прогрес та активність учнів, генерують звіти для аналізу ефективності навчання.

1.1 Дослідження систем управління навчанням

1.1.1 Функціонал та роль модулів тестування в LMS

Невід’ємним компонентом переважної більшості систем управління навчанням є модуль тестування за допомогою якого проводиться оцінювання. Функціонал цього модулю забезпечує одну з найважливіших цілей освітнього процесу – вимірювання рівня засвоєння знань студентами. Модулі тестування дозволяють створювати, проводити та аналізувати різноманітні контрольні заходи, що є основою для об’єктивного оцінювання навчальних досягнень, надання зворотного зв'язку та прийняття рішень щодо подальшого навчання.

Функції модулів тестування в сучасних LMS є досить широкими і зазвичай включають інструменти для створення тестів. Найбільш розповсюдженим є підтримка великої кількості типів тестових завдань: від класичних питань з однією чи кількома правильними відповідями, завдань на встановлення відповідності чи послідовності, до питань з короткою текстовою відповіддю, есе, числових та обчислювальних завдань. Важливою складовою є банк питань, що дозволяє зберігати, групувати та повторно використовувати тестові завдання для різних ситуацій, що значно підвищує ефективність роботи викладача. При створенні тесту доступні гнучкі налаштування: обмеження часу на виконання, визначення кількості спроб, можливість перемішування питань та варіантів відповідей [2].

Під час проведення тестування, LMS створює зручне середовище для студентів, а після завершення – надає інструменти для аналізу результатів. Для об’єктивних типів питань (де правильна відповідь чітко визначена), система здійснює автоматичне оцінювання, що миттєво розраховує бал студента. Для суб’єктивних питань (есе, розгорнуті відповіді) передбачено зручний інтерфейс для ручної перевірки та коментування викладачем. Усі отримані оцінки, як правило, автоматично передаються до електронного журналу платформи.

1.1.2 Класифікація та характеристика тестових завдань

Якщо казати про конкретну класифікацію питань, то кожен тип питань має унікальні характеристики, які визначають його придатність для перевірки конкретних знань або навичок. Наразі, існує велика вибірка типів питань, яка дозволяє покрити майже всі потреби в оцінювальному процесі:

– питання з однією правильною відповіддю (Single Choice Question), структура якого передбачає наявність запитання, до якого додається перелік можливих варіантів відповідей. З цього переліку лише один варіант є вірним, тоді як інші виступають як дистрактори (неправильні, але

правдоподібні варіанти), які потрібні для перевірки точності знань студента. SCQ є перевіреним і гарним варіантом для оцінки більшості факторів при навчанні;

– питання з кількома правильними відповідями (Multiple Choice Questions), які розширюють звичайний SCQ, дозволяючи студенту вибрати дві або більше правильних відповідей із запропонованого списку. MCQ є більш комплексним варіантом порівняно з SCQ, оскільки може вимагати часткового оцінювання (наприклад, зарахування балів за кожну правильно обрану відповідь або зняття балів за неправильно обрані). MCQ дозволяє зменшити ймовірність випадкового вгадування, але є концептуально складнішими, оскільки не завжди можна підібрати гарну комбінацію відповідей, щоб питання не було занадто простим або складним;

– питання типу «Правда/Неправда» (True/False), котрі представляють з себе твердження, яке студент повинен оцінити як вірне або хибне. Є найпростішим бінарним типом питання, яке часто використовується для швидкої перевірки розуміння теми або розвіювання поширених помилок. Перевагою є простота створення, але висока ймовірність вгадування робить цей тип менш надійним для оцінки знань при умові використанні його як єдиного інструменту;

– питання на встановлення відповідності (Matching Questions). Вони потребують від студента встановити логічних зв’язок між елементами двох або більше списків. Зазвичай один список містить терміни, поняття, імена, а інший – їхні визначення або характеристики. Таким чином гарно перевіряється асоціативне мислення та знання взаємозв’язків.

– питання на встановлення послідовності (Ordering/Sequencing Questions), де задачею є розташування елементів у певному логічному, хронологічному або ієрархічному порядку. Це можуть бути етапи процесу, історичні події, кроки алгоритму тощо;

– питання з розгорнутою відповіддю (Open-ended Questions), що дають студенту можливість сформулювати власну, розгорнуту відповідь на

поставлене питання. Цей тип є найменш структурованим і призначений для оцінки навичок критичного мислення, аналізу та аргументації. Через свою природу, розгорнуті питання вимагають ручної перевірки викладачем, оскільки автоматизовані системи аналізу тексту (навіть з використанням елементів штучного інтелекту) поки що не можуть повноцінно замінити експертну оцінку змісту, логіки та стилю відповіді.

Для досягнення ефективності та об’єктивності тестування, доцільним є комбінування різних типів питань. Наприклад, для перевірки загальних знань та розуміння теми ідеально підходять питання з однією правильною відповіддю. Водночас, для перевірки того, чи знають студенти чіткі факти, питання типу «Правда/Неправда» є ефективним і простим у використанні інструментом. Проте, для оцінки аналітичних здібностей або формулювання власних думок, гарним варіантом є питання з розгорнутою відповіддю. Хоча їхня перевірка вимагає ручного втручання, вони надають можливість оцінити глибину розуміння студентом матеріалу та його здатність застосовувати здобуті знання.

Тож, інтеграція в систему тестування цих трьох базових, але комплексних типів питань, дозволяє створити збалансований інструмент оцінювання, який додатково гарантує високу швидкість автоматизованої перевірки. Ці типи питань є найбільш універсальними та можуть бути реалізовані практично в будь-якій системі тестування, від найпростіших форм до повноцінних LMS, що робить їх надійним фундаментом для програмної генерації тестів.

1.2 Вибір платформи для програмного завантаження тестів

1.2.1 Екосистема Google та її функціональні можливості

Однією з найбільш розповсюджених та доступних цифрових освітніх екосистем на сьогодні є Google Workspace for Education. Ця платформа є

набором взаємопов'язаних хмарних інструментів Google (таких як Gmail, Drive, Docs, Sheets, Meet тощо), спеціально адаптованих для потреб навчальних закладів і, що важливо, надається їм на безкоштовній основі.

Організатором взаємодії між викладачем та студентами в цій екосистемі виступає Google Classroom. Це сервіс, що дозволяє організовувати групи студентів, публікувати завдання і навчальні матеріали, відстежувати успішність студентів. Приклад інтерфейсу класу можна побачити на рисунку 1.1.

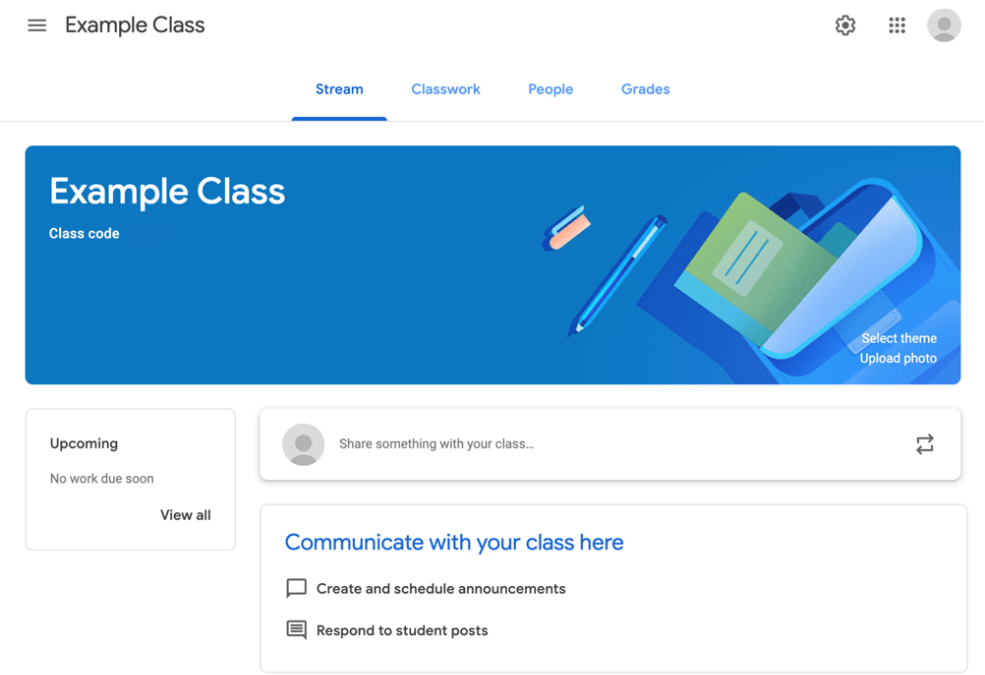

Рисунок 1.1 – Приклад інтерфейсу класу в Google Classroom

Google Classroom, на відміну інших LMS, не функціонує ізольовано, а активно використовує можливості інших сервісів: завдання можуть бути файлами з Google Drive, виконуватися в Google Docs чи Sheets, обговорення відбуватися в Meet, а результати зберігатися в одному конкретному місці. Це створює єдиний робочий простір, доступний з будь-якого пристрою, який підключений до мережі. Для викладачів та студентів, подібна система

дозволяє не витрачати час на технічне опанування застосунку, дозволяючи зосередитись на навчальному процесі [3].

В умовах дистанційної освіти, коли потрібно оцінити роботу студента або учня, викладачі часто використовують тестування або різного роду вікторини. Ключовим вбудованим інструментом в системі Google Classroom виступає Google Forms. Цей сервіс інтегрується напряму, дозволяючи легко створювати завдання з тестами, де форма слугує засобом перевірки знань.

За допомогою свого простого та зрозумілого вебінтерфейсу Google Forms, який можна побачити на рисунку 1.2, надає достатній функціонал для створення базових та середньої складності тестів. Викладач може використовувати різноманітні типи питань, що підходять для перевірки знань: питання з одним або кількома правильними варіантами відповідей, питання з короткою чи розгорнутою текстовою відповіддю, шкали, сітки тощо. Для більшості типів питань є можливість вказати правильні відповіді та призначити кількість балів, що дозволяє системі автоматично перевіряти роботи та виставляти оцінки, економлячи час викладача [4].

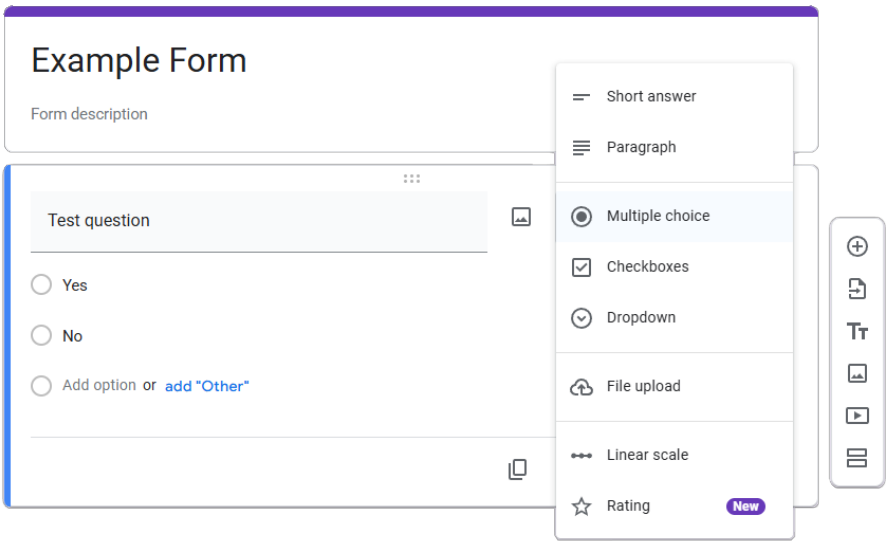

Рисунок 1.2 – Приклад інтерфейсу створення тесту в Google Forms

1.2.2 Переваги Google Forms API для автоматизації тестів

Незважаючи на функціональну глибину інструментів тестування в інших LMS (таких як Moodle, Blackboard), вибір саме Google Forms API як цільової платформи для програмної генерації тестів у даній роботі є достатнім для досягнення мети – повної автоматизації процесу. Хоча деякі LMS можуть надавати API, можливості програмного створення тестів з нуля або наповнення їх контентом із зовнішніх джерел через ці API часто є складнішими або менш придатними для повної автоматизації, ніж Google Forms API. Google Forms надає добре документований REST API, спеціально призначений для програмного створення та керування формами, включаючи додавання питань та окремих налаштувань [5].

Створені форми в застосунку Google Forms автоматично зберігаються на Google Drive користувача. Цей спосіб збереження дозволяє не прив’язувати згенеровані тести до конкретної системи та легко поширювати їх задля будь-яких потреб. Гарним бонусом виступає інтеграція з системою Google, яка надає універсальну авторизацію, спрощуючи доступ, редагування та поширення тестів для користувачів.

Тому в процесі аналізу було вирішено сфокусуватись на використанні саме Google Forms API для подальшої реалізації системи.

2 LLM ДЛЯ АВТОМАТИЗАЦІЇ СТВОРЕННЯ ТЕСТІВ

2.1 Поняття, архітектура та принцип дії великих мовних моделей

Великі мовні моделі (Large Language Models) є класом моделей машинного навчання, які є набагато більшими за обсягом тренувальних даних, кількістю параметрів та об’ємом потенціальних можливостей від стандартних моделей.

Сучасні LLM можуть містити від сотень мільйонів до трильйонів параметрів. Моделі навчаються на величезних, різноманітних об’ємах текстових даних, що охоплюють книги, статті, вебсторінки, вихідний код програм, наукові публікації тощо. Масштабний процес навчання дозволяє їм вивчати складні мовні закономірності, граматику, синтаксис, семантику, логічні зв’язки та різні стилі мовлення. Розвиток LLM позначив значний прорив у галузі штучного інтелекту, особливо в обробці природної мови.

Основне призначення великих мовних моделей – обробка та генерація людської мови на рівні, що наближається до людського. Вони виконують широкий спектр завдань, включаючи розуміння тексту, відповіді на питання, узагальнення інформації, переклад, створення нових текстів, написання коду, аналіз тональності і так далі.

2.1.1 Архітектура сучасних LLM

Переважна більшість сучасних великих мовних моделей побудовані на базі архітектури Трансформеру (Transformer), представленої у статті «Attention Is All You Need». Архітектура стала революційною, ефективно обробляючи послідовності даних та долаючи обмеження попередніх рекурентних (RNN) і згорткових (CNN) нейронних мереж. Схематично, архітектура представлена на рисунку 2.1 [6].

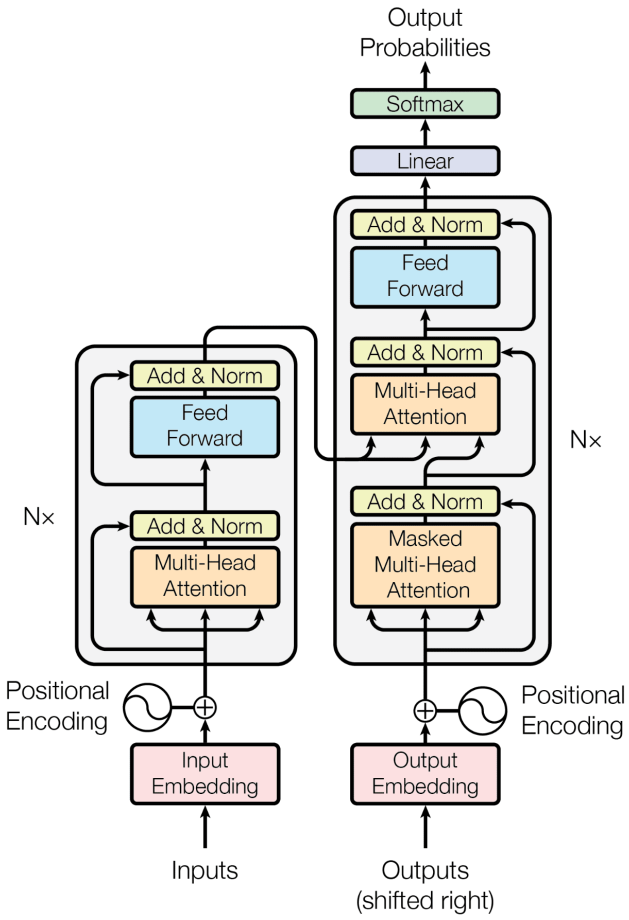

Рисунок 2.1 – Архітектура моделі Трансформеру

Головною відмінністю від інших архітектур став механізм уваги (attention), а якщо бути більш точним – само-уваги (self-attention). На відміну від RNN, які обробляють токени послідовно, механізм само-уваги дозволяє моделі одночасно зважувати важливість усіх токенів у вхідній послідовності при створенні представлення для кожного окремого токена. Для розуміння значення слова в реченні модель може «звернути увагу» на будь-які інші слова в цьому ж реченні, незалежно від їхньої відстані.

Ефективність механізму уваги досягається завдяки багатоголовій увазі (multi-head attention). Замість одного набору ваг для обчислення уваги, багатоголова увага паралельно використовує декілька «голів» уваги. Кожна «голова» навчається фокусуватися на різних аспектах взаємозв’язків між токенами. Наприклад, одна голова може відстежувати синтаксичні зв’язки,

інша – семантичні. Результати роботи всіх голів потім комбінуються та лінійно трансформуються у представлення, яке можна вважати результатом роботи моделі.

Архітектура Трансформеру не містить рекурентних або згорткових шарів, тому не має вбудованого способу врахування порядку даних в послідовності. Для вирішення цієї проблеми, до векторних представлень вхідних токенів додається позиційне кодування (positional encoding). Позиційне кодування – це вектори, що несуть інформацію про абсолютну або відносну позицію кожного токена в послідовності, дозволяючи моделі розрізняти слова на основі їхнього розташування.

Кожен блок енкодера або декодера в Трансформері, окрім шарів уваги, містить повнозв’язні нейронні мережі прямого поширення (feed-forward networks), які застосовуються до кожного токена окремо. Для стабілізації навчання глибоких мереж та полегшення проходження градієнтів використовуються залишкові з’єднання (residual connections) навколо кожного підшару з подальшою нормалізацією шарів (layer normalization).

Наразі, велика кількість LLM використовують модифіковані версії оригінальної архітектури Трансформера. Багато генеративних моделей, подібних до GPT (Generative Pre-trained Transformer), використовують лише декодерну частину Трансформера [7]. У таких моделях кожен токен може «відвідувати» лише попередні токени в послідовності через масковану само-увагу, що є природним для завдання генерації тексту, де наступний токен передбачається на основі попередніх. Інші моделі, як BERT, використовують лише енкодер для завдань розуміння мови.

2.1.2 Процес обробки тексту

Перед поданням тексту до LLM, він проходить етап токенізації – розбиття вхідної текстової послідовності на менші одиниці, названі

токенами. Токени можуть бути словами, частинами слів, або окремими символами. LLM працюють із фіксованим словником токенів.

Використання алгоритмів токенізації заснованих на аналізі частин окремих слів, наприклад Byte Pair Encoding (BPE), про який детальніше описано в [8], WordPiece (використовується в BERT) або SentencePiece (використовується в моделях Gemini), є стандартною практикою [9]. Методи дозволяють ефективно працювати з великими словниками, обробляти рідкісні слова шляхом розбиття їх на відомі частини слів та зменшувати кількість невідомих токенів (out-of-vocabulary tokens). Кожен токен зі словника конвертується у відповідний числовий вектор (embedding) перед подачею на вхід моделі.

2.1.3 Процес навчання моделей

Навчання сучасних LLM – складний та довготривалий процес, що включає в себе попереднє навчання (pre-training) та тонке налаштування (fine-tuning). Перший етап – попереднє навчання. На ньому модель навчається на величезних нерозмічених текстових даних, що можуть налічувати мільярди слів і їх частин, для вивчення загальних закономірностей мови. Основна мета – навчити модель передбачати пропущені частини тексту. Для декодерних моделей (наприклад, GPT-подібних) це каузальне мовне моделювання (Causal Language Modeling), де модель навчається передбачати наступний токен у послідовності, маючи попередні токени як контекст [10]. Для енкодерних моделей (наприклад, BERT) популярним є масковане мовне моделювання (Masked Language Modeling), де деякі токени у вхідному реченні випадково замінюються певним спеціальним токеном, і модель навчається передбачати оригінальні токени [11].

Після попереднього навчання базова модель має детальне розуміння мови, але може бути неоптимальною для виконання конкретних завдань або

генерації відповідей у бажаному стилі. Наступним кроком є її адаптація через тонке налаштування.

Тонке налаштування під наглядом (Supervised Fine-Tuning) передбачає донавчання моделі на меншому, але якісному наборі даних, що складається з прикладів бажаної поведінки, таких як пари «інструкція-відповідь» або «запитання-відповідь». За допомогою цього, модель зможе краще слідувати інструкціям та генерувати відповіді кращої якості.

Для покращення якості та відповідності людським очікуванням, наразі використовується відносно нова технологія навчання з підкріпленням на основі зворотного зв'язку людини (Reinforcement Learning from Human Feedback) [12]. Процес застосування RLHF починається зі збору даних, де об'єктом стають декілька відповідей на один запит, а люди-експерти оцінюють ці відповіді від кращої до гіршої. На основі цих порівнянь навчається окрема модель винагороди (Reward Model), яка прогнозує, наскільки «гарною» є відповідь LLM з емпіричної точки зору – з погляду людини. Завершальним етапом є тонке налаштування попередньо навченої SFT-моделі за допомогою обраного алгоритму навчання з підкріпленням. Схематично, зобразити цей процес можна на рисунку 2.2.

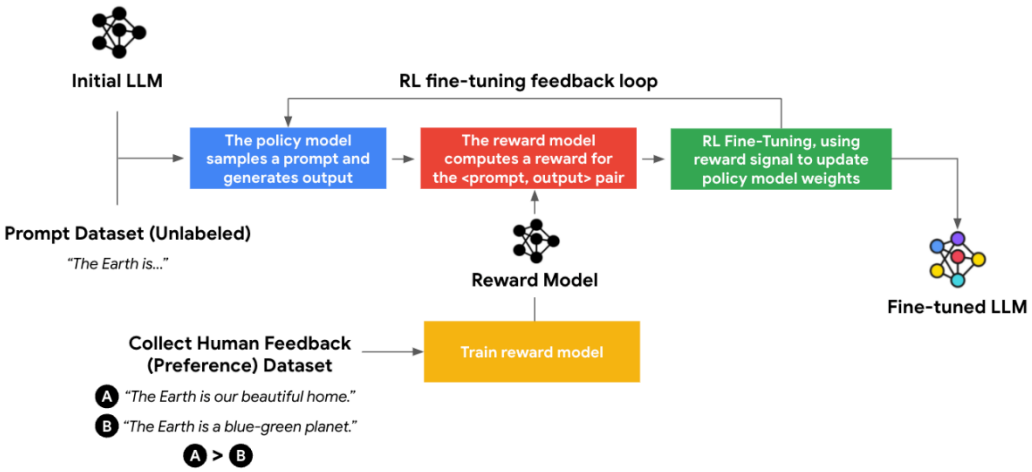

Рисунок 2.2 – Схема роботи RLHF підходу

Останнім часом з’являються альтернативні підходи до RLHF, наприклад Direct Preference Optimization, які намагаються досягти подібних результатів прямішим шляхом, без явного навчання моделі винагороди [13].

2.1.4 Генерація відповіді від моделей

Принцип дії LLM під час генерації тексту полягає у послідовному передбаченні наступного токена на основі вхідного контексту (промпту) та вже згенерованої частини послідовності. Отримавши промпт, модель спочатку токенізує його, а потім переводить його у векторні представлення. Вектори далі проходять обробку через шари Трансформера [14].

На кожному кроці генерації останній шар декодера видає розподіл ймовірностей, для кожного токена у словнику моделі як потенційне продовження послідовності. Спрощено, процес можна представити як на рисунку 2.1.

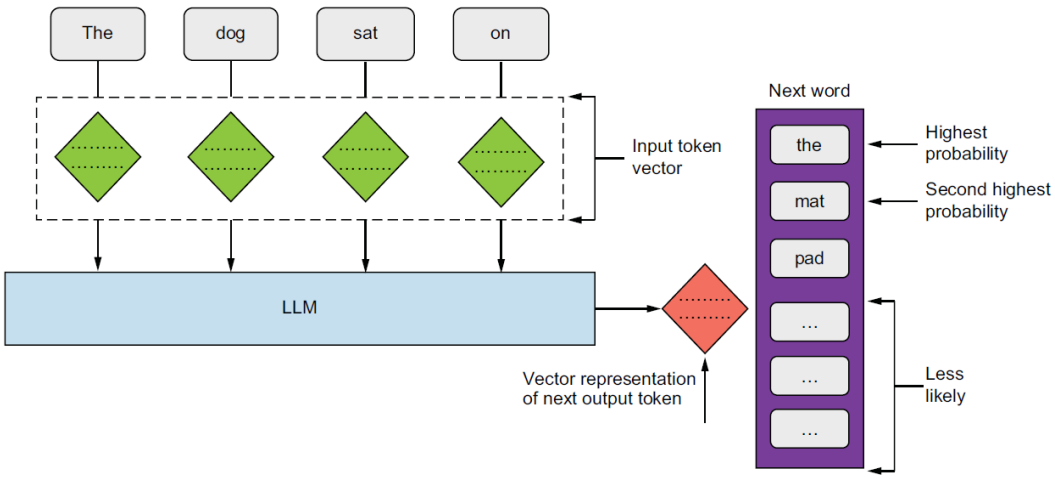

Рисунок 2.3 – Схема процесу прогнозування наступного токену

Після обрання потрібного токена з фінального розподілу, модель додає його до виходу і повторює процес, використовуючи оновлену послідовність як вхід для генерації наступного токена. Вибір токена не

завжди є детермінованим, тобто вибором найбільш ймовірного токена через «жадібний пошук».

Для контролю різноманітності та креативності генерації використовуються різні стратегії семплювання. Наприклад, параметр температури (Temperature scaling) контролює випадковість передбачень: нижча температура робить вивід більш детермінованим та сфокусованим, вища – більш випадковим та креативним.

Top-k семплювання обмежує вибір наступного токена k найімовірнішими токенами. Top-p семплювання вибирає наступний токен з найменшого можливого набору токенів, сукупна ймовірність яких перевищує поріг p.

Ітеративний процес генерації триває до досягнення спеціального токена кінця послідовності або максимальної заданої довжини генерації, дозволяючи моделі створювати зв’язний, логічний та контекстуально відповідний текст.

2.1.5 Процес навчання моделей

Дві важливі характеристики сучасних LLM – закони масштабування та емерджентні властивості.

Закони масштабування (Scaling Laws) описують, як продуктивність LLM, вимірювана як втрати на тестових даних або якість виконання певних завдань, передбачувано покращується зі збільшенням трьох ключових факторів: розміру моделі (кількості параметрів), обсягу навчальних даних та обчислювальних ресурсів, витрачених на навчання.

Закони дозволяють прогнозувати ефективність більших моделей ще до їх фактичного навчання.

Емерджентні властивості (Emergent Abilities) – це здібності, які з’являються у великих моделях, але відсутні або значно менш виражені у менших моделях того ж сімейства. Властивості не програмуються явно, а

виникають як побічний ефект масштабування. Прикладами можуть слугувати здатність до виконання арифметичних операцій, відповіді на питання з кількома кроками логічного висновку, або здатність до навчання «на льоту» (in-context learning / few-shot learning), коли модель демонструє здатність виконувати нове завдання на основі кількох прикладів, наданих у промпті, без зміни своїх ваг.

2.2 Актуальність застосування великих мовних моделей для автоматизації освітнього процесу

Швидкий розвиток та зростаючі можливості великих мовних моделей в частині розуміння складного контексту та генерації тексту, роблять їх надзвичайно актуальними для застосування у освітніх проектах. Потреба в ефективній обробці знань, створенні навчальних матеріалів та оцінюванні рівня їх засвоєння є постійною.

Зазвичай, процес створення оцінювальних завдань вимагає великої кількості часових ресурсів. В умовах зростаючої кількості інформації та потреби в окремій перевірці кожного зі студентів або учнів, автоматизація рутинних задач є не просто важливим, а необхідним рішенням для підвищення ефективності та забезпечення якісної освіти.

Одним з найбільш перспективних напрямків застосування LLM в освіті є автоматизоване створення навчальних матеріалів або, наприклад, тестових завдань. Базуючись на вхідному тексті (наприклад, параграфі з підручника, лекційними нотатками чи науковою статтею), LLM здатні аналізувати зміст, виділяти ключові поняття і факти, а потім генерувати тестові питання різних форматів. Це не тільки значно скорочує час на підготовку, але й дозволяє викладачам більше зосередитися на аналізі результатів навчання та індивідуальній роботі зі студентами, підвищуючи загальну якість освітнього процесу.

2.3 Вибір LLM для реалізації системи генерації тестів

2.3.1 Обмеження та критерії відбору моделі

Для успішної побудови проекту з автоматизованої генерації тестових завдань, потрібно врахувати наступні аспекти:

– модель повинна мати велике контексне вікно;

– модель повинна ефективно та швидко обробляти великий обсяг вхідної інформації (тобто, моделі з доступом по API);

– здатність формувати структурований вихід (structured output) у відповідності до поставленої задачі;

– орієнтація на моделі з оптимально низькою вартістю експлуатації для забезпечення економічної доцільності та широкої доступності розроблюваного рішення.

При виборі підходу до використання великої мовної моделі було відхилено варіант локальної розгортки моделей (self-hosted) через значні вимоги до обчислювальних ресурсів, складності розгортання та обслуговування, що виходить за рамки даної роботи.

2.3.2 LLM Gemini та її переваги

Відповідаючи критеріям, наведеним у попередньому пункті, для цілей даної роботи було здійснено аналіз низки великих мовних моделей, доступних через API та які належать до бюджетного цінового сегменту за 1 мільйон використаних токенів. Найпопулярніші з них, разом із ключовими характеристиками, представлені у таблиці 2.1. З цього переліку, за сукупністю таких факторів, як вартість, розмір контекстного вікна та індекс інтелектуальної продуктивності (intelligence index), оптимальним вибором стала модель Gemini Flash 2.0 [15]. Дані з таблиці актуальні на травень 2025 року.

Таблиця 2.1 – Порівняльна характеристика LLM

Назва моделі	Засновник моделі	Величина контекстного вікна (у токенах)	Індекс інтелектуальної продуктивності	Змішана ціна за мільйон токенів (у $)
Gemini 2.0 Flash-Lite	Google	1 мільйон	41	0,13
Gemini 2.0 Flash	Google	1 мільйон	48	0,17
GPT-4.1 nano	OpenAI	1 мільйон	41	0,17
GPT-4o mini	OpenAI	128 тисяч	36	0,26
Llama 4 Scout	Meta	10 мільйонів	43	0,27
Qwen3 4B	Alibaba	32 тисячі	35	0,19

Gemini є сімейством великих мовних моделей, розроблених компанією Google. Ці моделі створювалися з акцентом на високу продуктивність, ефективність та здатність працювати з різними типами даних, хоча для цілей роботи достатньо їх текстових можливостей [16].

Однією з найбільших переваг саме цієї групи моделей, що має значення для генерації тестів, є її велике контекстне вікно. Контекстне вікно визначає обсяг тексту (кількість токенів), який модель може одночасно обробляти та враховувати під час аналізу чи генерації. Велике контекстне вікно у цих моделях дозволяє подавати на вхід моделі значні фрагменти навчального матеріалу: цілі параграфи, розділи або навіть невеликі статті. Наявність великої кількості інформації гарантує різноманітність та вищу якість тестів [17].

Схематично, що зображено на рисунку 2.2, контекстне вікно представляє обмежений набір токенів, розташованих «ліворуч» і «праворуч» від поточного (цільового) токена у послідовності. Тому, модель одночасно тримає в пам’яті і обробляє лише ті частини тексту, що потрапляють у межі цього вікна.

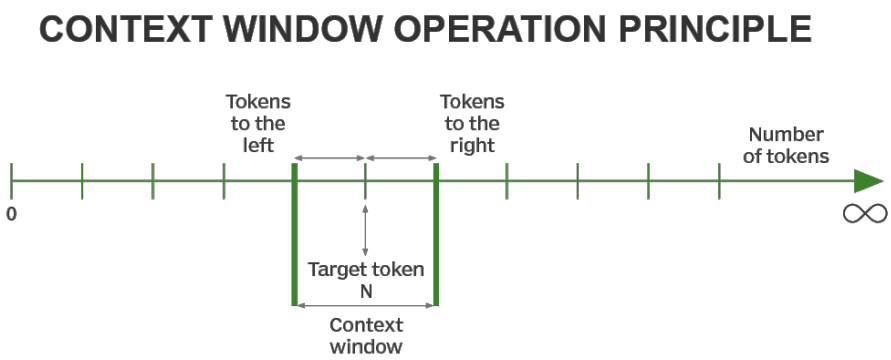

Рисунок 2.4 – Принцип дії контекстного вікна

До великого контекстного вікна додається технологічна сумісність Gemini з інфраструктурою Google. Модель представлена в середовищі Google Cloud Platform (GCP) і доступна через зручний API. Обрана платформа для вивантаження та представлення тестів (Google Forms) підв’язана до GCP, так само як і модель для генерації потрібних нам тестів, що створює нерозривний зв’язок між використанням цих технологій – це допоможе спростити процес розробки, розгортання на потенційного масштабування проєкту.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

Цей розділ присвячено детальному опису практичної реалізації розробленої системи автоматичного створення тестів. В основі системи лежить архітектура, що об’єднує декілька технологічних компонентів, кожен з яких виконує специфічну роль у загальному процесі генерації та формування тестів.

Головними складовими системи виступають наступних п’ять компонентів:

– авторизація в застосунку використовуючи можливості GCP, для можливості розміщення тестів на Google акаунті користувача;

– генерація питань базуючись на вхідному текстовому матеріалі за допомогою API великої мовної моделі Gemini;

– алгоритмічне створення й завантаження згенерованих тестів з використанням Google Forms API;

– збереження необхідної інформації до бази даних для можливості відновлення сесій і повторної генерації форм;

– зв’язок усіх компонентів системи за допомогою UI фреймворку Flet.

3.1 Налаштування авторизації в застосунку

Для забезпечення можливості програмного створення форми або тестів, система передбачає механізм авторизації через обліковий запис Google. Цей процес є необхідним, оскільки лише він може надати застосунку права проводити операції по створенню файлів та їх модифікації в середовищі Google Drive, де і зберігаються усі створені користувачем Google форми. Використання протоколу OAuth 2.0 для авторизації дозволяє користувачу свідомо надати дозволи застосунку через знайомий та захищений інтерфейс Google і гарантує що застосунок не отримує прямого

доступу до облікових даних користувача [18]. Принцип дії цього механізму можна побачити на рисунку 3.1.

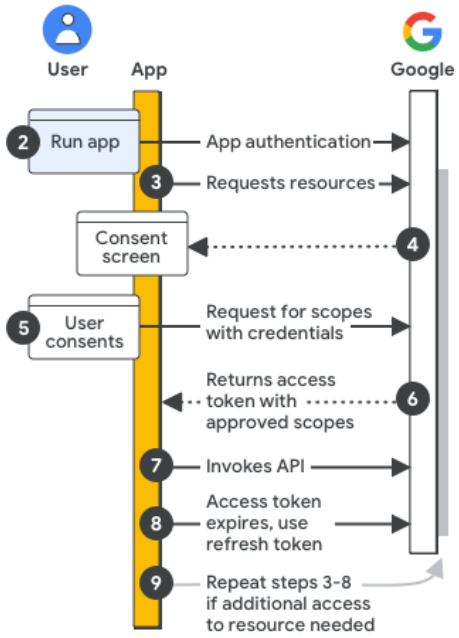

Рисунок 3.1 – Принцип роботи механізму авторизації OAuth 2.0

Для забезпечення безпеки користувача та застосунку, використовується принцип мінімально необхідних привілеїв (Principle of least privilege) [19]. Застосунок очікує лише конкретні дозволи (scopes) від користувача. До цих дозволів входить:

– дозвіл на створення нових файлів на Google Drive користувача;

– дозвіл на програмну модифікацію змісту Google Forms (для додавання питань, налаштування правильних відповідей та балів);

– дозвіл на відображення електронної пошти у візуальному інтерфейсі застосунку для спрощення процесу ідентифікації активного акаунту.

Важливо наголосити на тому, що взаємодія застосунку з будь-якими частинами Google простору (Google Drive, Google Forms) обмежується лише тими формами і тестами, які створюються безпосередньо самим застосунком під час активної сесії роботи користувача над генерацією

конкретного тесту. Після того, як тест було згенеровано, наповнено питаннями та збережено, він стає повноцінним документом, що знаходиться лише під контролем користувача. Застосунок не зберігає довготривалого доступу до вже створених форм і не взаємодіє з їх вмістом ніяким чином.

3.2 Взаємодія застосунку з Gemini API для генерації тестів

В основі генерації тестових питань головну роль відіграє використання можливостей великих мовних моделей Gemini, базуючись на вхідному тексті від користувача застосунком. Для того щоб чітко описати процес створення запиту, достатньо розглянути наступні етапи: підготовка та обробка вхідних текстових даних, формування запитів до API моделі з використанням розробленого промпту, а також обробку та валідацію отриманої відповіді від моделі.

3.2.1 Обробка вхідних текстових даних

Першим кроком у процесі генерації тестів виступає зчитування та форматування вхідного тексту, на базі якого буде генеруватись збірка тестів. Наразі, в системі реалізовані три найбільш популярні та поширені формати текстових файлів: звичайний текст (формат .txt), документів Microsoft Word (формат .docx) та файлів формату PDF (формат .pdf).

Для обробки звичайних текстових документів використовуються звичайні методи мови Python для зчитування інформації. Для документів формату Microsoft Word, було долучено окремий зовнішній модуль python-docx, який дозволяє аналізувати вміст текстовий вміст абзаців та таблиць документу.

Обробка формату PDF є складнішою в порівнянні з попередніми методами. Для неї також використовується зовнішня бібліотека PyPDF2, яка надає методи для вилучення тексту з кожної сторінки та об’єднання його в

загальний блок. Якість зчитаного тексту сильно залежить від структури та способу створення самого документу: якщо документ містить відскановані зображення без шару розпізнаного тексту або в ньому присутня специфічна для інших форматів верстка, то це може сильно вплинути на якість тексту який буде подаватись в модель.

Для того щоб запобігти потенційних проблем з якістю зчитаного тексту, в UI застосунку передбачена функція перегляду та редагування відповідного матеріалу. Алгоритм простий: користувач завантажує файл і на головному екрані з’являється кнопка для редагування, натискаючи на яку, створюється модальне вікно в якому буде представлений зчитаний застосунком текст. Якщо користувач помітив якісь аномалії або хоче дописати якусь частину інформації до документу – він може це зробити просто використовуючи функціонал застосунку.

3.2.2 Формування запиту до API та інженерія промптів

Після того, як вхідний текстовий файл було оброблено, наступним етапом є формування потрібного запиту до API моделі. Якість та чіткість згенерованих тестів напряму залежить від того, як сформовані інструкції до моделі. Процес написання відповідних інструкцій або промптів, має назву інженерія промптів (prompt engineering). Основою запиту до моделі є системний промпт (system prompt) або ж головна інструкція, яка задає задачу, роль та поведінку моделі на момент сесії.

Хоча не існує універсальних правил або обмежень на написання цих інструкцій, існують загальні практики або рекомендації, дотримання яких дозволить суттєво підвищити якість відповідей моделі, наприклад:

– визначення ролі для моделі, яке дозволяє моделі краще адаптувати стиль та зміст самої відповіді;

– послідовність, логічність та чіткість викладання інструкцій допомагає моделі фокусуватись на виконанні конкретної задачі, не відхиляючись від потрібного напрямку;

– формат відповіді моделі (якщо чітко вказати параметри, які ми очікуємо отримати на виході, наприклад, у JSON форматі, модель не буде витрачати свої ресурси та час на формування пов’язаних між собою словосполучень, що дозволить зменшити вихідну кількість токенів та мінімізувати витрати на валідацію інформації).

З усієї текстової інформації, яка була отримана на попередньому кроці, формується вставка в системний або юзер промпт (user prompt). По суті, юзер промпт – це прямий запит від користувача, саме те з чого системі потрібно згенерувати список питань.

У рамках цього проекту було прийнято рішення сфокусуватись на генерації трьох найбільш поширених форматів тестів: питання з єдиною правильною відповіддю, питання на хибність твердження та відкриті питання які передбачають коротку письмову відповідь від користувача. Вибір цієї трійки обумовлений декількома факторами. По-перше, ці типи питань є найбільш поширеними в освітній практиці та дозволяють комплексно оцінити різні аспекти засвоєння матеріалу – від знання фактів, до здатності формулювати власні думки. По-друге, сучасні мовні моделі демонструють доволі високу якість генерації відповідних типів за умови коректних складених інструкцій – якщо вказати джерело для генерації неправильних відповідей, у випадку з тестами де потрібно обирати правильну відповідь, то модель буде чітко слідкувати інструкціям та намагатись знаходити найближчі по темі і сенсу слова.

Тому, у вихідному системному промпті чітко вказані наступні інструкції: роль – як методист, який спеціалізується на створенні тестів; послідовна обробка вхідного тексту від користувача та інструкції того, який формат питань і їх кількість потрібно генерувати.

3.2.3 Обробка та валідація відповіді від моделі

Як тільки потрібний запит був сформований і застосунок отримав відповідь – згенеровані тести потребують подальшої обробки та валідації. Незважаючи на активний прогрес у розвитку LLM, їхні відповіді не завжди можуть бути структурованими або повністю відповідати очікуваному формату, навіть при детально прописаних інструкціях у промпті. Можуть виникати проблеми з кількістю згенерованих питань, структурою окремих завдань, типами даних у полях тощо. Ігнорування подібних ситуацій може призвести до помилок на наступних етапах роботи застосунку, наприклад, під час завантаження даних через Google Forms API.

Тому, в архітектурі системи передбачено етап валідації даних, отриманих від моделі, з використанням валідаційних моделей. Для цього в проекті застосовано бібліотеку Pydantic, яка дозволяє визначати чіткі схеми даних (data schemas) для очікуваної відповіді. Повний спектр можливостей валідаційної моделі представлений у вигляді схеми на рисунку 3.2 [20].

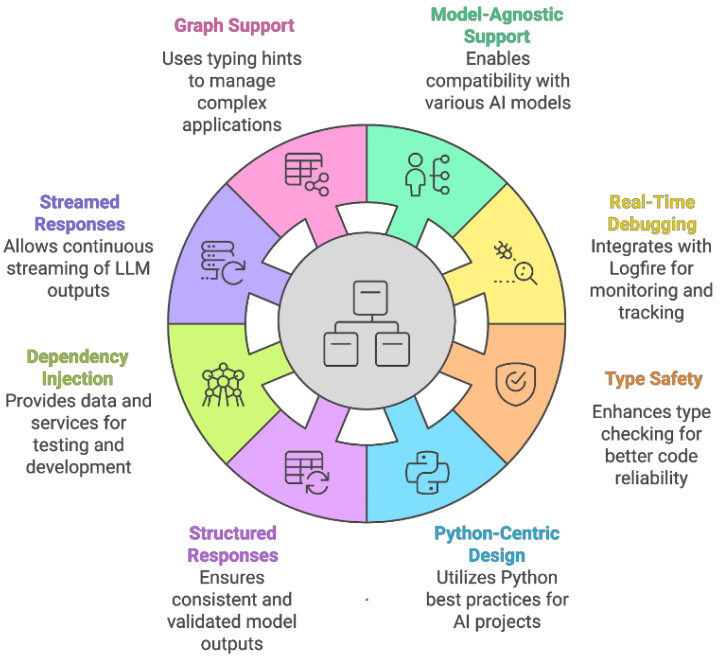

Рисунок 3.2 – Особливості бібліотеки Pydantic

Використання Pydantic-моделей дає змогу автоматично перевірити відповідність структури отриманих даних (наприклад, що кожне питання має текстове формулювання, визначений тип, список варіантів відповідей для тестів, правильну відповідь тощо) та типів даних у кожному полі. Якщо дані не відповідають заданій схемі: відсутнє обов'язкове поле, неправильний тип даних, невідповідна кількість варіантів відповідей – Pydantic відмічає всі помилки.

3.3 Використання Google Forms API для вивантаження тестів

Google Forms API є методом зв'язку згенерованого тесту в застосунку і форми, яка буде створена в хмарному середовищі. Це API має в собі потрібний базовий функціонал для створення та редагування форм, але й має обмеження на частину функціоналу, який представлений лише у вебверсії Google Forms.

3.3.1 Обробка та валідація відповіді від моделі

Першим кроком є програмне створення нової Google форми на акаунті користувача і налаштування її зв'язку через унікальний ідентифікатор. Після успішного створення, для активації функціоналу оцінювання, форму необхідно перевести у режим тесту (quiz). Це досягається шляхом надсилання запиту batchUpdate, який містить операцію оновлення налаштувань форми. Запит для ввімкнення режиму тесту представлений на лістингу 3.1.

Лістинг 3.1 – Структура запиту для оновлення налаштування форми

```
{
  "updateSettings": {
    "settings": {
      "quizSettings": {
```

Продовження лістингу 3.1

```
        "isQuiz": True
      },
    "updateMask": "quizSettings.isQuiz"
  }
}
```

У наведеному вище запиті, updateSettings вказує на намір змінити налаштування форми. Вкладений об'єкт settings містить поле quizSettings, де параметр isQuiz встановлюється в True, що й активує режим тесту. Поле updateMask гарантує, що буде оновлено лише цей конкретний параметр налаштувань, залишаючи інші загальні налаштування форми без змін.

3.3.2 Наповнення форми та налаштування оцінювання

Після створення та базової конфігурації форми відбувається її наповнення тестовими завданнями. На етапі генерації тестів застосовуються внутрішні маппери (mappers), які визначають необхідний тип питання для Google Forms на основі згенерованого контенту. Залежно від типу питання, формується відповідна JSON-структура, що передається у запиті batchUpdate до API для модифікації вже створеної форми:

– для питань, що передбачають вибір одного правильного варіанту з кількох або питань на хибність твердження використовується тип choiceQuestion;

– для питань, де відповіддю має бути текст, застосовується тип textQuestion.

Встановлення оцінки відбувається для кожного тестового завдання окремо: необхідно оновити відповідний елемент форми, передавши у тілі запиту об’єкт grading_body та вказавши маску оновлення. Кількість балів за правильну відповідь визначається параметром pointValue в об’єкті

grading_body. Запит для оновлення параметрів оцінювання наведений в лістингу 3.2.

Лістинг 3.2 – Структура запиту для налаштування оцінки тесту

```
{
  "updateItem": {
    "location": {
      "index": item_index
    },
    "item": {
      "questionItem": {
        "question": {
          "grading": grading_body
        }
      }
    },
    "updateMask": "questionItem.question.grading"
  }
}
```

Для текстових питань (textQuestion) у grading_body додатково формується поле generalFeedback – воно використовується для надання прикладу правильної, на думку моделі, відповіді, що допомагає зорієнтувати користувача. У доповненні, цей тип питань містять в собі механізм, що дозволяє остаточне оцінювання викладачем. Це пов’язано з тим, що точне співпадіння відповіді студента з еталонною текстовою відповіддю, особливо якщо вона складається з кількох слів або речень, є малоймовірним, тому автоматична оцінка може бути некоректною.

Додатково, в системі передбачено два прапорці для розширення функціональності:

– прапорець для автоматичного додавання на початку тесту питання типу «коротка текстова відповідь» для введення імені студента. Це поле

робиться обов’язковим для спрощення ідентифікації. JSON-структура для створення такого елемента наведена в лістингу 3.3;

– прапорець для встановлення обов’язковості виконання усіх тестів. Для цього, у випадку оновлення тестів, встановлюється параметр required як True.

Лістинг 3.3 – Структура запиту для налаштування оцінки тесту

```
{
  "updateItem": {
    "location": {
      "index": 0
    },
    "item": {
      "title": "Full Name",
      "questionItem": {
        "question": {
          "required": True,
          "textQuestion": {
            "paragraph": False
          }
        }
      }
    }
  }
}
```

3.3.3 Обмеження програмного функціоналу створення форм

На поточний момент Google Forms API не підтримує програмне створення та налаштування всіх типів питань та функцій, доступних у вебінтерфейсі Google Forms. Наприклад, складніші типи питань, такі як сітки з варіантами відповідей, шкали, або завантаження файлів, можуть мати обмежену або відсутню підтримку через API.

Проте, Google Forms API є інструментом, що розвивається. Очікується, що в майбутньому його функціонал буде розширено – це, у свою чергу, дозволить розширити можливості розробленої системи. Потенційні напрямки включають підтримку нових форматів тестів (наприклад, питання на відповідність, завантаження файлів), більш гнучкі та специфічні налаштування системи оцінювання (наприклад, часткове оцінювання, де результати тестів з відкритою відповіддю будуть перевірятись на відсоткове співпадіння з коректною відповіддю), а також розширені можливості для управління відповідями та аналітики безпосередньо через API.

3.4 Реалізація бази даних

Для забезпечення можливості відновлення попередніх сесій роботи та повторного використання згенерованих тестів, система передбачає механізм збереження необхідної інформації у локальній базі даних.

Для потреб застосунку було обрано SQLite3. Таке рішення обумовлене кількома факторами:

– SQLite3 є легкою, файловою СУБД, що не потребує окремих серверних процесів, що значно спрощує розгортання та використання застосунку;

– структура даних, що зберігається в базі, не передбачає складних взаємозв’язків чи будь-яких транзакційних операцій, для яких зазвичай використовуються більш потужні SQL-сервери (наприклад, PostgreSQL або MySQL).

Для потенційного перенесення застосунку у вебформат та розширення його функціоналу, наприклад, для реалізації семантичного пошуку по згенерованих питаннях чи матеріалах, альтернативним варіантом могли б слугувати векторні NoSQL бази даних. Вони оптимізовані для зберігання та пошуку векторних представлень даних, що є актуальним при роботі з великими мовними моделями та їхніми вихідними даними.

Вся інформація про сесії генерації тестів зберігається в єдиній таблиці з назвою sessions. Ця таблиця містить наступні поля:

– id – унікальний ідентифікатор сесії (первинний ключ);

– filename – назву вихідного файлу, на основі якого генерувався тест;

– content – повний зчитаний та оброблений текст з вихідного файлу;

– scq_count – числове значення кількості згенерованих питань з вибором однієї правильної відповіді;

– tf_count – числове значення кількості згенерованих питань на перевірку хибності твердження;

– open_count – числове значення кількості згенерованих питань з відкритою відповіддю;

– add_full_name – бінарний прапорець (0 або 1), що вказує, чи було додано до тесту поле для введення повного імені студента. Значення за замовчуванням – 1 (було додано);

– make_all_mandatory – бінарний прапорець (0 або 1), що визначає, чи були всі питання зроблені обов’язковими для відповіді. Значення за замовчуванням – 0 (не обов’язкові);

– language – текстове поле, що вказує мову генерації тесту. Підтримувані значення: «ENG» (англійська) та «UKR» (українська). Значення за замовчуванням – «ENG»;

– generated_json – текстове поле, що зберігає JSON-структуру, яка є повною репрезентацією згенерованого тесту. Це поле містить усі питання, варіанти відповідей, правильні відповіді та інші параметри, необхідні для відтворення тесту;

– model_name – текстове поле для збереження назви великої мовної моделі, що використовувалася для генерації питань;

– timestamp – поле типу дата-час, що автоматично фіксує час створення запису про сесію.

Ключовим полем для функціоналу відновлення сесії є generated_json. Воно зберігає деталізовану структуру тесту, згенеровану моделлю Gemini.

Наявність цієї інформації дозволяє системі відтворити сесію та створити новий екземпляр Google Форми з тими самими питаннями та налаштуваннями, без необхідності повторного звернення до великої мовної моделі. Це суттєво економить час та ресурси, особливо при роботі з об’ємними текстовими матеріалами. Створення нової форми при відновленні сесії обумовлено принципом роботи застосунку: після завершення генерації та збереження тесту, застосунок не зберігає жодного активного зв’язку з уже створеною Google формою. Такий підхід обрано задля підвищення безпеки даних користувача та уникнення потенційних помилок, пов’язаних із модифікацією вже існуючих файлів на Google Drive користувача без його явного контролю на момент модифікації.

Система надає користувачеві можливість керувати збереженими сесіями. Передбачено функціонал для видалення будь-якої обраної сесії або декількох сесій з бази даних, якщо збережена інформація більше не є актуальною для користувача.

3.5 Побудова графічного інтерфейсу

Однією з важливих частин розробленої системи є графічний інтерфейс користувача, який забезпечує взаємодію користувача з усіма функціональними можливостями застосунку. Головною задачею при розробці GUI було створення інтуїтивно зрозумілого, зручного та візуально привабливого застосунку. Такий інтерфейс повинен дозволяти виконувати усі передбачені операції: завантаження файлів, налаштування параметрів генерації тестів, відображення обробленого тексту, ініціювання процесу генерації, а також перегляд та управління збереженими сесіями роботи.

Для реалізації графічного інтерфейсу було обрано мову програмування Python. Таке рішення сприяло уніфікації стеку технологій, що використовуються в проекті, дозволило зменшити кількість зовнішніх залежностей та спростити загальний процес розробки і подальшої

підтримки системи. Використання єдиної мови для основної логіки застосунку та для GUI мінімізує складності, пов’язані з інтеграцією різних компонентів. Серед доступних Python-фреймворків для створення GUI було обрано Flet.

Flet є сучасним фреймворком, що дозволяє розробляти інтерактивні крос-платформні графічні інтерфейси, використовуючи Python, при цьому для рендерингу інтерфейсу залучаються можливості Flutter, що забезпечує високу продуктивність та сучасний вигляд елементів керування.

При виборі інструментарію для GUI розглядалися й інші поширені Python-фреймворки. Наприклад, Tkinter, як стандартна бібліотека Python, пропонує базовий функціонал і не потребує додаткових залежностей, проте часто вимагає значних зусиль для стилізації елементів задля досягнення сучасного вигляду (приклад інтерфейсу наведений на рисунку 3.3).

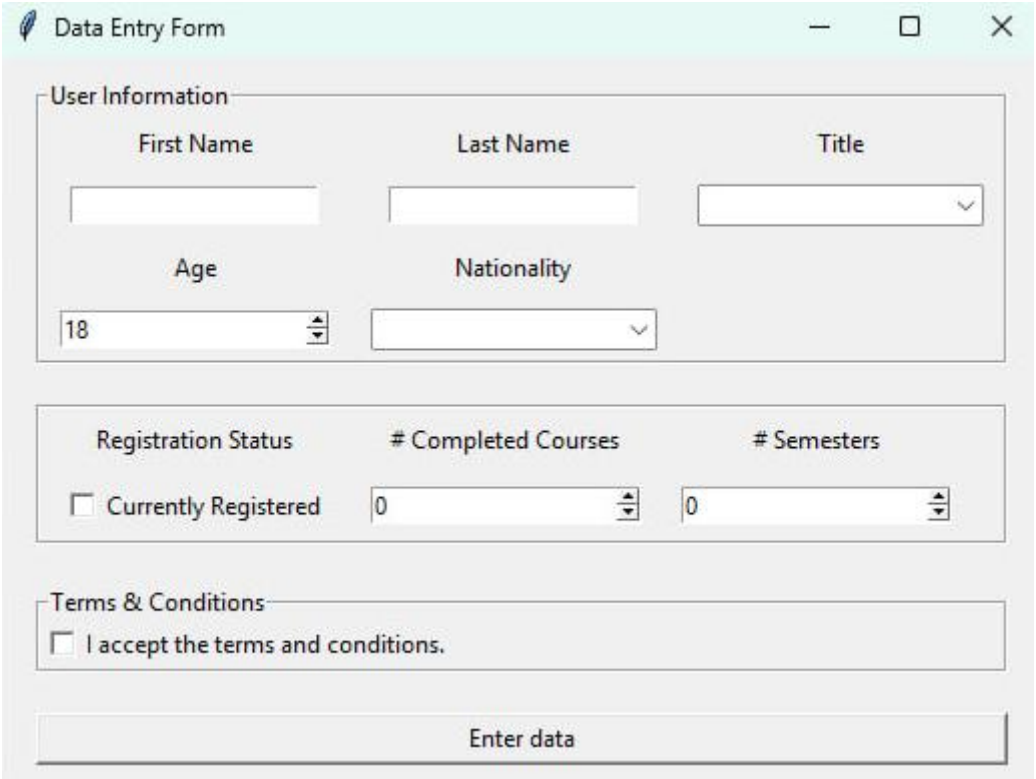

Рисунок 3.3 – Приклад інтерфейсу фреймворку Tkinter

Альтернативою є PyQt або PySide, Python-надбудови до потужної бібліотеки Qt, які надають широкий спектр віджетів для складних інтерфейсів, але можуть збільшувати розмір кінцевого застосунку та мати специфічний процес налаштування взаємодії між компонентами (приклад інтерфейсу наведений на рисунку 3.4).

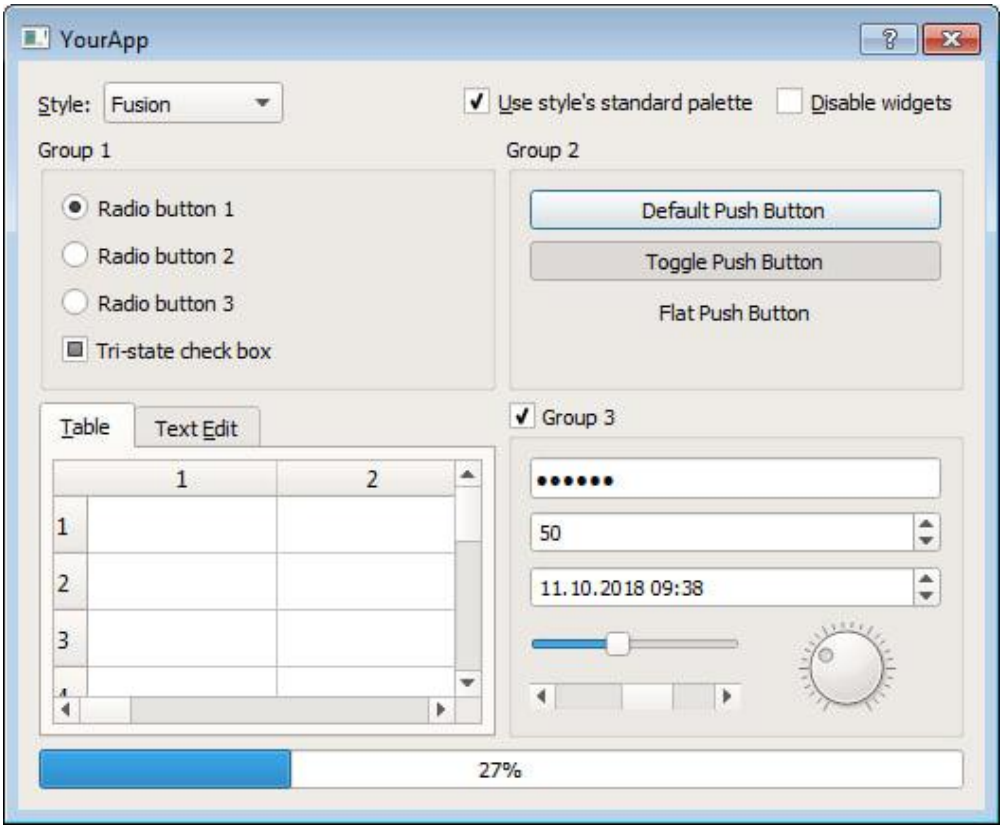

Рисунок 3.4 – Приклад інтерфейсу фреймворку PyQt

На їхньому фоні, Flet виявився оптимальним вибором для даного проекту. Обраний фреймворк надав декілька ключових переваг:

– по-перше, завдяки використанню Flutter, він дозволяє створювати інтерфейси з сучасним дизайном та компонентним підходом до їх побудови, що спрощує розробку та подальшу підтримку;

– по-друге, Flet пропонує відносно простий та декларативний API, що дозволяє розробнику зосередитись на логіці взаємодії елементів інтерфейсу,

а не на низькорівневих деталях їх відображення. Це, в свою чергу, прискорює процес прототипування та реалізації.

Хоча Flet є відносно новим інструментом, він активно розвивається та підтримується спільнотою.

Для реалізації функціоналу застосунку було використано стандартні компоненти Flet, такі як контейнери для структурування розмітки (Container, Column, Row), елементи для введення даних (TextField), кнопки (ElevatedButton, IconButton), елементи для відображення тексту (Text) та таблиць (DataTable, DataRow, DataCell). Такий набір дозволив реалізувати усі заплановані інтерактивні елементи та забезпечити належне відображення інформації.

Кінцевою метою розробки було створення єдиного виконуваного файлу (наприклад, .exe для Windows), який можна було б легко поширювати серед користувачів без необхідності встановлення Python чи будь-яких додаткових бібліотек. Flet у поєднанні з інструментом PyInstaller надає зручний та практичний механізм для досягнення цієї мети. Процес пакування дозволяє включити усі необхідні залежності, включаючи інтерпретатор Python та сторонні бібліотеки, в один файл. При цьому PyInstaller дозволяє ефективно керувати ресурсами застосунку: секретні файли можуть бути інтегровані та захищені всередині виконуваного файлу, а файли, з якими застосунку потрібно працювати під час виконання, наприклад, локальна база даних SQLite3, компактно розміщуються поруч з ним у виконуваній директорії.

4 РЕЗУЛЬТАТИ ТА ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ

У цьому розділі буде представлено результати роботи та послідовність дій при роботі з застосунком (user-flow) для майбутнього користувача.

4.1 Демонстрація роботи застосунку.

При вході в систему користувача буде зустрічати головний екран застосунку – вікно генерації (рисунок 4.1).

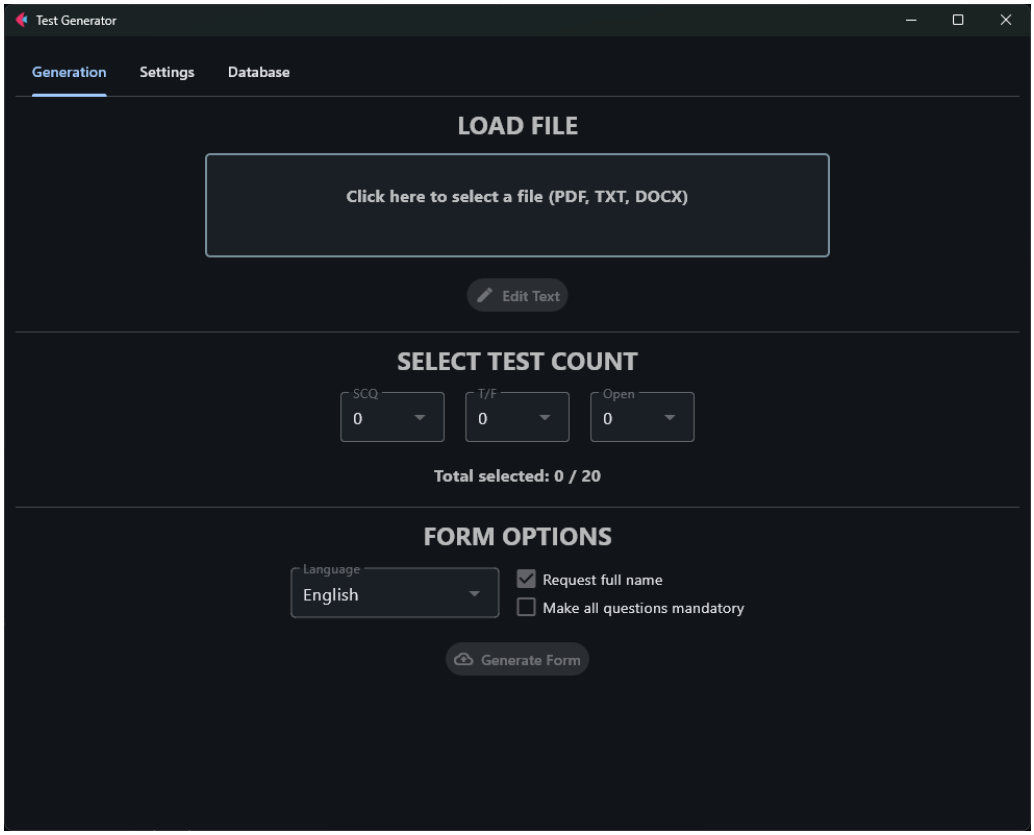

Рисунок 4.1 – Скріншот стартового вікна застосунку

На головному інтерфейсі розташовані три блоки:

– поле для завантаження текстового матеріалу;

– блок вибору формату та кількості потрібних питань;

– додаткові налаштування для форми.

Але, перед тим як починати процес генерації тестів, користувачу потрібно авторизуватись та налаштувати потрібні параметри для генерації.

Перейшовши у вікно налаштувань, користувач побачить екран з текстовими полями, що показано на рисунку 4.2.

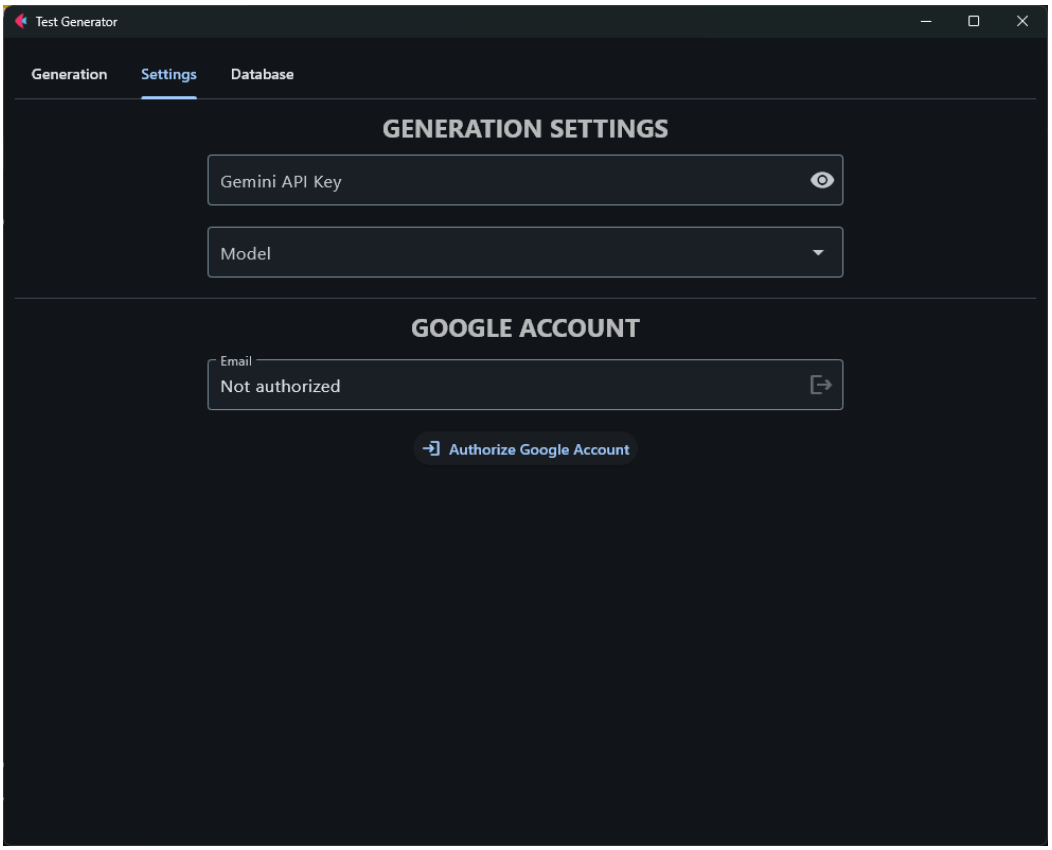

Рисунок 4.2 – Скріншот вікна налаштувань

У вікні налаштувань знаходяться три параметри:

– секретний API ключ для доступу до моделей Gemini;

– назва моделі, яка буде використовуватись для генерації;

– Google акаунт на хмарному сховищі якого буде зберігатись згенерована форма.

Секретний ключ для генерації можна отримати зареєструвавшись на офіційному сайті розробника моделі Gemini. Ввівши значення ключа в

застосунок, він захищається від можливості зовнішнього копіювання, що надає додаткові гарантії захисту при користуванні застосунком.

Для того, щоб вибрати модель для генерації, користувач обирає бажану модель з випадаючого списку. Наразі, доступні дві моделі: Gemini 2.0 Flash та Gemini 2.0 Flash-Lite.

Наступним кроком йде авторизація в застосунку. Натиснувши на кнопку для авторизації, користувач буде перенаправлений в браузер за замовчуванням, де його зустріне екран вибору Google-акаунту. Після підтвердження і надання потрібних прав застосунку, котрі користувач зобов’язаний прочитати перед користуванням застосунку, в полі з електронною поштою буде відображено авторизований акаунт.

Вигляд налаштувань після вибору усіх потрібних параметрів буде виглядати так, як зображено на рисунку 4.3.

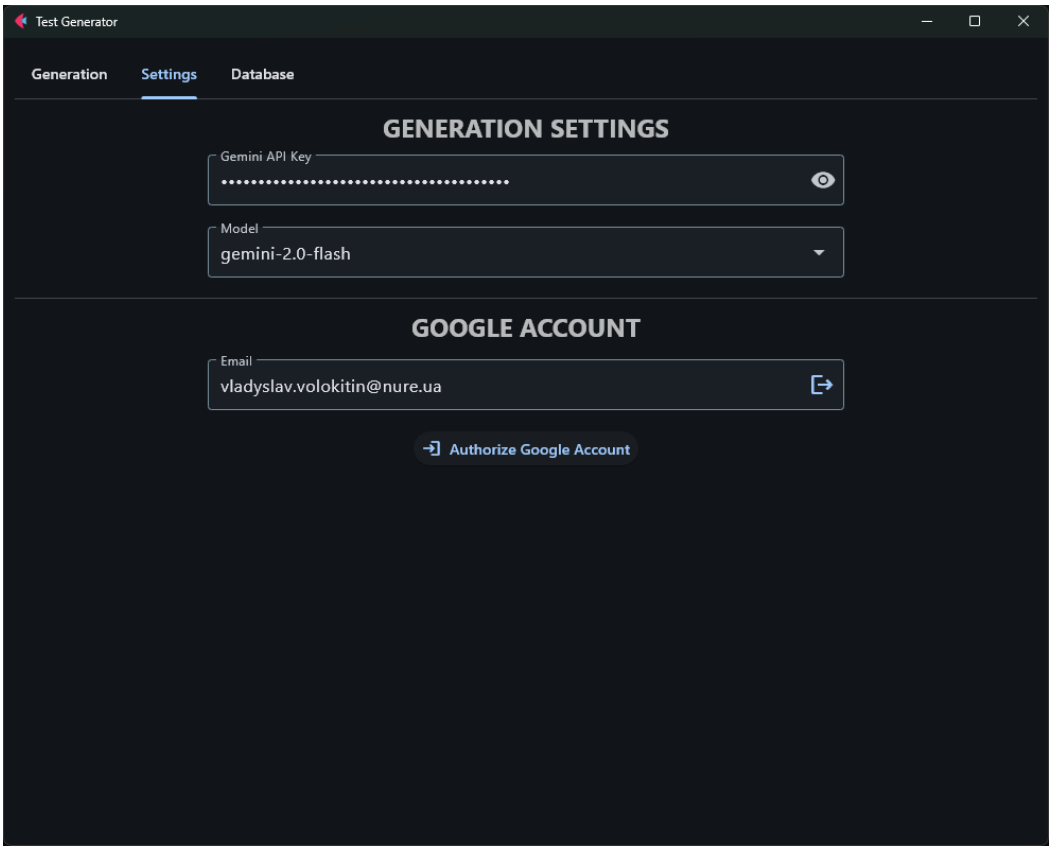

Рисунок 4.3 – Скріншот вікна налаштувань з заповненими даними

В будь-який момент, коли користувач захоче вийти з акаунту в застосунку, він може натиснути кнопку «Reset», яка виведе повідомлення з підтвердженням про вихід з акаунту.

Налаштувавши потрібні параметри, користувач може повернутись до головного екрану. Для того щоб розблокувати функціонал вибору потрібних параметрів для форми, потрібно завантажити текстовий файл, на базі якого буде генеруватись тест.

Поле для завантаження, наразі, підтримує лише три формати текстових файлів: звичайний текстовий формат, Microsoft Word документ та PDF. Після натискання на поле з завантаженням, користувачу відкриється файловий провідник, в якому він зможе обрати потрібний йому файл. Як тільки файл буде завантажений, система автоматично зчитає увесь текстовий матеріал та висвітить назву завантаженого файлу, як це зображено на рисунку 4.4.

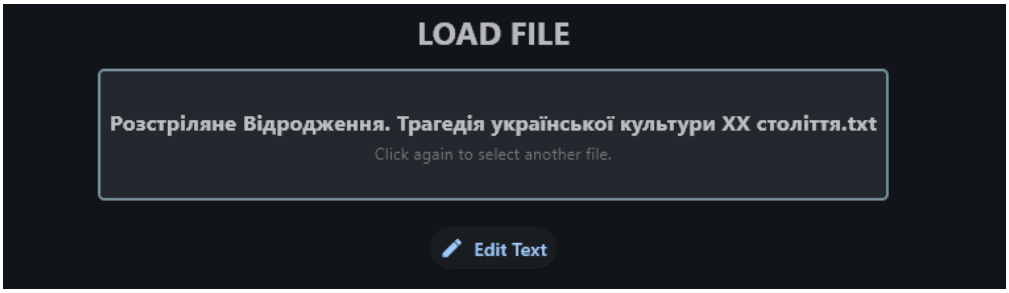

Рисунок 4.4 – Скріншот з прикладом завантаженого файлу

Після завантаження користувачу розблокований функціонал налаштування згенерованої форми. Перед тим, як генерувати форму, користувачу потрібно впевнитись, що файл який він надав системі, був зчитаний коректним чином: для цього достатньо натиснути кнопку для редагування тексту, яка стала активною після завантаження файлу. Результатом буде відображене вікно з текстом та можливістю редагувати контент, як продемонстровано на рисунку 4.5.

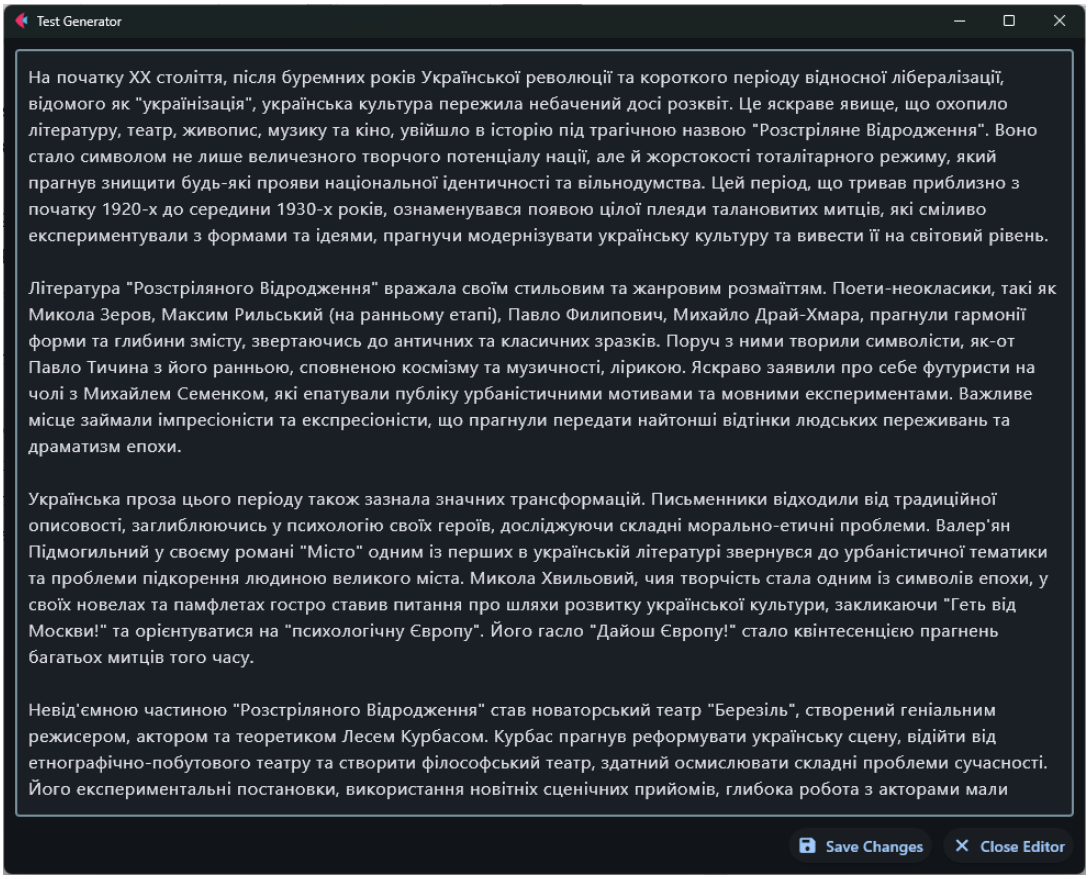

Рисунок 4.5 – Скріншот з прикладом зчитаного файлу

Якщо представлений текст відповідає вмісту текстового документу – користувач може продовжувати процес створення форми.

Перейшовши до налаштування самої форми, користувача зустрічає функціонал вибору типу та кількості питань. Наразі, у застосунку присутні три типи питань: вибір однієї правильної відповіді, питання на хибність твердження та питання з розгорнутою відповіддю. Для того щоб розпочати генерацію, користувачу потрібно обрати хоча б одне питання з двадцяти. Обмеження в двадцять питань було виставлене в результаті експериментальних досліджень, про що буде згадано більш детально у пункті 4.3.

Тож, обравши бажану кількість питань відповідних типів з випадаючого меню, користувач може бачити відповідні зміни в інтерфейсі блоку, що продемонстровано на рисунку 4.6.

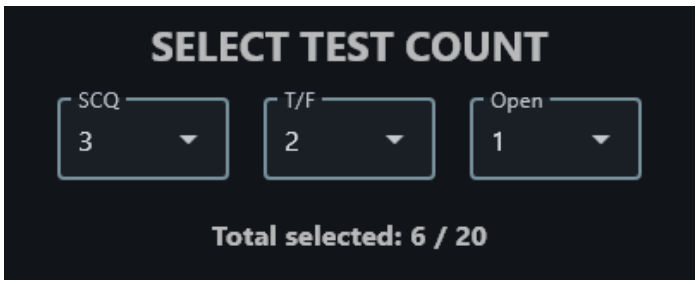

Рисунок 4.6 – Скріншот заповненого блоку з питаннями

Після вибору потрібної кількості питань, користувач може запустити процес генерації тесту. Після натискання відповідної кнопки, протягом 15 секунд (в залежності від кількості питань та розміру текстового матеріалу, час генерації може збільшитись або зменшитись) користувач побачить повідомлення та новостворені елементи інтерфейсу, як зображено на рисунку 4.7.

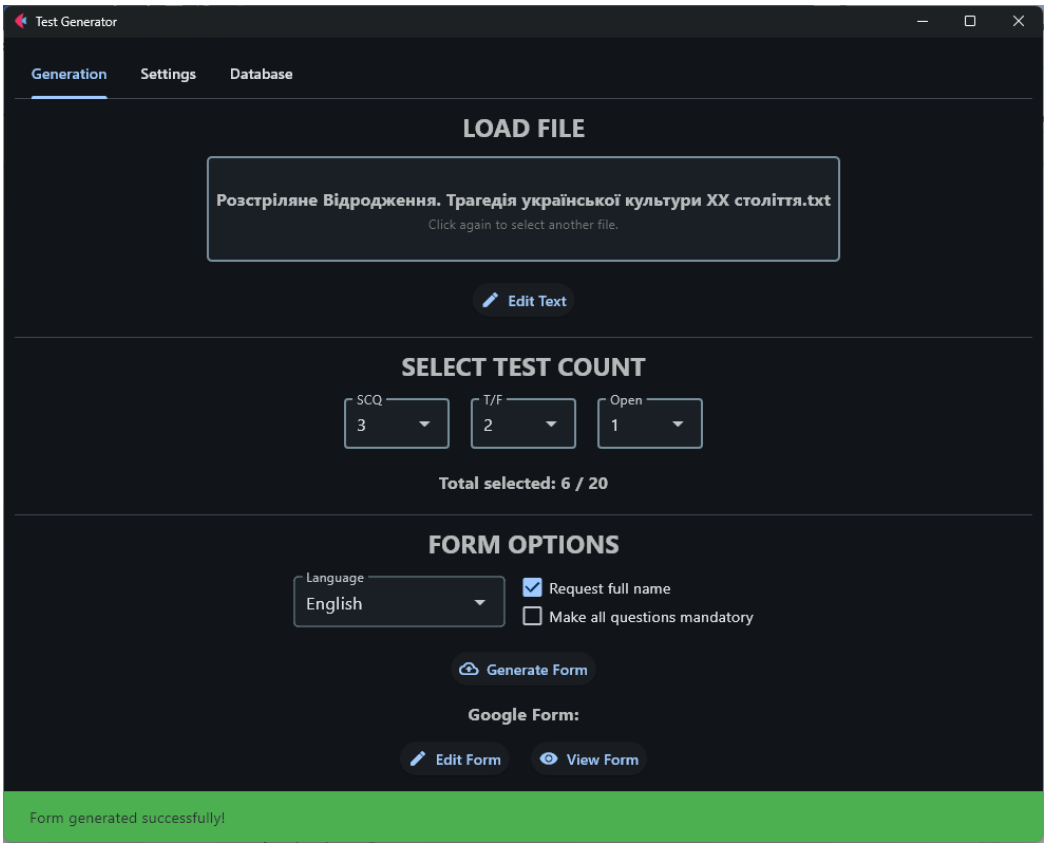

Рисунок 4.7 – Скріншот вікна після успішної генерації тесту

У новостворених елементах інтерфейсу присутні дві кнопки. Одна з яких веде на сторінку зі створеною формою у форматі редагування, а друга – у форматі перегляду.

Натискаючи на кнопку з переглядом, застосунок автоматично перенаправляє користувача на форму у браузері за замовчуванням. Приклад створеної форми зображений на рисунку 4.8.

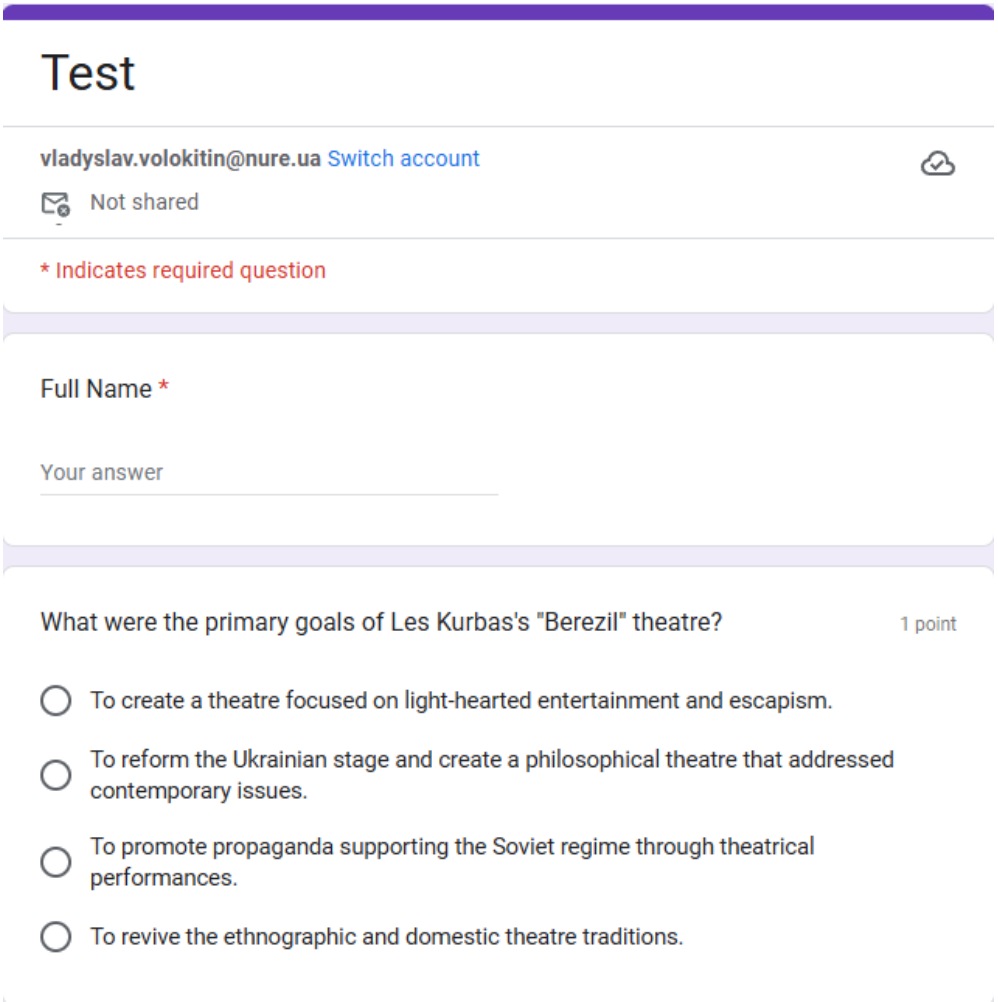

Рисунок 4.8 – Скріншот з прикладом створеної форми у форматі перегляду

В застосунку присутні додаткові параметри для налаштування форм: мова, якою буде згенерований тест, прапорець для створення окремого питання з ім’ям для спрощення ідентифікації студента та прапорець для

встановлення всіх питань у тесті обов’язковими до виконання. Ці додаткові налаштування можуть допомогти різним категоріям користувачів.

Якщо користувачем є викладач і йому, наприклад, потрібно провести швидке оцінювання студентів, він може створити форму вказавши обидва прапорці, як показано на рисунку 4.9.

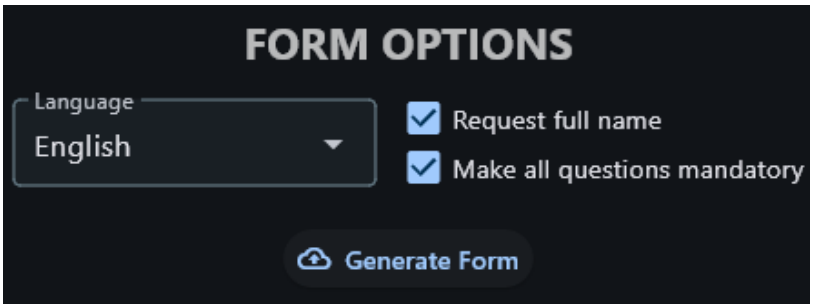

Рисунок 4.9 – Скріншот додаткових налаштувань з обраними прапорцями

Таким чином, згенерований тест буде мати окреме питання з ім’ям і тест не можна буде завершити до тих пір, поки студент не відповість на усі потрібні питання.

Але, тут є певне обмеження зі сторони програмного функціоналу. Нажаль, в API Google Forms не передбачені деякі функціональні можливості, які доступні у вебверсії Google Forms: вказавши прапорець для створення ім’я, застосунок не гарантує, що хтось зі студентів не порушить принцип академічної доброчесності. Наприклад, для налаштування збору електронних адрес студентів та обмеження однієї спроби на відповідь, що є обов’язковими для контролю за процесом тестування, викладачу слід скористатися вбудованими опціями самої Google форми.

Тому, для уникнення таких ситуацій, у функціоналі передбачена кнопка для перегляду форми у вигляді редагування. При натисканні на відповідну кнопку, застосунок перенаправляє користувача на окремий URL, доступ до якої може мати лише сам користувач. Приклад екрану редагування форми можна побачити на рисунку 4.10.

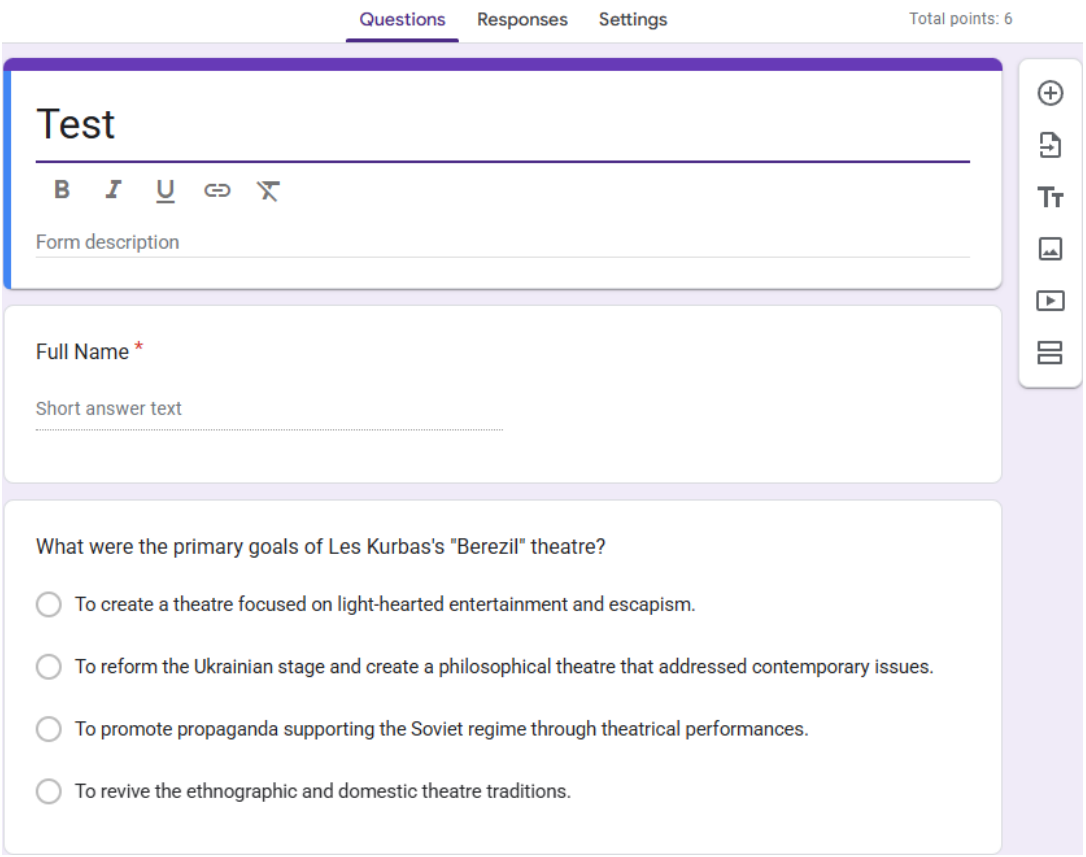

Рисунок 4.10 – Скріншот з прикладом створеної форми у форматі редагування

Перейшовши на вікно «Settings» у формі, користувач має змогу ознайомитись з додатковими параметрами, які не доступні в реалізації застосунку.

Ці налаштування дають змогу значно розширити контроль над тестуванням: обмежити кількість спроб, налаштувати перемішування питань чи порядок варіантів відповідей, а також встановити специфічні правила збору інформації про респондентів. Головним з цих параметрів для викладачів є виставлення результатів та збір електронних адрес. Коли студент виконує завдання, він повинен виконати його лише один раз без змоги поділитися або скопіювати правильні відповіді. Тому, викладачу потрібно змінити один з параметрів так, як це буде показано на рисунку 4.11.

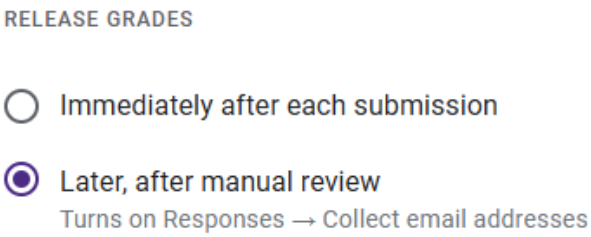

Рисунок 4.11 – Скріншот параметру для виставлення оцінок

У результаті, в режимі перегляду з’явиться додаткове поле для підтвердження використання електронної пошти під час відправки відповіді на тест, як показано на рисунку 4.12.

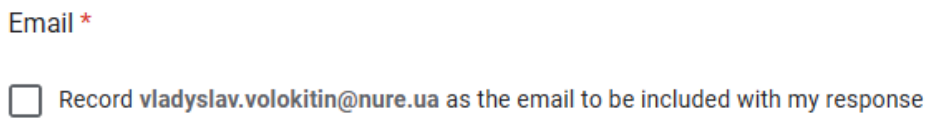

Рисунок 4.12 – Скріншот прапорця у формі після коригування параметрів виставлення оцінки

Якщо користувачем є студент, то він, наприклад, може використовувати застосунок у цілях покращення якості засвоєння матеріалу. Тоді, для зручності, користувач може прибрати додаткові прапорці з налаштувань та розпочати генерацію тестів для самоперевірки.

Подібний метод надає можливість не тільки оцінити власне розуміння теми, але й отримати орієнтир для корекції знань та кращого формулювання думок. Самостійна робота у такому вигляді сприяє активному навчанню та допомагає виявити прогалини, на які варто звернути додаткову увагу при підготовці. Більше того, систематичне тренування за допомогою подібних інструментів дозволяє адаптуватися до формату тестових завдань та зменшити стрес під час реальних контрольних заходів.

В умовах питань з розгорнутою відповіддю, після проходження тесту будуть наведені приклади правильних відповідей, які були згенеровані моделлю (рисунок 4.13).

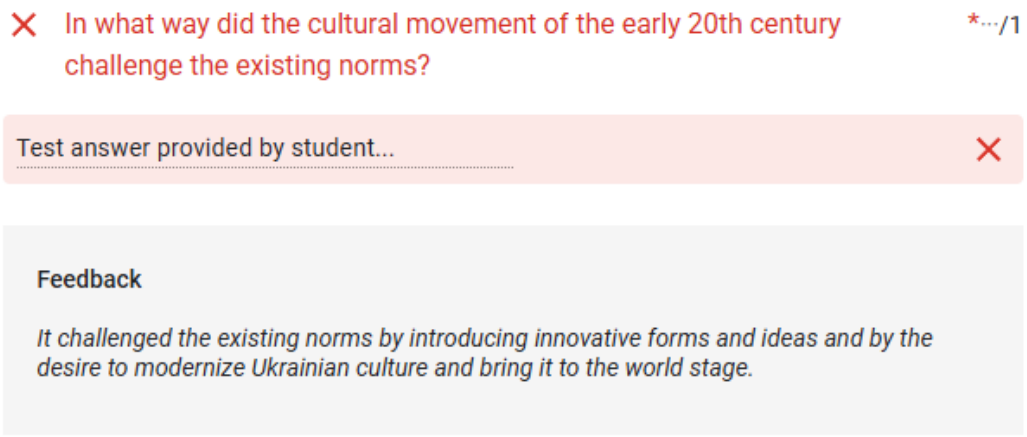

Рисунок 4.13 – Скріншот оцінки питання з текстовою відповіддю

Всі інші типи питань перевіряються автоматично.

Процес самооцінки для студента може бути реалізований по багатьом стратегіях, але найбільш корисними вважаються ті, які дозволяють запам’ятати ключові точки в темі (наприклад, сформувати тест лише з питань з розгорнутою відповіддю, для того щоб ретельно перевірити знання теми без додаткових підказок).

4.2 Взаємодія користувача з базою даних

В системі застосунку передбачена взаємодія користувача з базою даних. При переході на вікно з сесіями, перед користувачем з’явиться таблиця, як показано на рисунку 4.14. У цій таблиці можна спостерігати усі змістовні параметри які були використані під час попередніх сесій генерації тестів, окрім бінарних прапорців, котрі відповідають за додавання поля для імені або встановлення всіх питань обов’язковими.

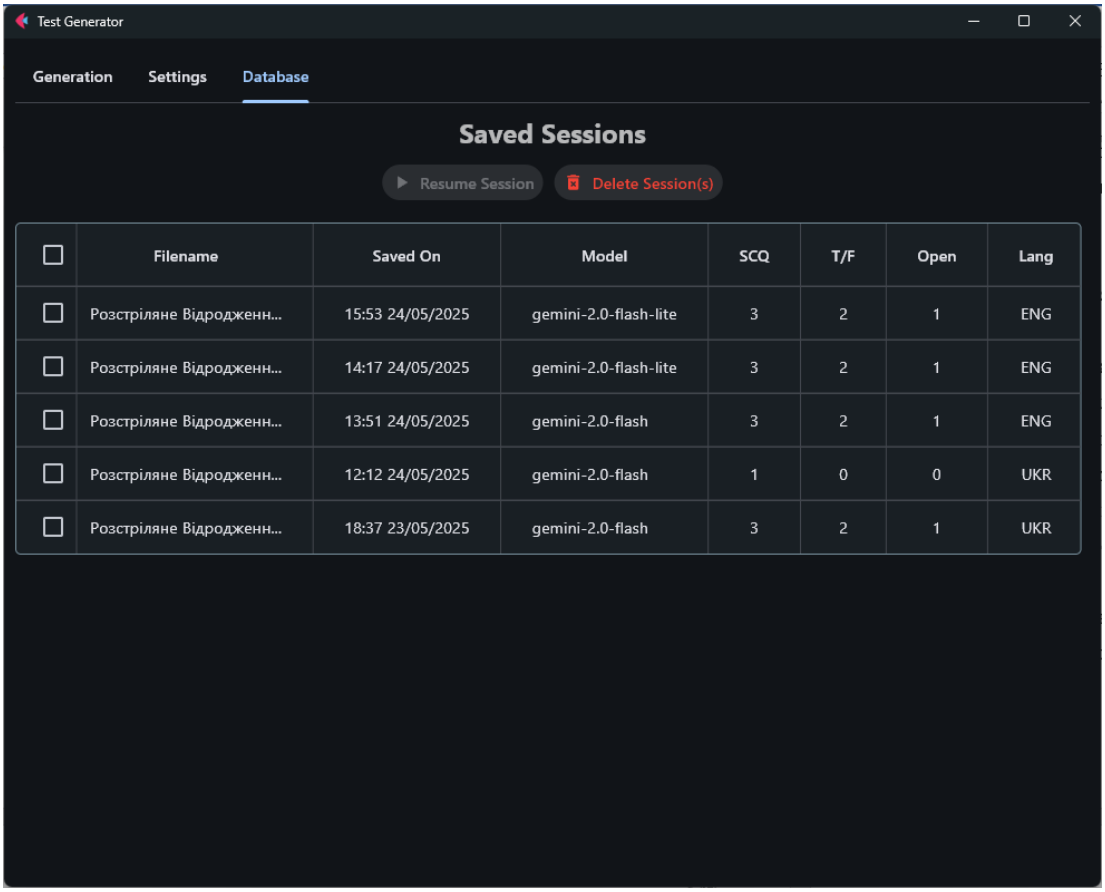

Рисунок 4.14 – Скріншот екрану зі збереженими сесіями

Якщо користувач має намір відновити сесію з історії, йому потрібно виділити прапорець напроти бажаного рядку і натиснути на кнопку відновлення сесії (вона стане активною після натискання на відповідний прапорець).

Після натискання, користувач буде перенаправлений на головний екран, де усі елементи інтерфейсу відновлять свій стан до того, який був збережений під час генерації питань. Після невеликої затримки, буде створена нова форма, але з такими ж питаннями, як і в збереженій сесії.

Відновлення сесії не створює окремий запис в базі даних, але якщо користувач натисне на генерацію нового тесту – це вже буде вважатись як новий екземпляр, оскільки модель згенерує повністю або частково нову відповідь (в залежності від того, чи був змінений текстовий контент).

Окрім відновлення сесій, у користувача є можливість видаляти сесії з бази даних. Для цього достатньо вибрати один, декілька або усі прапорці одразу і натиснути на кнопку видалення, яка розблокується після вибору конкретної кількості прапорців.

Після натискання на кнопку, застосунок відкриє модальне вікно для підтвердження видалення, як це показано на рисунку 4.15.

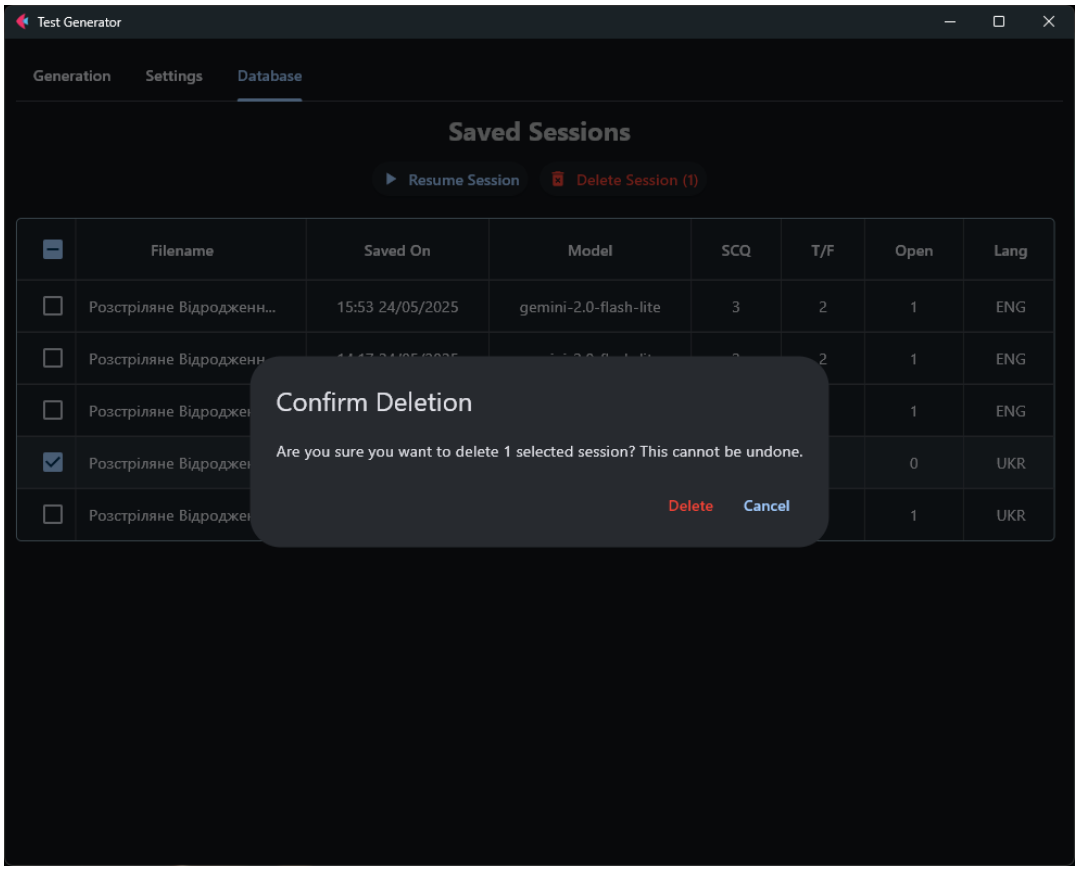

Рисунок 4.15 – Скріншот модального вікна для підтвердження видалення сесії

Якщо користувач погоджується з видаленням обраних сесій – система видаляє потрібну кількість екземплярів за індексом та миттєво оновлює інтерфейс, демонструючи таблицю та повідомлення про успішне видалення (рисунок 4.16).

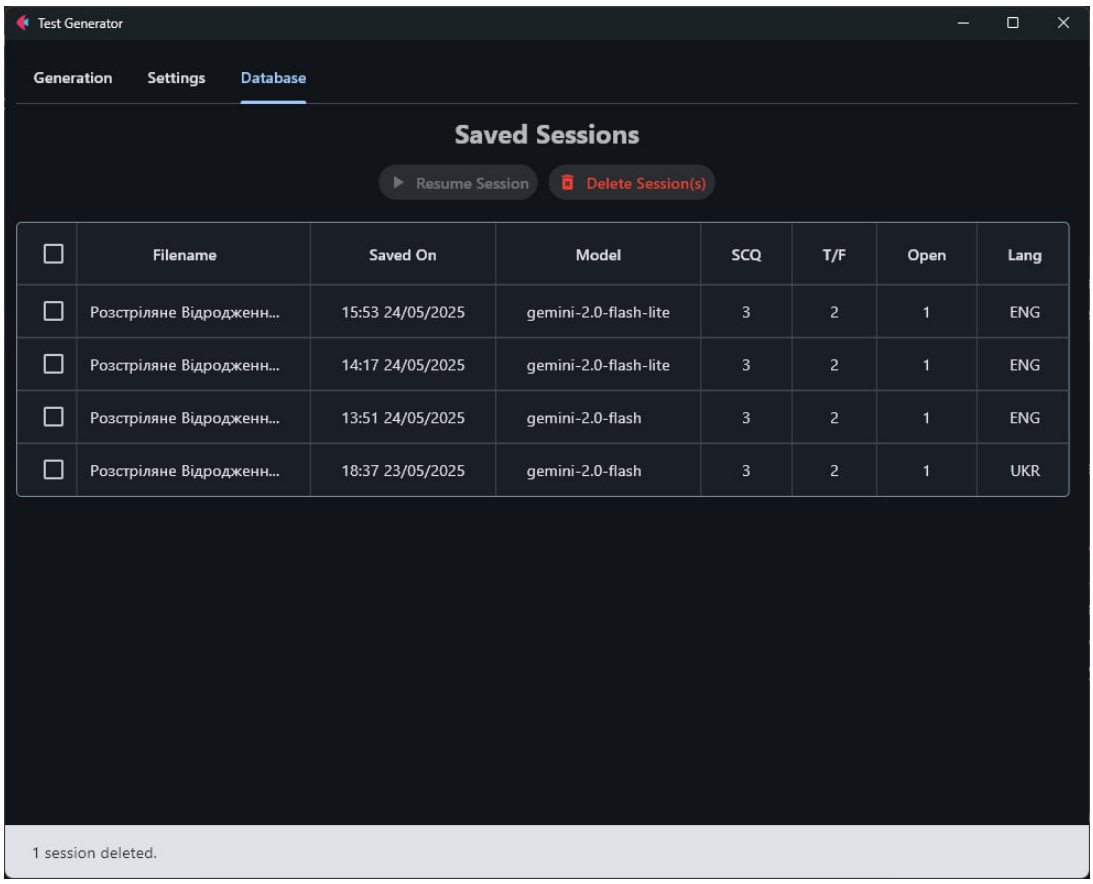

Рисунок 4.16 – Скріншот оновленого вікна після видалення екземпляру

Функціонал взаємодії з базою даних потенційно можна розширити, додавши короткий перегляд питань в самому інтерфейсі, без потреби створювати нову форму. Це буде зручно для тих ситуацій, коли кількість згенерованих людиною тестів буде занадто великою, щоб легко орієнтуватись по історії.

4.3 Рекомендації по використанню застосунку на базі експериментальних досліджень

Ефективність функціонування застосунку для генерації питань залежить від низки факторів, основними з яких є характеристики вхідного текстового матеріалу.

Для досягнення оптимальних результатів та забезпечення високої якості згенерованих питань, рекомендується враховувати декілька аспектів, про які буде наведено інформацію нижче.

4.3.1 Вплив обсягу контексту на якість генерації

Якість згенерованих питань напряму корелює з обсягом та релевантністю наданого текстового матеріалу. Незважаючи на постійне вдосконалення моделей штучного інтелекту, їхня здатність генерувати якісний контент значною мірою залежить від достатньої кількості вхідної інформації.

У випадку запиту на велику кількість питань (наприклад, 20 питань) при обмеженому обсязі вхідного тексту (умовно, два абзаци), модель стикається з дефіцитом інформації для формування різноманітних та змістовних питань. Це може призвести до феномену «галюцинацій», коли модель генерує нерелевантні, повторювані або фактично некоректні питання, виходячи за межі наданого контексту.

Запобігання галюцинаціям є складним завданням, яке часто вимагає впровадження додаткових модулів валідації. Ці модулі, хоча й здатні покращити достовірність згенерованих даних, можуть збільшити обчислювальні витрати та час генерації.

Для забезпечення максимальної якості згенерованих питань, рекомендується дотримуватися принципу адекватного співвідношення обсягу тексту та кількості запитуваних питань.

Оптимальним підходом є генерація невеликої кількості різнотипних питань, що дозволяє моделі зосередитися на головній темі матеріалу. Як загальне правило, слід керуватися принципом: на кожне питання бажано мати принаймні один невеликий абзац релевантного тексту.

4.3.2 Опрацювання великих обсягів текстових матеріалів

Застосунок розроблений з урахуванням можливості обробки значних обсягів текстової інформації. Проте, при роботі з надмірно великими документами (наприклад, цілою книгою або об’ємною статтею), існує ризик, що модель може зосередитися лише на певних фрагментах тексту, ігноруючи загальний контекст або інші важливі розділи. Це може призвести до генерації питань, які охоплюють лише обмежену частину матеріалу, а не увесь його зміст.

Для ефективної роботи з великими обсягами даних та для спрямованої перевірки конкретних тем, рекомендується використовувати вбудовані функції редагування тексту. Цей функціонал дозволяє користувачеві вибірково обрізати або виділяти певні частини тексту, тим самим направляючи модель на необхідні частини інформації.

Такий підхід є особливо корисним, коли метою є перевірка знань з конкретного розділу або теми, а не з усього курсу чи матеріалу. Виокремлення релевантних фрагментів тексту дозволяє підвищити точність та цілеспрямованість згенерованих питань.

ВИСНОВКИ

У ході написання кваліфікаційної роботи було успішно досліджено актуальну проблему значних часових витрат викладачів на створення тестових завдань та розроблено програмне рішення для її автоматизації. Незважаючи на функціональність модулів тестування в існуючих системах управління навчанням, процес підготовки якісного оцінювального контенту залишався доволі складним. Стрімкий розвиток великих мовних моделей надав перспективні інструменти для вирішення цієї задачі, що визначило мету роботи – дослідження та розробку програмної системи для автоматичної генерації тестів на основі текстових матеріалів з використанням LLM Gemini та Google Forms API.

Проведений аналіз предметної області та існуючих рішень підтвердив, що хоча традиційні LMS пропонують розширені інструменти тестування, їхні API часто мають обмеження щодо повної програмної автоматизації створення тестів. Натомість система Google Forms, завдяки своєму гнучкому та добре документованому API, виявилася оптимальною платформою для реалізації поставленої мети, забезпечуючи програмне створення та наповнення форм згенерованим контентом.

На основі теоретичних досліджень архітектури та принципів функціонування LLM були сформульовані ключові вимоги до моделі: велике контекстне вікно для обробки об’ємних текстів, доступність через API для інтеграції та здатність генерувати структуровані відповіді. Цим критеріям найкраще відповідала модель Gemini від Google, чиї характеристики у поєднанні з технологічною сумісністю з Google Forms, забезпечили ефективну взаємодію в рамках єдиної системи Google.

Було спроектовано та реалізовано інструмент автоматизованої генерації тестів, компоненти якого включають в себе модулі для авторизації користувача, обробки вхідних текстових файлів, взаємодії з LLM Gemini для генерації тестових завдань на основі розроблених промптів, валідації

отриманих даних, а також програмного створення та наповнення тестів у Google Forms. Реалізовано збереження даних у локальній базі даних для відновлення сесій та розроблено графічний інтерфейс для зручності користувача.

Практична реалізація та експериментальне тестування розробленого застосунку довели його функціональність та практичну цінність для освітнього процесу. Інструмент продемонстрував здатність суттєво спростити та прискорити підготовку оцінювальних матеріалів для викладачів, хоча як варіант, він може слугувати ефективним інструментом для самоперевірки студентів. В ході експериментальних досліджень були сформульовані рекомендації щодо оптимального використання системи: щодо співвідношення обсягу вхідного тексту та кількості генерованих питань для уникнення «галюцинацій» моделі та забезпечення якості тестів.

Але, існують певні обмеження, зумовлені природою використаних технологій: потенційна неточність або нерелевантність згенерованих LLM питань, де існує залежність не лише від стабільності роботи моделей, а й від якості поданого матеріалу людиною. Напрямками подальшого розвитку проекту можуть стати розширення підтримки інших типів тестових завдань, вдосконалення механізмів валідації згенерованого контенту, дослідження можливостей для більш глибокої семантичної обробки тексту та потенційна інтеграція з іншими освітніми платформами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ferdianto F., Dwiniasih. Learning Management System (LMS) schoology: why it's important and what it looks like. *Journal of physics: conference series*. 2019. Vol. 1360. URL: https://doi.org/10.1088/1742-6596/1360/1/012034 (date of access: 20.05.2025).

2. Abazi-Bexheti L., Jajaga E., Aposotolova M. Online testing module in LMS. *International conference on information technology interfaces ITI2013*, Cavtat / Dubrovnik, 24 June 2013. Tetovo, Macedonia, 2013. URL: https://www.researchgate.net/publication/255719218_Online_Testing_Module_in_LMS (date of access: 23.05.2025).

3. Rani T. J., Beutlin M. R. Effectiveness of Google Classroom as a tool for teaching and learning. *International journal of evidence based nursing*. 2020. URL: https://doi.org/10.37628/ijebn.v3i2.1606 (date of access: 20.05.2025).

4. Chekhratova O. Assessing students' academic achievements using Google Forms. *Humanities science current issues*. 2023. Vol. 2, no. 62. P. 318–325. URL: https://doi.org/10.24919/2308-4863/62-2-52 (date of access: 22.05.2025).

5. Overview | Google Forms. *Google for Developers*. URL: https://developers.google.com/workspace/forms/api/guides (date of access: 22.05.2025).

6. Attention is all you need / A. Vaswani et al. *Advances in neural information processing systems 30 (NIPS 2017)*, 12 June 2017. URL: https://arxiv.org/abs/1706.03762 (date of access: 21.05.2025).

7. Ardi M. Meet GPT, the decoder-only transformer | towards data science. *Towards Data Science*. URL: https://towardsdatascience.com/meet-gpt-the-decoder-only-transformer-12f4a7918b36/ (date of access: 22.05.2025).

8. Scaffold-BPE: enhancing byte pair encoding for large language models with simple and effective scaffold token removal / H. Lian et al. *Proceedings of*

the AAAI conference on artificial intelligence. 2025. Vol. 39, no. 23. URL: https://doi.org/10.1609/aaai.v39i23.34633 (date of access: 24.05.2025).

9. Kudo T., Richardson J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*, Brussels, Belgium. Stroudsburg, PA, USA, 2018. URL: https://doi.org/10.18653/v1/d18-2012 (date of access: 26.05.2025).

10. CausaLM: causal model explanation through counterfactual language models / A. Feder et al. *Computational linguistics*. 2021. URL: https://doi.org/10.1162/coli_a_00404 (date of access: 23.05.2025).

11. Masked language modeling and the distributional hypothesis: order word matters pre-training for little / K. Sinha et al. *Proceedings of the 2021 conference on empirical methods in natural language processing*, Online and Punta Cana, Dominican Republic. Stroudsburg, PA, USA, 2021. URL: https://doi.org/10.18653/v1/2021.emnlp-main.230 (date of access: 21.05.2025).

12. Interactive reinforcement learning from demonstration and human evaluative feedback / G. Li et al. *2018 27th IEEE international symposium on robot and human interactive communication (RO-MAN)*, Nanjing, 27–31 August 2018. 2018. URL: https://doi.org/10.1109/roman.2018.8525837 (date of access: 27.05.2025).

13. Amini A., Vieira T., Cotterell R. Direct preference optimization with an offset. *Findings of the association for computational linguistics ACL 2024*, Bangkok, Thailand and virtual meeting. Stroudsburg, PA, USA, 2024. P. 9954–9972. URL: https://doi.org/10.18653/v1/2024.findings-acl.592 (date of access: 26.05.2025).

14. Gomede E. The art of prediction: how LLMs master next-token generation. *Medium*. URL: https://python.plainenglish.io/the-art-of-prediction-

how-llms-master-next-token-generation-b8f81dc16de2 (date of access: 24.05.2025).

15. LLM leaderboard - compare GPT-4o, Llama 3, Mistral, Gemini & other models. *AI Model & API Providers Analysis | Artificial Analysis*. URL: https://artificialanalysis.ai/leaderboards/models (date of access: 22.05.2025).

16. Islam R., Ahmed I. Gemini-the most powerful LLM: Myth or Truth. *2024 5th information communication technologies conference (ICTC)*, Nanjing, China, 10–12 May 2024. 2024. URL: https://doi.org/10.1109/ictc61510.2024.10602253 (date of access: 20.05.2025).

17. Martineau K. Why larger LLM context windows are all the rage. *IBM Research*. URL: https://research.ibm.com/blog/larger-context-window (date of access: 21.05.2025).

18. Learn about authentication and authorization. *Google for Developers*. URL: https://developers.google.com/workspace/guides/auth-overview (date of access: 22.05.2025).

19. Schneider F. B. Least privilege and more. *IEEE security & privacy*. 2003. Vol. 1, no. 5. URL: https://doi.org/10.1109/msecp.2003.1236236 (date of access: 24.05.2025).

20. Tripathi S. A practioner's guide to pydanticai agents. *ADaSci*. URL: https://adasci.org/a-practioners-guide-to-pydanticai-agents/ (date of access: 25.05.2025).