

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)

_____ Інтелектуальна QA-система стійкої підтримки освітнього процесу в
_____ умовах критичної нестабільної інфраструктури
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-4

_____ Поліна ЩЕРБИНА
(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. Олена ГРИНЬОВА
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

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

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Щербині Поліні Олександрівні _____
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальна QA-система стійкої підтримки освітнього процесу в умовах критичної нестабільної інфраструктури

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

2. Термін подання здобувачем роботи до екзаменаційної комісії 8 липня 2026 р.

3. Вихідні дані до роботи Навчальні матеріали у форматах PDF, DOCX, TXT, база організаційних питань, дані про користувачів системи, сучасні бібліотеки для обробки природної мови, моделі ШІ

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі _____

2) Постановка задачі _____

3) Методи та підходи до розробки інтелектуальних систем _____

4) Проектування системи _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	10.06.2026	виконано
3	Аналіз існуючих рішень	11.06.2026	виконано
4	Визначення вимог до систем та вибір архітектури	13.06.2026	виконано
5	Програмна реалізація	15.06.2026	виконано
6	Написання пояснювальної записки	17.06.2026	виконано
7	Підготовка презентації та доповіді	23.06.2026	виконано
8	Попередній захист	04.07.2026	виконано
9	Захист перед ЕК	08.07.2026	

Дата видачі завдання 8 червня 2026 р.

Здобувач _____

(підпис)

Керівник роботи _____

(підпис)

ст.викл. Олена ГРИНЬОВА

(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 63 с., 9 рис., 1 табл., 1 дод., 21 джерело.

АВТОНОМНІСТЬ, ДИСТАНЦІЙНЕ НАВЧАННЯ, ІНТЕЛЕКТУАЛЬНА НАДБУДОВА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ЛОКАЛЬНЕ ЗБЕРІГАННЯ ДАНИХ, СЕМАНТИЧНИЙ ПОШУК, СТІЙКІСТЬ ІНФРАСТРУКТУРИ, NLP, QA-СИСТЕМА, RAG.

Об'єкт дослідження – процес інформаційної підтримки учасників освітнього процесу в умовах нестабільної інфраструктури з використанням технологій штучного інтелекту.

Предмет дослідження – методи та програмні засоби для побудови інтелектуальної RAG-системи, що працює з неструктурованими навчальними матеріалами різних форматів, забезпечує локальне зберігання даних та автономне функціонування.

Мета роботи – розробити інтелектуальну надбудову над платформою дистанційного навчання, яка забезпечує стійку підтримку освітнього процесу в умовах критичної нестабільності інфраструктури.

Методи дослідження – аналіз предметної галузі та огляд існуючих рішень; методи обробки природної мови (токенізація, лематизація, векторне представлення тексту, семантичний аналіз); методи інформаційного пошуку (TF-IDF, BM25, косинусна подібність, наближений пошук найближчих сусідів); аналіз архітектури RAG; методи веб-програмування (Python, Flask).

Протягом написання роботи досліджено сучасні методи побудови інтелектуальних QA-систем, зокрема підходи rule-based, IR-based, генеративні та гібридні (RAG). Розглядаються методи обробки природної мови, алгоритми пошуку релевантних фрагментів та підходи до забезпечення стійкості системи в умовах нестабільної інфраструктури.

ABSTRACT

Bachelor's thesis contains: 63 pp., 9 fig., 1 tabl., 1 ann., 21 references.

AUTONOMY, CLIENT-SERVER ARCHITECTURE, DISTANCE LEARNING, INFRASTRUCTURE RESILIENCE, INTELLIGENT ADD-ON, LOCAL DATA STORAGE, NLP, QA SYSTEM, RAG, SEMANTIC SEARCH.

Object of the study – is the process of information support for educational process participants under unstable infrastructure using AI technologies.

Subject of the study – are methods and software tools for building an intelligent RAG system that works with unstructured educational materials, ensures local data storage and autonomous functioning.

Objective of the study – is to develop an intelligent add-on over the distance learning platform that ensures sustainable support of the educational process under critical infrastructure instability.

Research methods – include subject area analysis, review of existing solutions, NLP methods (tokenization, lemmatization, text vectorization, semantic analysis), information retrieval methods (TF-IDF, BM25, cosine similarity, ANN), RAG architecture analysis, and web programming methods (Python, Flask).

During the writing of the work, modern methods of building intelligent QA systems (rule-based, IR-based, generative, and hybrid RAG approaches) are studied, along with NLP methods, relevant fragment search algorithms, and approaches to ensuring system resilience under unstable infrastructure.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та постановка задачі.....	11
1.1 Аналіз предметної галузі.....	11
1.2 QA-системи.....	13
1.3 Аналіз існуючих рішень	15
1.4 Постановка задачі.....	18
2 Методи та підходи до розробки інтелектуальних qa-систем.....	20
2.1 Підходи до побудови QA-систем	20
2.2 Методи обробки природної мови (NLP).....	23
2.3 Алгоритми пошуку релевантних фрагментів.....	27
2.3.1 Традиційні методи пошуку	27
2.3.2 Пошук на основі векторних представлень	28
2.3.3 Наближений пошук найближчих сусідів (ANN)	29
2.3.4 Ранжування результатів пошуку	30
2.4 Генеративні моделі та підхід RAG	31
2.5 Методи роботи з навчальними матеріалами	35
2.6 Підходи до забезпечення стійкості системи	37
3 Проектування та програмна реалізація системи	40
3.1 Архітектура системи.....	40
3.2 Реалізація основної QA-системи	43
3.2.1 Принцип роботи QA-системи	43
3.2.2 Структура зберігання даних	44
3.2.3 Автентифікація та розмежування прав доступу	45
3.3 Реалізація RAG-компонента	46
3.3.1 Архітектура RAG	46
3.3.2 Розбиття тексту на фрагменти та індексація.....	47
3.3.3 Семантичний пошук	48

3.3.4 Автоматична генерація питань.....	49
3.4 Веб-інтерфейс користувача.....	51
3.4.1 Чат для студентів	52
3.4.2 Адміністративна панель	55
Висновки	57
Перелік джерел посилання	60
Додаток А Відомість кваліфікаційної роботи.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІІІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

ANN – Approximate Nearest Neighbors – наближений пошук найближчих сусідів;

API – Application Programming Interface – програмний інтерфейс застосунку;

BERT – Bidirectional Encoder Representations from Transformers – двонаправлений енкодер представлень від трансформерів;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

HTML – HyperText Markup Language – мова гіпертекстової розмітки;

IR – Information Retrieval – інформаційний пошук;

JSON – JavaScript Object Notation – текстовий формат обміну даними;

LLM – Large Language Model – велика мовна модель;

LMS – Learning Management System – система управління навчанням;

NLP – Natural Language Processing – обробка природної мови;

PDF – Portable Document Format – портативний формат документів;

QA – Question Answering – система запитань-відповідей;

RAG – Retrieval-Augmented Generation – генерація з доповненням пошуку;

TF-IDF – Term Frequency – Inverse Document Frequency – частота терміна – обернена частота документа.

ВСТУП

У сучасному освітньому середовищі інформаційні технології відіграють ключову роль. Платформи дистанційного навчання стали невід'ємною частиною університетської інфраструктури, забезпечуючи студентів навчальними матеріалами, завданнями та зворотним зв'язком. Однак в умовах критичної нестабільності інфраструктури, спричиненої перебоями електропостачання, пошкодженням серверного обладнання, чи проблемами з доступом до мережі Інтернет, ці системи часто стають недоступними.

У реальних умовах часто виникають ситуації, коли студент має доступ до електроенергії та навчальних матеріалів, але не має можливості звернутися до викладача через його тимчасову недоступність, або навпаки. Також поширеною проблемою є необхідність самостійного пошуку відповідей у великій кількості навчальних файлів (лекції, методичні вказівки, презентації тощо), що є трудомістким і неефективним процесом.

Одним із перспективних напрямів вирішення зазначених проблем є створення інтелектуальної QA-системи, яка виступає як надбудова над системами дистанційного навчання та забезпечує автоматизовану підтримку освітнього процесу. Така система повинна працювати незалежно від основної платформи, встановлюватися на пристроях користувачів та забезпечувати доступ до навчальних матеріалів навіть у разі недоступності серверної інфраструктури.

У межах даної роботи було поставлено мету розробити та впровадити інтелектуальну надбудову над платформою дистанційного навчання, яка забезпечує стійку підтримку освітнього процесу в умовах критичної нестабільності інфраструктури шляхом надання швидких відповідей на запитання студентів на основі завантажених викладачем навчальних матеріалів.

Під час виконання роботи реалізовано такі основні етапи:

- аналіз проблематики університетської інфраструктури та обґрунтування необхідності створення інтелектуальної надбудови над DL;
- огляд сучасних RAG-систем та методів семантичного пошуку, вибір оптимальної архітектури;
- розробка модуля індексації та пошуку для роботи з файлами різних форматів (PDF, Excel, DOC);
- створення веб-інтерфейсу для студентів (чат із наданням відповідей) та адміністративної панелі для викладачів (керування курсами, завантаження файлів, ручне додавання питань);
- проведення серії експериментів та аналіз отриманих результатів.

У результаті буде отримано узагальнену картину ефективності запропонованої системи, зроблено висновки щодо доцільності її використання як надбудови над платформами дистанційного навчання в умовах нестабільної інфраструктури.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Починаючи з 2020 року, система вищої освіти України зазнала кардинальних змін, спричинених спочатку пандемією COVID-19 [1], а згодом – повномасштабною військовою агресією. Ці події змусили заклади вищої освіти екстрено переходити на дистанційні та змішані форми навчання [2]. Велика кількість українських університетів використовують дистанційний або змішаний формат навчання через безпекову ситуацію. Харківський національний університет радіоелектроніки, який є базою для дипломування, не є винятком – більшість занять проводиться онлайн із використанням платформ DL, Google Meet та Microsoft Teams.

Дистанційне навчання, попри свою очевидну необхідність у кризових умовах, створює низку додаткових викликів [3]. По-перше, воно вимагає від усіх учасників освітнього процесу стабільного доступу до інтернету та наявності відповідного технічного обладнання. По-друге, суттєво знижується можливість безпосереднього спілкування між студентами та викладачами, що ускладнює оперативне вирішення організаційних питань. По-третє, зростає навантаження на викладачів, які змушені відповідати на велику кількість однотипних питань студентів щодо термінів здачі робіт, вимог до оформлення, регламентів тощо.

Додатковим ускладнюючим фактором є нестабільність інфраструктури, яка проявляється у перебоях електропостачання та нестабільному або повністю відсутньому доступу до мережі Інтернет [4]. У таких умовах студенти не завжди мають можливість своєчасно отримувати навчальні матеріали або звертатися за консультацією до викладача, що негативно впливає на якість освітнього процесу. Особливо гостро ця проблема стоїть для прифронтових регіонів, де інфраструктура зазнає найбільших руйнувань.

Важливою є проблема асинхронності доступу до ресурсів. В умовах нестабільної інфраструктури доступ до електроенергії та інтернету є несинхронізованим між учасниками освітнього процесу. У той час як студент має доступ до електропостачання та мережі Інтернет, викладач може бути тимчасово позбавлений такої можливості, і навпаки. Це створює додаткові труднощі у комунікації та своєчасному отриманні відповідей на запитання. Студенти часто не можуть спланувати свій час, оскільки не знають, коли саме буде доступ до мережі для отримання консультації або завантаження виконаних робіт, що призводить до пропуску дедлайнів та зниження успішності.

У зв'язку з цим особливої актуальності набуває використання автоматизованих інтелектуальних систем, здатних забезпечити підтримку навчального процесу навіть за умов обмеженого доступу до ресурсів. Такі системи дозволяють оперативно надавати відповіді на типові запитання студентів, зменшують навантаження на викладачів та підвищують доступність освітньої інформації. Крім того, вони можуть виконувати роль альтернативного сховища навчальних матеріалів, забезпечуючи доступ до них навіть при недоступності основного сервера дистанційного навчання.

У контексті описаної вище проблеми асинхронності доступу, використання інтелектуальної QA-системи (Question Answering System) є особливо доцільним, оскільки вона дозволяє оперативно надавати відповіді незалежно від доступності викладача в конкретний момент часу. Система може працювати цілодобово, не вимагаючи одночасної присутності обох сторін комунікації, що є критично важливим в умовах, коли графіки доступу до світла у студентів та викладачів можуть не збігатися. Крім того, така система здатна аналізувати великі обсяги навчальних матеріалів (лекції, методичні вказівки, презентації) та надавати точні відповіді із зазначенням конкретного джерела, що значно економить час студента.

При цьому важливо підкреслити, що подібні системи не є заміною викладача, а виконують допоміжну функцію, забезпечуючи підтримку

освітнього процесу, підвищення його безперервності та доступності. Вони беруть на себе відповіді на типові, повторювані організаційні та навчальні питання, звільняючи час викладача для більш змістовної комунікації, індивідуальної роботи зі студентами та вирішення складних навчальних завдань, які потребують творчого підходу та безпосередньої взаємодії.

1.2 QA-системи

QA-система (Question Answering System) – це інтелектуальна інформаційна система, призначена для автоматичного пошуку та формування відповідей на запитання користувачів, сформульовані природною мовою [5]. На відміну від традиційних пошукових систем, які повертають список документів або посилань, що потенційно містять відповідь, QA-системи надають безпосередньо готову відповідь, що значно економить час користувача та підвищує зручність роботи з інформацією [6].

QA-системи набули значного поширення в різноманітних сферах діяльності людини. У сфері обслуговування клієнтів вони використовуються як чат-боти банків, мобільних операторів та інтернет-магазинів. В освіті вони виступають у ролі віртуальних асистентів студентів, допомагаючи з організаційними та навчальними питаннями. У медицині QA-системи застосовуються для діагностичних систем та підтримки прийняття рішень лікарями. У юридичній сфері вони допомагають аналізувати законодавство та знаходити релевантні норми. У контексті освітнього процесу QA-системи дозволяють автоматизувати відповіді на типові запитання студентів, що особливо актуально в умовах дистанційного навчання та нестабільної інфраструктури (рисунк. 1.1).

Основними етапами роботи QA-системи є: спочатку виконується обробка запиту користувача з використанням методів аналізу природної мови (токенізація, нормалізація, виділення ключових слів), після чого здійснюється пошук релевантної інформації в базі знань або документах. На

основі знайдених даних формується відповідь (шляхом витягування готового фрагмента або генерації нового тексту), яка потім повертається користувачу у зручному для сприйняття вигляді.

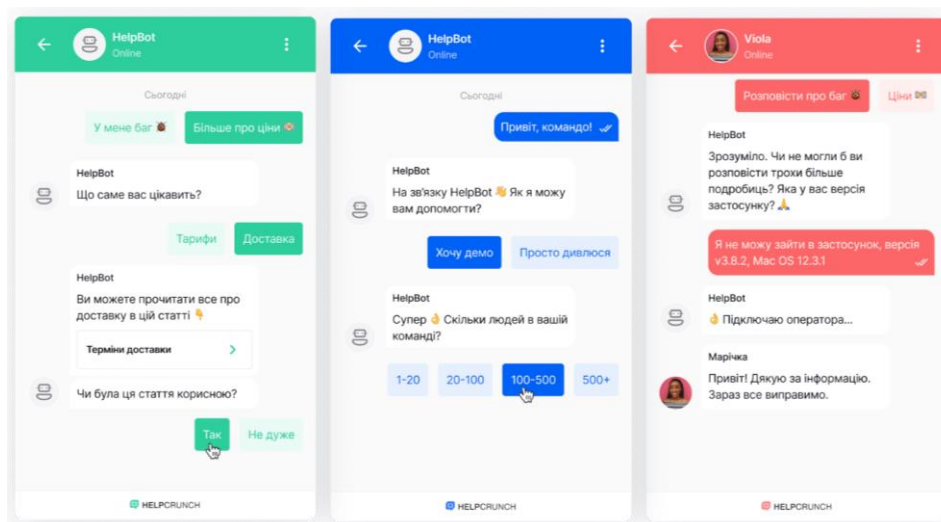


Рисунок 1.1 – QA-системи у різних сферах

Сучасні QA-системи можуть використовувати методи обробки природної мови (NLP), машинного навчання та семантичного аналізу, що дозволяє підвищити точність і релевантність відповідей, навіть коли запитання сформульоване нечітко або містить помилки.

За способом формування відповіді QA-системи поділяються на:

- системи з фіксованими відповідями, що базуються на заздалегідь підготовленій базі знань;
- інформаційно-пошукові системи, які здійснюють пошук відповідей у документах або базах даних;
- генеративні системи, що формують відповіді динамічно за допомогою моделей штучного інтелекту [7].

За умовами функціонування QA-системи можуть бути:

- онлайн-системами, які потребують постійного доступу до мережі Інтернет;

– локальними або гібридними, здатними частково функціонувати автономно.

У сфері освіти QA-системи виступають ефективним інструментом підтримки навчального процесу, забезпечуючи швидкий доступ до інформації, зменшуючи навантаження на викладачів та підтримуючи асинхронну взаємодію між студентами та викладачами. Вони можуть використовуватися для автоматизації відповідей на організаційні питання (дедлайни, вимоги до оформлення, розклад), а також для допомоги в навчанні (пояснення складних понять, пошук інформації в лекційних матеріалах). Особливої актуальності вони набувають в умовах нестабільної інфраструктури, коли доступ до викладача або освітніх ресурсів є обмеженим. У таких ситуаціях QA-системи можуть забезпечити безперервність отримання інформації, підвищити ефективність навчання та знизити рівень стресу у студентів, пов'язаний з невизначеністю.

1.3 Аналіз існуючих рішень

У сучасному освітньому середовищі існує значна кількість цифрових платформ та інтелектуальних систем, що забезпечують підтримку навчального процесу. Серед них можна виділити як системи управління навчанням (LMS), так і інтелектуальні QA-системи та чат-боти.

ChatGPT та Claude – сучасні генеративні QA-системи, які використовують моделі штучного інтелекту для формування відповідей на запитання користувачів. Вони здатні вести діалог природною мовою, пояснювати складні теми, генерувати текст та виконувати творчі завдання. Завдяки цим можливостям такі системи активно використовуються в освіті як віртуальні асистенти для студентів. Однак вони мають суттєві недоліки в контексті роботи в умовах нестабільної інфраструктури: повна залежність від постійного доступу до інтернету, неможливість працювати із закритими університетськими даними, а також ризик недостовірних відповідей (так

звані «галюцинації»), що є критичним для освітнього процесу, де точність інформації має першорядне значення [8].

Moodle – це одна з найпоширеніших систем управління навчанням у світі, яка широко використовується в закладах освіти, включаючи Харківський національний університет радіоелектроніки. Moodle надає інструменти для створення курсів, розміщення навчальних матеріалів у різних форматах, проведення тестувань, організації форумів для комунікації між студентами та викладачами. Вона підтримує гнучке налаштування та розширення функціональності через плагіни. Однак ключовим недоліком Moodle є відсутність вбудованої інтелектуальної QA-системи – відповіді на запитання студентів надаються викладачами вручну, що створює додаткове навантаження та не гарантує миттєвої відповіді, особливо в умовах асинхронного доступу до ресурсів [9].

Google Classroom – це безкоштовний хмарний сервіс від Google для організації навчального процесу, який тісно інтегрується з іншими сервісами екосистеми Google (Google Drive, Google Docs, Google Meet, Google Calendar). Він дозволяє викладачам створювати курси, роздавати завдання, збирати виконані роботи та виставляти оцінки. Google Classroom має стрічку коментарів, де студенти можуть ставити запитання, але, як і Moodle, не має вбудованої інтелектуальної QA-системи. Викладачі змушені відповідати на кожне запитання вручну, що створює додаткове навантаження. Крім того, сервіс є повністю хмарним, тому залежить від стабільного доступу до інтернету та недоступний при відключенні електропостачання або падінні серверів Google.

Проведений аналіз показує, що жодне з існуючих рішень повною мірою не задовольняє вимогам до системи, яка має забезпечувати стійку підтримку освітнього процесу в умовах нестабільної інфраструктури (таблиця. 1.1). Загальні генеративні моделі (ChatGPT, Claude) не мають доступу до закритих даних університету та повністю залежать від наявності інтернет-з'єднання. Існуючі LMS-

платформи (Moodle, Google Classroom) не забезпечують автоматичних відповідей на запитання студентів та залежать від центрального сервера, що робить їх недоступними при його падінні або відключенні електроенергії.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Критерій	ChatGPT / Claude	Moodle	Google Classroom	Пропонована система
Тип системи	Генеративна QA	LMS	LMS	Інтелектуальна QA- система
Доступ до закритих даних університету	Ні	Так	Ні	Так
Прив'язка відповіді до джерела	Ні	Ні	Ні	Так
Автоматичні відповіді на запитання	Так	Ні (вручну)	Ні (вручну)	Так
Розмежування прав доступу	Ні	Так	Так	Так

Спеціалізовані освітні чат-боти часто є комерційними продуктами, складними у налаштуванні та не інтегруються з університетською інфраструктурою. Це обґрунтовує необхідність розробки власної інтелектуальної надбудови над платформою дистанційного навчання, яка поєднуватиме переваги локальної роботи, доступу до закритих даних, автоматичних відповідей із прив'язкою до джерела та розмежування прав доступу [10].

1.4 Постановка задачі

На основі проведеного аналізу можна визначити ключові недоліки існуючих рішень, які необхідно усунути. По-перше, це залежність від інтернету та центрального сервера, більшість систем стають недоступними при відключенні електропостачання або падінні сервера дистанційного навчання. По-друге, відсутність доступу до закритих даних, зовнішні моделі не можуть аналізувати навчальні матеріали, які зберігаються на серверах університету. По-третє, відсутність автоматичних відповідей, LMS-платформи вимагають ручного втручання викладача для відповіді на кожне запитання. По-четверте, відсутність прив'язки до джерела, студент отримує відповідь, але не знає, з якого саме файлу вона взята.

Виходячи з виявлених недоліків, основною задачею кваліфікаційної роботи є розробка інтелектуальної надбудови над платформою дистанційного навчання, яка забезпечує стійку підтримку освітнього процесу в умовах критичної нестабільності інфраструктури. Така система повинна функціонувати автономно, не залежати від стану основного сервера DL та мати можливість локального зберігання всіх необхідних даних.

Для досягнення поставленої мети необхідно вирішити наступні конкретні задачі. Перш за все, потрібно розробити архітектуру системи, яка працює незалежно від основного сервера та забезпечує локальне зберігання даних. Далі слід реалізувати модуль індексації та пошуку для роботи з файлами різних форматів (PDF, Excel, DOC тощо), що дозволить системі аналізувати навчальні матеріали, завантажені викладачем. Наступним кроком є забезпечення розмежування прав доступу, щоб кожен викладач мав можливість редагувати лише власні курси, а інші викладачі могли їх лише переглядати.

Також необхідно створити веб-інтерфейс для студентів у вигляді чату з можливістю отримання відповідей із посиланням на конкретне джерело,

що дозволить студенту не лише отримати відповідь, але й звернутися до оригінального файлу для детальнішого ознайомлення.

Крім того, потрібно реалізувати адміністративну панель для викладачів, яка забезпечує керування курсами, завантаження файлів, ручне додавання питань та автоматичну генерацію FAQ на основі завантажених матеріалів.

На завершальному етапі слід провести тестування системи на реальних навчальних матеріалах та оцінити її ефективність, зокрема точність відповідей, швидкодію та стійкість до відсутності інтернет-з'єднання.

2 МЕТОДИ ТА ПІДХОДИ ДО РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ QA-СИСТЕМ

2.1 Підходи до побудови QA-систем

У сучасній практиці створення QA-систем склалося кілька основних підходів, які відрізняються за способом зберігання знань, методом пошуку відповідей та технологіями, що використовуються. Вибір конкретного підходу залежить від вимог до точності, швидкодії, автономності та складності реалізації системи.

Правила-орієнтовані (rule-based) QA-системи є найбільш простими та традиційними. Вони базуються на заздалегідь підготовленій базі знань, де кожному питанню (або набору варіантів формулювань) поставлено у відповідність конкретна відповідь. Пошук відповіді здійснюється шляхом порівняння запиту користувача з еталонними питаннями або їх варіантами з використанням методів нечіткого порівняння рядків [11].

Перевагами цього підходу є простота реалізації, висока точність відповідей на питання з бази знань, контрольованість та передбачуваність роботи, а також відсутність потреби в доступі до інтернету або складних обчислювальних ресурсах. Недоліками є необхідність ручного наповнення та підтримки бази знань, обмеженість кола питань, на які система може відповісти, та неможливість відповідати на питання, яких немає в базі.

Такі системи доцільно використовувати для вузькоспеціалізованих завдань з обмеженою та стабільною тематикою питань. Прикладом є FAQ-системи на сайтах організацій або чат-боти для відповідей на типові організаційні запитання.

Іншим поширеним підходом є системи, що базуються на методах інформаційного пошуку (Information Retrieval, IR). У таких системах відповідь формується шляхом пошуку найбільш релевантних документів або фрагментів тексту у базі знань. Вони не потребують попереднього

створення бази пар «питання-відповідь» – достатньо надати документи, у яких потрібно шукати. Типова IR-based QA-система складається з трьох основних етапів. На першому етапі виконується попередня обробка документів та вилучення з них потенційних відповідей (наприклад, виділення іменників, дат, назв). На другому етапі здійснюється пошук документів або фрагментів, релевантних до запиту користувача, з використанням методів, таких як TF-IDF або BM25. На третьому етапі зі знайдених фрагментів витягується безпосередньо відповідь [12].

Перевагами IR-based систем є здатність працювати з великими обсягами неструктурованих даних, можливість знаходити відповіді на питання, які не були заздалегідь передбачені, та відносна простота розширення бази знань (достатньо додати нові документи). Недоліками є менша точність порівняно з rule-based системами, залежність від якості пошукового алгоритму, а також відсутність гарантії, що знайдений фрагмент дійсно містить правильну відповідь.

З розвитком технологій штучного інтелекту широкого поширення набули генеративні QA-системи, що базуються на великих мовних моделях (Large Language Models, LLM). Такі системи здатні формувати відповіді природною мовою, враховуючи контекст запиту, що значно підвищує зручність взаємодії з користувачем. Генеративні системи використовують великі мовні моделі (Large Language Models, LLM), такі як GPT, Gemini, Claude, для динамічного формування відповідей на запитання користувачів. Ці моделі навчаються на величезних корпусах текстів (мільярди слів) і здатні генерувати зв'язний, граматично правильний текст, вести діалог, відповідати на запитання широкого спектру тем [13].

Перевагами генеративних систем є універсальність – здатність відповідати на різноманітні питання, природність діалогу, здатність узагальнювати та робити висновки на основі наданої інформації. Однак вони мають і суттєві недоліки в контексті освітніх застосувань. По-перше, це ризик недостовірних відповідей («галюцинацій») – модель може

генерувати правдоподібну, але фактично неправильну інформацію. По-друге, залежність від доступу до інтернету (більшість потужних LLM доступні лише через хмарні API). По-третє, неможливість доступу до закритих даних – модель не має доступу до актуальних або закритих даних, якщо вони не надані додатково. По-четверте, високі обчислювальні вимоги для локального розгортання.

Гібридні системи поєднують елементи різних підходів для компенсації недоліків кожного з них. Найбільш поширеною є комбінація інформаційно-пошукового та генеративного підходів, яка отримала назву RAG (Retrieval-Augmented Generation). У цьому підході спочатку виконується пошук релевантних фрагментів документів (як у IR-based системах), а потім знайдені фрагменти подаються як контекст до генеративної моделі, яка формує відповідь на основі цього контексту.

Порівняно з іншими підходами, RAG має низку важливих переваг. Він не потребує ручного створення бази пар «питання-відповідь» – достатньо завантажити матеріали у вигляді документів. На відміну від інформаційно-пошукових систем, цей підхід забезпечує природне та зв'язне формулювання відповіді. Водночас, порівняно з чистими генеративними моделями, RAG значно знижує ризик галюцинацій, оскільки відповідь формується на основі наданого контексту. Додатковою перевагою є можливість роботи із закритими даними та забезпечення прив'язки відповіді до джерела інформації.

З урахуванням особливостей освітнього середовища та вимог до системи, що розробляється в межах даної роботи, найбільш доцільним є використання гібридного підходу, зокрема Retrieval-Augmented Generation (RAG).

Даний підхід поєднує переваги інформаційного пошуку та генеративних моделей, забезпечуючи формування відповідей на основі конкретних навчальних матеріалів. Це дозволяє досягти високої точності,

забезпечити прив'язку до джерела інформації та обмежити генерацію відповідей лише релевантними даними.

Крім того, використання RAG-підходу є особливо доцільним в умовах нестабільної інфраструктури, оскільки він дозволяє працювати з локальними базами знань, зменшуючи залежність від зовнішніх ресурсів.

Таким чином, аналіз підходів до побудови QA-систем показує, що гібридні методи, зокрема RAG, є найбільш ефективними для реалізації інтелектуальної системи підтримки освітнього процесу.

2.2 Методи обробки природної мови (NLP)

Обробка природної мови (Natural Language Processing, NLP) є ключовим компонентом сучасних інтелектуальних QA-систем, оскільки забезпечує можливість аналізу, розуміння та інтерпретації текстових запитів користувачів. Використання методів NLP дозволяє перетворювати неструктурований текст у форму, придатну для подальшої обробки та пошуку релевантної інформації. Без методів NLP сучасні системи не змогли б ефективно опрацьовувати запити користувачів та знаходити релевантні відповіді в масивах текстів.

Токенізація (tokenization) – це процес розбиття суцільного тексту на окремі значущі елементи, які називаються токенами. Токенами можуть бути окремі слова, числа, розділові знаки, аббревіатури або інші символи, які мають смислове значення в контексті. Цей етап є першим і обов'язковим у будь-якій системі обробки природної мови, оскільки подальший аналіз тексту виконується саме на рівні окремих токенів.

Існує кілька підходів до виконання токенізації. Найпростіший підхід полягає в розбитті тексту за пробілами та розділовими знаками – у цьому випадку кожне слово, відокремлене пробілом, стає окремим токеном, а розділові знаки або приєднуються до слів, або виділяються в окремі токени. Більш складні підходи використовують регулярні вирази для визначення

меж токенів з урахуванням особливостей конкретної мови. Найбільш точними є спеціалізовані бібліотеки токенізації, які враховують правила правопису, розпізнають аббревіатури, числа з комами, електронні адреси, дати та інші складні випадки.

Для української мови токенізація має свої особливості, пов'язані з наявністю апострофів (наприклад, «з'ясувати», «комп'ютер»), дефісів (наприклад, «науково-дослідний», «відео-урок»), а також складних випадків словотворення, де правильне розбиття на токени впливає на якість подальшого аналізу. Коректна токенізація є критично важливою, оскільки помилки на цьому етапі призводять до помилок на всіх наступних етапах обробки тексту.

Лематизація (lemmatization) – це процес зведення різних слів до їхньої базової, словникової форми, яка називається лемою. Лема – це канонічна форма слова, яка відповідає його поданню у словнику. Наприклад, слова «навчаюся», «навчався», «навчатиметься», «навчалися» мають спільну лему «навчатися», а слова «студента», «студенту», «студентом» – спільну лему «студент» [14].

Лематизація відрізняється від стемінгу (простішого методу відкидання закінчень) тим, що вона враховує контекст та морфологічні особливості мови. Стемінг може повернути некоректну (неіснуючу) основу слова, тоді як лематизація завжди повертає реальне слово, наявне в словнику. Наприклад, для слова «кращі» стемінг може дати основу «кращ», що не є словом, тоді як лематизація поверне коректну лему «кращий».

Для виконання лематизації використовуються словникові бази (лексикони), які містять відповідності між різними словформами та їх лемами, а також граматичні правила певної мови. Для української мови існують спеціалізовані лематизатори, які враховують складну морфологію української мови (відмінки, роди, числа, часи, дієвідміни тощо). Застосування лематизації дозволяє значно зменшити кількість унікальних

слів у тексті без втрати смислового значення, що спрощує подальший аналіз та порівняння текстів.

Векторне представлення тексту (embedding) – це спосіб перетворення текстових даних (символів, слів, речень, документів) у числові вектори фіксованої довжини. Це необхідно для того, щоб комп'ютерні алгоритми могли «розуміти» текст – аналізувати, порівнювати, шукати схожі тексти, групувати їх за сенсом [15].

Словникові ембедінги (word embeddings) ставлять кожному слову у відповідність вектор чисел (зазвичай довжиною від 100 до 300 вимірів). Головна властивість таких векторів полягає в тому, що семантично близькі слова мають близькі вектори у цьому просторі. Наприклад, вектори слів «комп'ютер», «ноутбук», «лептоп» будуть розташовані близько один до одного, тоді як вектор слова «стіл» буде значно далі. Більше того, між векторами слів можна виконувати арифметичні операції: наприклад, вектор «король» мінус вектор «чоловік» плюс вектор «жінка» дає вектор, близький до вектора «королева».

Контекстуальні ембедінги (contextual embeddings) є більш сучасним та точним підходом, у якому векторне представлення слова залежить від контексту його вживання. Наприклад, слово «банк» у реченнях «фінансовий банк» та «банк даних» отримає різні вектори, оскільки в цих контекстах воно має різне значення. Це дозволяє значно підвищити точність семантичного аналізу порівняно зі словниковими ембедінгами, де кожне слово має єдиний вектор незалежно від контексту.

Для створення ембедінгів використовуються нейромережеві моделі, які навчаються на величезних корпусах текстів (мільярди слів). Під час навчання модель аналізує, які слова та в яких контекстах зустрічаються разом, і на основі цього будує векторний простір, де близькими є слова зі схожими контекстами вживання. Найбільш відомими моделями для створення словникових ембедінгів є Word2Vec, GloVe та FastText. Для

створення контекстуальних ембедінгів найчастіше використовуються моделі сімейства BERT та їх модифікації.

Семантичний аналіз (semantic analysis) – це сукупність методів, спрямованих на виявлення та формалізацію смислового змісту текстів. На відміну від синтаксичного аналізу, який вивчає граматичну структуру речень, семантичний аналіз зосереджується на значенні слів, словосполучень і речень у цілому.

Визначення семантичної близькості текстів (semantic similarity) – одна з ключових задач семантичного аналізу в контексті QA-систем. Вона полягає у визначенні того, наскільки два текстові фрагменти (наприклад, запит користувача та питання з бази знань) є близькими за змістом. Рішення цієї задачі базується на порівнянні векторних представлень текстів, отриманих за допомогою ембедінгів. Чим ближче вектори у просторі, тим більш семантично близькими вважаються тексти.

Виділення ключових слів та тем (keyword extraction, topic modeling) – це задача визначення найбільш важливих слів або тем, які характеризують текст. У QA-системах виділення ключових слів використовується для покращення пошуку: замість порівняння всього тексту запиту з усіма питаннями, система може порівнювати лише ключові слова та фрази, що прискорює роботу та підвищує точність.

Аналіз тональності (Sentiment Analysis) – це задача визначення емоційного забарвлення тексту (позитивне, негативне, нейтральне, а також складніші емоції: гнів, сум, радість, здивування). В освітніх QA-системах аналіз тональності може використовуватися для виявлення студентів, які відчують фрустрацію або потребують додаткової підтримки. Наприклад, якщо студент ставить питання агресивним або роздратованим тоном, система може запропонувати звернутися до викладача безпосередньо або надати розгорнуту, дружню відповідь.

У контексті побудови QA-систем семантичний аналіз використовується на всіх етапах роботи з текстом: від розуміння запиту

користувача до оцінки релевантності знайдених відповідей. Без методів семантичного аналізу система не змогла б відрізнити питання про «термін задачі роботи» від «терміну дії договору» або зрозуміти, що «дедлайн» та «крайній термін» означають одне й те саме.

2.3 Алгоритми пошуку релевантних фрагментів

Пошук релевантних фрагментів є ключовим етапом у будь-якій QA-системі, особливо в системах, побудованих за принципом RAG (Retrieval-Augmented Generation). Від якості цього етапу залежить, наскільки точну та доречну відповідь зможе згенерувати система на наступному етапі. У цьому підрозділі розглядаються основні алгоритми та підходи до пошуку релевантних фрагментів у текстовому корпусі.

2.3.1 Традиційні методи пошуку

Метод TF-IDF (Term Frequency – Inverse Document Frequency) є одним із найбільш поширених класичних методів інформаційного пошуку (2.1). Він базується на двох ключових показниках. Частота терміну в документі (Term Frequency, TF) показує, як часто слово зустрічається в конкретному документі – чим частіше, тим важливішим вважається це слово для даного документа. Обернена частота терміну в колекції (Inverse Document Frequency, IDF) показує, наскільки рідкісним є слово в усьому корпусі документів – чим рідше слово зустрічається в інших документах, тим більшу вагу воно отримує [16].

$$w_{t,d} = \text{tf}_{t,d} \cdot \log\left(\frac{N}{\text{df}_t}\right), \quad (2.1)$$

де $\text{tf}_{t,d}$ – частота терміна в документі;

N – загальна кількість документів у корпусі;

df_t – кількість документів, що містять цей термін.

Добуток цих двох показників ($TF \times IDF$) дає числову вагу кожного слова в кожному документі. TF-IDF є ефективним для пошуку ключових слів та ранжування документів за релевантністю до запиту, але він не враховує порядок слів та їх семантичний зв'язок.

Алгоритм BM25 (Окарі BM25) є вдосконаленням підходу TF-IDF і вважається одним із найефективніших традиційних алгоритмів ранжування документів. Він враховує три основні фактори: частоту терміна в документі (TF), обернену частоту терміна в колекції (IDF) та нормалізацію довжини документа. Нормалізація довжини дозволяє усунути недолік, при якому довгі документи мають не виправдану перевагу просто через більшу кількість слів. BM25 має два регульовані параметри (k_1 та b), які дозволяють налаштовувати вплив частоти терміна та довжини документа на кінцевий результат. Цей алгоритм широко використовується в пошукових системах завдяки своїй ефективності, швидкодії та відносній простоті реалізації [17].

2.3.2 Пошук на основі векторних представлень

Косинусна подібність (Cosine Similarity) є стандартною метрикою для порівняння векторних представлень текстів. Вона обчислює косинус кута між двома векторами в багатовимірному просторі. Значення косинусної подібності знаходиться в діапазоні від 0 до 1 для додатних векторів (якими є вектори частот слів або ембедінги), де 1 означає, що вектори співнапрямлені (тексти максимально схожі), а 0 – що вектори ортогональні (схожість відсутня). Для порівняння запиту користувача з документами спочатку обчислюються векторні представлення запиту та кожного документа (або фрагмента), а потім розраховується косинусна подібність між ними. Документи з найбільшими значеннями косинусної подібності вважаються найбільш релевантними [18].

Пошук найближчих сусідів (k-NN, k-Nearest Neighbors) – це підхід, у якому для заданого вектора-запиту знаходяться k найближчих до нього векторів у багатовимірному просторі за вибраною метрикою відстані. У контексті пошуку релевантних фрагментів це означає, що система знаходить k фрагментів документів, векторні представлення яких є найбільш близькими до векторного представлення запиту користувача. Точний пошук k-найближчих сусідів (exact k-NN) гарантує знаходження найближчих векторів, але для великих масивів даних (мільйони векторів) він стає надто повільним, оскільки вимагає порівняння з усіма векторами в базі.

2.3.3 Наближений пошук найближчих сусідів (ANN)

Для роботи з великими масивами даних (типовий розмір векторної бази в сучасних RAG-системах може становити мільйони або навіть десятки мільйонів векторів) використовуються алгоритми наближеного пошуку найближчих сусідів (Approximate Nearest Neighbors, ANN). Вони не гарантують знаходження абсолютно найближчих векторів, але знаходять достатньо близькі вектори за значно менший час порівняно з точним пошуком. Компроміс між швидкістю та точністю регулюється параметрами алгоритму.

Метод HNSW (Hierarchical Navigable Small World) є одним із найбільш ефективних алгоритмів ANN. Він будує багаторівневу графову структуру, де на верхніх рівнях знаходяться «опорні» точки з довгими зв'язками, які дозволяють швидко наблизитися до цільової області, а на нижніх рівнях – детальна структура для точного пошуку. HNSW забезпечує високу швидкість пошуку (логарифмічну залежність від кількості векторів) та добру точність, що робить його популярним вибором для векторних баз даних.

Метод IVF (Inverted File Index) – це підхід, заснований на кластеризації векторів. Усі вектори в базі даних розбиваються на K кластерів (наприклад, за допомогою алгоритму k-Means). Під час пошуку спочатку визначається, до яких кластерів належить вектор-запит (зазвичай обирається кілька найближчих кластерів), а потім пошук виконується лише серед векторів цих кластерів, ігноруючи решту бази. Це значно прискорює пошук, оскільки замість перебору всіх векторів перебираються лише вектори з обраних кластерів. IVF часто комбінується з методом PQ (Product Quantization) для додаткового стиснення векторів та прискорення обчислень.

2.3.4 Ранжування результатів пошуку

Після того як система знайшла набір потенційно релевантних фрагментів, необхідно їх упорядкувати (ранжувати) за ступенем релевантності до запиту користувача. Найпростішим підходом є ранжування за значенням косинусної подібності або іншої метрики близькості – чим ближче вектор фрагмента до вектора запиту, тим вище він розташовується в результатах.

У більш складних системах можуть використовуватися методи переранжування (re-ranking).

На першому етапі за допомогою швидкого, але менш точного методу (наприклад, BM25 або IVF) відбирається відносно невелика кількість кандидатів (наприклад, 100–200 фрагментів).

На другому етапі ці кандидати повторно оцінюються більш точним, але повільнішим методом (наприклад, крос-енкодером на основі BERT), який аналізує пару «запит-фрагмент» комплексно. Це дозволяє досягти високої точності при збереженні прийнятної швидкодії [19].

2.4 Генеративні моделі та підхід RAG

Після того як система знайшла релевантні фрагменти документів, необхідно сформулювати на їх основі зв'язну, граматично правильну та змістовну відповідь. Для цього використовуються генеративні моделі. У цьому підрозділі розглядаються архітектура трансформера, яка лежить в основі сучасних генеративних моделей, а також підхід RAG (Retrieval-Augmented Generation), який поєднує пошук та генерацію.

Архітектура трансформера (Transformer) була представлена в 2017 році в статті «Attention Is All You Need» та стала основою для більшості сучасних великих мовних моделей, включаючи GPT, BERT, LLaMA, Claude та інші. До появи трансформера домінуючими підходами в обробці послідовностей були рекурентні нейронні мережі (RNN) та згорткові нейронні мережі (CNN). RNN обробляли послідовності крок за кроком, що унеможливлювало паралелізацію обчислень та обмежувало довжину послідовностей, які могли бути опрацьовані. Трансформер вирішив обидві ці проблеми.

Механізм самоуваги (self-attention) є ключовим компонентом архітектури трансформера. Він дозволяє моделі визначати важливість різних слів у послідовності відносно одне одного. Для кожного слова в реченні обчислюються три вектори: вектор запиту (query, Q), вектор ключа (key, K) та вектор значення (value, V). Вага уваги між двома словами обчислюється як скалярний добуток вектора запиту першого слова та вектора ключа другого слова, після чого застосовується функція softmax для отримання ймовірнісного розподілу. Вихідне представлення слова є зваженою сумою векторів значень усіх слів, де ваги визначаються обчисленими коефіцієнтами уваги. Це дозволяє моделі «зосереджуватися» на найбільш релевантних частинах вхідної послідовності.

Багатоголовий механізм самоуваги (multi-head attention) розширює механізм самоуваги, використовуючи кілька незалежних «голов» уваги,

кожна з яких має власні навчувані матриці параметрів Q , K , V . Результати всіх голів конкатенуються та лінійно перетворюються. Це дозволяє моделі одночасно фокусуватися на різних типах взаємозв'язків між словами. Наприклад, одна голова може навчатися синтаксичним зв'язкам, друга голова – семантичним, третя – анафоричним (зв'язок займенника з іменником).

Позиційне кодування (positional encoding) є ще одним важливим компонентом трансформера. Оскільки механізм самоуваги не враховує порядок слів у послідовності (він є інваріантним до перестановок), необхідно додати інформацію про позицію кожного слова. У трансформері використовуються синусоїдальні функції для створення векторів позиційного кодування, які додаються до векторів ембедінгів слів. Це дозволяє моделі розрізняти, наприклад, речення «кот з'їв мишу» та «миша з'їла кота».

Трансформер складається з двох основних частин: енкодера (encoder) та декодера (decoder). Енкодер перетворює вхідну послідовність у проміжне представлення (набір векторів). Декодер генерує вихідну послідовність (наприклад, відповідь) на основі виходу енкодера та раніше згенерованих слів. Моделі типу BERT використовують лише енкодер (для розуміння тексту), моделі типу GPT – лише декодер (для генерації тексту), а моделі типу T5 – повний трансформер (енкодер + декодер).

RAG (Retrieval-Augmented Generation) – це підхід, який поєднує компонент пошуку (retrieval) з компонентом генерації (generation) для покращення якості відповідей мовних моделей. Він був запропонований дослідниками з Facebook AI (Meta) у 2020 році та швидко став стандартом для побудови систем, які потребують доступу до актуальних або специфічних знань, відсутніх у навчальному корпусі моделі.

RAG складається з трьох основних етапів, які виконуються послідовно.

Етап 1 – індексація (Indexing). На цьому етапі всі документи, з якими система має працювати (навчальні матеріали, методичні вказівки, лекції, презентації тощо), попередньо обробляються. Кожен документ (або окремі фрагменти документів) перетворюється на векторне представлення (ембедінг) за допомогою моделі ембедінгів (наприклад, BERT, Sentence-BERT або іншої спеціалізованої моделі). Отримані вектори зберігаються у векторній базі даних (наприклад, FAISS, Chroma, Pinecone), яка підтримує ефективний пошук за семантичною близькістю. Цей етап виконується один раз перед початком роботи системи та оновлюється лише тоді, коли змінюються або додаються нові документи.

Етап 2 – пошук (Retrieval). Коли користувач задає запит (наприклад, «Коли потрібно здати курсову роботу?»), система перетворює цей запит на векторне представлення за допомогою тієї самої моделі ембедінгів, яка використовувалася на етапі індексації. Потім виконується пошук у векторній базі даних для знаходження k найбільш семантично близьких векторів (де k – параметр системи, зазвичай від 3 до 10). Для цього використовуються алгоритми наближеного пошуку найближчих сусідів (ANN), розглянуті в підрозділі 2.3.3. Знайдені вектори відповідають конкретним фрагментам документів, які повертаються як контекст. Таким чином, система отримує релевантні фрагменти, які потенційно містять відповідь на запит користувача.

Етап 3 – генерація (Generation). На цьому етапі знайдені на попередньому кроці фрагменти (контекст) підставляються в спеціально сконструйований промпт (запит) до великої мовної моделі (LLM). Промпт зазвичай має структуру: «Використовуючи наступний контекст, дай відповідь на запитання користувача. Контекст: [фрагмент 1], [фрагмент 2], Запитання: [запит користувача]. Відповідь:». Модель генерує відповідь на основі наданого контексту. Оскільки модель бачить конкретні фрагменти, вона може навести цитати або вказати, з якого саме джерела взято інформацію. Важливо, що модель генерує відповідь переважно на основі

наданого контексту, що значно знижує ризик галюцинацій (генерації недостовірної інформації).

RAG має кілька ключових переваг, які роблять його особливо придатним для створення освітніх QA-систем, особливо в умовах нестабільної інфраструктури.

По-перше, прив'язка до джерел. RAG-система може не лише надати відповідь, але й вказати, з якого саме документа або навіть фрагмента взято інформацію. Це дозволяє студенту звернутися до першоджерела для детальнішого вивчення матеріалу, що є критично важливим в освітньому процесі.

По-друге, можливість роботи із закритими даними. Оскільки RAG-система використовує лише ті документи, які були попередньо проіндексовані, вона може працювати з конфіденційними або закритими матеріалами (наприклад, навчальними матеріалами університету, які не публікуються у відкритому доступі), не передаючи їх зовнішнім моделям або хмарним сервісам. Це особливо важливо для закладів освіти, які мають власні закриті репозиторії навчальних матеріалів.

По-третє, актуальність відповідей. RAG-система завжди ґрунтується на наданих документах. Коли викладач оновлює навчальні матеріали (наприклад, змінює дедлайн або додає нові вимоги до оформлення робіт), достатньо переіндексувати оновлені документи (або додати нові), і система почне використовувати актуальну інформацію без необхідності донавчання або перенавчання мовної моделі.

По-четверте, зменшення галюцинацій. На відміну від чистих генеративних систем (наприклад, ChatGPT без підключених джерел), які можуть генерувати правдоподібну, але неправильну відповідь, RAG-система генерує відповідь строго на основі наданого контексту. Якщо запит користувача виходить за межі наявних документів, система може коректно повідомити про відсутність інформації, замість того щоб «вигадувати» відповідь.

По-п'яте, локальне розгортання. RAG-система може бути розгорнута повністю локально – з використанням локальних моделей ембедінгів (наприклад, BERT) та локальних генеративних моделей (наприклад, LLaMA або інших відкритих LLM). Це робить систему стійкою до відключень інтернету та відсутності доступу до хмарних сервісів, що є критично важливим в умовах нестабільної інфраструктури.

Таким чином, RAG є оптимальним архітектурним підходом для створення інтелектуальної надбудови над платформою дистанційного навчання, оскільки він поєднує точність пошуку, гнучкість генерації, прив'язку до джерел та можливість локального розгортання.

2.5 Методи роботи з навчальними матеріалами

Для того щоб інтелектуальна QA-система могла працювати з навчальними матеріалами, які завантажують викладачі, необхідно вирішити низку завдань, пов'язаних з обробкою файлів різних форматів, розбиттям текстів на фрагменти та створенням індексів для ефективного пошуку.

Навчальні матеріали в університетському середовищі представлені в різних форматах. Найбільш поширеними є PDF (лекції, методичні вказівки, наукові статті), DOC/DOCX (текстові документи, завдання), презентації (слайди лекцій), а також звичайні текстові файли. Кожен із цих форматів має свою внутрішню структуру, тому для їх обробки потрібні спеціалізовані бібліотеки.

Для роботи з PDF-файлами використовуються бібліотеки, які дозволяють витягувати текстовий вміст сторінок, зберігаючи при цьому інформацію про структуру документа (розділи, підрозділи, заголовки). Для DOC/DOCX-файлів використовуються бібліотеки, які працюють з внутрішнім XML-поданням документів Microsoft Word. Для Excel-файлів використовуються бібліотеки, які дозволяють читати дані з окремих аркушів та комірок, а також витягувати текстові значення з формул.

Після витягування тексту з файлу необхідно виконати його попередню обробку: видалити зайві пробіли, розділові знаки, спеціальні символи, а також розділити текст на окремі речення або абзаци. Це необхідно для подальшого розбиття на фрагменти.

Оскільки більшість моделей ембедінгів та генеративних моделей мають обмеження на довжину вхідного тексту (так званий контекст, який для багатьох моделей становить від 512 до 4096 токенів), довгі навчальні матеріали (наприклад, підручник на кілька сотень сторінок) не можуть бути оброблені цілком. Тому текст необхідно розбивати на менші фрагменти – цей процес називається *chunking* (розбиття на фрагменти або «*chunks*») [20].

Існує кілька стратегій розбиття тексту на фрагменти. Найпростішою є фіксоване розбиття за кількістю символів або токенів – текст просто нарізається на шматки однакового розміру. Цей метод простий у реалізації, але може розривати речення або абзаци посередині, що погіршує якість пошуку.

Більш досконалим підходом є семантичне розбиття, при якому межі фрагментів проводяться по межах речень, абзаців або логічних розділів документа. Наприклад, фрагментом може бути окремий абзац, або група з кількох абзаців об'єднаних спільною темою. Для визначення меж фрагментів можуть використовуватися заголовки розділів, маркери списків, або аналіз когезії тексту (лексичних зв'язків між сусідніми реченнями).

Важливим параметром є розмір фрагмента. Занадто маленькі фрагменти (наприклад, одне речення) можуть втрачати контекст – окреме речення поза контекстом може бути незрозумілим. Занадто великі фрагменти можуть не вміщатися в ліміт контексту моделі або містити занадто багато нерелевантної інформації. Оптимальний розмір фрагмента залежить від конкретного застосування та використовуваних моделей і зазвичай підбирається експериментально [21].

Після того як текст розбито на фрагменти, кожен фрагмент необхідно перетворити на векторне представлення (ембедінг) за допомогою моделі

ембедінгів. Цей процес називається індексацією. Отримані вектори зберігаються у векторній базі даних разом з метаданими: назвою документа, номером сторінки (якщо доступно), позицією фрагмента в документі.

Індексація виконується один раз при завантаженні навчальних матеріалів викладачем. При оновленні матеріалів (наприклад, додається новий PDF-файл) індексація виконується лише для нового файлу. При видаленні або зміні матеріалів відповідні вектори видаляються з бази або оновлюються. Це забезпечує актуальність даних без необхідності повної переіндексації всіх матеріалів.

Після створення індексу система готова до пошуку: отримавши запит користувача, вона перетворює його на вектор за допомогою тієї самої моделі ембедінгів і знаходить найближчі вектори в базі, які відповідають найбільш релевантним фрагментам навчальних матеріалів.

2.6 Підходи до забезпечення стійкості системи

Однією з ключових вимог до системи, що розробляється в рамках даної кваліфікаційної роботи, є її здатність функціонувати в умовах критичної нестабільності інфраструктури. Як було зазначено в розділі 1, до таких умов належать перебої електропостачання, падіння сервера дистанційного навчання, нестабільний або відсутній доступ до інтернету. У цьому підрозділі розглядаються підходи до забезпечення стійкості системи до таких збоїв.

Основою стійкості системи є її автономність – здатність працювати без доступу до зовнішніх мереж та серверів. Автономність досягається шляхом локального зберігання всіх даних (навчальних матеріалів, векторних індексів, бази питань-відповідей) на пристрої користувача.

У клієнт-серверній архітектурі автономність означає, що клієнтська частина системи повинна містити локальну копію всіх необхідних даних. Якщо сервер стає недоступним (через падіння сервера DL або відключення

електроенергії), клієнт продовжує працювати з локальною копією, хоча і з обмеженою функціональністю (наприклад, не може отримувати оновлення від викладача).

Локальне зберігання даних реалізується з використанням вбудованих баз даних або файлової системи. Для зберігання структурованих даних (питання, відповіді, метадані) може використовуватися JSON, SQLite або інша легковагова база даних. Для зберігання векторних індексів використовуються векторні бази даних, які підтримують локальний режим роботи (наприклад, FAISS або Chroma в локальному режимі). Для зберігання оригінальних файлів (PDF, DOC, Excel) використовується файлова система.

Кешування є важливим механізмом для підвищення стійкості та швидкодії системи. Суть кешування полягає в тому, що результати дорогих обчислень (наприклад, векторне представлення запиту або відповідь, згенерована LLM) зберігаються локально після першого виконання. При повторному надходженні того самого або дуже схожого запиту система може повернути результат із кешу, не виконуючи пошук та генерацію заново.

Кешування особливо корисне в освітніх системах, оскільки багато студентів ставлять одні й ті самі питання. Наприклад, питання «Коли дедлайн ПЗ-1?» може бути задане десятками студентів. Без кешування система щоразу виконуватиме пошук та генерацію (або навіть звернення до зовнішньої LLM). З кешуванням – повертатиме готову відповідь миттєво.

Синхронізація вирішує проблему узгодження даних між клієнтом та сервером. Коли відновлюється доступ до сервера (наприклад, після відключення електроенергії), клієнт повинен перевірити, чи не змінилися дані на сервері (викладач міг додати нові матеріали або оновити відповіді). Для цього використовуються механізми синхронізації: клієнт періодично перевіряє, чи є оновлення, або сервер надсилає повідомлення про зміни (наприклад, за допомогою механізмів push-повідомлень).

Клієнт-серверна архітектура є оптимальним вибором для системи, що розробляється, оскільки вона дозволяє поєднати переваги централізованого управління (викладачі керують курсами через адмін-панель) та децентралізованої стійкості (студенти можуть працювати з локальною копією при недоступності сервера).

У цій архітектурі серверна частина відповідає за зберігання основного репозиторію навчальних матеріалів, управління користувачами та розмежування прав доступу (автор курсу може редагувати, інші викладачі можуть лише переглядати). Сервер також надає API для синхронізації даних з клієнтами.

Клієнтська частина встановлюється на пристрої користувача (ноутбук студента або викладача). Вона містить локальну копію даних, необхідних для роботи: векторний індекс навчальних матеріалів курсів, на які підписаний студент, а також базові компоненти для пошуку та генерації. Клієнт може працювати повністю автономно, синхронізуючи дані з сервером при відновленні з'єднання.

Такий підхід забезпечує стійкість до збоїв: при падінні сервера ДЛ або відключенні інтернету студент продовжує отримувати доступ до навчальних матеріалів та відповідей на запитання з локальної копії. Єдине, що стає недоступним – це оновлення матеріалів викладачем (але це не критично для короткочасних збоїв).

З ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У цьому розділі розглядаються архітектурні рішення, програмна реалізація та особливості функціонування розробленої інтелектуальної надбудови над платформою дистанційного навчання. Система отримала назву SmartQA та реалізована з використанням мови програмування Python, веб-фреймворку Flask та сучасних технологій обробки природної мови.

3.1 Архітектура системи

Розроблена система SmartQA має клієнт-серверну архітектуру з підтримкою автономної роботи клієнтської частини (рисунок 3.1). Такий підхід забезпечує стійкість функціонування в умовах нестабільної інфраструктури, коли доступ до сервера може бути тимчасово відсутнім через перебої електропостачання або пошкодження мережевого обладнання.

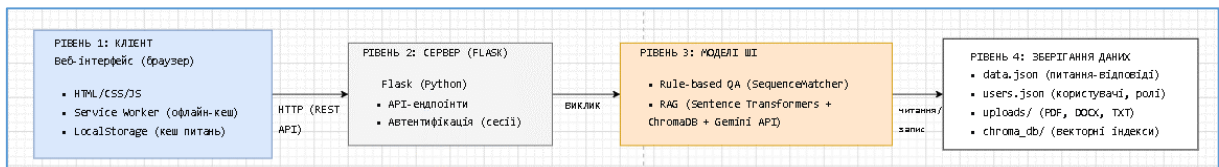


Рисунок 3.1 – Клієнт-серверна архітектура

Загальна архітектура системи SmartQA складається з чотирьох основних рівнів:

– рівень представлення – веб-інтерфейс користувача, реалізований на HTML, CSS та JavaScript, що забезпечує взаємодію з системою через браузер;

- рівень логіки обробки – серверна частина на Flask, яка приймає запити, виконує автентифікацію, координацію між компонентами та формування відповідей;
- рівень моделей штучного інтелекту – два ключові компоненти: rule-based та RAG-система;
- рівень зберігання даних – файли data.json (база питань-відповідей), users.json (користувачі) та векторна база даних ChromaDB (індекси документів для семантичного пошуку).

Серверна частина системи реалізована на фреймворку Flask мовою Python і виконує зберігання основної бази питань-відповідей, управління користувачами та їх автентифікацію, обробку API-запитів від клієнтської частини через RESTful endpoints, індексацію завантажених викладачами навчальних матеріалів у векторну базу даних, а також генерацію відповідей на основі rule-based пошуку (точний або нечіткий збіг з еталонними питаннями) та семантичного пошуку з використанням RAG-підходу.

Клієнтська частина являє собою веб-застосунок, що виконується в браузері користувача. Вона реалізована з використанням HTML5, CSS3 та JavaScript без застосування сторонніх фреймворків, що забезпечує легкість розгортання та високу швидкодію, що є необхідним під час відсутності інтернет-з'єднання. Ключовою особливістю клієнтської частини є наявність Service Worker, який забезпечує кешування статичних ресурсів та можливість роботи в офлайн-режимі. Крім того, клієнтська частина використовує локальне сховище браузера для збереження кешованої копії бази питань-відповідей, що дозволяє виконувати пошук навіть за відсутності інтернет-з'єднання.

Система складається з двох незалежних, але інтегрованих підсистем, які доповнюють одна одну:

- правило-орієнтована (rule-based) підсистема базується на точному або нечіткому зіставленні запитань з еталонною базою пар «питання-відповідь». Використовує бібліотеку difflib (SequenceMatcher) для

обчислення коефіцієнта подібності рядків. Ця підсистема є основною для відповідей на типові, заздалегідь внесені викладачем організаційні питання;

- семантична підсистема (RAG) – використовує методи векторного представлення тексту (Sentence Transformers) та пошуку найближчих сусідів (ChromaDB) для знаходження відповідей у довільних документах, завантажених викладачем. Ця підсистема активується тільки якщо rule-based компонент не знайшов відповіді.

Взаємодія між компонентами системи відбувається за наступною схемою. Користувач через веб-інтерфейс надсилає запит на сервер за допомогою HTTP-запиту. Сервер отримує запит, визначає поточний предмет та спочатку викликає правило-орієнтовану підсистему для пошуку відповіді в базі питань-відповідей. Якщо правило-орієнтований пошук знаходить відповідь, результат негайно повертається клієнту. Якщо ж відповідь не знайдено, сервер викликає семантичну підсистему, яка виконує пошук у завантажених документах та формує відповідь на основі найбільш релевантних знайдених фрагментів. У разі, коли жоден з компонентів не зміг знайти відповідь, система повертає повідомлення про відсутність інформації з пропозицією перефразувати питання або звернутися до викладача.

Для забезпечення модульності та полегшення подальшого розширення системи, кожен компонент винесено в окремий файл:

- app.py – головний файл застосунку, що містить маршрути Flask та API-ендпоінти;
- model.py – реалізація rule-based QA-системи (QASystem) та управління користувачами;
- rag_system.py – реалізація RAG-системи (RAGSystem) з підтримкою обробки файлів, векторизації та семантичного пошуку;
- preprocess.py – допоміжні функції для попередньої обробки тексту (токенізація, видалення стоп-слів);

– service-worker.js – скрипт для кешування статичних ресурсів та забезпечення офлайн-роботи.

Уся конфігурація системи зберігається в текстових файлах структурованого формату, що дозволяє легко редагувати базу питань та додавати нових користувачів без необхідності внесення змін до програмного коду.

3.2 Реалізація основної QA-системи

QA-система призначена для відповідей на типові організаційні питання, заздалегідь внесені викладачем або адміністратором. Вона реалізує правило-орієнтований (rule-based) підхід із використанням алгоритму нечіткого порівняння рядків, що дозволяє знаходити відповіді навіть при незначних відхиленнях у формулюванні запитання. Такий підхід забезпечує високу точність відповіді на питання, які є в базі знань, та повну контрольованість роботи системи.

3.2.1 Принцип роботи QA-системи

Основою роботи QA-системи є алгоритм нечіткого порівняння текстових рядків, реалізований за допомогою бібліотеки difflib. Перед порівнянням текст запиту проходить нормалізацію: всі літери переводяться в нижній регістр, видаляються зайві розділові знаки та пробіли. Це дозволяє зменшити вплив випадкових відмінностей у написанні. Програмний код реалізації функції нормалізації тексту поданий у лістингу 3.1.

Лістинг 3.1 – Функція нормалізації тексту

```
def normalize_text(self, text):  
    if not text:  
        return ""  
    text = text.lower().strip()
```

Продовження лістингу 3.1

```
text = re.sub(r'[.!?,:;()"\']$', '', text)
text = re.sub(r'\s+', ' ', text)
return text
```

Поріг схожості встановлено на рівні 0.7, якщо коефіцієнт подібності між запитом користувача та еталонним питанням (або одним із його варіантів) перевищує цей поріг, питання вважається знайденим. Такий підхід дозволяє коректно обробляти незначні відхилення у формулюваннях, описки та синонімічні заміни. Значення порогу 0.7 було обрано експериментально як оптимальне: він забезпечує достатню чутливість для розпізнавання схожих запитань і водночас запобігає хибним спрацьовуванням.

Якщо відповідь на запит не знайдено, система виконує додаткову перевірку: аналізується наявність у запиті ключових слів, характерних для навчальних питань («що таке», «поясни», «чому», «алгоритм», «метод», «функція», «клас», «змінна», «інтеграл», «похідна», «матриця», «теорема» тощо). У разі виявлення таких слів система повідомляє користувача, що це навчальне питання виходить за межі її компетенції, і рекомендує звернутися до викладача. Це дозволяє чітко розмежувати організаційні питання (на які система може відповісти) та навчальні (які потребують участі викладача). Такий механізм є важливим, оскільки система не призначена для виконання функцій викладача, а лише для допоміжної підтримки освітнього процесу.

3.2.2 Структура зберігання даних

Усі питання-відповіді зберігаються у файлі data.json. Кожен запис має наступну структуру:

- унікальний ідентифікатор питання;
- назва предмета, до якого належить питання;

- еталонний текст питання;
- текст відповіді;
- масив альтернативних варіантів формулювань питання;
- статус активності (активне/неактивне);
- ідентифікатор автора (викладача, який додав питання);
- дата та час створення.

Така структура дозволяє гнучко керувати базою знань: для кожного питання можна додати кілька варіантів формулювань, що розширює можливості пошуку без необхідності створювати окремі записи. Статус активності дає змогу тимчасово приховувати застарілі питання без їх фізичного видалення, що корисно в разі зміни організаційних умов у семестрі.

Викладачі можуть додавати, редагувати та видаляти питання через адміністративну панель, причому кожна зміна автоматично зберігається у файл. Для роботи з даними передбачено відповідні методи: додавання нового питання (з автоматичним призначенням унікального ідентифікатора), оновлення існуючого (з перевіркою прав доступу) та видалення з подальшою перенумерацією ідентифікаторів для підтримання цілісності даних.

3.2.3 Автентифікація та розмежування прав доступу

Система підтримує два типи користувачів: адміністратор та викладач. Дані користувачів зберігаються у структурованому текстовому форматі з хешованими паролями із застосуванням алгоритму SHA-256, що забезпечує безпеку зберігання облікових даних. Адміністратор має повний доступ до всіх питань: він може створювати, редагувати, видаляти та змінювати будь-яке питання незалежно від предмета та автора.

Викладач, у свою чергу, може створювати нові питання (при цьому йому автоматично присвоюється авторство), редагувати та видаляти лише

ті питання, де він є автором, а також переглядати питання інших викладачів для ознайомлення. Загальні питання, створені адміністратором, можуть редагуватися тільки адміністратором, що гарантує цілісність базової інформації про навчальний процес. Перевірка прав доступу виконується на сервері для кожної операції редагування або видалення, що унеможливорює несанкціонований доступ навіть при спробі прямого звернення до API.

3.3 Реалізація RAG-компонента

Для роботи з довільними навчальними документами (лекціями, методичними вказівками, презентаціями) було реалізовано RAG-компонент. Він дозволяє викладачам завантажувати документи у форматах PDF, DOCX та TXT, які автоматично індексуються та стають доступними для семантичного пошуку. RAG-компонент активується тільки в разі, якщо правило-орієнтована підсистема не змогла знайти відповідь на запит користувача.

3.3.1 Архітектура RAG

Клас RAGSystem інкапсулює всю логіку роботи з документами. При ініціалізації завантажувється модель ембедінгів all-MiniLM-L6-v2 з бібліотеки sentence-transformers, яка працює повністю локально, без необхідності доступу до інтернету. Також створюється підключення до векторної бази даних ChromaDB, яка зберігає всі індекси на диску. Такий підхід забезпечує автономність роботи системи навіть після перезавантаження сервера.

Система підтримує наступні типи файлів:

- PDF, витягування тексту за допомогою бібліотеки PyPDF2;
- DOCX, витягування тексту за допомогою бібліотеки python-docx;
- TXT, читання тексту з підтримкою кодувань UTF-8 та cp1251.

Кожен завантажений файл зберігається у папці uploads, а його текстовий вміст – у векторній базі ChromaDB.

3.3.2 Розбиття тексту на фрагменти та індексація

Оскільки більшість моделей обробки природної мови мають обмеження на довжину вхідного тексту (для використовуваної моделі ембедінгів – до 512 токенів), довгі документи не можуть бути оброблені цілком. Тому текст необхідно розбивати на менші фрагменти – цей процес називається chunking (рисунок 3.2).

У системі реалізовано наступний підхід до розбиття: спочатку текст розділяється на речення, після чого речення групуються у фрагменти розміром до 500 символів. Для збереження контексту на межах фрагментів передбачено перекриття між сусідніми фрагментами розміром 100 символів. Кожен фрагмент зберігається в базі разом з метаданими, які включають назву файлу та порядковий номер фрагмента.

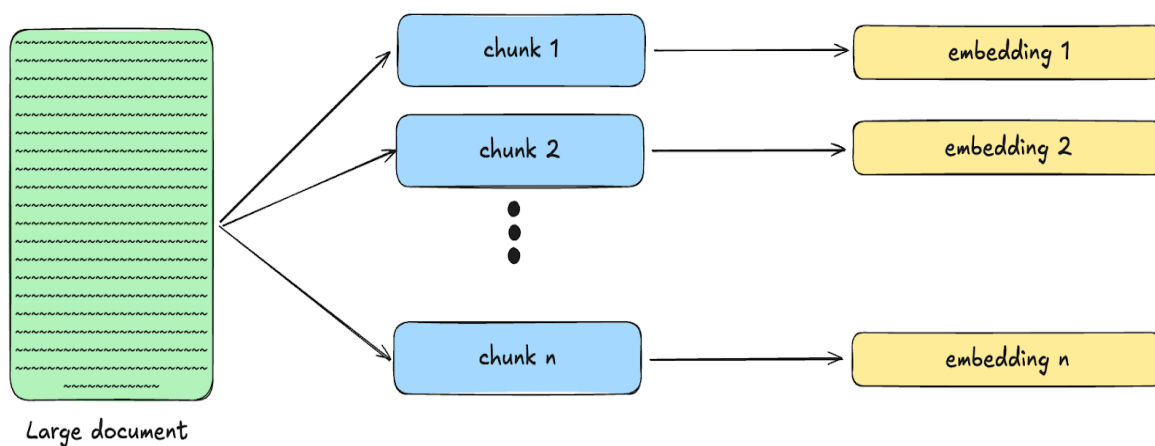


Рисунок 3.2 – Процес chunking

Програмний код реалізації функції розбиття тексту на фрагменти, поданий у лістингу 3.2.

Лістинг 3.2 – Функція розбиття тексту на фрагменти

```
def split_text_into_chunks(self, text: str, chunk_size: int
= 500, overlap: int = 100) -> List[str]:
    if not text:
        return []
    sentences = re.split(r'[.!?]+', text)
    sentences = [s.strip() for s in sentences if s.strip()]
    chunks = []
    current_chunk = ""
    for sentence in sentences:
        if len(current_chunk) + len(sentence) < chunk_size:
            current_chunk += sentence + ". "
        else:
            if current_chunk:
                chunks.append(current_chunk.strip())
                words = current_chunk.split()
                overlap_text = " ".join(words[-overlap:]) if overlap > 0
                and len(words) > overlap else ""
                current_chunk = overlap_text + " " + sentence + ". "
            if current_chunk:
                chunks.append(current_chunk.strip())
    return chunks
```

Після розбиття кожен фрагмент перетворюється на векторне представлення (ембедінг). Отримані вектори разом з текстом фрагмента та метаданими зберігаються у векторній базі ChromaDB. Індексація виконується один раз при завантаженні документа, а при видаленні документа відповідні вектори автоматично видаляються з бази.

3.3.3 Семантичний пошук

Для пошуку найбільш релевантних фрагментів за запитом користувача система виконує наступні кроки:

- отриманий запит перетворюється на векторне представлення за допомогою тієї самої моделі ембедінгів;
- виконується пошук у векторній базі ChromaDB для знаходження топ-5 найбільш семантично близьких векторів;
- використовується косинусна подібність як метрика близькості;
- знайдені фрагменти повертаються разом з назвами файлів-джерел та коефіцієнтами схожості.

Такий підхід дозволяє знаходити релевантну інформацію навіть тоді, коли запит користувача не збігається дослівно з текстом документа, але є близьким за змістом.

3.3.4 Автоматична генерація питань

Для мінімізації ручної праці викладача та оперативного наповнення первинної бази знань (rule-based компонента) інтелектуальна QA-система містить підсистему автоматичної генерації пар «запитання-відповідь» на основі завантажених неструктурованих навчальних матеріалів. Цей модуль реалізує парадигму synthetic data generation і виконує функцію інтелектуального бутстрапінгу бази знань для нових або оновлених навчальних курсів.

Процес автоматичного синтезу знань спроектовано як дворівневу процедуру з деградацією функціоналу (graceful degradation) у разі виникнення інфраструктурних обмежень.

Основний режим – хмарна генерація за допомогою генеративних моделей (Gemini API).

За наявності стабільного Інтернет-з'єднання та дійсного ключа доступу (API key) система використовує потенціал великих мовних моделей (LLM) для глибокого семантичного аналізу тексту. Процедура інтелектуальної генерації складається з таких етапів:

– препроцесинг та контекстне вікнування. Оскільки обробка великих документів у реальному часі може створювати надмірне навантаження та затримки (latency), обсяг вхідного текстового масиву обмежується першими 4000 символами, що еквівалентно найбільш інформативній вступній або організаційно-методичній частині документа (наприклад, syllabus курсу, перші розділи методичних вказівок);

– семантичний промптинг (Prompt Engineering). До Gemini API надсилається спеціалізований системний запит (prompt), який інструктує модель діяти в ролі експерта-методолога. Модель аналізує витягнутий текст і фокусується на виявленні: організаційних дедлайнів, умов колоквиумів та критеріїв оцінювання, ключових термінів, концептів та фундаментальних питань предметної галузі, типових процедурних запитань, які найчастіше виникають у студентів під час опанування дисципліни;

– форматування та валідація. API повертає структурований масив даних у форматі JSON, що містить синтезовані пари когнітивно точних запитань та вичерпних відповідей на них;

– інтеграція в базу знань. Згенеровані об'єкти автоматично десеріалізуються на стороні сервера та імпортуються до локального файлу data.json. Це дозволяє надалі обробляти ці запити за допомогою швидкодіючої правило-орієнтованої підсистеми з використанням алгоритмів нечіткого порівняння рядків (fuzzy string matching) із лінійним порогом схожесті \$0.7\$, суттєво економлячи обчислювальні ресурси при повторних запитах.

Резервний автономний режим – евристичний синтаксичний аналіз (Rule-based Templates).

В умовах аварійного відключення електромереж, відсутності Інтернет-з'єднання (офлайн-режим) або блокування доступу до хмарних сервісів система демонструє стійкість архітектури і перемикається на локальний резервний алгоритм.

Замість генеративної моделі запускається локальний синтаксичний парсер, який функціонує за правилами регулярних виразів (Regex) та шаблонів:

- евристичний аналіз структурних маркерів. Алгоритм сканує текст на предмет наявності специфічних синтаксичних конструкцій. Наприклад, виділяються рядки, що містять двокрапку («:»), які в навчальних текстах часто відокремлюють дефініцію від терміна або умову від її виконання (наприклад, «Дедлайн для ПЗ-1: 15 жовтня» або «Програмне забезпечення: Python 3.10»);

- генерація на основі лінгвістичних шаблонів: Виявлені компоненти трансформуються в прості запитання за допомогою статичних шаблонів підстановки. Рядок виду [Концепт]: [Опис] трансформується в пару: запитання: «Що таке [Концепт]?» або «Які параметри має [Концепт]?» та відповідь: «[Опис]»;

- збереження автономності. Попри меншу лінгвістичну гнучкість порівняння з LLM, цей підхід повністю виконує інженерну вимогу відмовостійкості (fault tolerance). Наповнення бази знань не зупиняється, система залишається на 100% функціональною локально на машині користувача (викладача або адміністратора), забезпечуючи безперервність підготовки системи до взаємодії зі студентами.

3.4 Веб-інтерфейс користувача

Веб-інтерфейс системи SmartQA забезпечує зручну взаємодію користувачів з системою через браузер. Він складається з декількох основних сторінок: головна сторінка з чатом для студентів, адміністративна панель для керування питаннями-відповідями, панель управління RAG-компонентом для роботи з документами, а також сторінки автентифікації та реєстрації. Усі сторінки мають адаптивний дизайн, що забезпечує коректне відображення на різних пристроях.

3.4.1 Чат для студентів

Головна сторінка системи є чат-інтерфейсом, через який студенти можуть ставити запитання та отримувати відповіді (рисунок 3.3). У верхній частині сторінки розташований перелік доступних предметів, що дозволяє користувачу обирати контекст для пошуку.

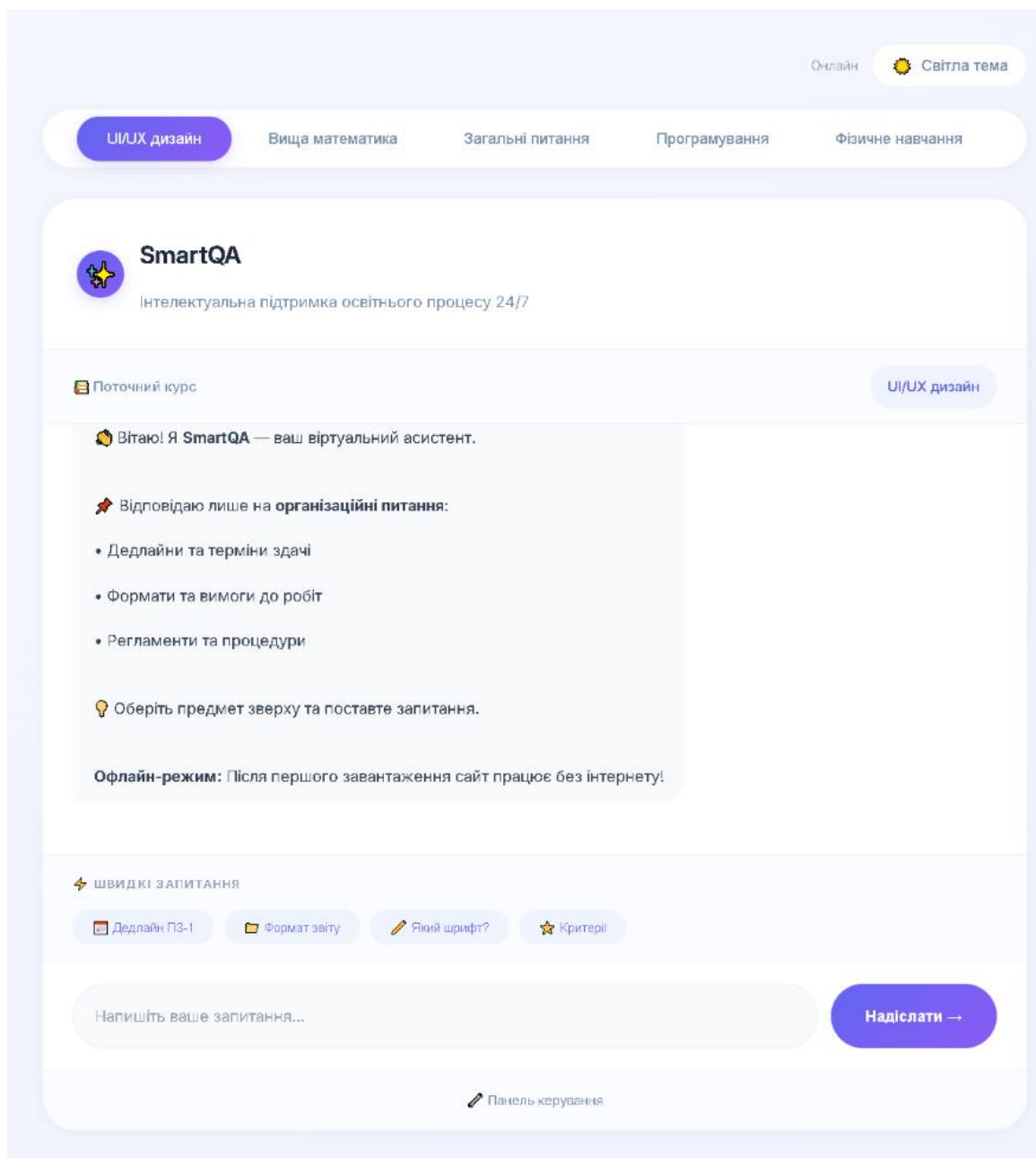


Рисунок 3.3 – Головна сторінка

Користувач обирає предмет з переліку доступних курсів, після чого вводить запитання у текстове поле. Система надсилає запит на сервер, який виконує пошук відповіді спочатку в rule-based базі, а за необхідності – в RAG-компоненті.

Чат підтримує відображення історії повідомлень протягом поточної сесії, що дозволяє студенту відстежувати послідовність запитань та відповідей. Інтерфейс також має індикатор набору тексту, який повідомляє користувача про те, що система обробляє запит. У разі відсутності відповіді в обох підсистемах, система повертає повідомлення з пропозицією перефразувати запитання або звернутися до викладача.

Для зручності користувачів реалізовано автодоповнення при введенні запитань – підказки формуються на основі наявних питань та їх варіантів з бази даних (рисунок 3.4).

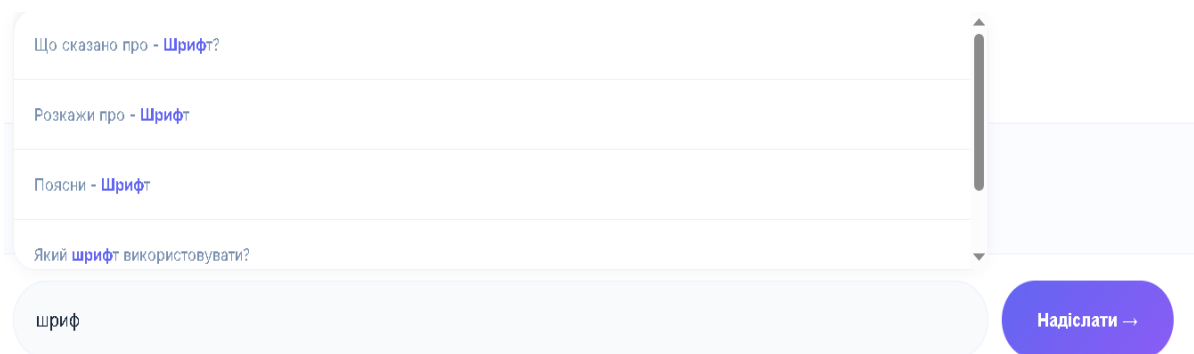


Рисунок 3.4 – Автодоповнення при введенні запитів

Інтерфейс підтримує перемикання між світлою та темною темою оформлення. При натисканні на відповідну кнопку змінюються значення CSS-змінних, які визначають кольори фону, тексту, елементів інтерфейсу та повідомлень. Вибір теми зберігається в локальному сховищі браузера, тому при наступному відвідуванні сторінка автоматично завантажується в обраній темі (рисунок 3.5).

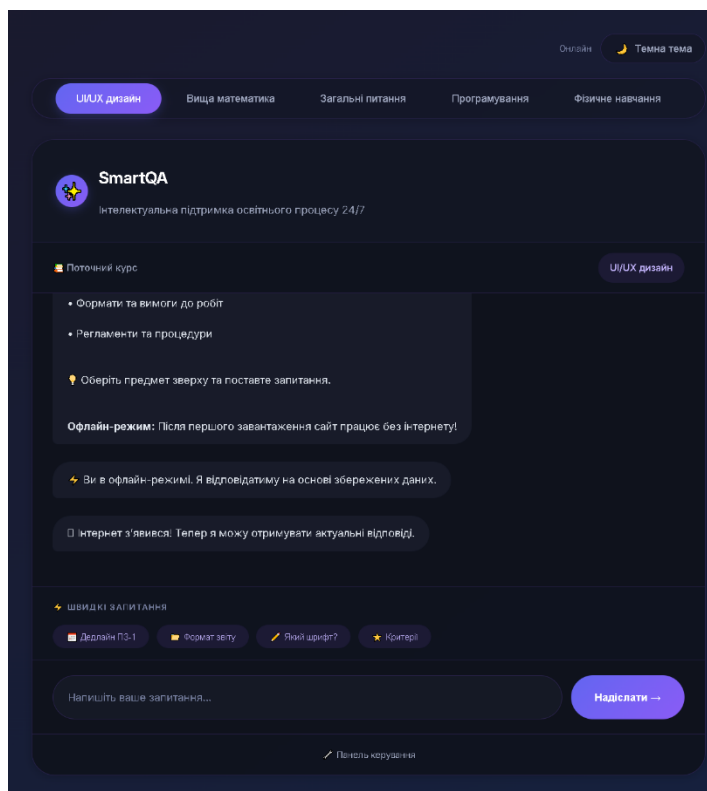


Рисунок 3.5 – Темна тема оформлення

Особливістю головної сторінки є підтримка офлайн-режиму (рисунок 3.6).

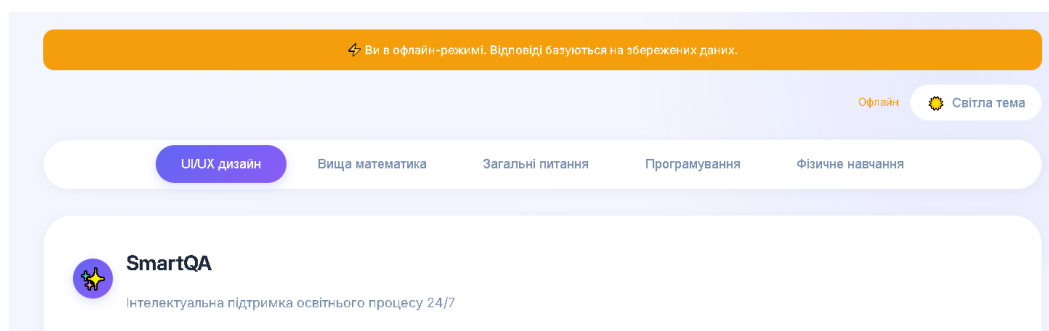


Рисунок 3.6 – Робота в офлайн-режимі

Якщо інтернет-з'єднання відсутнє, система автоматично перемикається на пошук відповідей у кешованій локальній копії бази

питань-відповідей. Користувач бачить відповідний індикатор статусу підключення.

3.4.2 Адміністративна панель

Основна адміністративна панель призначена для керування базою питань-відповідей (рисунок 3.7).

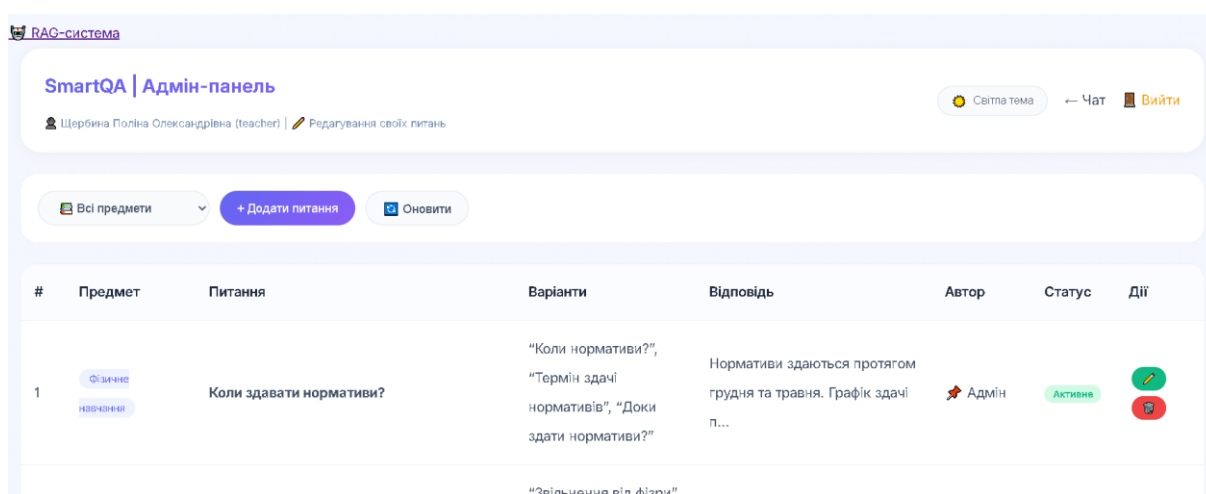


Рисунок 3.7 – Основна адміністративна панель

Вона має табличне подання даних із можливістю фільтрації за предметом, додавання нових питань, редагування та видалення існуючих. При редагуванні або створенні питання відкривається модальне вікно з формою, яка містить поля для введення предмета, еталонного питання, відповіді та списку варіантів формулювань (мінімум три). Варіанти можна динамічно додавати та видаляти. Крім того, передбачено прапорець для активації або деактивації питання, що дозволяє тимчасово приховувати застарілі запитання без їх видалення.

RAG-адміністративна панель додатково надає можливість завантажувати навчальні матеріали у форматах PDF, DOCX та TXT,

переглядати список завантажених документів, видаляти їх та виконувати тестовий семантичний пошук (рисунок 3.8).

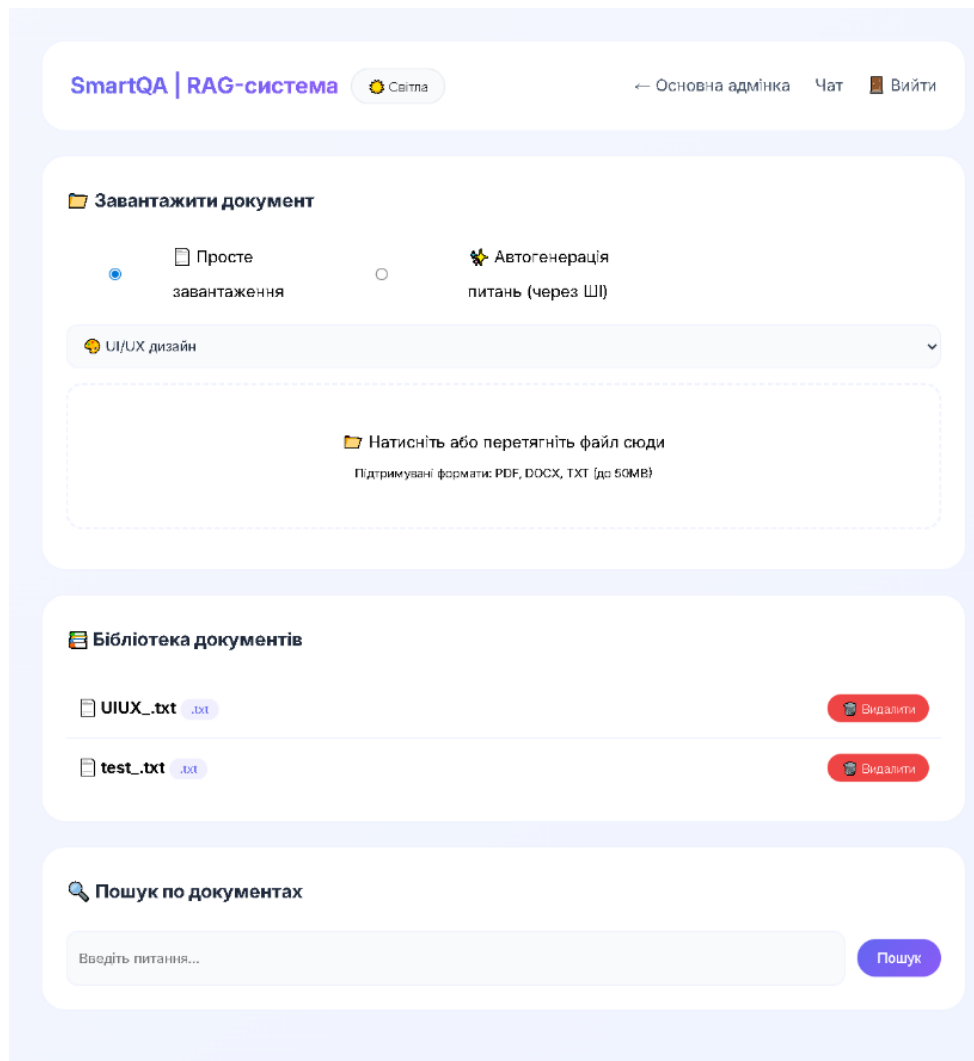


Рисунок 3.8 – RAG-адміністративна панель

Розроблений веб-інтерфейс забезпечує інтуїтивно зрозумілу взаємодію з усіма компонентами системи, не вимагаючи від користувача спеціальних технічних знань. Поєднання чату для студентів, адміністративних панелей для викладачів та підтримки офлайн-режиму робить систему SmartQA універсальним інструментом для автоматизації відповідей на організаційні запитання в умовах нестабільної інфраструктури.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено проблему стійкої підтримки освітнього процесу в умовах критичної нестабільності інфраструктури. Особливу увагу було приділено можливості використання сучасних технологій штучного інтелекту для автоматизації відповідей на запитання студентів на основі завантажених викладачем навчальних матеріалів.

Під час наукового дослідження було проаналізовано предметну галузь, визначено основні проблеми, що виникають при дистанційному навчанні в умовах перебоїв електропостачання, падіння серверів дистанційного навчання та асинхронного доступу студентів і викладачів до ресурсів.

Було проведено порівняльний аналіз існуючих рішень (ChatGPT, Claude, Moodle, Google Classroom), визначено їхні переваги та недоліки. Виявлено, що більшість практичних рішень не забезпечують роботи із закритими університетськими даними, прив'язки відповіді до конкретного джерела та автономної роботи без доступу до інтернету. Зокрема, генеративні системи повністю залежать від інтернету та мають ризик недостовірних відповідей («галюцинацій»), а системи управління навчанням не забезпечують автоматичних відповідей та залежать від центрального сервера.

У рамках теоретичного дослідження було розглянуто чотири основні підходи до побудови QA-систем: правило-орієнтовані, інформаційно-пошукові, генеративні та гібридні. Обґрунтовано, що для створення освітньої QA-системи в умовах нестабільної інфраструктури найбільш оптимальним є гібридний підхід RAG, який поєднує переваги інформаційного пошуку (робота з довільними документами, прив'язка до джерел) та генеративних моделей (природне формулювання відповіді) при мінімізації ризику галюцинацій.

Детально проаналізовано методи обробки природної мови: токенизацію, лематизацію, векторне представлення тексту (ембедінги) та семантичний аналіз. Розглянуто алгоритми пошуку релевантних фрагментів: TF-IDF (з наведенням математичної формули), BM25 як вдосконалення TF-IDF з нормалізацією довжини документа, косинусну подібність для порівняння векторних представлень, а також алгоритми наближеного пошуку найближчих сусідів HNSW та IVF. Досліджено архітектуру RAG, яка складається з трьох етапів: індексація, пошук та генерація. Визначено ключові переваги RAG для освітніх систем: прив'язка відповіді до джерела, можливість роботи із закритими даними, актуальність відповідей, зменшення ризику галюцинацій та можливість локального розгортання.

Крім того, розглянуто методи роботи з навчальними матеріалами різних форматів (PDF, DOCX, TXT) та підходи до забезпечення стійкості системи в умовах нестабільної інфраструктури, зокрема автономність, локальне зберігання даних, кешування, синхронізацію та клієнт-серверну архітектуру.

По завершенню роботи було розроблено концепцію інтелектуальної надбудови над платформою дистанційного навчання, що функціонує автономно, не залежить від стану основного сервера та має можливість локального зберігання всіх необхідних даних. Розроблено програмний прототип системи SmartQA, який реалізує правило-орієнтовану підсистему на основі нечіткого порівняння рядків з порогом схожості 0.7 для відповідей на типові організаційні питання. Ця підсистема використовує базу даних, де кожне питання має еталонний текст, масив альтернативних варіантів формулювань, відповідь та ідентифікатор автора.

Також реалізовано RAG-компонент для семантичного пошуку в завантажених документах з використанням ChromaDB та моделі sentence-transformers, яка працює повністю локально. Система підтримує завантаження файлів у форматах PDF, DOCX та TXT, розбиття тексту на

фрагменти (chunking) з перекриттям для збереження контексту, а також автоматичну генерацію питань-відповідей через резервний режим на основі шаблонів. Система забезпечує зручний веб-інтерфейс із чатом для студентів (автодоповнення, дві кольорові теми, індикатор статусу підключення), адміністративними панелями для керування питаннями та документами, а також підтримку офлайн-режиму.

Ключовою особливістю системи є підтримка офлайн-режиму. За відсутності інтернет-з'єднання система автоматично перемикається на офлайн-режим, виконуючи пошук відповідей у локальному кеші. Крім того, реалізовано автентифікацію користувачів з хешуванням паролів та розмежування прав доступу: адміністратор має повний доступ до всіх питань, викладач – лише до власних.

Узагальнюючи, дослідження, проведене в рамках кваліфікаційної роботи, дозволило поглиблено вивчити тему інтелектуальних QA-систем в освіті, визначити оптимальні підходи до забезпечення стійкості системи в умовах нестабільної інфраструктури та реалізувати діючий програмний прототип. Розроблена система може бути впроваджена в навчальний процес Харківського національного університету радіоелектроніки для забезпечення безперервної підтримки студентів навіть за відсутності стабільного доступу до інтернету. Система дозволяє значно скоротити час пошуку відповідей на організаційні запитання, знизити навантаження на викладачів та підвищити доступність освітньої інформації в кризових умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Asante M. O., Liyanapathirana S. W. S., Pinheiro R. Public service resilience in a post-covid-19 world: digital transformation in nordic higher education. *The impact of covid-19 on the institutional fabric of higher education*. Cham, 2023. P. 245–267. URL: https://doi.org/10.1007/978-3-031-26393-4_10 (date of access: 10.06.2026).
2. The Difference between Emergency Remote Teaching and Online Learning / C. B. Hodges et al. *Handbook of Research in Online Learning*. 2024. P. 511–522. URL: https://doi.org/10.1163/9789004702813_021 (date of access: 10.06.2026).
3. Hurzhy T., Hynova O. Leveraging LLMS for automated test and glossary generation in education. Proceedings of the *International Workshop "Chatbot, Gaming & AI Techniques Applied in Student Digital Education"* (May, 12-13, 2025, Uzhhorod, Ukraine). Uzhhorod: Uzhhorod National University, 2025. 28-32p. URL: <https://dspace.uzhnu.edu.ua/jspui/handle/lib/74262> (date of access: 10.06.2026).
4. Деякі питання організації дистанційного навчання : Наказ МОН України від 08.09.2020 № 1115 : станом на 28 жовт. 2025 р. URL: <https://zakon.rada.gov.ua/laws/show/z0941-20#Text> (date of access: 10.06.2026).
5. Introduction. *Natural language processing*. 2021. P. 3–24. URL: <https://doi.org/10.1017/9781108332873.002> (date of access: 12.06.2026). (date of access: 11.06.2026).
6. Hirschman L., Gaizauskas R. Natural language question answering: the view from here. *Natural language engineering*. 2001. Vol. 7, no. 4. P. 275–300. <https://doi.org/10.1017/s1351324901002807> (date of access: 11.06.2026).
7. Aggarwal C. C., Zhai C. An introduction to text mining. *Mining text data*. Boston, MA, 2012. P. 1–10. URL: https://doi.org/10.1007/978-1-4614-3223-4_1 (date of access: 11.06.2026).

8. A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions / L. Huang et al. *ACM transactions on information systems*. 2024. URL: <https://doi.org/10.1145/3703155> (date of access: 11.06.2026).

9. John T., Möller A. L. On-Demand internationalization for learning management system moodle. *HCI international 2024 posters*. Cham, 2024. P. 95–102. URL: https://doi.org/10.1007/978-3-031-61953-3_11 (date of access: 11.06.2026).

10. Mollick E. R., Mollick L. Using AI to implement effective teaching strategies in classrooms: five strategies, including prompts. *SSRN electronic journal*. 2023. URL: <https://doi.org/10.2139/ssrn.4391243> (date of access: 13.06.2026).

11. Strengths and weaknesses of llm-based and rule-based NLP technologies and their potential synergies / N. N. Karanikolas et al. *Electronics*. 2025. Vol. 14, no. 15. P. 3064. URL: <https://doi.org/10.3390/electronics14153064> (date of access: 13.06.2026).

12. Kamil M., Çakır D. Advances in transformer-based semantic search: techniques, benchmarks, and future directions. *Turkish journal of mathematics and computer science*. 2025. Vol. 17, no. 1. P. 145–166. URL: <https://doi.org/10.47000/tjmcs.1633092> (date of access: 13.06.2026).

13. Retrieval-Augmented generation for educational application: a systematic survey / Z. Li et al. *Computers and education: artificial intelligence*. 2025. P. 100417. URL: <https://doi.org/10.1016/j.caeai.2025.100417> (date of access: 13.06.2026).

14. Large vocabulary size improves large language models / S. Takase et al. *Findings of the association for computational linguistics: ACL 2025*, Vienna, Austria. Stroudsburg, PA, USA, 2025. P. 1015–1026. URL: <https://doi.org/10.18653/v1/2025.findings-acl.57> (date of access: 13.06.2026).

15. Word embeddings: a comprehensive survey / A. Pak et al. *Computación y sistemas*. 2024. Vol. 28, no. 4. URL: <https://doi.org/10.13053/cys-28-4-5225> (date of access: 12.06.2026). (date of access: 13.06.2026).

16. Щербина П.О., Гриньова О.Є. Розробка QA-системи підтримки дистанційного навчання в умовах нестабільної інфраструктури. *Радіoeлектроніка та молодь у XXI столітті: матеріали 30-го Міжнар.молодіж.форуму, 22-24 квітня 2026 р.* Харків: ХНУРЕ, 2026 Т. 6. С. 138–140.

17. Robertson S., Zaragoza H. The probabilistic relevance framework: BM25 and beyond. *Foundations and trends® in information retrieval*. 2009. Vol. 3, no. 4. P. 333–389. URL: <https://doi.org/10.1561/15000000019> (date of access: 17.06.2026).

18. Pennington J., Socher R., Manning C. Glove: global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, Doha, Qatar. Stroudsburg, PA, USA, 2014. URL: <https://doi.org/10.3115/v1/d14-1162> (date of access: 17.06.2026).

19. The evolution of reranking models in information retrieval: from heuristic methods to large language models / T. Pandit et al. *Communications in computer and information science*. Singapore, 2026. P. 56–67. URL: https://doi.org/10.1007/978-981-95-4788-3_6 (date of access: 17.06.2026).

20. Retrieval-Augmented generation for educational application: a systematic survey / Z. Li et al. *Computers and education: artificial intelligence*. 2025. P. 100417. URL: <https://doi.org/10.1016/j.caeai.2025.100417> (date of access: 12.06.2026). (date of access: 17.06.2026).

21. A survey of large language models / W. X. Zhao et al. *arXiv*. 2023. URL: <https://doi.org/10.48550/arXiv.2303.18223> (date of access: 17.06.2026).