МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ім. Ю. М. Потебні

ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)

(рівень вищої освіти)

на тему **Розробка програмного забезпечення для автоматизації бізнес-процесів**

Виконав: студент 4 курсу, групи 6.1212-пзс

спеціальності 121 Інженерія програмного забезпечення

(код і назва спеціальності)

освітньої програми Програмне забезпечення систем

(код і назва освітньої програми)

О.О. Бєлов

(ініціали та прізвище)

Керівник асистент, д-р філос.(PhD) О.О. Жуков

(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Рецензент директор ТОВ «Дісітел»

П. О. Лютий

(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бєлову Олександру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка програмного забезпечення для автоматизації бізнес процесів

керівник роботи Жуков Олександр Олексійович, д-р філософії (PhD)
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ЗНУ від 23.12.2025 р. № 1830-с

2. Строк подання студентом кваліфікаційної роботи 02.06.2026
3. Вихідні дані бакалаврської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - огляд та аналіз існуючих рішень та аналогів;
   - дослідження технології RAG (Retrieval-Augmented Generation) та особливостей опрацювання нормативних документів;
   - створення програмного продукту та його опис;
   - перелік вимог для роботи програми;
   - розробка висновків та пропозицій.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів магістерської роботи	Примітка
1.	Аналіз предметної області (пожежна безпека, нормативна база)	01.03-10.03	виконано
2.	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником.	09.03	виконано
3.	Огляд та збір літератури (NLP, великі мовні моделі, RAG)	11.03-25.03	виконано
4.	Аналіз програм-аналогів	16.03-28.03	виконано
5.	Дослідження технології RAG та особливостей нормативних документів як корпусу	26.03-08.04	виконано
6.	Порівняння та вибір інструментів (векторне сховище, модель векторизації, LLM)	06.04-12.04	виконано
7.	Проєктування архітектури системи та схеми бази даних	10.04-16.04	виконано
8.	Формування корпусу нормативних документів	13.04-18.04	виконано
9.	Розробка модуля підготовки документів (парсинг, сегментація, метадані)	17.04-25.04	виконано
10.	Розробка модуля індексації та семантичного пошуку (FAISS)	24.04-02.05	виконано
11.	Розробка модуля генерації відповідей із цитуванням джерел	01.05-08.05	виконано
12.	Розробка діалогового агента з інструментами (tool use)	06.05-13.05	виконано
13.	Реалізація Telegram-бота	12.05-17.05	виконано
14.	Реалізація CRM-системи (серверна частина та веб-інтерфейс)	14.05-22.05	виконано
15.	Тестування якості відповідей, аналіз результатів	21.05-27.05	виконано
16.	Рефакторинг коду.	25.05-29.05	виконано
17.	Оформлення пояснювальної записки	23.05-31.05	виконано
18.	Оформлення презентації і отримання рецензії.	28.05-31.05	виконано

Студент _____ _____О.О. Бєлов
Керівник роботи _____ _____О. О. Жуков
Нормоконтроль пройдено
Нормоконтролер _____ _____І.А. Скрипник

АНОТАЦІЯ

Сторінок — 82.

Рисунків — 14.

Джерел — 26.

Бєлов О.О Розробка програмного забезпечення для автоматизації бізнес-процесів: кваліфікаційна робота бакалавра спеціальності 121 Інженерія програмного забезпечення / наук. керівник О.О. Жуков. Запоріжжя : ЗНУ, 2026, 82с.

Обсяг нормативної бази з пожежної безпеки зростає, а ручний пошук вимог забирає чимало часу — звідси й потреба в автоматизації. Втім, напряму прикласти великі мовні моделі до юридично значущих текстів заважають їхні галюцинації, застарілість знань і нездатність до перевірюваного цитування. Актуальність роботи саме в тому, щоб дослідити й практично реалізувати підхід, який ці обмеження знімає. Мета роботи — створити інтелектуальний чат-бот на основі технології RAG для достовірного пошуку та опрацювання нормативних документів з інтеграцією в CRM-систему. Для цього проаналізовано архітектури великих мовних моделей і підхід RAG, досліджено особливості нормативних документів як корпусу, виконано порівняльний аналіз і вибір інструментів, спроєктовано та реалізовано RAG-пайплайн із цитуванням джерел, діалогового агента й інтеграцію з CRM-системою, а також проведено оцінювання якості відповідей. Систему побудовано на власному RAG-пайплайні (PyMuPDF, sentence-transformers, FAISS, Claude API) з обов'язковим цитуванням пунктів документів, діалоговим агентом на виклику інструментів і CRM-інтеграцією через Telegram-бот та вебінтерфейс.

Ключові слова: *RAG, велика мовна модель (LLM), векторна база даних, embeddings, нормативні документи, семантичний пошук, чат-бот, пожежна безпека.*

ABSTRACT

Pages — 82.

Drawings — 14.

Sources — 26.

Bielov O.O. Development of Software for Business Process Automation: qualification thesis of the bachelor of specialty 121 Software Engineering / Science Manager O.O. Zhukov Zaporizhzhia: ZNU, 2026, 82 p.

The growing volume of regulatory documentation in the field of fire safety, combined with the labor-intensive nature of manual requirements search, necessitates automation. At the same time, the direct application of large language models (LLMs) to legally significant texts is constrained by their susceptibility to hallucinations, knowledge staleness, and the inability to provide verifiable citation. The relevance of the work lies in the investigation and practical implementation of an approach that eliminates these limitations. The objective of the thesis is the development of an intelligent chatbot based on RAG technology for reliable search and processing of normative documents with integration into a CRM system. To achieve this objective, the architectures of large language models and the RAG approach are analyzed; the characteristics of normative documents as a corpus are examined; a comparative analysis and selection of tools are performed; a RAG pipeline with source citation, a conversational agent, and CRM system integration are designed and implemented; and response quality evaluation is conducted. The thesis formulates requirements for the RAG pipeline derived from the properties of the normative corpus, including hierarchical structure, cross-references, document versioning, and legal precision. A system is implemented on the basis of a custom RAG pipeline (PyMuPDF, sentence-transformers, FAISS, Claude API) with mandatory citation of document clauses, a tool-calling conversational agent, and CRM integration via a Telegram bot and a web interface.

Keywords: *RAG, large language model (LLM), vector database, embeddings, normative documents, semantic search, chatbot, fire safety.*

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ТЕХНОЛОГІЙ NLP ТА ПІДХОДУ RAG 11
1.1 Розвиток NLP та великих мовних моделей 11
1.2 Обмеження LLM у роботі з нормативними документами 16
1.3 Векторні представлення тексту 19
1.4 Архітектура RAG 22
1.4.1 Етап індексації 23
1.4.2 Етап пошуку 25
1.4.3 Етап генерації 26
1.5 Особливості нормативних документів як корпусу 27
1.6 Висновки до розділу 1 31
2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИБІР ІНСТРУМЕНТІВ 33
2.1 Огляд аналогів 33
2.2 Підходи до побудови RAG-пайплайна: фреймворки та власна реалізація 38
2.3 Порівняння векторних сховищ 40
2.4 Вибір LLM та моделі векторизації з урахуванням української мови 42
2.5 Вибір каналу взаємодії та CRM-інтеграції 44
2.6 Висновки до розділу 2 45
3 РОЗРОБКА ЧАТ-БОТА НА ОСНОВІ RAG 46
3.1 Опис проблеми та вимоги до системи 46
3.2 Архітектура системи 48
3.3 Модуль підготовки нормативних документів 50
3.4 Модуль індексації та семантичного пошуку 54
3.5 Модуль генерації відповідей із цитуванням 56
3.6 Інтеграція з CRM-системою 64
3.7 Тестування якості відповідей 71

3.7.1 Методика експерименту 71
3.7.2 Результати експерименту 73
3.7.3 Обмеження експерименту та інтерпретація 75
3.8 Висновки до розділу 3 76
ВИСНОВКИ 78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 80

ВСТУП

Актуальність теми. Сферу пожежної безпеки в Україні регулює велика кількість документів різного рівня – від Кодексу цивільного захисту до державних стандартів (ДСТУ), будівельних норм (ДБН) і правил пожежної безпеки (НАПБ). Ці акти об’ємні, мають складну ієрархічну будову й раз у раз переглядаються. Щоб відповісти навіть на цілком конкретне запитання — скільки вогнегасників потрібно для приміщення певної площі, — доводиться звірятися одразу з кількома документами. Досвідченому фахівцю це вартує часу, а клієнту без спеціальних знань така робота фактично не під силу.

Поява великих мовних моделей (Large Language Models, LLM) дала змогу отримувати відповіді природною мовою. Та застосовувати їх до юридично значущих текстів напряму не виходить. Моделі схильні до галюцинацій — упевнено видають правдоподібні, але хибні твердження; іхні знання обмежені датою навчання; перевірюваного посилання на першоджерело вони не надають. А в нормативний сфері, де ціна помилки висока, відповідь без посилання на конкретний пункт документа цінності не має.

Розв’язок пропнує технологія генерації з доповненням пошуком (Retrieval-Augmented Generation, RAG). Відповідь тут будується не з «пам’яті» моделі, а з фрагментів реальних документів, відібраних у заздалегідь сформованому корпусі, — тож кожне твердження можна супроводити точним посиланням на документ і сторінку. Якщо доповнити цей механізм діалоговим інтерфейсом і вбудувати його в CRM-систему підприємства, відкривається можливість автоматизувати і консультування клієнтів, і частинку роботи менеджерів. Цим і зумовлена актуальність теми.

Мета роботи — створення інтелектуального чат-бота на основі технології RAG для достовірного пошуку та опрацювання нормативних документів у сфері пожежної безпеки з інтеграцією в CRM-систему підприємства.

Для досягнення мети поставлено такі завдання:

Проаналізувати розвиток мовних моделей та їхні обмеження у застосуванні до нормативних документів;

Дослідити архітектуру RAG і методи підготовки нормативних текстів до індексації(сегментацію, роботу з метаданими);

Виконати порівняльний аналіз інструментів побудови RAG-систем — векторних сховищ, моделей векторизації з урахуваннням української мови та великих мовних моделей — і обгрунтувати вибір;

Спроєктувати архітектуру системи: конвеєр опрацювання документів, серверну частину, діалогового агента, CRM-інтерфейс;

Розробити RAG-пайплайн — парсинг PDF-документів, сегментацію зі збереженням структурних метаданих, векторну індексацію, семантичний пошук і генерацію відповідей з обов'язковим цитуванням джерел;

Реалізувати діалогового агента з інструментами там інтеграцію з месенджером Telegram і CRM-системою;

Провести тестування якості відповідей, зокрема порівняти систему з мовною моделлю без RAG.

Об'єкт дослідження — процес автоматизованого пошуку та опрацювання інформації в корпусах нормативних документів.

Предмет дослідження – методи та засоби побудови RAG- систем для роботи з нормативними документами.

Методи дослідження. У роботі використано аналіз наукової літератури та технічної документації, порівняльний аналіз програми засобів, методи опрацювання природної мови (векторні представлення тексту, семантичний пошук), архітектурне моделювання, а також експериментальне оцінювання якості відповідей.

Практичне значення. Систему доведено до робочого прототипу й упроваджено на підприємстві з продажу протипожежного обладнання. Бот у Telegram консультує клієнтов за нормативними документами (ДСТУ, НАПБ

тощо) із посиланням на конкретні пункти, підбирає обладнання за параметрами об’єкта та оформлює заявки. CRM-система, своєю чергою, дає менеджерам дашборд, ведення заявок і замовлень, каталог товарів та довідковий AI-чат. На практиці це скорочує час на пошук нормативних вимог і зменшує ризик надати клієнтові недостовірну відповідь.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. Перший розділ присвячено технологіям NLP, обмеженням LLM, архітектурі RAG та особливостям нормативних документів як корпусу. У другому розглянуто наявні рішення й обгрунтовано вибір інструментів. Третій описує проєктування, реалізацію та тестування системи.

1 АНАЛІЗ ТЕХНОЛОГІЙ NLP ТА ПІДХОДУ RAG

1.1 Розвиток NLP та великих мовних моделей

Опрацювання природньої мови (Natural Language Processing, NLP) — міждисциплінарна галузь на стику лінгвистики, математики та комп'ютерних наук, мета якої навчити обчислювальні системи розуміти, аналізувати й породжувати людську мову [5]. За кілька десятиліть вона пройшла помітну еволюцію: від наборів жорстко закодованих правил до самонавчальних нейромереж, які «перетравлюють» мільярди слів і розв'язують широке коло мовних задач без додаткової розмітки.

Правилові та статистичні підходи. Спершу в NLP домінував підхід, що спирався на явно задані лінгвістичні правила. Морфологічні аналізатори, граматичні парсери, тематичні словники — усе це фахівці-лінгвісти укладали вручну. Платою за точність була трудомісткість: кожна нова мова чи предметна галузь потребувала окремої роботи, а перенесення на інші домени давалися тяжко. [5].

Наступний етап — статистичне NLP — припав на 1980–1990-ті й змістив акцент на навчання моделей за корпусами текстів. Приховані марковські моделі (HMM), умовні випадкові поля (CRF) та n-грамові мовні моделі навчилися автоматично вилучати закономірності з великих масивів даних. Відтворюваність результатів зросла, залежність від ручної розмітки зменшилася. Проте кожна задача все ще вимагала окремого конструювання ознак (feature engineering), а врахувати широкий контекст такі методи майже не вміли [5].

Дистрибутивна семантика та векторні представлення слів. Теоретичним фундаментом сучасних методів є дистрибутивна семантика з її тезою: значення слова визначається тим, у яких контекстах воно вживається. Звідси й виросли методи векторного представлення (embedding) слів —

щільних числових векторів у багатовимірному просторі, де схожі за змістом слова стоять поруч.

Помітним кроком тут стала робота Mikolov et al. (2013) з архітектурою word2vec — ефективним способом навчати векторні представлення слів на великих корпусах [4]. word2vec несподівано добре вловлював семантичні й синтаксичні аналогії; хрестоматійний приклад — арифметика над векторами, коли vector("king") − vector("man") + vector("woman") ≈ vector("queen"). В основі лежали дві архітектури, Continuous Bag-of-Words (CBOW) та Skip-gram, що дозволяли навчатися на корпусах набагато більших, ніж раніше [4]. Власне, word2vec уперше показав річ, яка згодом стала засадничою: щільні вектори, здобуті самонавчанням на нерозміченому тексті, несуть змістовну семантику і годяться як вхід для найрізноманітніших задач NLP.

Рекурентні мережі та модель кодувальник–декодувальник. Із поширенням глибокого навчання до NLP природно прийшли рекурентні нейронні мережі (RNN). Архітектури на кшталт довгої короткочасної пам'яті (LSTM) та вентильних рекурентних блоків (GRU) опрацьовували текст послідовно, токен за токеном, накопичуючи відомості про попередній контекст у прихованому стані.

На рекурентних мережах виросла парадигма кодувальник–декодувальник (seq2seq). Кодувальник стискав вхідну послідовність у фіксований вектор контексту, а декодувальник із цього вектора розгортав вихідну. У машинному перекладі та підсумовуванні текстів seq2seq дала справді гарні результати [5]. Та була проблема: чим довший вхід, тим гірше фіксований вектор утримував усю потрібну інформацію. До того ж послідовна природа RNN не давала ефективно паралелізувати навчання, а отже — обмежувала масштаб. Саме ці дві проблеми й підштовхнули до пошуку іншої архітектури.

Архітектура Transformer і механізм уваги. У 2017 році Vaswani et al. запропонували архітектуру Transformer, що перевернула підходи до побудови мовних моделей [1]. Її головна ідея — взагалі відмовитися від рекурентних

з'єднань на користь механізму уваги (attention). Завдяки йому будь-який елемент послідовності напряму взаємодіє з будь-яким іншим, хоч би як далеко вони стояли в тексті.

Серцевиною архітектури є масштабований скалярний добуток уваги (scaled dot-product attention):

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) \cdot V$$

де Q — матриця запитів (queries), K — матриця ключів (keys), V — матриця значень (values), d_k — розмірність векторів ключів. Ділення на $\sqrt{d_k}$ стримує надмірне зростання скалярних добутків за великих розмірностей і тим самим стабілізує навчання [1].

Transformer застосовує багатоголову увагу (multi-head attention): операція уваги виконується паралельно в кількох «головах» з власними проєкціями Q, K, V, а результати конкатенуються й лінійно перетворюються. Так модель одночасно вловлює різні типи залежностей — і семантичні, і синтаксичні. Окрім уваги, кожен блок містить позиційне кодування (воно компенсує відсутність рекурентності) та позиційно-незалежні повнозв'язні шари (feed-forward) [1].

Головний козир Transformer проти рекурентних мереж — можливість опрацьовувати всі токени послідовності паралельно. На сучасних прискорювачах (GPU/TPU) це дає величезний виграш у швидкості й відкриває шлях до навчання на значно більших корпусах та до помітно глибших моделей.

Надзвичайно важливою рисою Transformer стала його масштабованість: нарощування параметрів і обсягу навчальних даних стабільно покращувало якість на широкому колі задач, що й створило підґрунтя для наступного покоління мовних моделей.

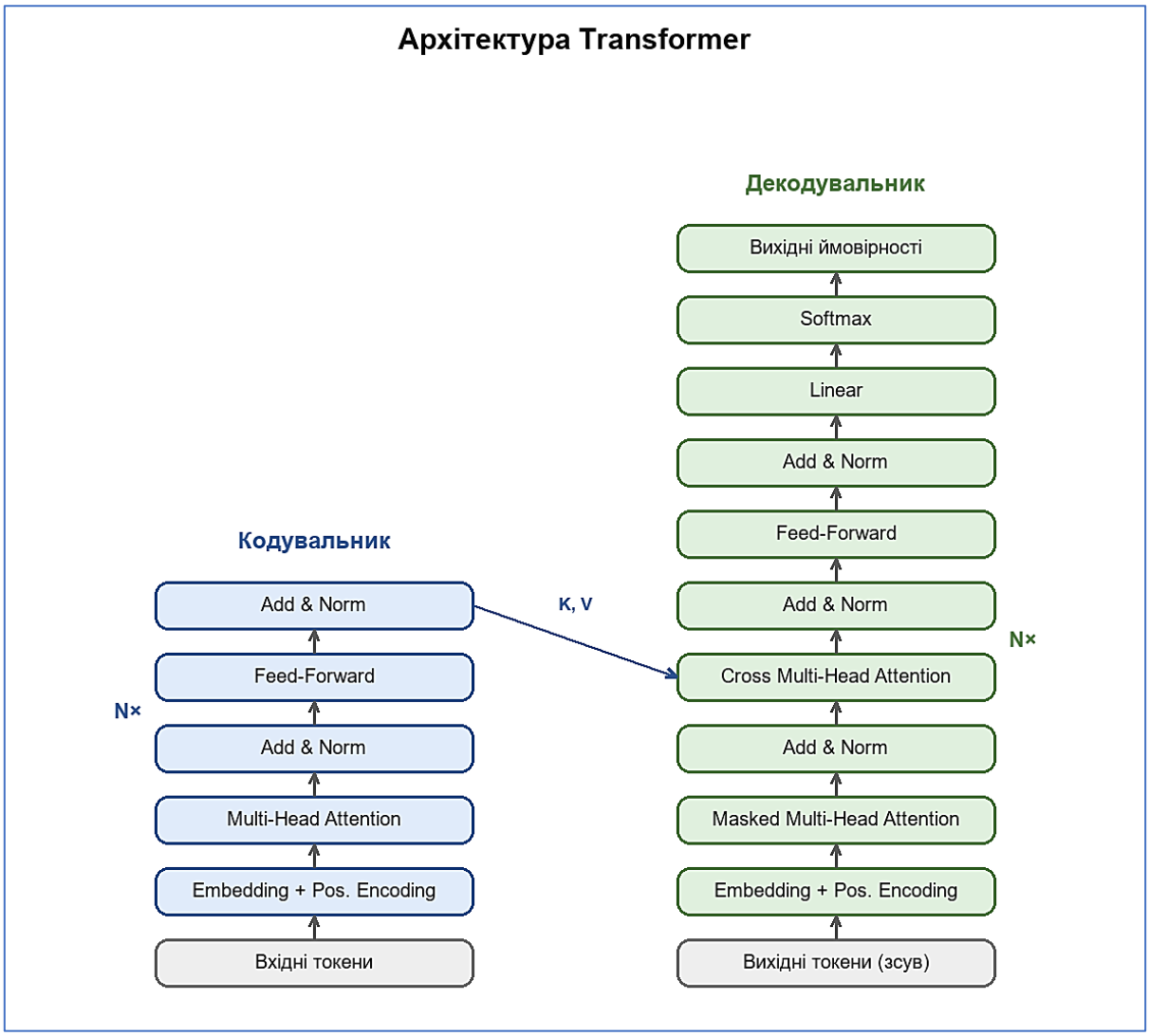


Рисунок *1.1 — Узагальнена схема архітектури Transformer*

BERT і двонапрямне переднавчання. Спираючись на Transformer, Devlin et al. (2019) представили модель BERT (Bidirectional Encoder Representations from Transformers) і разом з нею — нову парадигму «переднавчання (pre-training) → донавчання (fine-tuning)» [2].

Що відрізняло BERT від попередників — двонапрямність переднавчання: модель ураховує контекст токена з обох боків, і лівий, і правий. Досягається це двома цілями навчання — маскованим мовним моделюванням (MLM), де модель відновлює навмисно приховані токени, і передбаченням наступного речення (NSP) [2]. Обидві задачі розв'язуються на

нерозміченому тексті, тож корпус можна брати будь-якого обсягу, без ручної анотації.

Пройшовши переднавчання, модель набуває загального чуття мовних структур, семантики й синтаксису. Далі під конкретну задачу — класифікацію, вилучення іменованих сутностей, відповіді на запитання — достатньо надбудувати кілька шарів і донавчити на порівняно невеликому розміченому наборі [2]. Схема «одна модель на всі задачі через донавчання» виявилася куди економнішою, ніж окрема архітектура під кожну потребу, і швидко стала галузевим стандартом.

Практичне значення цієї парадигми важко переоцінити: вона різко знизила поріг входу. Розробникові більше не треба було збирати велетенський розмічений корпус під кожне нове завдання — і мовні моделі пішли у промислові системи.

GPT-серія та авторегресивна генерація. Поряд із двонапрямними енкодерними моделями розвивалася інша гілка — авторегресивні моделі на декодерній частині Transformer. Серія GPT (Generative Pre-trained Transformer) від OpenAI послідовно масштабувала саме цей підхід.

Суть авторегресії проста: модель передбачає наступний токен за всіма попередніми — $P(w_t \mid w_1, \ldots, w_{t-1})$. Причинна маска (causal masking) не дає увазі «зазирати» в майбутні позиції, чим і зберігає авторегресивний характер генерації. Передавчання відбувається на задачі мовного моделювання, яка теж обходиться без розміченого корпусу [3].

По-справжньому якісний стрибок дала GPT-3, описана в Brown et al. (2020) [3]. Маючи безпрецедентну на той час кількість параметрів, вона показала few-shot навчання: кількох прикладів просто у вхідному запиті (а часом і жодного — варіант zero-shot) вистачало, щоб розв'язати нову задачу без донавчання. Автори показали, що зі зростанням масштабу few-shot ефективність різко підвищується [3]. Це переписало уявлення про можливості мовних моделей і поставило під питання саму потребу в окремих архітектурах під кожну задачу NLP.

Сучасні великі мовні моделі. Далі великі мовні моделі (LLM) навчили слухатися інструкцій користувача, сформульованих вільно. Для цього застосовують інструкційне налаштування (instruction tuning) — донавчання на парах «інструкція → бажана відповідь». А щоб поведінка моделі відповідала людським уявленням про корисність і безпеку, додають навчання з підкріпленням від людських оцінок (RLHF): оцінювачі ранжують відповіді, на цих оцінках навчається модель нагороди, і вона вже спрямовує подальше навчання. На виході — моделі, що ведуть діалог, виконують складні інструкції й відмовляються від некоректних запитів.

Серед сучасних прикладів — GPT-4 (OpenAI) та Claude (Anthropic). Обидві демонструють багатокрокове міркування, роботу з довгим контекстом і розв'язання різнотипних задач у межах однієї архітектури.

Для нормативних і регуляторних документів LLM мають окрему вагу. Вони розбираються у складних юридичних та технічних формулюваннях, вихоплюють ключові норми й вимоги, стискають великі обсяги тексту, відповідають на запитання щодо змісту. Порівняно з пошуком за ключовими словами чи ручним переглядом це інший рівень автоматизації. Та є застереження: щоб користуватися LLM у предметній галузі, потрібні механізми, які гарантують актуальність і достовірність відповідей. Саме їм і присвячено наступні підрозділи.

1.2 Обмеження LLM у роботі з нормативними документами

Здібності великих мовних моделей вражають: вони пишуть зв'язні тексти, відповідають на запитання, узагальнюють документи, розв'язують непрості міркувальні задачі. Але щойно йдеться про галузі, де ціняться точність, відповідальність і перевірюваність — а робота з нормативними, правовими та стандартизаційними документами саме така, — спливає низка принципових обмежень. Без додаткових технічних заходів застосовувати LLM тут не можна.

Проблема галюцинацій. Найглибший недолік LLM — схильність вигадувати інформацію, що звучить переконливо, але не відповідає дійсності. Це явище називають галюцинацією (hallucination). Як зазначено в огляді Ji et al. [6], коріння його — у самій статистичній природі генерації: на кожному кроці модель добирає найімовірніше продовження послідовності токенів, спираючись на засвоєний під час навчання розподіл, а не звіряється з надійним джерелом. Тобто вона відтворює те, що «звучить доречно» в контексті навчальних даних, — і не обов'язково те, що є правдою.

Для нормативної сфери це може дорого коштувати. На запит про вимогу конкретного пункту стандарту модель здатна видати розлогу відповідь із покликанням на пункт, якого або немає в документі, або який насправді містить зовсім іншу норму. А якщо такий висновок перекочує до юридично значущого документа, наслідки — від репутаційних втрат до санкцій за порушення вимог пожежної безпеки.

Застарілість знань. Кожна LLM навчається на даних, зібраних до певної дати, — це її межа знань (knowledge cutoff). Усе, що змінилося в нормативно-правовій базі пізніше, лишається для моделі невидимим. А база ця живе: до законів вносять зміни, органи стандартизації випускають нові редакції. Розрив між черговим перенавчанням великої моделі та актуальним станом норм сягає від кількох місяців до кількох років — і цього досить, щоб параметричні знання моделі стали непридатними для відповіді про чинні вимоги. Та навіть за регулярного перенавчання його вартість і тривалість для сучасних великих моделей лишаються надто високими.

Відсутність вузькогалузевих і внутрішніх документів. Навчальні корпуси LLM складають здебільшого з відкритих текстів — вебсторінок, відкритих книжкових колекцій, наукових публікацій. Натомість чимала частина нормативної бази підприємств і галузей або взагалі закрита, або представлена в мережі вкрай скупо: корпоративні регламенти, внутрішні стандарти, галузеві технічні умови, повні тексти платних національних стандартів. Не «бачивши» цих документів під час навчання, модель не

відтворить їхнього змісту й не зрозуміє як слід запитань, що на них спираються.

Обмежене контекстне вікно. Сучасні LLM працюють у межах контекстного вікна (context window) — граничної кількості токенів, яку можна подати на вхід [3]. Реальний корпус нормативки — це сотні тисяч, а то й мільйони токенів. Умістити все це в один запит технічно неможливо. Та навіть якби вікно дозволяло, якість падає з обсягом: деталі з віддалених частин контексту модель утримує гірше. Тож ідея «завантажити весь корпус у промпт» не працює ні технічно, ні за якістю.

Неможливість перевірюваного цитування. У параметричній пам'яті LLM знання лежать у вигляді вагів нейромережі, а не структурованого сховища документів із метаданими. Через це модель не здатна точно сказати, з якого документа, розділу чи пункту взяте те чи інше твердження. А коли вона все ж наводить посилання, ті можуть бути неточні або й цілком вигадані — ті самі галюцинації, тільки у формі «галюцинованих цитат». Для нормативної сфери це вирок: відповідальний виконавець мусить не просто дати правильну відповідь, а й назвати документ і пункт, на якому ґрунтується вимога. Описані недоліки систематизовані в таблиці. (таблиця 1.1).

Таблиця *1.1 — Обмеження LLM та їхні наслідки для роботи з нормативними документами*

Обмеження	**Причина**	**Наслідок для нормативної сфери**
Галюцинації	Статистична природа генерації токенів	Вигадані посилання на неіснуючі пункти стандартів
Межа знань (knowledge cutoff)	Фіксований час збирання навчального корпусу	Відповіді на основі застарілих норм і вимог
Відсутність галузевих документів	Непублічний характер частини нормативної бази	Неможливість відповісти на запитання щодо галузевої специфіки
Обмежене контекстне вікно	Архітектурні обмеження моделі	Неможливість охопити весь корпус в одному запиті
Відсутність перевірюваного цитування	Параметричний характер пам'яті моделі	Неможливість верифікувати джерело відповіді

Перелічене — не випадкові вади, а системна властивість архітектури LLM. Там, де точність, актуальність і верифікованість відповіді обов'язкові, «чиста» LLM без механізму заземлення (grounding) непридатна.

Підхід, що прямо б'є по цих обмеженнях, — генерація з доповненням пошуком (Retrieval-Augmented Generation, RAG), запропонована Lewis et al. [7]. Замість покладатися на параметричну пам'ять, RAG-система під кожен запит дістає релевантні фрагменти з контрольованого корпусу й подає їх моделі разом із запитом. Так забезпечуються і актуальність, і перевірюваність, і прив'язка відповіді до конкретного джерела. Докладно архітектуру RAG розглянуто в підрозділі 1.4.

1.3 Векторні представлення тексту

Технічна основа, що взагалі робить можливим семантичний пошук у RAG, — векторне представлення (embedding): відображення текстових одиниць у числові вектори, де змістова близькість стає близькістю геометричною. Без розуміння того, як ці представлення утворюються, не спроєктувати добрий модуль пошуку.

Концепція векторного представлення. Векторне представлення — це відображення текстового об'єкта (слова, речення чи більшого фрагмента) у щільний вектор фіксованої розмірності в деякому семантичному просторі. Розріджені представлення на кшталт one-hot encoding чи TF-IDF відводять окрему координату під кожне слово словника; щільні ж вектори розміщують тексти в неперервному просторі так, що схожі за змістом опиняються поруч [5].

Для пошуку це властивість засаднича. Хай запит «вимоги до пожежної сигналізації» і фрагмент «норми встановлення димових сповіщувачів» майже не перетинаються лексично — за змістом вони суміжні, тож і вектори їхні будуть близькі. Власне, цим семантичний пошук і переважає пошук за ключовими словами, який ловить лише буквальні збіги. Геометрію цієї

близькості показано на рисунку 1.2: вектор запиту й вектори релевантних фрагментів утворюють малий кут (висока косинусна подібність), а нерелевантні відхилені на більший кут.

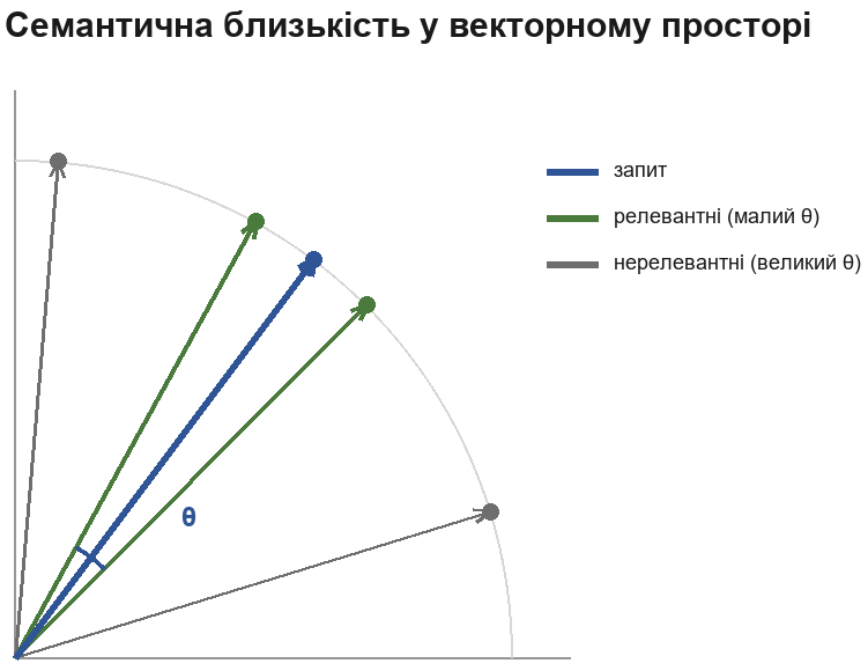


Рисунок *1.2 — Семантична близькість запиту й фрагментів у векторному просторі*

Косинусна подібність. Звична міра близькості двох векторів у семантичному пошуку — косинусна подібність (cosine similarity). Для векторів A та B розмірності n її обчислюють так:

$$\cos(\theta) = (A \cdot B) / (\|A\| \cdot \|B\|) = \Sigma\, A_i \cdot B_i \,/\, (\sqrt{(\Sigma\, A_i^2)} \cdot \sqrt{(\Sigma\, B_i^2)})$$

де чисельник — скалярний добуток векторів, а знаменник — добуток їхніх евклідових норм. Значення лежить у межах від −1 до +1; що ближче до 1, то схожіші тексти.

На практиці важливий один нюанс: якщо вектори заздалегідь нормовані до одиничної довжини ($\|A\| = \|B\| = 1$), знаменник перетворюється на одиницю, і косинусна подібність зводиться до простого скалярного добутку. Останній рахується значно дешевше — а це критично, коли вектор запиту звіряється з тисячами чи мільйонами збережених векторів.

Від векторів слів до векторів речень. Перші вдалі методи векторизації (word2vec [4], GloVe) працювали на рівні окремих слів. Вектор речення з них діставали усередненням векторів слів — і тут крилася проблема: усереднення стирає порядок слів, синтаксис і контекстну залежність значень. Скажімо, «обладнання відповідає нормам» і «обладнання не відповідає нормам» дадуть майже однакові усереднені вектори, хоча зміст протилежний.

Зрушення дала модель Sentence-BERT від Reimers і Gurevych [8]. В її основі — сіамська (siamese) архітектура: дві однакові гілки трансформера незалежно кодують кожне з двох речень, а далі вектори зіставляються через функцію втрат, що враховує семантичну близькість. Так модель навчається будувати щільні вектори речень, які схоплюють зміст цілого висловлювання, а не окремих лексем.

Багатомовні моделі векторизації. Більшість ранніх моделей векторизації орієнтувалися передусім на англійську. Для документів іншими мовами — української зокрема — це обертається відчутною втратою якості: слабко представлена в навчанні мова дістає невиразні вектори з розмитою семантичною геометрією.

Щоб обійти це, з'явився клас багатомовних моделей (multilingual embedding models). Їх навчають одразу на корпусах багатьох мов, вирівнюючи міжмовний семантичний простір так, щоб однакові за змістом речення різними мовами діставали близькі вектори. Один із прикладів — paraphrase-multilingual-MiniLM-L12-v2 із сімейства Sentence-Transformers [13]. Вона дає вектори розмірності 384, підтримує понад 50 мов (українську серед них) і доволі компактна за кількістю параметрів та часом виконання. Невелика розмірність — це менші вимоги до пам'яті й обчислень при зберіганні та

пошуку, що неабияк важить при розгортанні на звичайному серверному обладнанні без графічних прискорювачів.

Семантичний пошук як основа retrieval. Готові вектори фрагментів зберігають у векторних індексах — структурах, заточених під пошук найближчих сусідів у багатовимірному просторі. Коли надходить запит, його тією самою моделлю перетворюють на вектор і шукають фрагменти з найбільшою косинусною подібністю. Це і є серцевина етапу пошуку (retrieval) у RAG: система знаходить потрібні фрагменти нормативних документів навіть тоді, коли формулювання запиту далеке від того, що написано в тексті.

1.4 Архітектура RAG

Retrieval-Augmented Generation (RAG) — архітектурна парадигма, що зводить докупи два різні типи пам'яті. Перший — параметрична пам'ять великої мовної моделі, закодована у вагах під час навчання. Другий — непараметрична пам'ять у вигляді зовнішнього індексу документів, який можна оновлювати окремо від моделі [7]. Така зв'язка знімає головні слабкості суто параметричних систем — насамперед застиглість знань і схильність вигадувати, — лишаючи моделі те, що вона робить добре: синтез, узагальнення й природне формулювання відповіді.

Конвеєр RAG має три послідовні етапи. Індексація (indexing): корпус документів переводиться у вектори й лягає в пошуковий індекс. Пошук (retrieval): запит зіставляється з індексом, відбираються найрелевантніші фрагменти. Генерація (generation): знайдене разом із запитом передається моделі, і та будує підсумкову відповідь, спираючись на контекст. Етапи утворюють ланцюг, де якість кожної наступної ланки залежить від попередньої [11].

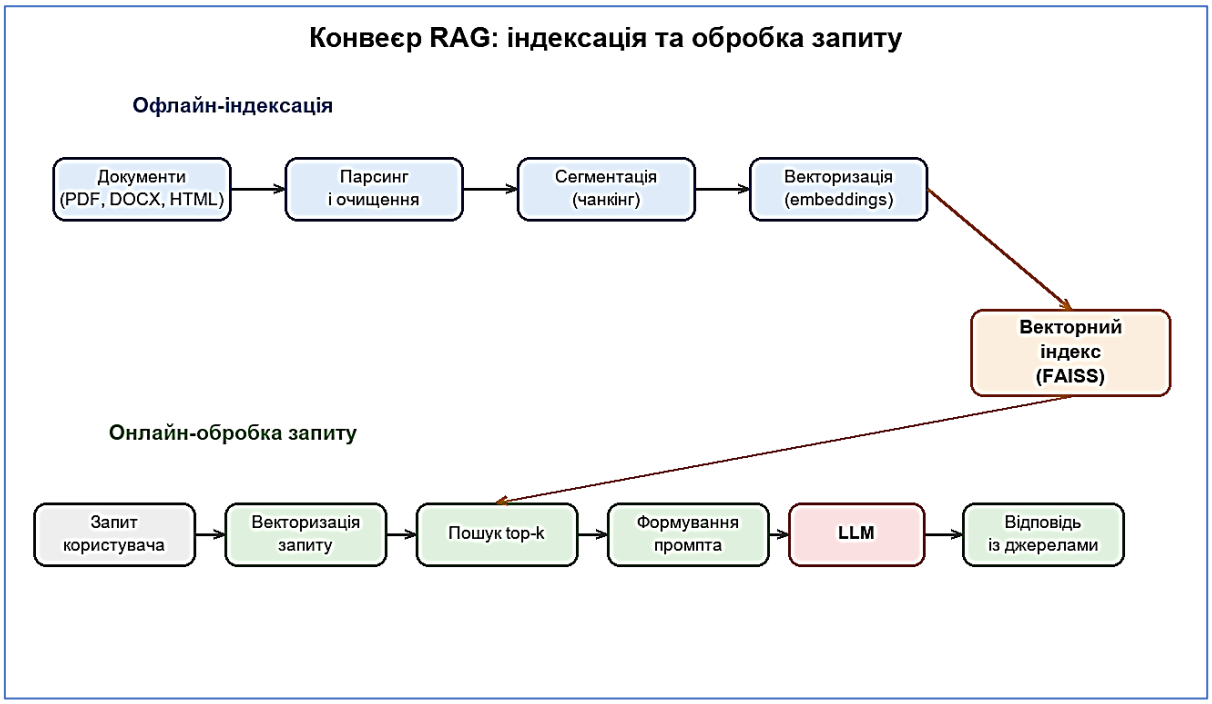


Рисунок *1.3 — Узагальнена схема RAG-конвеєра*

1.4.1 Етап індексації

Індексація — підготовчий етап, що відбувається поза прямою взаємодією з користувачем. Її завдання — перетворити неструктурований чи напівструктурований корпус на пошукову структуру, придатну для семантичних запитів у реальному часі.

Завантаження і парсинг документів. Спершу документи треба завантажити й вилучити з них текст. Реальні корпуси різнорідні: PDF, DOCX, HTML — і кожен формат потребує свого парсера. PDF буває зі складною верстkою — кілька колонок, таблиці, вбудовані зображення, — і лінійно вилучити текст звідти непросто. DOCX несе структурну інформацію у стилях заголовків. HTML вимагає прибрати розмітку й навігаційні елементи. Нормативно-правові акти нерідко виходять водночас у PDF офіційних видань і в HTML на урядових порталах, тож парсери доводиться застосовувати паралельно.

Очищення тексту. Вилучений текст треба почистити від артефактів конвертації: повторюваних колонтитулів і номерів сторінок, що влізають усередину тексту після посторінкового парсингу PDF; дефісів переносу, які

розривають слова; зайвих пробілів і порожніх рядків. Брудний текст псує векторні представлення і породжує фрагменти, де змістовна частина перемішана зі службовою.

Сегментація (чанкінг). Чистий текст ріжуть на менші фрагменти (чанки), і кожен далі індексується та повертається окремо. Потреба в цьому має дві причини: контекстне вікно моделі обмежене й цілий документ у нього не вкласти; до того ж для якісного пошуку фрагмент має бути тематично однорідним, аби його вектор точно відбивав один конкретний аспект.

Стратегій дві. Фіксована сегментація з перекриттям (fixed-size chunking with overlap) ділить текст на шматки заданої довжини зі зсувом, меншим за розмір фрагмента, через що сусідні фрагменти частково накладаються. Вона проста й передбачувана, але байдужа до змісту: межа може розірвати речення, абзац чи логічну одиницю. Структурно-орієнтована сегментація (structure-aware chunking) натомість бере за межі природну ієрархію документа — заголовки розділів, статей, пунктів, — і так зберігає цілісність кожної одиниці [11].

Розмір фрагмента — питання компромісу: завеликий додає шуму в контекст, замалий губить зв'язність і самодостатність. Оптимум зазвичай підбирають емпірично, і він залежить від характеру корпусу [11].

Метадані фрагментів. Під час індексації кожному фрагменту приписують метадані про походження: назву документа-джерела, номер сторінки, ідентифікатор розділу чи пункту. Вони зберігаються поруч із вектором і повертаються при пошуку. Без них не реалізувати цитування джерел у відповідях.

Векторизація і побудова індексу. Сегментовані фрагменти йдуть до моделі векторного представлення, яка обертає кожен на щільний вектор. Вектори разом із метаданими лягають у векторний індекс. Для великих корпусів використовують алгоритми наближеного пошуку найближчих сусідів (ANN): вони помітно дешевші за точний перебір ціною невеликої втрати точності. Поширена відкрита бібліотека для таких індексів — FAISS

(Facebook AI Similarity Search), розрахована на пошук подібності аж до мільярдних колекцій векторів [10].

1.4.2 Етап пошуку

Пошук іде в реальному часі на кожен запит. Завдання — відібрати з індексу невелику кількість фрагментів, найрелевантніших до запиту.

Щільний пошук (dense retrieval). Тут і запит, і документи — щільні вектори в одному семантичному просторі. Запит кодують тією ж моделлю, що й при індексації, і шукають фрагменти, найближчі до нього за мірою подібності. Архітектура бі-енкодера (bi-encoder), де запит і фрагменти кодуються незалежно, дає змогу знаходити близьке за змістом навіть без лексичного збігу [9]. Особливо добре щільний пошук працює на перефразованих чи узагальнених запитах.

Розріджений пошук (sparse retrieval). Цей варіант успадковує класику інформаційного пошуку й оперує лексичними ознаками: запит і документи — розріджені вектори в просторі термінів. Базовий алгоритм — BM25, імовірнісна функція ранжування, що зважує частоту терміна в документі та обернену частоту в корпусі [12]. Розріджений пошук виграє там, де в запиті є специфічні технічні терміни, абревіатури, числові позначення чи власні назви: точний збіг ідентифікатора надійніший за семантичну близькість. Скажімо, шукаючи конкретний номер пункту нормативного акта, лексичний пошук відшукає потрібний фрагмент певніше.

Гібридний пошук. Щільний і розріджений підходи доповнюють один одного — кожен сильний на своєму класі запитів. Гібридний пошук зводить їхні оцінки докупи через злиття результатів; поширений прийом — Reciprocal Rank Fusion (RRF), що об'єднує ранжовані списки без нормалізації вихідних оцінок [11].

Параметр top-k. Скільки фрагментів піде на генерацію, задає гіперпараметр top-k. Більше k — повніший відбір, але занадто велике значення забиває контекст нерелевантним і псує генерацію, «розбавляючи» корисне.

Оптимум залежить від корпусу, довжини фрагментів і розміру контекстного вікна.

Реранжування (re-ranking). Щоб підняти точність, після первинного пошуку інколи додають реранжування. Крос-енкодер (cross-encoder), на відміну від бі-енкодера, опрацьовує пару «запит — фрагмент» разом, моделюючи їхню взаємодію на рівні уваги, — і дає точнішу оцінку релевантності [11]. Коштує він дорожче, тож його напускають не на весь індекс, а лише на невеликий набір кандидатів первинного пошуку.

1.4.3 Етап генерації

Генерація завершує конвеєр: на вхід — відібрані фрагменти й запит, на вихід — природномовна відповідь.

Складання промпта. З моделлю спілкуються через структурований промпт із кількох частин: системна інструкція (роль, обмеження, вимоги до формату), знайдені фрагменти з ідентифікаторами джерел і власне запит користувача. Порядок та оформлення цих частин позначаються на якості відповіді.

Заземлення (grounding). Важливіша складова системної інструкції — директива заземлення: будувати відповідь лише з фрагментів, поданих у контексті. Це головний запобіжник проти галюцинацій у RAG: обмеживши джерело фактів наданим контекстом, система менше ризикує видати правдоподібне, але хибне чи застаріле твердження.

Цитування джерел. Із заземленням нерозривно пов'язане цитування: інструкція велить моделі супроводжувати кожне твердження ідентифікатором фрагмента-джерела — назвою документа і, де є, номером пункту чи сторінки. Відповідь стає перевірюваною: користувач може піти й звіритися з першоджерелом. Для нормативних документів це принципово, адже відповідальність за коректне застосування норм не перекладеш на автоматизовану систему.

Поведінка за відсутності відповіді в контексті. Окремо варто продумати, як модель поводиться, коли в жодному фрагменті немає достатньої інформації. Якщо інструкція цього не обумовлює, модель може потягтися до параметричних знань — і підважити весь принцип заземлення. Правильно налаштована система натомість прямо каже, що відповіді в наявному корпусі немає, і не вигадує тверджень без опори на контекст.

Температура генерації. Температура керує тим, наскільки випадково модель добирає токени. Для фактологічних задач, де треба точно відтворити норму, число чи процедурну вимогу, її тримають низькою — щоб модель радше обрала точне формулювання, ніж його «майже правильний» варіант.

1.5 Особливості нормативних документів як корпусу

Якість RAG-системи залежить не лише від алгоритмів пошуку й генерації, а й чи не насамперед — від властивостей корпусу, з яким вона працює. Нормативні документи — окремий клас текстів, помітно відмінний від загальних вебкорпусів, новинних статей чи документації до програм. У цьому підрозділі ці відмінності розібрано й перетворено на вимоги до конвеєра опрацювання. Аналіз спирається на реальний корпус системи — нормативну базу з пожежної безпеки України (таблиця 1.2).

Таблиця *1.2 — Склад корпусу нормативних документів системи*

Документ	Тип	Роль у корпусі
Кодекс цивільного захисту України (№ 5403-VI від 02.10.2012)	Кодекс (закон)	Рамкові правові засади цивільного захисту та пожежної безпеки
НАПБ А.01.001-2014. Правила пожежної безпеки в Україні	Правила (НАПБ)	Основні правила пожежної безпеки: режим, експлуатація, розміщення засобів пожежогасіння
Наказ МВС № 25 від 15.01.2018. Правила експлуатації та типові	Наказ (НПА)	Норми належності вогнегасників — основа розрахунку їх кількості

норми належності вогнегасників		
Наказ МНС № 338 від 18.05.2009. Правила улаштування та експлуатації систем раннього виявлення НС та оповіщення	Наказ (НПА)	Вимоги до систем раннього виявлення надзвичайних ситуацій та оповіщення
ДБН В.1.1-7-2016. Пожежна безпека об'єктів будівництва. Загальні вимоги	ДБН (будівельні норми)	Протипожежні вимоги до об'єктів будівництва
ДБН В.2.5-64. Внутрішній водопровід та каналізація	ДБН (будівельні норми)	Вимоги до внутрішнього протипожежного водопроводу
ДСТУ 8828-2019. Пожежна безпека. Загальні положення	ДСТУ (стандарт)	Загальні положення та вимоги пожежної безпеки
ДСТУ 2272-2006. Пожежна безпека. Терміни та визначення основних понять	ДСТУ (стандарт)	Термінологічна база предметної області
ДСТУ EN 54-23:2015. Системи пожежної сигналізації. Сповіщувачі пожежні світлові	ДСТУ (гармонізований EN)	Вимоги до світлових пожежних сповіщувачів

Після сегментації (фрагменти по 800 символів із перекриттям 150 символів) корпус налічує 2625 фрагменти. Далі розглянемо ті властивості цих документів, що прямо позначаються на проєктних рішеннях RAG-конвеєра.

Ієрархічна структура. Нормативні документи укладено жорстко й багаторівнево. Стандарти та правила йдуть за схемою «розділ → підрозділ → пункт → підпункт» зі наскрізною нумерацією на кшталт «4.2», «3.18»; закони — «розділ → глава → стаття → частина». Важливо, що ця нумерація не просто навігація, а юридично значущий ідентифікатор: у фаховому обігу вимогу цитують саме за номером пункту чи статті, а не за переказом змісту. Звідси два наслідки для системи. По-перше, структурні одиниці — природні межі

сегментації: фрагмент, що збігається з пунктом або групою суміжних пунктів, є самодостатньою одиницею знання. По-друге, метадані фрагмента мусять нести ідентифікатор структурної одиниці (номер пункту чи статті) і номер сторінки — без них відповідь годі верифікувати, а отже, вона втрачає юридичну цінність.

Перехресні посилання. Документи системно посилаються один на одного: терміни визначаються в одному (ДСТУ 2272:2006 [23]), а вживаються у вимогах інших [20–22]; будівельні норми відсилають до стандартів щодо методів випробувань; правила пожежної безпеки спираються на Кодекс [19]. Повна відповідь на практичне запитання часто складається з фрагментів кількох документів. Класичний RAG-конвеєр частково це покриває: вибірка top-k природно зачерпує фрагменти різних документів корпусу. Повне ж розв'язання — побудова графа зв'язків між структурними одиницями (підхід GraphRAG) — виходить за межі роботи й лишається напрямом подальшого розвитку.

Версійність і чинність. Нормативна база рухлива: до документів вносять зміни окремими актами, документи втрачають чинність і поступаються новим редакціям. Скажімо, чинний текст НАПБ А.01.001-2014 несе позначки про зміни за наказом МВС України від 31.07.2017 № 657 [20] — пункти супроводжуються примітками про перенумерацію підпунктів. Відповідь за нечинною редакцією не просто марна — вона шкідлива, бо створює оманливе відчуття відповідності. Для системи це означає: корпус формуємо лише з чинних редакцій із офіційних джерел; кожне оновлення документа тягне за собою переіндексацію; а на перспективу метадані фрагментів варто доповнити датою чинності редакції.

Юридична модальність і числова точність. Мова нормативних документів чітко розрізняє модальні конструкції: «повинен» і «не допускається» — це обов'язкові вимоги, «слід» — настанова, «рекомендується» — необов'язкове положення, «дозволяється» — дозвіл з умовами. Підмінити модальність (поставити «рекомендується» там, де стоїть «не допускається») означає змінити юридичний зміст вимоги. Так само не терпить вільного переказу числова точність: нормативи кількості, площі, відстаней, тиску мають відтворюватися дослівно. Для генерації звідси жорсткі вимоги до промпта: числові значення й модальні конструкції модель зобов'язана переносити з фрагментів буквально, а температуру слід тримати низькою.

Технічні особливості подання. Офіційні видання нормативних документів ходять переважно у PDF зі складною верстkою: колонтитули з позначенням документа на кожній сторінці, таблиці, формули в додатках. Частина документів існує лише як скановані копії без текстового шару — їх довелося б розпізнавати оптично, і в цій роботі вони не розглядаються (до корпусу взято тільки документи з текстовим шаром). Окрема практична морока — кодування. Під час формування корпусу трапився PDF, де кириличний текст зберігався у Windows-1251, а декодувався як Latin-1; через це вилучений текст вироджувався в беззмістовний потік латинських символів із діакритиками (приміром, заголовок «ПОЖЕЖНА БЕЗПЕКА ОБ'ЄКТІВ» поставав як «ÏÎÆÅÆÍÀ ÁÅÇÏÅÊÀ ÎÁ'ªÊÒ²Â»). Семантично порожні фрагменти індексуються без жодної помилки виконання, але не знаходяться нічим — дефект дається взнаки лише як «тиха» деградація пошуку. Звідси вимога до конвеєра підготовки: автоматичний контроль якості вилученого тексту, найпростіший варіант якого — статистична перевірка частки кириличних символів у видобутому тексті кожного документа.

Мовний аспект. Корпус цілком україномовний, а це вимагає багатомовної моделі векторизації (підрозділ 1.3). Додає складності й специфічна лексика: абревіатури класів документів (ДСТУ, ДБН, НАПБ),

позначення обладнання (наприклад, ВП-6 — вогнегасник порошковий), номери документів. Усе це — точні лексичні ознаки, для яких семантичний пошук менш надійний за точний збіг, що знову схиляє до гібридного (щільного + розрідженого) пошуку як напряму вдосконалення.

Підсумовуючи, властивості нормативних документів задають такі вимоги до RAG-конвеєра:

1) сегментація має зберігати прив'язку кожного фрагмента до структурної одиниці документа та сторінки (метадані: документ, сторінка, пункт);

2) формат відповіді має містити посилання на документ і пункт — без них відповідь юридично знецінена;

3) корпус має складатися лише з чинних редакцій; оновлення документів супроводжується переіндексацією;

4) генерація має дослівно відтворювати числові значення та модальні конструкції; температура — низька; за відсутності відповіді в корпусі — явна відмова;

5) конвеєр підготовки має включати автоматичний контроль якості вилучення тексту (зокрема перевірку кодування);

6) модель векторизації має бути багатомовною з підтвердженою підтримкою української мови.

Саме ці вимоги й визначають проєктні рішення, ухвалені в розділі 3.

1.6 Висновки до розділу 1

У розділі розібрано технологічне підґрунтя роботи. Показано, що сучасні великі мовні моделі на архітектурі Transformer добре розуміють і породжують природну мову, проте мають системні обмеження — галюцинації, застарілість знань, обмежене контекстне вікно й неможливість перевірюваного цитування, — які не дають застосовувати їх до нормативних документів напряму.

Розглянуто RAG як підхід, що ці обмеження знімає, поєднуючи параметричну пам'ять моделі з непараметричним індексом документів: етапи індексації, пошуку й генерації утворюють конвеєр, чия якість тримається на якості кожної ланки. Векторні представлення тексту й косинусна подібність становлять математичну основу семантичного пошуку; для україномовного корпусу потрібна багатомовна модель векторизації.

Головний результат розділу — аналіз нормативних документів як специфічного класу корпусу (підрозділ 1.5): їхня ієрархічна структура, перехресні посилання, версійність, юридична модальність і технічні особливості подання зведені у шість вимог до RAG-конвеєра, що й лягають в основу проєктних рішень розділу 3.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИБІР ІНСТРУМЕНТІВ

2.1 Огляд аналогів

Щоб окреслити місце розроблюваної системи серед наявних, розглянемо кілька продуктів зі схожими функціями — діалоговий пошук по нормативних чи правових документах, відповіді з посиланнями на першоджерела, робота з регуляторним контентом. Оцінюватимемо їх за чотирма критеріями: чи заземлено відповіді на конкретний корпус; чи цитуються першоджерела; чи підтримано українську нормативну базу (ДСТУ, НАПБ, ДБН); чи можлива технічна інтеграція у власні системи підприємства — передусім у CRM та месенджери. До розгляду взято п'ять аналогів: ChatGPT, Google NotebookLM, LIGA360 / ЛІГА:ЗАКОН, Harvey та CoCounsel.

ChatGPT (OpenAI). ChatGPT — універсальний діалоговий асистент на великій мовній моделі, що веде природномовну розмову в будь-якій предметній області. Його зручно взяти за базовий орієнтир (baseline), з яким порівнюватимемо спеціалізовані рішення. Відповіді він будує зі знань, закладених у параметри під час навчання, і до жодного корпусу актуальних нормативних документів не прив'язаний.

Популярність ChatGPT тримається на загальнодоступності та якісній генерації тексту. Однак щойно справа доходить до галузевої нормативки з пожежної безпеки, спливає головна вада — відповіді ні на чому зовнішньому й перевіреному не заземлені.

Переваги:

- загальнодоступність та зручний інтерфейс, що не вимагає спеціальної підготовки;
- висока якість генерації природномовного тексту та підтримка багатьох мов;
- широкі загальні знання, що охоплюють різноманітні предметні області.

Недоліки:

- схильність до галюцинацій — генерація правдоподібних, але фактично хибних тверджень без попередження;
- відсутність перевірюваних посилань на конкретні пункти нормативних документів;
- знання обмежені датою навчання, актуальні зміни нормативної бази не відображаються;
- українська галузева нормативна база (ДСТУ, НАПБ, ДБН) представлена у навчальних даних фрагментарно; інтеграція з бізнес-процесами підприємства відсутня.

Google NotebookLM. Google NotebookLM влаштовано інакше: він заземлює відповіді виключно на документах, які завантажив користувач. На відміну від звичайних LLM-чатів, система відповідає на підставі наданого корпусу й супроводжує відповіді посиланнями на конкретні фрагменти — рівень верифікованості тут помітно вищий, ніж у базового ChatGPT.

Концептуально це близько до RAG, проте NotebookLM — закрита хмарна платформа для персонального чи командного користування через браузер. Для корпоративного сценарію, де асистента треба вбудувати в Telegram-бот або CRM підприємства, цього замало.

Переваги:

- заземлення відповідей виключно на завантаженому користувачем корпусі документів;
- автоматичне цитування джерел із зазначенням фрагментів тексту;
- простота використання без програмування та налаштування інфраструктури.
- Недоліки:
- закрита платформа без можливості вбудувати функціональність у власні продукти (Telegram-бот, CRM) та бізнес-процеси;
- обмеження на кількість і формат завантажуваних документів;

- відсутність спеціалізованого опрацювання ієрархічної структури нормативних документів (розділ — стаття — пункт);
- залежність від стороннього хмарного сервісу та пов'язані ризики конфіденційності.

LIGA360 / ЛІГА:ЗАКОН. LIGA360 / ЛІГА:ЗАКОН — комерційна інформаційно-правова платформа з повнотекстовою базою українського законодавства та інтелектуальним пошуком нормативних актів. Орієнтована вона передусім на правників і вміє контролювати чинність редакцій, відстежувати зміни, шукати за реквізитами. З-поміж розглянутих аналогів це єдиний продукт, що цілеспрямовано охоплює українське законодавче поле.

Та попри актуальність і офіційність бази, LIGA360 лишається інструментом пошуку й перегляду документів — не діалоговим асистентом, який синтезує відповідь на конкретне прикладне питання. Для підбору протипожежного обладнання чи оформлення заявки в CRM знайдені фрагменти доведеться ще довго опрацьовувати вручну.

Переваги:

- офіційна та регулярно актуалізована база українського законодавства;
- контроль чинності редакцій документів і відстеження змін;
- орієнтація на українське правове поле.
- Недоліки:
- закрита екосистема за передплатою; інтеграція у власну CRM-систему підприємства та кастомізація діалогового сценарію неможливі;
- орієнтація на професійних юристів, а не на технічних спеціалістів або менеджерів з продажу;
- технічні нормативні документи (повні тексти ДСТУ, ДБН галузевого характеру) представлені обмежено;
- сценарії підбору обладнання чи автоматизованого оформлення заявки в межах платформи не реалізуються.

Harvey. Harvey — спеціалізований AI-асистент для юридичних фірм на моделях OpenAI, побудований на RAG поверх внутрішніх документів клієнта. Він автоматизує типову юридичну рутину: аналіз договорів, підготовку меморандумів, пошук у базі знань фірми. Корпоративний рівень безпеки й вузька спеціалізація роблять Harvey одним із найзріліших прикладів галузевого RAG.

Технологічно це найближчий до нашого підходу продукт — і водночас закритий корпоративний інструмент, доступний здебільшого великим юрфірмам. Розгорнути його у себе, адаптувати під українську нормативку чи під'єднати до довільної CRM не вийде.

Переваги:

- предметна спеціалізація на юридичних задачах;
- робота з власним корпусом документів клієнта в межах RAG-архітектури;
- корпоративний рівень безпеки та контролю доступу до даних.
- Недоліки:
- закритий продукт, орієнтований на великі юридичні фірми; недоступний для малого та середнього бізнесу;
- відсутність підтримки української нормативної бази (ДСТУ, НАПБ, ДБН);
- неможливість самостійного розгортання та кастомізації;
- відсутність механізмів інтеграції із зовнішніми бізнес-системами (CRM, месенджери).

CoCounsel (Casetext / Thomson Reuters). CoCounsel — AI-асистент юриста від Casetext, нині частина екосистеми Thomson Reuters. Він аналізує юридичні документи, шукає в правовій базі, готує чернетки процесуальних паперів. Вирізняють його глибока інтеграція з правовими базами Thomson Reuters і спеціалізація на юридичному аналізі.

При цьому CoCounsel — закрита комерційна платформа, і методологічно, і технологічно зорієнтована на правову систему США.

Прикласти її до українських технічних норм пожежної безпеки чи інтегрувати у вітчизняні корпоративні системи практично неможливо.

Переваги:

- глибока інтеграція з великими правовими базами даних;
- набір спеціалізованих юридичних навичок: аналіз договорів, пошук прецедентів, підготовка чернеток.
- Недоліки:
- орієнтація на правову систему США; непридатність для українських технічних норм пожежної безпеки;
- закрита комерційна платформа без можливості самостійного розгортання або кастомізації;
- висока вартість, що робить продукт недоступним для малого бізнесу.

Висновок з огляду аналогів

Зіставлення розглянутих аналогів за чотирма критеріями оцінювання наведено в таблиці 2.1.

Таблиця *2.1 — Порівняння аналогів за ключовими критеріями*

Рішення	**Заземлення на корпус**	**Цитування першоджерел**	**Українська норм. база**	**Інтеграція (CRM/месенджер)**	**Доступність для МСБ**
ChatGPT	–	–	частково	–	висока
Google NotebookLM	+	+	–	–	середня
LIGA360 / ЛІГА:ЗАКОН	+	частково	+ (правова)	–	низька
Harvey	+	+	–	–	низька
CoCounsel	+	+	–	–	низька
Розроблювана система	+	+	+	+	висока

Зведена картина оголює незайняту нішу. Універсальні асистенти (ChatGPT) не дають ні заземлення, ні цитування. Платформи із заземленням (NotebookLM) закриті й у бізнес-процеси не вбудовуються. Українські правові платформи (LIGA360) не покривають технічну нормативку й не синтезують

відповідь у діалозі. Спеціалізовані юридичні AI (Harvey, CoCounsel) закриті, дорогі та заточені під чужі юрисдикції. Жодне рішення не зводить водночас усі чотири речі: заземлення на українській технічній нормативній базі з цитуванням пунктів; діалог у масовому месенджері; інтеграцію з CRM підприємства; доступність для малого та середнього бізнесу. Саме це поєднання й реалізує розроблювана система.

2.2 Підходи до побудови RAG-пайплайна: фреймворки та власна реалізація

Зібрати RAG-пайплайн означає розв'язати кілька типових інженерних задач: завантажити й попередньо опрацювати документи, посегментувати текст, векторизувати фрагменти, проіндексувати їх і навчитися шукати, нарешті — згенерувати відповідь з урахуванням знайденого. На практиці вибір стоїть між готовим спеціалізованим фреймворком, що дає компоненти під кожен етап, і власною реалізацією з окремих бібліотек.

LangChain [24] — чи не найпопулярніший універсальний фреймворк для застосунків на LLM. У ньому є готові компоненти для завантаження документів різних форматів, сегментації, побудови ретриверів, складання ланцюгів опрацювання (chains) та агентів, а широка екосистема інтеграцій дозволяє швидко підключати векторні сховища, LLM і зовнішні сервіси. Зворотний бік — громіздкий граф залежностей, через який важче керувати середовищем; бурхливий розвиток бібліотеки веде до частих змін API між версіями; а багатошарові абстракції, що прискорюють старт, водночас ускладнюють налагодження й точний контроль над промптами.

LlamaIndex [25] заточено передусім під індексацію та пошук: конектори до джерел даних, готові стратегії сегментації, структури індексів. Для пошукових систем поверх великих корпусів він зручний, але успадковує ті самі болячки, що й LangChain, — чималий обсяг залежностей і абстракції, які ховають деталі реалізації.

Власна реалізація на окремих бібліотеках — PyMuPDF для вилучення тексту з PDF [16], sentence-transformers для векторизації [13], FAISS для індексації та пошуку [15], прямий виклик SDK обраної LLM [14] — віддає повний контроль над кожним етапом. Логіка стає прозорою, налагодження — простішим, а стратегію сегментації можна підлаштувати під структуру нормативних документів. Мінімум залежностей полегшує супровід і відтворюваність. Платити за це доводиться більшим обсягом власного коду й відсутністю готових інтеграцій.

Таблиця *2.2 — Порівняння підходів до побудови RAG-пайплайна*

Критерій	**LangChain**	**LlamaIndex**	**Власна реалізація**
Контроль над етапами конвеєра	Низький	Низький	Високий
Прозорість / налагоджуваність	Низька	Низька	Висока
Обсяг залежностей	Великий	Великий	Мінімальний
Поріг входу	Низький	Низький	Середній
Придатність до спеціалізованої сегментації нормативних текстів	Середня	Середня	Висока
Обсяг власного коду	Малий	Малий	Великий

Вибір та обґрунтування. Для цієї роботи обрано власну реалізацію RAG-пайплайна — з трьох міркувань. Перше — дослідницький характер задачі: вимоги до сегментації нормативних текстів із підрозділу 1.5 (зберегти прив'язку фрагмента до пункту й сторінки, проконтролювати якість вилучення) потребують повного контролю над етапом підготовки, який у фреймворках захований за абстракціями. Друге — масштаб корпусу: тисячі фрагментів і п'ять документів не створюють потреби в готових інтеграціях до десятків джерел, а це якраз головний козир фреймворків. Третє — промпти: заземлення й обов'язкове цитування вимагають точного контролю над системними інструкціями, чому багатошарові шаблони фреймворків лише заважають. Обсяг власного коду — ціна цього рішення — для конвеєра з

чотирьох модулів лишається помірним і з лишком окупається простотою налагодження.

2.3 Порівняння векторних сховищ

Векторне сховище — центральна ланка RAG-системи: воно тримає вектори фрагментів і забезпечує швидкий пошук найближчих сусідів за семантичним запитом. Що саме обрати, залежить від характеристик корпусу, потреби у фільтрації за атрибутами, моделі розгортання та операційних можливостей.

FAISS (Facebook AI Similarity Search) [10], [15] — бібліотека для ефективного пошуку подібності, яка живе прямо в адресному просторі застосунку й окремого серверного процесу не потребує. Вона підтримує і точний пошук (індекси IndexFlatIP, IndexFlatL2), і наближені ANN-алгоритми; індекс зберігається у файл і вантажиться при старті. Головне обмеження — немає вбудованої фільтрації за метаданими: атрибути фрагментів доводиться тримати окремо й фільтрувати на рівні застосунку. Зате на корпусах до мільйонів векторів швидкість пошуку дуже висока.

Chroma — вбудовувана векторна база для прототипування та малих і середніх корпусів: простий Python-API, персистентність, фільтри за метаданими з коробки, жодного окремого сервісу.

Qdrant — повноцінна векторна БД із власним сервером (REST/gRPC): розвинена фільтрація за метаданими, горизонтальне масштабування, багатий набір операторів запитів. Натомість її треба розгорнути, налаштувати й супроводжувати як окремий сервіс.

pgvector — розширення PostgreSQL, що додає тип для векторів та оператори подібності прямо в реляційну СУБД. Зручне, коли PostgreSQL у проєкті вже є: вектори лежать поруч із бізнес-даними й доступні через SQL. Щоправда, на великих корпусах без додаткових індексів за швидкістю воно поступається спеціалізованим рішенням.

Таблиця *2.3 — Порівняння векторних сховищ*

Критерій	FAISS	Chroma	Qdrant	pgvector
Модель розгортання	Бібліотека (in-process)	Вбудована БД	Окремий сервер	Розширення СУБД
Фільтрація за метаданими	Відсутня (вручну)	Є	Розвинена	Є (через SQL)
Персистентність	Файл на диску	Є	Є	Є
Масштабованість	Вертикальна	Обмежена	Горизонтальна	Вертикальна
Складність експлуатації	Низька	Низька	Висока	Середня
Придатність для корпусу ~10^3–10^4 фрагментів	Висока	Висока	Надлишкова	Висока

Коли корпус має порядок 10^3–10^4 фрагментів — а саме такий обсяг у зібранні нормативних документів однієї галузі, — продуктивнісний аргумент на користь розподілених рішень майже нічого не важить: усі розглянуті варіанти обслуговують такий обсяг із прийнятною затримкою. На передній план виходять простота інтеграції та операційні витрати.

Вибір та обґрунтування. Обрано FAISS. Для 2625 фрагментів цілком вистачає точного (а не наближеного) пошуку на пласкому індексі IndexFlatIP: за попередньо нормованих векторів скалярний добуток дорівнює косинусній подібності (підрозділ 1.3), а повний перебір такого обсягу займає мілісекунди. Заразом знімається й єдиний для цієї задачі недолік FAISS — відсутність фільтрації за метаданими: метадані фрагментів (документ, сторінка, пункт) лежать у паралельному JSON-файлі, де і-тий запис відповідає і-тому вектору індексу. Без окремого сервера розгортання теж простіше — індекс це лише два файли поряд із застосунком, а PostgreSQL у проєкті лишається суто під бізнес-дані CRM. Qdrant (окремий сервіс) і pgvector (розширення СУБД) відхилено як такі, що додають операційної складності без виграшу на цьому масштабі; Chroma поступається FAISS і швидкістю, і зрілістю екосистеми.

2.4 Вибір LLM та моделі векторизації з урахуванням української мови

Вимоги до великої мовної моделі. У RAG-системі для нормативних документів роль LLM не зводиться до написання тексту. Модель має відповідати лише на підставі наданих фрагментів, чітко називати джерела, а коли релевантного немає — так і казати, замість вигадувати правдоподібне. Додатково діалоговий агент усередині CRM потребує можливості викликати зовнішні функції, щоб дістатися до даних системи.

Звідси вимоги до LLM:

- якісна генерація українською мовою;
- точне слідування системним інструкціям (заземлення, обов'язкове цитування, відмова поза корпусом);
- підтримка механізму виклику інструментів (tool use / function calling) для агентських сценаріїв;
- достатнє контекстне вікно для розміщення фрагментів і діалогової історії;
- доступність через публічний API без розгортання власної інфраструктури;
- передбачувана вартість використання.

Кандидати LLM:

GPT-4o (OpenAI) — флагманська комерційна модель із якісною багатомовною генерацією, function calling і доступом через API.

Claude (Anthropic, сімейство *Sonnet*) [14] — комерційна модель із якісною багатомовною генерацією, розвиненим tool use і доброю керованістю через системні інструкції. Її особливість — ретельне дотримання обмежень, заданих у системному промпті, а це вагомий плюс там, де галюцинації неприпустимі.

Відкриті моделі (*Llama* 3 і подібні, 8–70 млрд параметрів) дають змогу розгорнути все локально, нічого не передаючи стороннім сервісам, — інколи це вирішальна вимога конфіденційності. Але в малих конфігураціях якість української генерації помітно відстає від флагманських комерційних моделей, а великі конфігурації тягнуть за собою власну GPU-інфраструктуру.

Вимоги до моделі векторизації та кандидати. Модель векторизації переводить і фрагменти документів, і запити користувача у вектори спільного семантичного простору; від її якості залежить точність пошуку. Вимоги до неї: надійна підтримка української; добра семантична близькість на коротких фрагментах; прийнятна розмірність вектора й швидкість на CPU; відкрита ліцензія.

paraphrase-multilingual-MiniLM-L12-v2 [13] — компактна багатомовна модель сімейства Sentence-Transformers (~50 мов, українська серед них). Розмірність 384 означає невеликі вимоги до пам'яті та швидку індексацію й пошук на CPU.

multilingual-e5-base — багатомовна модель із розмірністю 768: потенційно якісніше представлення ціною подвоєних обчислень і обсягу індексу.

LaBSE — модель вирівнювання речень для ~100 мов (розмірність 768), первісно зроблена для пошуку перекладних пар; для монолінгвального семантичного пошуку її спеціалізація пасує менше.

Таблиця *2.4 — Порівняння моделей векторизації*

Критерій	paraphrase-multilingual-MiniLM-L12-v2	multilingual-e5-base LaBSE	LaBSE
Підтримка української мови	Є (~50 мов)	Є	Є (~100 мов)
Розмірність вектора	384	768	768
Відносна швидкість на CPU	Висока	Середня	Середня
Ліцензія / доступність	Відкрита, sentence-transformers	Відкрита, sentence-transformers	Відкрита, sentence-transformers

Вибір та обґрунтування. Як LLM обрано Claude (Anthropic, модель сімейства Sonnet) [14]. Вирішили два чинники, прямо пов'язані з вимогами підрозділу 1.5. Перший — керованість системними інструкціями: модель послідовно тримається директив заземлення, дослівно відтворює числові значення й чесно відмовляє, коли відповіді в корпусі немає. Другий — розвинений tool use, на якому й тримається діалоговий агент (пошук у нормах, підбір обладнання, оформлення заявки — кожне як окремий інструмент). Локальні відкриті моделі відхилено через слабшу українську генерацію в доступних конфігураціях і потребу в GPU, що суперечить вимозі простого розгортання.

Як модель векторизації обрано paraphrase-multilingual-MiniLM-L12-v2 [13]. Підтверджена підтримка української закриває вимогу 6 підрозділу 1.5; розмірність 384 удвічі скорочує і обсяг індексу, і час пошуку проти альтернатив; а індексація 2625 фрагментів на CPU триває десятки секунд, що робить переіндексацію при оновленні документів (вимога 3 підрозділу 1.5) дешевою. Моделі з розмірністю 768 на цьому корпусі не дали б виграшу, який виправдав би подвоєння витрат.

2.5 Вибір каналу взаємодії та CRM-інтеграції

Систему використовують дві категорії людей із різними сценаріями: клієнт ставить запитання й оформлює заявки, а менеджер опрацьовує ці заявки, веде каталог і користується довідковим пошуком по нормах. Під кожну категорію обрано свій канал.

Для клієнта таким каналом став месенджер Telegram. Чому саме він: месенджер масово поширений в Україні, тож не доведеться змушувати клієнта ставити окремий застосунок; платформа дає безкоштовний Bot API [18] з готовими діалоговими примітивами — текстом, кнопками й механізмом передавання контакту (request_contact), яким зручно зареєструвати клієнта без ручного введення номера; режим довгого опитування (long polling) дозволяє

запустити бота без публічної IP-адреси та налаштування webhook, що спрощує розгортання. Власний вебвіджет чату довелося б писати з нуля, та й готової авдиторії він би не мав.

Менеджер працює у вебзастосунку CRM — односторінковому застосунку React поверх REST API на FastAPI [17]. Бот при цьому зроблено тонким клієнтом: бізнес-логіки в ньому немає, він звертається до того самого серверного API, що й CRM. Запити бота автентифікуються окремим внутрішнім токеном, відмінним від JWT менеджерів, — так розмежовано канали доступу. Заявка, створена агентом у діалозі (підрозділ 3.5), одразу з'являється в CRM зі статусом «очікує підтвердження», а зміна статусу менеджером запускає сповіщення клієнтові через Telegram API. Ця схема зшиває діалоговий канал із бізнес-процесом без дублювання даних: єдині джерела істини — реляційна БД (клієнти, заявки, каталог) і векторний індекс (нормативний корпус). Деталі інтеграції — у підрозділі 3.6.

2.6 Висновки до розділу 2

Огляд аналогів виявив, що рішення, яке поєднало б заземлений на українській технічній нормативці діалоговий пошук із цитуванням пунктів, масовий месенджерний канал, інтеграцію з CRM і доступність для малого бізнесу, просто відсутнє, — це й окреслює нішу розроблюваної системи.

За підсумками порівняння обрано: власну реалізацію RAG-пайплайна (повний контроль над сегментацією нормативних текстів і промптами); бібліотеку FAISS із точним індексом IndexFlatIP як векторне сховище (корпус ~$2 \cdot 10^3$ фрагментів не вимагає серверних рішень); LLM Claude сімейства Sonnet (керованість інструкціями заземлення і tool use для агента); модель векторизації paraphrase-multilingual-MiniLM-L12-v2 (багатомовність, компактність, індексація на CPU); Telegram як клієнтський канал і вебCRM на React/FastAPI як інтерфейс менеджера. Кожен вибір спирається на вимоги, сформульовані в підрозділі 1.5.

3 РОЗРОБКА ЧАТ-БОТА НА ОСНОВІ RAG

3.1 Опис проблеми та вимоги до системи

Систему створено для підприємства, що працює у сфері протипожежного обладнання. Перед нею стоять дві пов'язані прикладні задачі. Перша — давати клієнтам і менеджерам достовірні відповіді на запитання з пожежної безпеки, спираючись на чинну нормативну базу України та обов'язково посилаючись на конкретні документи й пункти. Друга — автоматизувати типовий бізнес-сценарій: підібрати протипожежне обладнання за параметрами об'єкта згідно з нормами, оформити заявку й передати її менеджеру в CRM для подальшого опрацювання.

Від звичайного питально-відповідального сценарію цю задачу відрізняє вимога юридичної достовірності: відповідь без перевірюваного посилання на чинний нормативний документ не лише марна, а й потенційно шкідлива (підрозділ 1.5). Саме це ставить RAG у центр рішення й задає перелік вимог, наведений далі.

Функціональні вимоги. Систему специфіковано такими функціональними вимогами (таблиця 3.1).

Таблиця *3.1 — Функціональні вимоги до системи*

Код	**Вимога**
FR-1	Відповідь на запитання з пожежної безпеки на основі корпусу нормативних документів
FR-2	Обов’язкове цитування джерела (документ, сторінка, за наявності — пункт)
FR-3	Явна відмова від відповіді за відсутності релевантної інформації в корпусі (заборона генерації поза корпусом)
FR-4	Розрахунок переліку протипожежного обладнання за параметрами об’єкту (тип, площа) згідно з нормами та каталогом

FR-5	Прямий підбір та замовлення конкретного товару за каталогу
FR-6	Клієнт має мати можливість оформлення заявки у діалозі з подальшою реєстрацією контакту
FR-7	Опрацювання заявок менеджером у CRM: перегляд, підтвердження, відхилення, відмітка оплати
FR-8	Автоматичне сповіщення клієнта про зміну статусу його заявки
FR-9	Довідковий пошук по нормах для менеджера у вебінтерфейсі CRM
FR-10	Керування каталогом товарів та перегляд клієнтської бази

Нефункціональні вимоги. Нефункціональні вимоги (таблиця 3.2) випливають як із характеру корпусу (підрозділ 1.5), так і з умов експлуатації прототипу на типовому серверному обладнанні без графічного прискорювача.

Таблиця *3.2 — Нефункціональні вимоги до системи*

Код	**Категорія**	**Вимога**
NFR-1	Достовірність	Відповіді заземлені на корпус; дослівне відтворення числових значень і модальних конструкцій норм
NFR-2	Продуктивність	Векторний пошук і генерація без GPU; індексація корпусу на CPU за прийнятний час
NFR-3	Актуальність	Можливість переіндексації корпусу при оновленні документів
NFR-4	Мова	Опрацювання україномовного корпусу та діалогу
NFR-5	Безпека	Розмежування доступу: автентифікація менеджерів (JWT) та внутрішній токен для бота; маскування персональних даних перед передаванням у зовнішню LLM
NFR-6	Зручність	Доступ для клієнта без встановлення окремого застосунку (месенджер)
NFR-7	Супроводжуваність	Мінімізація зовнішніх залежностей; модульна структура коду

Вимоги FR-1 — FR-3 та NFR-1 — це пряме перекладення властивостей нормативного корпусу (підрозділ 1.5) на мову специфікації; саме вони визначають проєктні рішення щодо модулів підготовки документів, пошуку та генерації (підрозділи 3.3–3.5).

3.2 Архітектура системи

Систему побудовано багатошарово, з чітким розподілом відповідальностей. Логічно вона ділиться на чотири шари: клієнтський (канали взаємодії), серверний (єдина бізнес-логіка та API), шар RAG-ядра (опрацювання нормативних документів) і шар даних (реляційне сховище та векторний індекс). Узагальнену архітектуру наведено на рисунку 3.1.

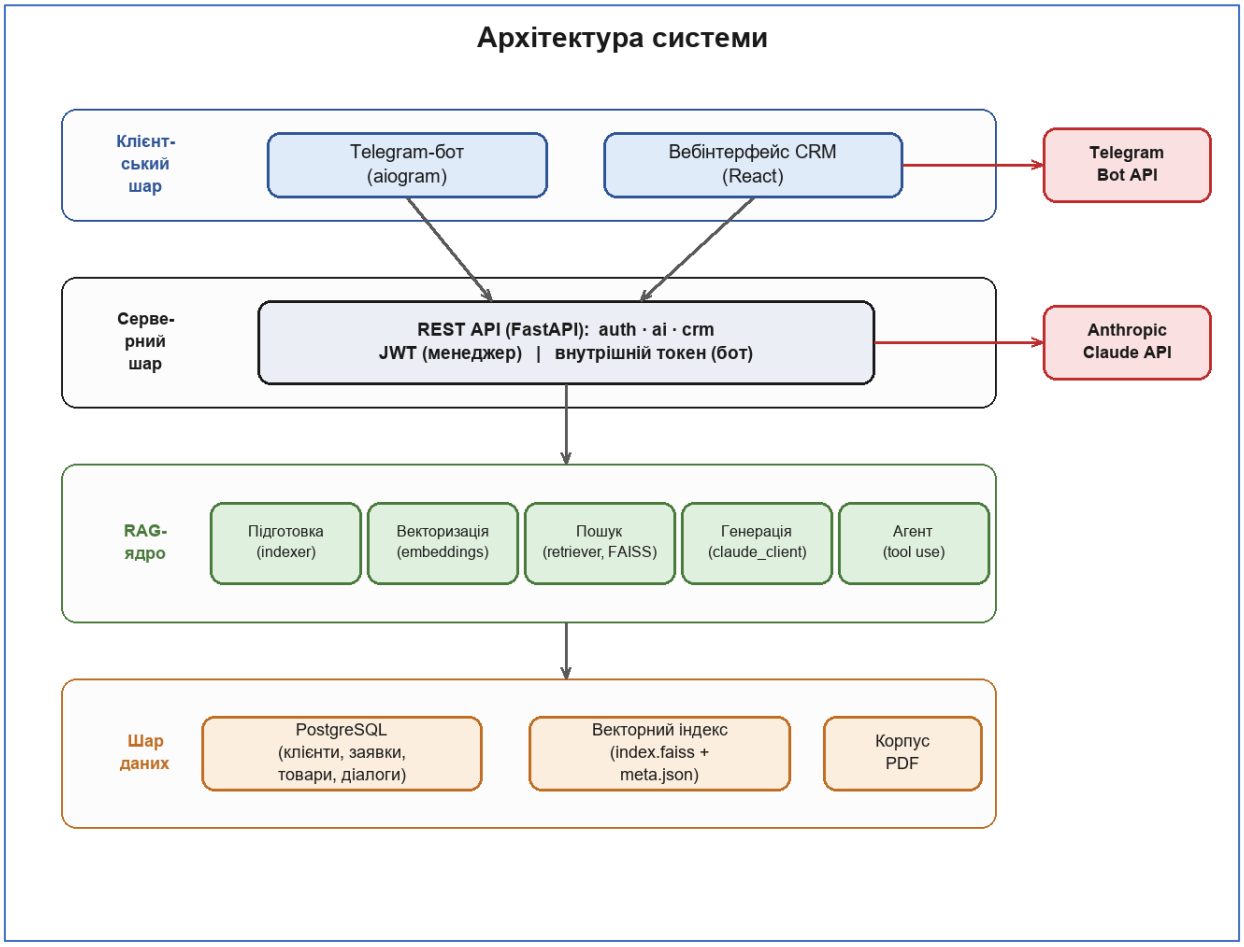


Рисунок *3.1 — Узагальнена архітектура системи*

Найважливіше архітектурне рішення — винести RAG-ядро в окремий шар, не залежний від каналів взаємодії. І Telegram-бот, і вебінтерфейс CRM

ходять до однієї й тієї самої серверної логіки через REST API, а сам бот зроблено тонким клієнтом без бізнес-логіки. Так усувається дублювання й лишається єдине джерело істини: реляційна база тримає бізнес-сутності (клієнти, заявки, товари, історія діалогу), а векторний індекс — опрацьований нормативний корпус.

Друге рішення — розвести конвеєр RAG на офлайн- та онлайн-частини (підрозділ 1.4). Офлайн-частина (модуль indexer) відпрацьовує одноразово при формуванні чи оновленні корпусу: будує векторний індекс і кладе його на диск. Онлайн-частина (retriever, claude_client, agent) живе в межах застосунку: при старті сервера індекс один раз вантажиться в пам'ять і далі обслуговує всі запити, більше не звертаючись до диска. Технологічний стек системи наведено в таблиці 3.3.

Таблиця *3.3 — Технологічний стек системи*

Шар / компонент	Технологія	Призначення
Серверна частина	Python, FastAPI [17]	REST API, бізнес-логіка
Парсинг PDF	PyMuPDF [16]	вилучення тексту з нормативних документів
Векторизація	sentence-transformers [13]	модель paraphrase-multilingual-MiniLM-L12-v2
Векторний пошук	FAISS [15]	індекс IndexFlatIP, семантичний пошук
Генеративна модель	Anthropic Claude [14]	генерація відповідей, виклик інструментів
Реляційна БД	PostgreSQL, SQLAlchemy	клієнти, заявки, товари, історія діалогу
Telegram-бот	aiogram 3 [18]	канал взаємодії з клієнтом
Вебінтерфейс CRM	React	інтерфейс менеджера

Маршрути API згруповано за трьома призначеннями: auth (автентифікація менеджерів), ai (звернення бота до RAG-ядра й агента) та crm (керування заявками, клієнтами, каталогом). Доступ розмежовано двома механізмами: запити менеджера автентифікуються токеном JWT, запити бота — окремим внутрішнім токеном у заголовку (вимога NFR-5). Так клієнтський канал ізольовано від адміністративного.

3.3 Модуль підготовки нормативних документів

Модуль підготовки документів (indexer) — це офлайн-частина RAG-конвеєра: він перетворює корпус PDF на векторний індекс. Тут безпосередньо втілюються вимоги підрозділу 1.5: зберегти прив'язку фрагмента до сторінки й пункту, сегментувати з урахуванням структури та проконтролювати якість вилученого тексту. Конвеєр складається з чотирьох послідовних кроків: вилучення тексту, сегментація, збагачення метаданими, векторизація та збереження індексу.

Вилучення тексту. Текст вилучається з PDF постранично за допомогою PyMuPDF. Посторінковість тут принципова: номер сторінки лягає в метадані кожного фрагмента й згодом стає в пригоді для цитування джерела (вимога FR-2). Основний цикл вилучення та формування записів показано в лістингу 3.1.

Лістинг *3.1 — Вилучення тексту та формування фрагментів (модуль* `indexer`)

```
    import fitz  # PyMuPDF
    records: list[dict] = []
    for pdf_path in sorted(pdf_dir.glob("*.pdf")):
        doc = fitz.open(pdf_path)
        for page_num, page in enumerate(doc, start=1):
            text = page.get_text("text")
            for piece in _chunk_text(text):
                if len(piece) < 30:          # пропускаємо
сміттєві уривки
                    continue
                records.append({
                    "document": pdf_path.name,    #
документ-джерело
                    "page": page_num,             # номер
сторінки
                    "clause": _detect_clause(piece),  #
номер пункту (за наявності)
                    "text": piece,
                })
        doc.close()
```

Сегментація. Суцільний текст сторінки ріжеться на перекривні фрагменти заданого розміру. За результатами підрозділу 1.4.1 взято фрагмент 800 символів із перекриттям 150 символів. Перекриття страхує від втрати змісту на межі: положення норми, що опинилося на стику двох фрагментів, цілком зберігається бодай в одному з них. Алгоритм (лістинг 3.2) переносить межу фрагмента на найближчий пробіл, аби не різати слова, а тоді зсуває початок наступного фрагмента назад на величину перекриття.

Лістинг *3.2 — Алгоритм сегментації з перекриттям*

```
    CHUNK_SIZE = 800       # розмір фрагмента у символах
    CHUNK_OVERLAP = 150    # перекриття, щоб не розривати
думку на межі

    def _chunk_text(text: str) -> list[str]:
        text = re.sub(r"\s+", " ", text).strip()   #
нормалізація пробілів
        if not text:
            return []
        chunks: list[str] = []
        start, n = 0, len(text)
        while start < n:
            end = min(start + CHUNK_SIZE, n)
            if end < n:
                space = text.rfind(" ", start, end)  # межа
по пробілу
                if space > start:
                    end = space
            chunks.append(text[start:end].strip())
            if end >= n:
                break
            start = end - CHUNK_OVERLAP              # крок
назад на перекриття
        return chunks
```

Такий підхід зводить разом надійність фіксованої сегментації й часткове врахування структури: нормалізація пробілів і перенесення межі на пробіл бережуть цілісність лексем, а помірний розмір фрагмента тримає тематичну однорідність. Для нашого корпусу фрагмент цього розміру здебільшого вкладається в один чи кілька суміжних пунктів — а саме цього й вимагає

самодостатність фрагмента (підрозділ 1.5). Принцип сегментації з перекриттям показано на рисунку 3.2.

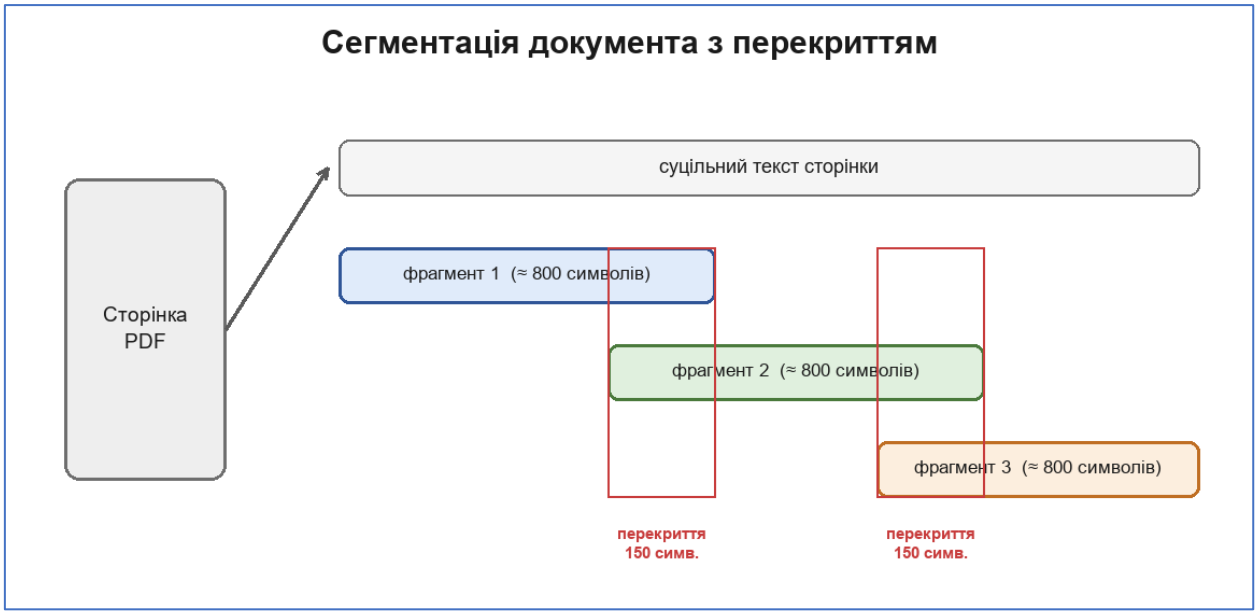


Рисунок *3.2 — Сегментація документа на перекривні фрагменти*

Збагачення метаданими. Кожному фрагменту приписується номер пункту, який евристика виловлює з тексту регулярним виразом: послідовності виду «4.2» чи «5.1.3», типові для нумерації пунктів стандартів і правил, розпізнаються як ідентифікатор структурної одиниці (лістинг 3.3). Працює це за принципом best-effort: коли номера не видно, поле отримує значення «відсутнє», і цитування йде за документом та сторінкою. Три поля метаданих разом — документ, сторінка, пункт — і складають повне посилання на джерело, чим реалізують вимогу FR-2.

Лістинг *3.3 — Виявлення номера пункту*

```
    # Пошук номера пункту виду 5.2 або 4.1.3 — для
посилання на джерело
    _CLAUSE_RE = re.compile(r"\b\d+(?:\.\d+){1,3}\b")

    def _detect_clause(chunk: str) -> str | None:
        m = _CLAUSE_RE.search(chunk)
        return m.group(0) if m else None
```

Контроль якості вилучення. Як уже зазначалося в підрозділі 1.5, частина PDF має дефекти кодування, через які кирилиця вилучається беззмістовним потоком латинських символів. Такий фрагмент індексується без жодної помилки виконання, але не знаходиться нічим — і пошук тихо деградує. Щоб ловити подібне, у конвеєр вбудовано статистичний контроль: для кожного документа рахується частка кириличних символів у вилученому тексті, і аномально низьке значення сигналізує про дефект кодування, через що документ доводиться замінити коректною копією. Під час формування корпусу цей контроль і викрив один документ із кодуванням Windows-1251, помилково декодованим як Latin-1; його замінили.

Векторизація та збереження. Зібрані фрагменти векторизуються (підрозділ 3.4) і лягають у два файли: бінарний індекс FAISS (index.faiss) та файл метаданих (meta.json), де і-тий запис відповідає і-тому вектору індексу. Це паралельне зберігання й компенсує те, що у FAISS немає вбудованої фільтрації за метаданими (підрозділ 2.3): знайшовши позицію вектора в індексі, система бере відповідний запис метаданих за тим самим порядковим номером. А оскільки індекс лежить у файлах, розгортання тривіальне — він просто копіюється разом із застосунком і вантажиться при старті.

Опрацювавши реальний корпус із п'яти нормативних документів (таблиця 1.2), отримали індекс із 2625 фрагментів. Будується він одноразово й перебудовується лише при оновленні корпусу (вимога NFR-3), а з огляду на швидкість векторизації (підрозділ 3.4) це недорога операція. Підсумкові параметри конфігурації RAG-конвеєра наведено в таблиці 3.4.

Таблиця *3.4 — Параметри конфігурації RAG-конвеєра*

Параметр	Значення
Розмір фрагмента	800
Перекриття фрагментів	150
Модель векторизації	paraphrase-multilingual-MiniLM-L12-v2
Розмірність вектора	384
Тип індексу FAISS	IndexFlatIP
Документів у корпусі	9
Фрагментів у корпусі	2625
Глибина пошуку (top-k)	8

3.4 Модуль індексації та семантичного пошуку

Модуль семантичного пошуку (retriever) реалізує етап retrieval: за текстом запиту повертає найрелевантніші фрагменти корпусу разом із метаданими. Складається він із двох частин — підсистеми векторизації (embeddings) і власне пошуку на FAISS.

Векторизація. Текст у вектори перетворює модель paraphrase-multilingual-MiniLM-L12-v2 (вибір обґрунтовано в підрозділі 2.4). Та сама функція векторизації працює і при індексації корпусу (підрозділ 3.3), і при опрацюванні запитів — а отже, фрагменти й запити гарантовано лежать в одному семантичному просторі. Важлива деталь реалізації — нормалізація векторів до одиничної довжини (лістинг 3.4): для нормованих векторів скалярний добуток дорівнює косинусній подібності (підрозділ 1.3), що дає змогу взяти швидкий індекс скалярного добутку FAISS і не рахувати косинус окремо. Модель вантажиться одноразово й кешується.

Лістинг 3.4 — Підсистема векторизації

```
  from functools import lru_cache
    from sentence_transformers import SentenceTransformer

    EMBED_MODEL_NAME = "paraphrase-multilingual-MiniLM-L12-
v2"

    @lru_cache(maxsize=1)
    def get_model() -> SentenceTransformer:
        return SentenceTransformer(EMBED_MODEL_NAME)   #
завантаження одноразове

    def embed(texts: list[str]):
        model = get_model()
        vectors = model.encode(
            texts,
            normalize_embeddings=True,   # одинична довжина
-> добуток = косинус
            convert_to_numpy=True,
            show_progress_bar=False,
        )
        return vectors.astype("float32")
```

Побудова індексу. Матриця нормованих векторів усіх фрагментів передається до індексу типу IndexFlatIP (Inner Product — скалярний добуток). Для корпусу близько 1800 фрагментів узято саме точний, а не наближений, індекс: повний перебір такої кількості векторів триває мілісекунди, тож городити ANN-структури немає сенсу. Це рішення повністю узгоджується з вибором, обґрунтованим у підрозділі 2.3.

Пошук. Пошуковий компонент оформлено класом, що інкапсулює завантаження індексу та виконання запитів (лістинг 3.5). Індекс і метадані вантажаться з диска одноразово, за принципом лінивої ініціалізації — при першому зверненні; далі вони тримаються в пам'яті, чим задовольняють вимогу продуктивності NFR-2. Під час пошуку текст запиту векторизується тією самою функцією embed, а індекс повертає позиції top-k найближчих фрагментів та їхні оцінки подібності. За цими позиціями з файлу метаданих відновлюються поля document, page, clause і text, які й формують результат.

Лістинг *3.5 — Семантичний пошук top-k*

```
    class Retriever:
        def __init__(self) -> None:
            self._index = None
            self._meta: list[dict] = []

        def load(self) -> None:
            faiss_dir = Path(settings.FAISS_DIR)
            self._index = faiss.read_index(str(faiss_dir /
"index.faiss"))
            self._meta = json.loads(
                (faiss_dir /
"meta.json").read_text(encoding="utf-8"))

        def search(self, query: str, top_k: int = 5) ->
list[dict]:
            if self._index is None:
                self.load()                          # ленива
ініціалізація
            query_vec = embed([query])               # та сама
модель, що й при індексації
            scores, idxs = self._index.search(query_vec,
top_k)
            results: list[dict] = []
```

```
            for score, i in zip(scores[0], idxs[0]):
                if i == -1:                          # менше
за top_k результатів
                    continue
                item = dict(self._meta[i])           #
відновлення метаданих за позицією
                item["score"] = round(float(score), 4)
                results.append(item)
            return results
```

Параметр top-k. Кількість фрагментів, що йдуть на генерацію, в системі дорівнює восьми. Це значення підібрано емпірично як компроміс (підрозділ 1.4.2): його досить, щоб охопити релевантне навіть тоді, коли відповідь розкидана між кількома пунктами чи документами (а для нормативного корпусу з його перехресними посиланнями це звична ситуація, підрозділ 1.5), і водночас воно не забиває контекст моделі зайвим.

Обмеження та напрям удосконалення. Реалізований пошук суто щільний (dense) — він спирається на семантичну близькість векторів. Як зазначалося в підрозділі 1.5, для точних лексичних ознак (номерів документів, позначень обладнання на кшталт ВП-6, номерів пунктів) надійніший лексичний збіг. Тож додавання розрідженого компонента (BM25), гібридизацію оцінок та реранжування крос-енкодером (підрозділ 1.4.2) винесено в напрям подальшого вдосконалення; на поточному обсязі корпусу щільний пошук top-8 дає прийнятну якість, що й підтвердило тестування (підрозділ 3.7).

3.5 Модуль генерації відповідей із цитуванням

Модуль генерації (claude_client) завершує RAG-конвеєр: зі знайдених фрагментів і запиту користувача мовна модель будує природномовну відповідь із посиланнями на джерела. Тут реалізуються вимоги FR-1 — FR-3, NFR-1 і NFR-5; саме в цьому місці властивості нормативного корпусу (підрозділ 1.5) обертаються на конкретні інструкції для моделі.

Системний промпт. Поведінку моделі задає системний промпт — стала інструкція з роллю, обмеженнями та форматом відповіді. Його складено так, щоб запрацювали три механізми з підрозділу 1.4.3: заземлення, обов'язкове цитування та коректна поведінка, коли відповіді немає. Фрагмент промпта показано в лістингу 3.6.

Лістинг *3.6 — Системний промпт модуля генерації (фрагмент)*

```
    SYSTEM_PROMPT = (
        "Ти — консультант з пожежної безпеки. Відповідай
простим текстом... "
        "Спершу дай корисну, конкретну відповідь по темі,
поєднуючи наведені "
        "фрагменти нормативних документів і загальновідому
інформацію. "
        "Де використовуєш конкретне положення з фрагмента —
став джерело у форматі "
        "[документ, стор. N]. Якщо частина відповіді —
загальні знання (не з наданих "
        "документів), коротко це познач. "
        "Не вигадуй посилань на документи, яких немає у
фрагментах. "
        "Відповідай українською, по суті."
    )
```

Лістинг 3.6 показує системний промт одиночного конвеєра answer_question (довідковий пошук менеджера). Він реалізує гібридну стратегію заземлення з огляду на специфіку прикладної задачі: кожне конкретне нормативне твердження модель зобов'язана супроводити посиланням у форматі [документ, стор. N] (заземлення на корпус), але має право доповнити відповідь загальновідомим — за умови, що явно це позначить. Пряма заборона вигадувати посилання на документи, яких немає у фрагментах, і є захистом від галюцинованих цитат (підрозділ 1.2). А вимога відповідати «по суті» й дослівно відтворювати положення норм відповідає NFR-1.

Формування контексту. Знайдені фрагменти зводяться в нумерований перелік, де кожен має заголовок із метаданими джерела — назву документа,

номер сторінки і, якщо є, номер пункту (лістинг 3.7). Ці заголовки модель і бере за зразок, будуючи посилання у відповіді, чим забезпечується перевірюваність (FR-2).

Лістинг *3.7 — Формування контексту з фрагментів*

```
    def build_context(chunks: list[dict]) -> str:
        lines = []
        for n, c in enumerate(chunks, start=1):
            clause = f", п. {c['clause']}" if
c.get("clause") else ""
            header = f"[Фрагмент {n}] {c['document']},
стор. {c['page']}{clause}"
            lines.append(f"{header}\n{c['text']}")
        return "\n\n".join(lines)
```

Конвеєр генерації відповіді. Повний конвеєр відповіді поєднує пошук, маскування персональних даних, складання промпта та виклик моделі (лістинг 3.8). Окремо варто спинитися на кроці маскування PII: перш ніж запит піде до зовнішньої мовної моделі, з нього вилучаються персональні дані (номери телефонів, адреси електронної пошти) — їх замінюють знеособленими позначками. Цей крок втілює вимогу безпеки NFR-5 і потрібен, бо запит користувача цілком може містити особисту інформацію, якій не місце у стороннього сервісу. Температуру генерації лишають на стандартному низькому рівні, що відповідає вимозі точності для фактологічних задач (підрозділ 1.4.3).

Лістинг *3.8 — Конвеєр генерації відповіді із цитуванням*

```
    def answer_question(question: str, top_k: int = 8) ->
dict:
        chunks = retriever.search(question, top_k=top_k)
# етап пошуку
        safe_question = mask_pii(question)
# маскування PII
        context = (build_context(chunks) if chunks
                   else "(релевантних фрагментів не
знайдено)")
        user_prompt = (
```

```
            f"Фрагменти нормативних
документів:\n{context}\n\n"
            f"Питання: {safe_question}\n\n"
            f"Дай корисну відповідь простим текстом,
посилайся на джерела."
        )
        message = _client.messages.create(
            model=MODEL, max_tokens=MAX_TOKENS,
            system=SYSTEM_PROMPT,
            messages=[{"role": "user", "content":
user_prompt}],
        )
        sources = [{"document": c["document"], "page":
c["page"],
                    "clause": c.get("clause")} for c in
chunks]
        return {"answer": message.content[0].text,
"sources": sources}
```

Поведінка за відсутності відповіді. Коли пошук не повертає жодного релевантного фрагмента, замість контексту в промпт підставляється явне повідомлення, що релевантних фрагментів немає. Разом із директивою системного промпта це й змушує модель сказати користувачеві, що відповіді в наявному корпусі немає, а не вигадувати щось із параметричної пам'яті (вимога FR-3).

Діалоговий агент. Описаний конвеєр answer_question обслуговує поодиноке питання-відповідь (довідковий пошук менеджера, вимога FR-9). Для клієнтського каналу, де потрібен зв'язний діалог із пам'яттю та діями (розрахунок, оформлення заявки), реалізовано діалоговий агент на механізмі виклику інструментів (tool use) обраної моделі (підрозділ 2.4). Агент перетворює бота з машини станів на природного співрозмовника: який інструмент викликати, модель вирішує сама, спираючись на історію діалогу.

Агенту дано шість інструментів: search_norms (пошук у нормах — обгортка над описаним конвеєром), calculate_equipment (розрахунок обладнання за нормами й каталогом), find_products (пошук товару в каталозі), ask_for_contact (запит контакту в нового клієнта), create_order (оформлення заявки), request_callback (замовлення дзвінка). Кожен інструмент описано

схемою у форматі, зрозумілому моделі. Диспетчеризацію реалізовано патерном «Команда + Реєстр» (Command + Registry): кожен інструмент — окрема функція-обробник, а словник зіставляє назву інструмента з обробником (лістинг 3.9), що прибирає розгалужений умовний оператор і спрощує додавання нових інструментів. Перелік інструментів та їх призначення наведено в таблиці 3.5.

Таблиця *3.5 — Інструменти діалогового агента*

Інструмент	Призначення	Основні параметри
search_norms	пошук у нормативних документах	текст запиту
calculate_equipment	розрахунок обладнання за нормами й каталогом	параметри об'єкта (тип, площа)
find_products	пошук товару в каталозі	назва / категорія
ask_for_contact	запит контактних даних нового клієнта	—
create_order	оформлення заявки на обладнання	контакт, перелік позицій
request_callback	замовлення дзвінка менеджера	контакт

Лістинг *3.9 — Реєстр інструментів агента (патерн Command + Registry)*

```
    TOOL_HANDLERS = {
        "search_norms": _tool_search_norms,
        "find_products": _tool_find_products,
        "calculate_equipment": _tool_calculate_equipment,
        "ask_for_contact": _tool_ask_for_contact,
        "create_order": _tool_create_order,
        "request_callback": _tool_request_callback,
    }

    def _run_tool(name: str, inp: dict, ctx: ToolContext) -
> str:
        handler = TOOL_HANDLERS.get(name)
        if handler is None:
            return "Невідомий інструмент."
        return handler(inp, ctx)
```

Працює агент циклічно (лістинг 3.10): історія діалогу береться з бази й передається моделі разом із переліком інструментів; якщо модель замість відповіді просить виклик інструмента, система виконує відповідний обробник,

повертає результат моделі й повторює крок; щойно модель видає текстову відповідь, цикл завершується, а відповідь лягає в історію. Пам'ять діалогу зберігається в реляційній БД за ідентифікатором користувача (останні повідомлення), чим і тримається безперервність розмови між запитами.

Лістинг *3.10 — Агентний цикл з пам'яттю та tool use*

```
    def converse(telegram_id, user_text, db):
        crud.add_dialog_message(db, telegram_id, "user",
user_text)
        history = crud.get_dialog_history(db, telegram_id,
limit=HISTORY_LIMIT)
        messages = [{"role": m.role, "content": m.content}
for m in history]
        ctx = ToolContext(telegram_id, db)
        for _ in range(MAX_TOOL_LOOPS):
            resp = _client.messages.create(model=MODEL,
system=system,
                                           tools=TOOLS,
messages=messages)
            if resp.stop_reason == "tool_use":          #
модель просить інструмент
                results = []
                for block in resp.content:
                    if block.type == "tool_use":
                        out = _run_tool(block.name,
block.input, ctx)
                        results.append({"type":
"tool_result",
                                        "tool_use_id":
block.id, "content": out})
                messages.append({"role": "assistant",
"content": resp.content})
                messages.append({"role": "user", "content":
results})
                continue
            text = "".join(b.text for b in resp.content if
b.type == "text")
            crud.add_dialog_message(db, telegram_id,
"assistant", text)
            return text, ctx.request_contact
```

Узагальнену схему агентного циклу наведено на рисунку 3.3.



Рисунок *3.3 — Агентний цикл діалогового агента з пам'яттю та виклику інструментів*

Така архітектура агента — це втілення концепції agentic RAG: пошук у нормах стає лише одним з інструментів, а модель сама оркеструє його застосування в межах ширшої задачі (консультація, розрахунок, оформлення). Тим самим RAG-ядро напряму зчіплюється з бізнес-сценарієм підприємства, описаним у підрозділі 3.6.

Дисципліна заземлення агента. Агент, на відміну від одиничного конвеєра answer_question, веде вільну розмову і теоретично здатен відповісти, не звернувшись до пошуку. Тому в його системний промт закладене обов’язкове правило: будь-яке нормативне твердження – правила розміщення, переодичність перевірок, числові значення (відстані, висоти) — має спиратися на результат search_norms і нести посилання [документ, стор. N]; якщо такого виклику не було, факт або не подається як норма, або прямо позначається як загальна інформація (лістинг 3.11). Так механізми заземлення та обов’язкового

цитування (підрозділ 1.4.3) переносяться з довідкового пошуку в діалоговий режим і програмно перекривають появу галюцинованих норм у вільній розмові. Діалоговий агент клієнтського каналу — окрема підсистема з власним системним промтом. (лістинг 3.11)

Лістинг *3.11 — Правило обов'язкового цитування в системному промпті агента*

```
SYSTEM_PROMPT = (
    "Ти — консультант компанії «Вогню Ні!» з пожежної
безпеки. ..."
    # роль, сценарії спілкування, розрахунку та оформлення
заявки
    "ОБОВ'ЯЗКОВЕ ПРАВИЛО: будь-яке нормативне твердження
(правила "
    "розміщення, періодичність перевірок, числові значення,
відстані, "
    "висоти) має спиратися на search_norms і
супроводжуватися посиланням "
    "[документ, стор. N]. Якщо ти не викликав search_norms
для факту — "
    "не подавай його як норму: або виклич search_norms і
дай цитату, або "
    "чітко познач «це загальна інформація, не з нормативних
документів». "
    "Не вигадуй цифр і норм поза знайденим."
)
```

Детермінований розрахунок обладнання. Найвідповідальнішу за точністю операцію — підрахунок кількості вогнегасників (вимога NFR-1) — інструмент calculate_equipment не довіряє мовній моделі, а рахує безпосередньо в коді за нормою (Наказ МВС № 25). Потрібну масу вогнегасної речовини він отримує як площу об'єкта, поділену на нормативні 10 м² та округлену вгору, а далі ділить її на заряд обраної моделі з каталогу моделі, щоб дістати кількість вогнегасників. Винесення цієї арифметики з мовної моделі в детермінований код дає відтворюваність (на однакових даних — однакова відповідь) і знимає ризик помилок генерації, а норму у відповіді наведено дослівно з посиланням на сторінку.

3.6 Інтеграція з CRM-системою

RAG-ядро з підрозділів 3.3–3.5 — це дослідницьке серце системи, але практичної ваги воно набуває лише вбудоване в робочий процес підприємства. У цьому підрозділі описано інтеграційно-інфраструктурний шар, що робить із ядра завершений продукт: модель даних, серверну частину, канали взаємодії та механізм, який зчіплює діалог із бізнес-процесом.

Модель даних. Реляційне сховище (PostgreSQL) тримає бізнес-сутності системи окремо від нормативного корпусу (той живе у векторному індексі). Схему бази даних наведено на рисунку 3.4.

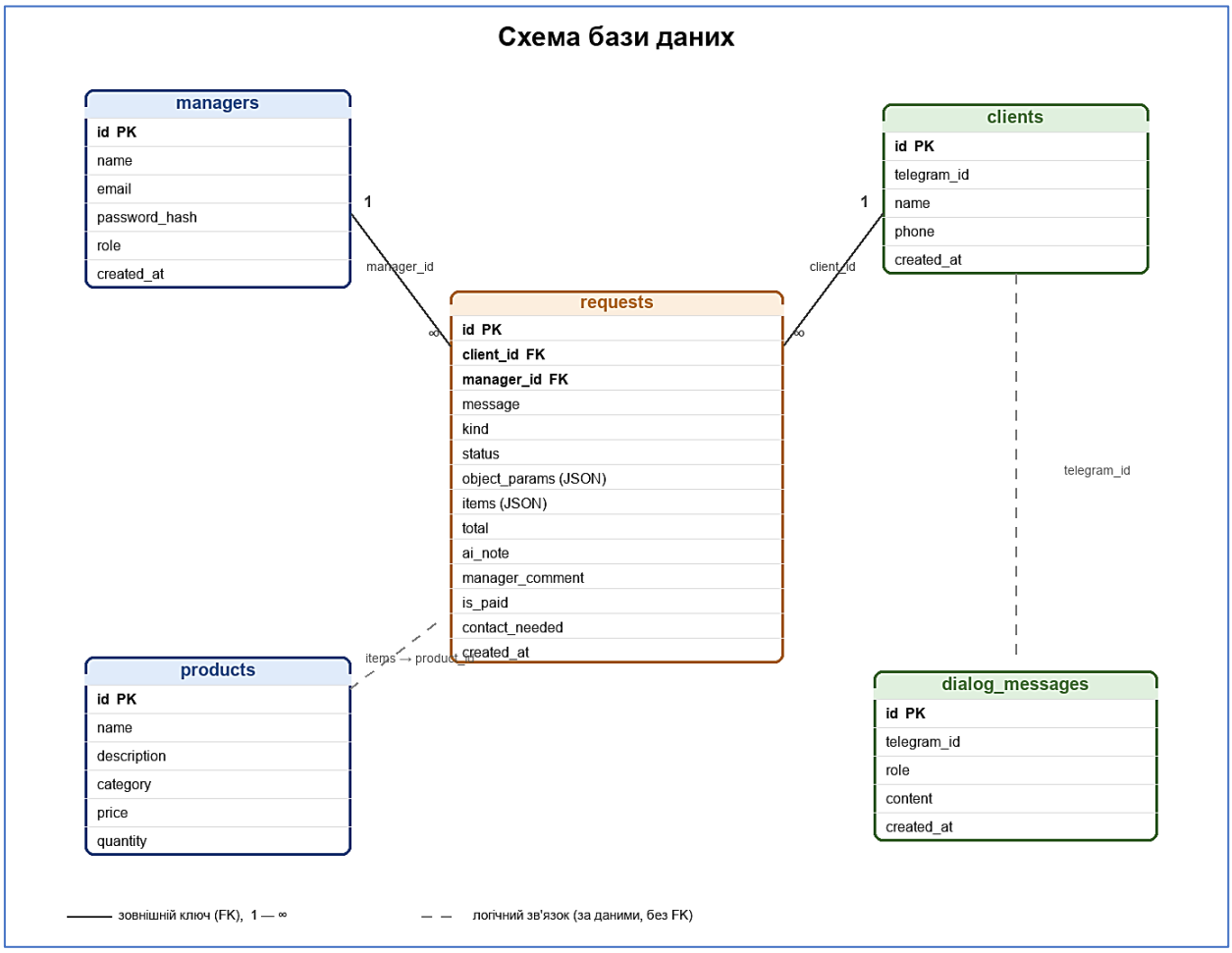


Рисунок *3.4 — Схема бази даних системи*

Центральна сутність — заявка (requests). Поле kind розрізняє два її типи: «order» — заявка на обладнання зі списком позицій (поле items у JSON) і «callback» — заявка на дзвінок менеджера. Поле status задає воронку

опрацювання: «pending» (очікує підтвердження) → «confirmed» (підтверджена, стає замовленням) → «done» (виконана), з окремою гілкою «rejected» (відхилена). Життєвий цикл заявки наведено на рисунку 3.5. Окреме булеве поле is_paid фіксує оплату. Параметри об'єкта (object_params) і підібраний перелік обладнання (items) зберігаються у JSON — формат пасує до неоднорідної структури цих даних.

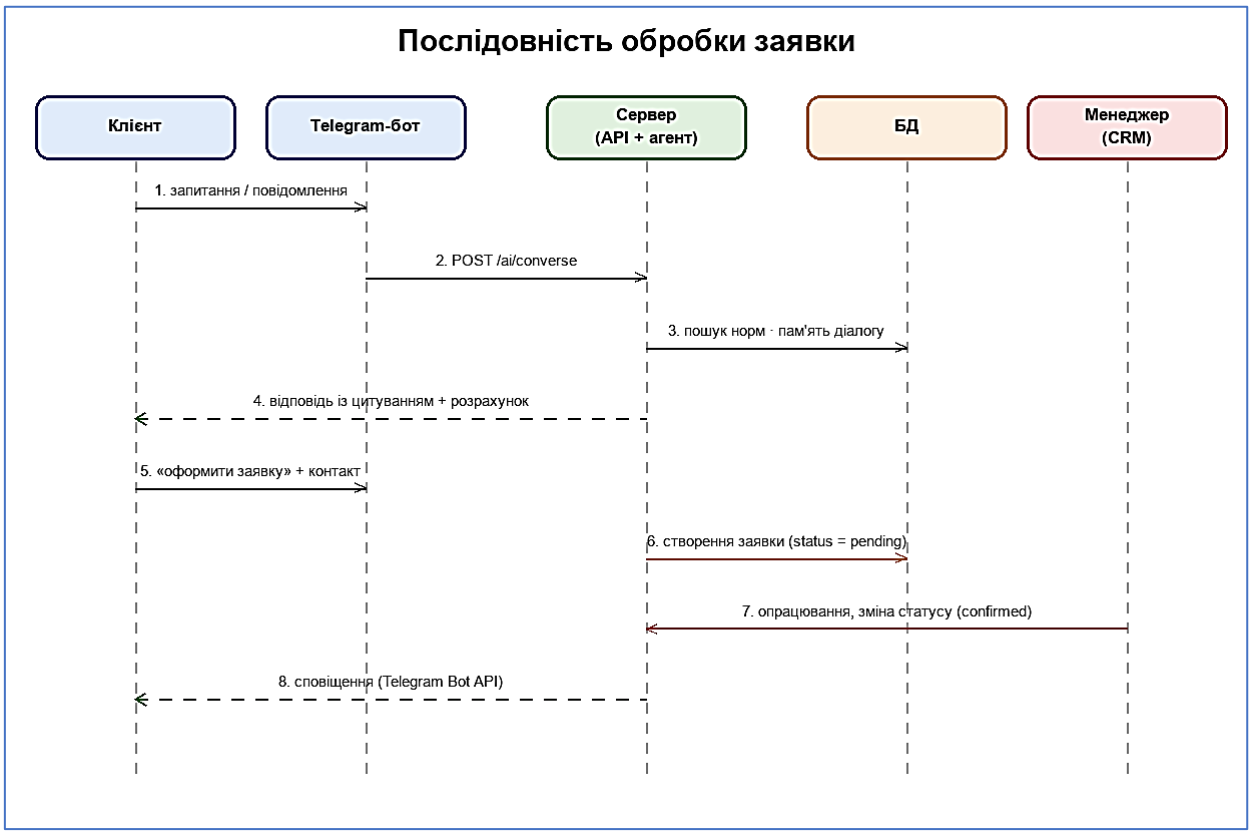


Рисунок *3.5 — Життєвий цикл (статуси) заявки*

Окрема сутність — пам'ять діалогу (dialog_messages). На відміну від заявки, вона прив'язана не до клієнта, а до ідентифікатора користувача в месенджері (telegram_id), адже діалог ведеться і з незареєстрованими користувачами. Такий поділ випливає з вимоги: клієнт потрапляє в базу лише після того, як надасть контактні дані.

Серверна частина. Серверну частину зроблено на FastAPI як набір REST-маршрутів, згрупованих за призначенням (auth, ai, crm). Бізнес-логіка зосереджена в єдиному шарі доступу до даних (модуль crud), який смикають усі маршрути, — так усувається дублювання логіки між каналами. Маршрути

групи ai обслуговують звернення бота (діалоговий агент, реєстрація клієнта, перелік заявок), маршрути crm — операції менеджера. Розмежування доступу тримається на системі залежностей FastAPI: crm вимагає дійсного токена JWT, ai — внутрішнього токена, який бот передає в заголовку запиту (вимога NFR-5). Основні маршрути REST API наведено в таблиці 3.6.

Таблиця *3.6 — Основні маршрути REST API*

Група	Маршрут	Призначення	Авторизація
auth	POST /auth/login	автентифікація менеджера	—
ai	POST /ai/converse	діалоговий агент (бот)	внутрішній токен
ai	GET /ai/requests	заявки клієнта	внутрішній токен
crm	GET /crm/requests	список заявок (pending)	JWT
crm	GET /crm/orders	список замовлень (confirmed)	JWT
crm	PATCH /crm/requests/{id}	зміна статусу заявки	JWT
crm	GET /crm/stats	зведені показники дашборда	JWT

Канал клієнта: Telegram-бот. Бота зроблено на aiogram тонким клієнтом: бізнес-логіки в ньому немає, а є дві функції — прийняти текстове повідомлення користувача й передати його серверному маршруту діалогового агента, після чого надіслати назад отриману відповідь. Уся логіка діалогу, пам'ять і виклик інструментів живуть на сервері (підрозділ 3.5). Щоб оформити заявку, бот користується вбудованим механізмом месенджера — кнопкою передавання контакту (request_contact), якою клієнт реєструється без ручного введення номера (вимога FR-6). Така мінімалістична архітектура робить бота стійким: без машини станів зникає цілий клас помилок, пов'язаних із втратою стану діалогу. Приклад діалогу клієнта з ботом наведено на рисунку 3.6.

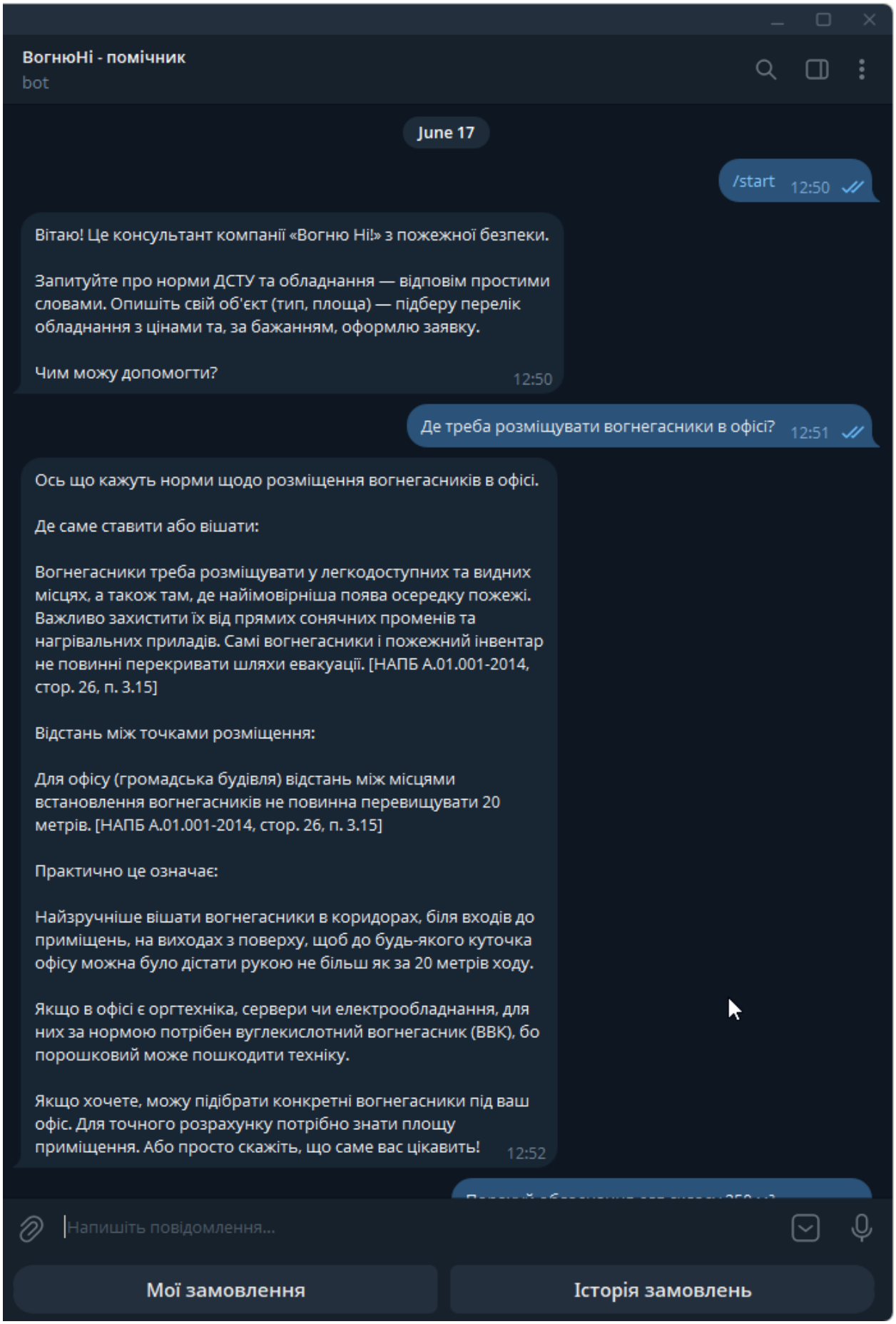


Рисунок *3.6 — Приклад діалогу клієнта з Telegram-ботом*

Канал менеджера: вебінтерфейс CRM. Інтерфейс менеджера — односторінковий вебзастосунок (React), що ходить до тих самих серверних маршрутів. У ньому є розділи: інформаційна панель зі зведеними показниками (кількість клієнтів, замовлень, нових заявок, виручка), список заявок із можливістю опрацювання, список підтверджених замовлень, база клієнтів, керування каталогом товарів і довідковий AI-чат для пошуку по нормах (вимоги FR-7, FR-9, FR-10). Інформаційну панель (дашборд) менеджера наведено на рисунку 3.7. Картка заявки показує параметри об'єкта, підібраний перелік обладнання з підсумковою вартістю, обґрунтування від AI-помічника та елементи керування статусом; її наведено на рисунку 3.8.

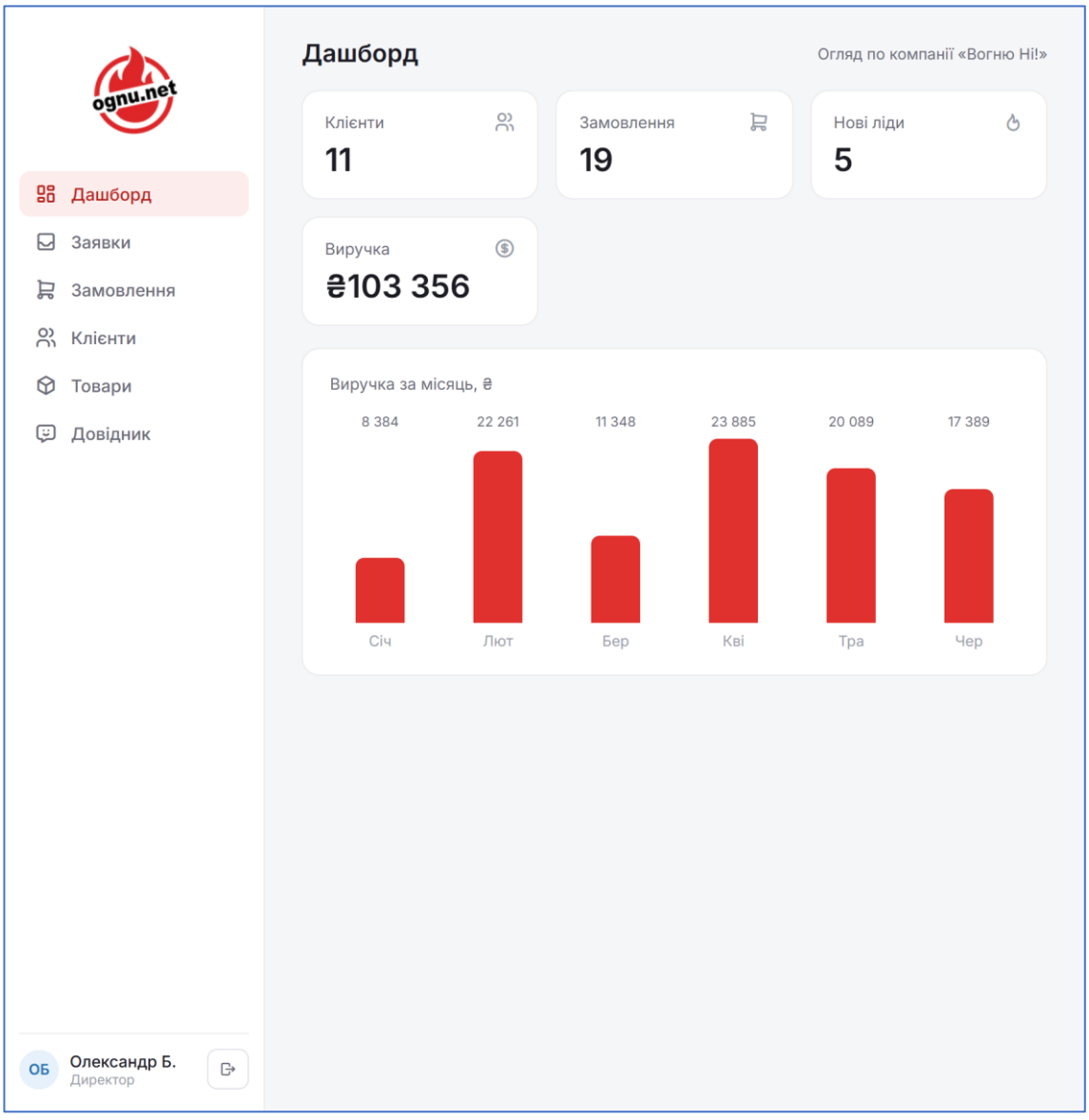


Рисунок *3.7 — Інформаційна панель (дашборд) CRM-системи*

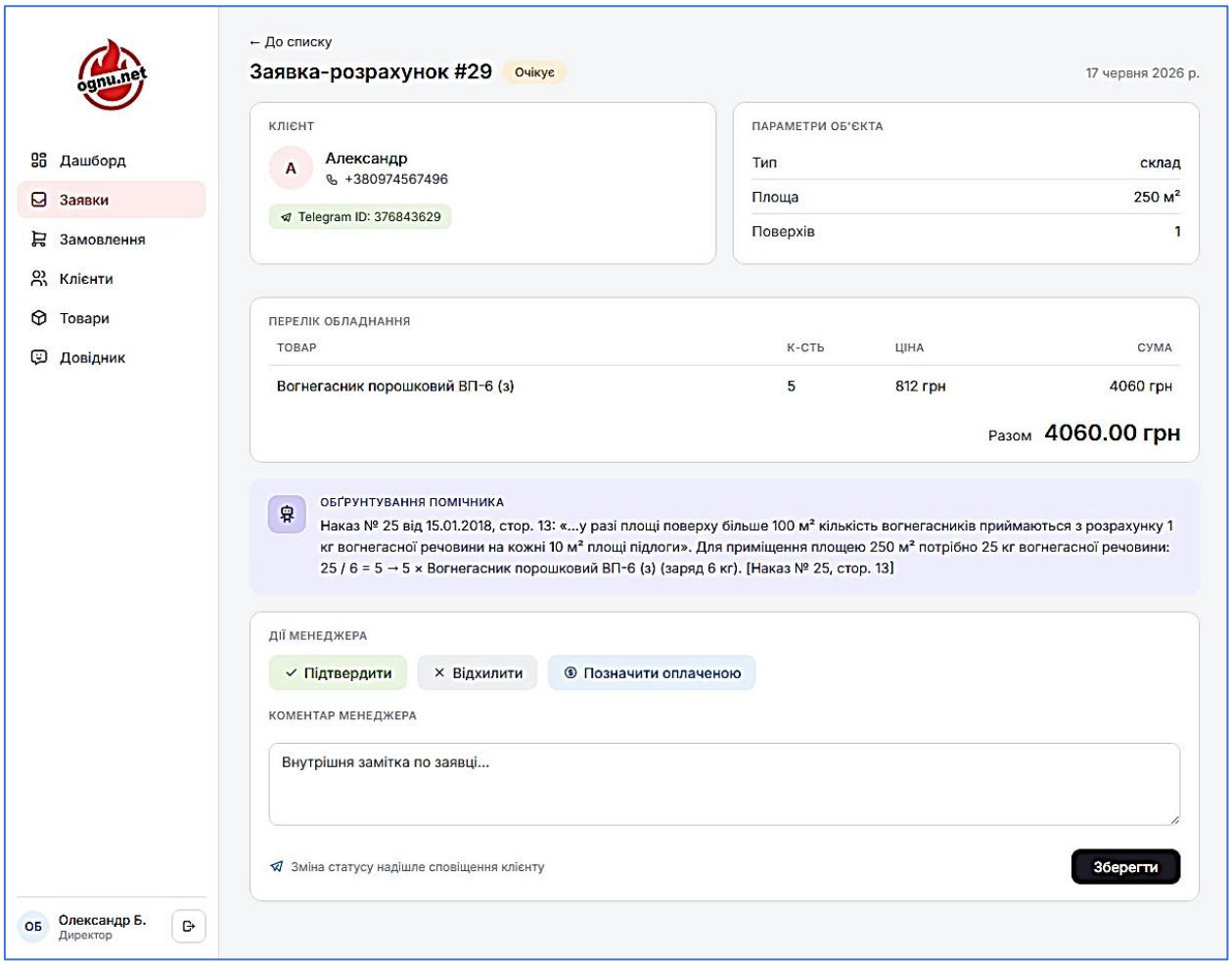


Рисунок *3.8 — Картка заявки з параметрами об'єкта, підібраним обладнанням та обґрунтуванням за нормами НАПБ*

Зв'язок діалогу з бізнес-процесом. Стрижень інтеграції — наскрізний зв'язок між діалоговим каналом і роботою менеджера. Заявка, оформлена клієнтом у діалозі з агентом (підрозділ 3.5), одразу зберігається в базі зі статусом «pending» і з'являється у списку заявок CRM. Опрацьовуючи її, менеджер змінює статус, а зміна статусу автоматично запускає сповіщення клієнтові в месенджері (вимога FR-8). Сповіщення влаштовано як пряме звернення серверної частини до Telegram Bot API через HTTP-клієнт, без залучення асинхронного фреймворку бота, — так повідомлення відправляється просто із синхронного коду серверних маршрутів. Тим самим замикається повний цикл: запит клієнта → діалог з агентом → заявка → опрацювання менеджером → сповіщення клієнта, — без дублювання даних

між каналами. Повну послідовність взаємодії компонентів у цьому циклі наведено на рисунку 3.9.

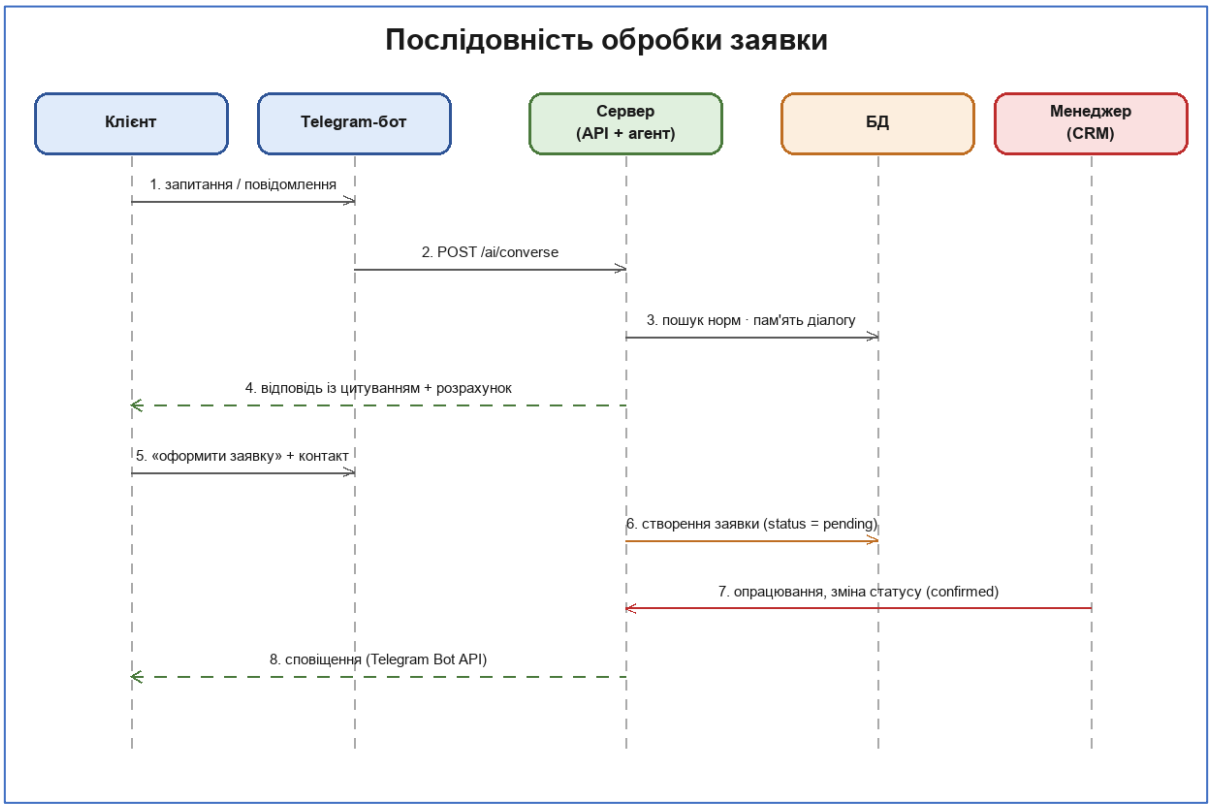


Рисунок *3.9 — Діаграма послідовності обробки заявки*

Розгортання. Систему розгорнуто трьома процесами: серверна частина (REST API та RAG-ядро), Telegram-бот і вебінтерфейс. Щоб спростити експлуатацію прототипу, написано скрипт-супервізор, який запускає всі компоненти й автоматично перезапускає той, що аварійно завершився. Векторний індекс вантажиться в пам'ять одноразово при старті серверної частини; оновити нормативний корпус можна запуском модуля індексації з подальшим перезапуском сервера (вимога NFR-3). Графічного прискорювача ця схема не потребує (вимога NFR-2): і векторизація запитів, і пошук ідуть на центральному процесорі.

3.7 Тестування якості відповідей

Мета тестування — експериментально перевірити центральну гіпотезу роботи: чи дає RAG-підхід достовірніші та перевірюваніші відповіді на запити з предметної галузі, ніж велика мовна модель без доповнення пошуком. Оскільки система працює з юридично значущими нормативними документами, важать не стилістичні риси відповіді, а її фактична обґрунтованість і наявність перевірюваного посилання на конкретний пункт чинного документа.

3.7.1 Методика експерименту

Для оцінювання зібрано тестовий набір із десяти запитань, що відтворюють типові звернення клієнтів і менеджерів до системи (розміщення та кількість вогнегасників, відповідальність за пожежні гідранти, вимоги до шляхів евакуації, визначення термінів, обов'язки суб'єктів господарювання, утримання сигналізації, періодичність перевірок, первинні засоби пожежогасіння). Кожне запитання супроводжує очікуваний документ-джерело — нормативний акт корпусу з релевантною відповіддю, що й дозволяє об'єктивно перевірити влучність пошуку.

Експеримент порівнює два режими опрацювання кожного запитання:

- RAG-система — повний конвеєр роботи (підрозділи 3.3–3.5): семантичний пошук вісьмох найрелевантніших фрагментів у векторному індексі з подальшою генерацією заземленої відповіді з обов'язковим цитуванням;

- «Гола» мовна модель — та сама модель (Claude) із загальною інструкцією консультанта з пожежної безпеки, але без доступу до корпусу; відповідь формується виключно з параметричних знань моделі.

Для оцінювання застосовано три метрики:

1. Влучність пошуку Recall@8 — частка запитань, для яких очікуваний документ-джерело потрапив до восьми знайдених фрагментів. Метрика

характеризує здатність етапу пошуку знаходити релевантне джерело й не залежить від подальшої генерації.

2. Частка цитування (RAG) — частка відповідей RAG-системи, що містять перевірюване посилання на джерело (назву документа, номер сторінки або пункту). Наявність посилання визначається автоматично за формальними ознаками у тексті відповіді.

3. Частка цитування («гола» LLM) — той самий показник для відповідей моделі без доповнення пошуком.

Експеримент реалізовано окремим скриптом, що звертається до робочої системи з повним індексом нормативного корпусу (1802 фрагменти, підрозділ 3.3). Скрипт послідовно проганяє кожне запитання обома режимами, автоматично рахує метрики й зберігає для кожного запитання знайдені документи, оцінку релевантності першого результату та фрагменти обох відповідей для подальшого якісного аналізу. Склад тестового набору наведено в таблиці 3.7.

Таблиця *3.7 — Тестовий набір запитань*

№	Запитання	Очікуваний документ	Recall@8	top-1
1	Де розміщувати вогнегасники на об'єкті?	НАПБ А.01.001-2014	+	0,74
2	Скільки вогнегасників на склад 250 м²?	НАПБ А.01.001-2014	+	0,75
3	Хто відповідає за пожежні гідранти?	НАПБ А.01.001-2014	+	0,74
4	Якими мають бути шляхи евакуації?	ДБН В.1.1-7:2016	+	0,80
5	Що таке пожежна безпека (визначення)?	ДСТУ 2272:2006	+	0,81
6	Обов'язки суб'єктів господарювання?	Кодекс цивільного захисту	+	0,84

7	Вимоги до утримання сигналізації?	НАПБ А.01.001-2014	+	0,75
8	Як часто перевіряють вогнегасники?	НАПБ А.01.001-2014	+	0,62
9	Кількість евакуаційних виходів?	ДБН В.1.1-7:2016	+	0,83
10	Що належить до первинних засобів?	НАПБ А.01.001-2014	+	0,78

3.7.2 Результати експерименту

Зведені результати експерименту наведено в таблиці 3.8.

Таблиця *3.8 — Результати порівняння RAG-системи та «голої» мовної моделі*

Метрика	RAG-система	«Гола» LLM
Влучність пошуку (Recall@8)	10/10 (100%)	—
Частка відповідей із цитуванням джерела	10/10 (100%)	0/10
Середня кількість джерел у відповіді	8	—

Етап семантичного пошуку показав влучність Recall@8 = 100%: для всіх десяти запитань очікуваний документ-джерело потрапив до знайдених фрагментів — і це попри відсутність буквального збігу формулювань запитання з текстом норми (скажімо, на запит «Як часто перевіряють вогнегасники?» система знайшла відповідні положення НАПБ А.01.001-2014, хоча в документі йдеться про періодичність технічного обслуговування). Оцінка релевантності першого результату для всіх запитань трималася в діапазоні 0,62–0,84, що свідчить про впевнене відокремлення релевантних фрагментів. Це узгоджується з вибором багатомовної моделі векторизації (підрозділ 2.4), здатної зіставляти близькі за змістом формулювання попри лексичні розбіжності. Розподіл оцінок top-1 за окремими питаннями наведено на рисунку 3.11.



Рисунок *3.11 — Розподіл оцінки релевантності top-1 за питаннями тестового набору*

Головний результат стосується перевірюваності відповідей. Усі десять відповідей RAG-системи містили посилання на конкретний пункт і сторінку чинного нормативного документа (частка цитування 100%), наприклад: «…норма забезпечення первинними засобами пожежогасіння становить два вогнегасники… на кожні 100 м² приміщення [НАПБ А.01.001-2014, стор. 63]». Натомість жодна з десяти відповідей «голої» моделі перевірюваного посилання на сторінку чи пункт не мала (частка цитування 0%).

Якісний аналіз відповідей «голої» моделі виявив характерну й найнебезпечнішу для предметної галузі ваду — галюцинацію нормативних посилань. Модель упевнено посилалася на документи, яких у чинній нормативній базі немає або які не є релевантними: «ДСТУ EN 3-7:2007», «НПБ Б.01.008-2004», «ДБН В.2.5-56:2010», «ДСТУ EN 2:2014», «Закон України „Про пожежну безпеку“ від 17.12.1993 № 3745-XII». Останній приклад показовий: названий закон справді існував, але втратив чинність із набуттям чинності Кодексу цивільного захисту України (2012), тобто модель оперує застарілими знаннями (підрозділ 1.2). Зовні така відповідь виглядає

компетентно й рясніє правдоподібними реквізитами, проте її посилання неможливо перевірити, а частина з них узагалі хибні — для юридично значущого контексту це неприйнятно.

Варто визнати: за фактичним змістом базові відповіді «голої» моделі подекуди були загалом правильні (приміром, визначення пожежної безпеки за ДСТУ 2272:2006 відтворено майже дослівно). Це й не дивно — модель велика й охоплює загальновідомі положення. Та для кінцевого користувача, клієнта чи менеджера, відповідь без перевірюваного джерела марна: її не процитуєш у комерційній пропозиції, не наведеш як обґрунтування перед замовником і не відрізниш правильне твердження від галюцинації. Саме цю різницю — між правдоподібністю та перевірюваністю — RAG-підхід і знімає.

Ключові показники тестування унаочнено на рисунку 3.10.

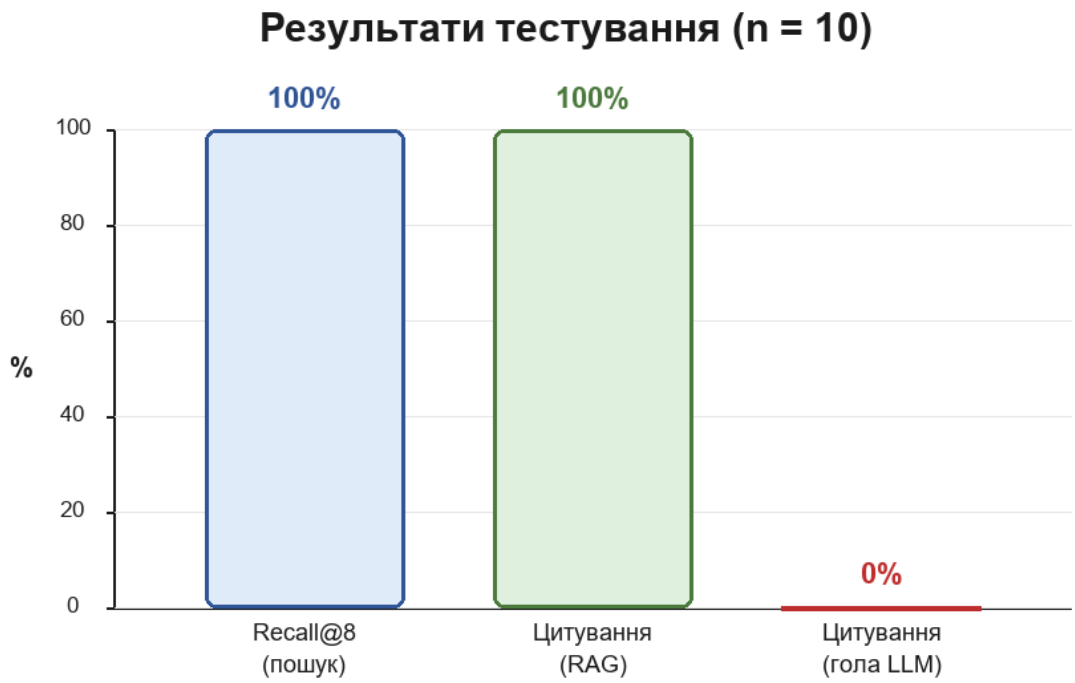


Рисунок *3.10 — Порівняння показників RAG-системи та «голої» мовної моделі*

3.7.3 Обмеження експерименту та інтерпретація

Проведений експеримент невеликий за масштабом (десять запитань) і на статистичну значущість метрик не претендує; його завдання —

демонстраційно перевірити гіпотезу на репрезентативних сценаріях використання. Метрика частки цитування зважає лише на формальну наявність посилання й не оцінює, чи правильний процитований пункт, — останнє перевірялося вибірково вручну для відповідей RAG-системи й підтвердилося. Розгорнуте оцінювання фактичної точності на великому розміченому наборі, а також метрики якості відповідей (наприклад, faithfulness і answer relevancy) лишаються напрямом подальшої роботи.

Та навіть за такого масштабу результати недвозначно підтверджують центральну тезу: за перевірюваністю відповідей RAG-система й «гола» модель різняться якісно, а не кількісно (100% проти 0%). І це прямий наслідок архітектурних рішень — заземлення генерації на знайдений контекст та обов'язкового цитування (підрозділ 3.5), а не випадкова властивість окремих запитань.

3.8 Висновки до розділу 3

У розділі описано проєктування та програмну реалізацію інтелектуального чат-бота на основі технології RAG з інтеграцією в CRM-систему.

1. На основі властивостей нормативного корпусу та сценаріїв використання сформульовано функціональні (FR-1–FR-10) та нефункціональні (NFR-1–NFR-7) вимоги до системи, що стали критеріями проєктних рішень і подальшого тестування.

2. Спроєктовано багатошарову архітектуру з виокремленням RAG-ядра в незалежний від каналів взаємодії шар і поділом конвеєра на офлайн-індексацію та онлайн-обслуговування запитів. Така організація дозволяє розвивати дослідницьке ядро незалежно від інтеграційного шару й повторно використовувати його в різних каналах (бот, вебінтерфейс).

3. Реалізовано RAG-ядро системи: модуль підготовки документів зі структурно-усвідомленою сегментацією та збереженням прив'язки фрагмента

до пункту й сторінки; модуль семантичного пошуку на основі багатомовної векторизації та індексу FAISS; модуль генерації заземлених відповідей з обов'язковим цитуванням джерел і маскуванням персональних даних. Усі ключові модулі наведено лістингами з реального коду системи.

4. Реалізовано діалогового агента на основі механізму виклику інструментів (agentic RAG), що поєднує пошук у нормах, розрахунок обладнання та оформлення заявки в єдиному природному діалозі з пам'яттю, а також інтеграційно-інфраструктурний шар: модель даних, серверну частину на FastAPI, канал клієнта (Telegram-бот як тонкий клієнт), канал менеджера (вебінтерфейс CRM) і наскрізний зв'язок діалогу з бізнес-процесом, що замикає цикл «запит клієнта - діалог - заявка - опрацювання менеджером - сповіщення клієнта».

5. Проведено експериментальне тестування якості відповідей на тестовому наборі запитань. Влучність семантичного пошуку склала Recall@8 = 100%, частка відповідей RAG-системи з перевірюваним цитуванням джерела — 100% проти 0% у «голої» мовної моделі. Якісний аналіз виявив схильність моделі без доповнення пошуком до галюцинації нормативних посилань (зокрема на застарілі та неіснуючі документи), що підтверджує практичну необхідність RAG-підходу для застосувань у юридично значущій предметній галузі.

Отриманий програмний продукт є робочим прототипом, що реалізує сформульовані вимоги та підтверджує винесену на дослідження гіпотезу про перевагу RAG-підходу над безпосереднім застосуванням великої мовної моделі за критерієм достовірності й перевірюваності відповідей.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну задачу — створення інтелектуального чат-бота на основі технології RAG для достовірного пошуку та опрацювання нормативних документів з пожежної безпеки, інтегрованого в CRM-систему. Отримано такі результати.

1. Проаналізовано розвиток технологій опрацювання природної мови та великих мовних моделей. Показано, що сучасні моделі на архітектурі Transformer добре розуміють і породжують текст, проте мають системні обмеження — галюцинації, застарілість знань, обмежене контекстне вікно й неможливість перевірюваного цитування, — які не дають застосовувати їх до юридично значущих нормативних документів напряму.

2. Досліджено архітектуру RAG та особливості нормативних документів як корпусу. З властивостей корпусу виведено шість вимог до конвеєра опрацювання: зберігати прив'язку фрагмента до пункту й сторінки; обов'язково цитувати джерело у відповіді; формувати корпус лише з чинних редакцій; дослівно відтворювати числові значення й модальні конструкції; контролювати якість вилучення тексту; застосовувати багатомовну векторизацію. Ці вимоги й лягли в основу всіх проєктних рішень.

3. Виконано порівняльний аналіз інструментів і обґрунтовано вибір: власна реалізація RAG-пайплайна (повний контроль над сегментацією нормативних текстів і промптами); FAISS із точним індексом IndexFlatIP як векторне сховище; мовна модель Claude (керованість інструкціями заземлення та механізм виклику інструментів); багатомовна модель векторизації paraphrase-multilingual-MiniLM-L12-v2. Кожен вибір спирається на сформульовані вимоги.

4. Спроєктовано багатошарову архітектуру з виокремленням RAG-ядра в незалежний від каналів взаємодії шар і поділом конвеєра на офлайн-індексацію та онлайн-обслуговування запитів. Реалізовано модуль підготовки документів зі структурно-усвідомленою сегментацією, збереженням

метаданих і контролем якості вилучення тексту; модуль семантичного пошуку на основі векторизації та FAISS; модуль генерації з обов'язковим цитуванням джерел, заземленням на корпус і маскуванням персональних даних.

5. Реалізовано діалогового агента на основі механізму виклику інструментів, що зводить пошук у нормах, розрахунок обладнання та оформлення заявки в єдиний природний діалог (концепція agentic RAG), а також інтеграцію з CRM-системою через Telegram-бот і вебінтерфейс менеджера із замкненим циклом «запит клієнта → діалог → заявка → опрацювання менеджером → сповіщення».

6. Проведено експериментальне оцінювання якості відповідей на тестовому наборі питань. Підтверджено перевагу RAG-підходу над мовною моделлю без доповнення пошуком за достовірністю й перевірюваністю: RAG-система цитує конкретні пункти нормативних документів, тоді як «гола» модель відповідає без перевірюваних посилань і схильна до галюцинацій.

Практичний результат роботи — робочий прототип системи, упроваджений для підприємства з продажу протипожежного обладнання.

Серед напрямів подальшого розвитку: гібридний пошук (поєднання семантичного й лексичного) з реранжуванням крос-енкодером, щоб підняти точність на лексично специфічних запитах (номери документів, позначення обладнання); підхід GraphRAG для опрацювання перехресних посилань між нормативними документами; відстеження версій і дат чинності документів на рівні метаданих фрагментів; розширення корпусу та автоматизація його оновлення з офіційних джерел.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS 2017), 2017. С. 5998–6008.

2. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT 2019. Minneapolis, 2019. С. 4171–4186.

3. Brown T. B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS 2020), 2020. Т. 33. С. 1877–1901.

4. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 10.06.2026).

5. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. (draft). Stanford University, 2024. URL: https://web.stanford.edu/~jurafsky/slp3 (дата звернення: 10.06.2026).

6. Ji Z., Lee N., Frieske R. et al. Survey of Hallucination in Natural Language Generation. // ACM Computing Surveys. 2023. Т. 55, № 12. С. 1–38.

7. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems (NeurIPS 2020), 2020. С. 9459–9474.

8. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of EMNLP-IJCNLP 2019. Hong Kong, 2019. С. 3982–3992.

9. Karpukhin V., Oğuz B., Min S. et al. Dense Passage Retrieval for Open-Domain Question Answering. Proceedings of EMNLP 2020, 2020. С. 6769–6781.

10. Johnson J., Douze M., Jégou H. Billion-scale Similarity Search with GPUs. // IEEE Transactions on Big Data. 2021. Т. 7, № 3. С. 535–547.

11. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. URL: https://arxiv.org/abs/2312.10997 (дата звернення: 10.06.2026).

12. Robertson S., Zaragoza H. The Probabilistic Relevance Framework: BM25 and Beyond. // Foundations and Trends in Information Retrieval. 2009. Т. 3, № 4. С. 333–389.

13. Sentence-Transformers. Model Card: paraphrase-multilingual-MiniLM-L12-v2. URL: https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 (дата звернення: 10.06.2026).

14. Anthropic. Tool Use (Function Calling). Claude Documentation. URL: https://platform.claude.com/docs/en/docs/build-with-claude/tool-use (дата звернення: 10.06.2026).

15. Faiss Documentation (Wiki). Facebook AI Research. URL: https://github.com/facebookresearch/faiss/wiki (дата звернення: 10.06.2026).

16. PyMuPDF Documentation. URL: https://pymupdf.readthedocs.io (дата звернення: 10.06.2026).

17. FastAPI Documentation. URL: https://fastapi.tiangolo.com (дата звернення: 10.06.2026).

18. aiogram Documentation. URL: https://docs.aiogram.dev (дата звернення: 10.06.2026).

19. Кодекс цивільного захисту України : Закон України від 02.10.2012 № 5403-VI. URL: https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 10.06.2026).

20. Про затвердження Правил пожежної безпеки в Україні (НАПБ А.01.001-2014) : наказ МВС України від 30.12.2014 № 1417. URL: https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 10.06.2026).

21. ДСТУ 8828:2019. Пожежна безпека. Загальні положення. Київ : ДП «УкрНДНЦ», 2019.

22. ДБН В.1.1-7:2016. Пожежна безпека об'єктів будівництва. Загальні вимоги. Київ : Мінрегіон України, 2017.

23. ДСТУ 2272:2006. Пожежна безпека. Терміни та визначення основних понять. Київ : Держспоживстандарт України, 2007.

24. LangChain Documentation. URL: https://python.langchain.com (дата звернення: 10.06.2026).

25. LlamaIndex Documentation. URL: https://docs.llamaindex.ai (дата звернення: 10.06.2026).

26. Бєлов О.О. Розробка програмної системи автоматизації CRM-процесів та експертного консультування у сфері пожежної безпеки. Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, університетська конференція «МОЛОДА НАУКА-2026» Том III «Секції Інженерного навчально-наукового інституту ім. Ю. М. Потебні Запорізького національного університету» : м. Запоріжжя, 20–24 квітня 2026 р. Запоріжжя, 2026. С. 81–82.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Бєлов Олександр Олександрович студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми “Програмне забезпечення систем”, адреса електронної пошти

ipz17bd-5@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «Розробка конструктора сайтів з використанням контейнеризації та підтримкою мультитенантної архітектури» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної добро чесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 10.06.2026 _____ Бєлов О. О.

(підпис) (прізвище та ініціали) (Студент)

Дата 10.06.2026 _____ Жуков О. О.

(підпис) (прізвище та ініціали) (Керівник)