

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

До захисту допущено

*Кафедрою математичного моделювання та аналізу даних
протокол № ____ від _____*

Завідувач кафедри _____ Володимир СТРУКОВ
(підпис) (імя, прізвище)

« ____ » _____ 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

РОЗРОБКА TELEGRAM-БОТА З ВИКОРИСТАННЯМ МЕТОДІВ ШТУЧНОГО
ІНТЕЛЕКТУ ДЛЯ ВИЯВЛЕННЯ ТА АНАЛІЗУ ДЕЗІНФОРМАЦІЇ

Спеціальність (спеціалізація) 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконавець


(підпис)

Максим ФУСАРЬ
(імя, прізвище)

Науковий керівник


(підпис)

Галина ПАДАЛКО
(імя, прізвище)

Харків – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

ЗАТВЕРДЖУЮ

Завідувач кафедри ММТАД

Володимир СТРУКОВ

підпис

ім'я, прізвище

“ _____ ” _____ 2026 року

З А В Д А Н Н Я НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувача першого (бакалаврського) рівня вищої освіти

Фусарь Максима Дмитровича

Спеціальність (спеціалізація) 122 «Комп'ютерні науки»,

Освітня програма Комп'ютерні системи та мережі

1. Тема роботи: Розробка Telegram-бота з використанням методів штучного інтелекту для виявлення та аналізу дезінформації

керівник роботи старший викладач з во, PhD Падалко Г.А.

затверджені наказом по Університету від 29 січня 2026 року № 4101-5/225

2. Строк здобувачем роботи “01” червня 2026 року

3. Вихідні дані до роботи: технічне завдання на розробку інтелектуальної системи автоматизованого фактчекінгу у форматі Telegram-бота з використанням архітектури RAG та великих мовних моделей (LLM).

4. Перелік питань, які потрібно розробити:

1. Дослідити проблему поширення дезінформації в месенджерах, виконати огляд методів автоматизованого фактчекінгу та технологій інтеграції LLM із зовнішніми базами знань.
2. Обґрунтувати вибір технологічного стека, спроектувати мікросервісну архітектуру гібридної RAG-системи та структуру локальної векторної бази знань.

3. Розробити конвеєр обробки даних у Telegram-боті та спроектувати аналітичну логіку системного промпту на базі методології Chain-of-Thought.
4. Виконати програмну реалізацію комплексу мовою Python, інтегрувавши механізми асинхронного семантичного пошуку, порогового відсікання шуму та захисту даних за принципом Human-in-the-Loop.
5. Провести автоматизоване тестування системи на реальному наборі даних, оцінити її роботу в крайових сценаріях та виконати аудит швидкодії й точності класифікації ІІІ.

5. План роботи

№ з/п	Назви етапів роботи	Строк виконання етапів роботи	Примітка
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи.	15.01.2026 – 30.01.2026	Виконано
2	Пошук та аналіз інформаційних джерел за темою КР.	30.01.2026 – 20.02.2026	Виконано
3	Вибір технологічного стеку, проєктування архітектури RAG-системи та структури векторної бази знань.	20.02.2026 – 18.03.2026	Виконано
4	Програмна реалізація Telegram-бота на базі Aiogram та аналітичного ядра з інтеграцією моделі Llama 3.1.	18.03.2026 – 02.04.2026	Виконано
5	Автоматизоване тестування точності класифікації, оцінка швидкодії та стабільності системи під навантаженням.	02.04.2026 – 19.04.2026	Виконано
6	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ІІІ.	19.04.2026 – 03.05.2026	Виконано

1. Дата видачі завдання 15 січня 2026 р.

Здобувач вищої освіти


(підпис)

М.Д. ФУСАРЬ
(ініціали, прізвище)

Керівник роботи


(підпис)

Г.А. ПАДАЛКО
(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка містить 73 сторінок, 3 розділи, 6 рисунків, 2 таблиці, 3 додатки та 25 джерел згідно з переліком посилань.

Метою роботи є розробка інтелектуальної системи (Telegram-бота) для автоматизованого фактчекінгу та протидії дезінформації. Програмно-апаратні рішення реалізовано мовою Python 3.12 з використанням фреймворку Aioogram, локальної векторної СУБД ChromaDB та великої мовної моделі Llama 3.1 (через Groq API) на базі архітектурного патерну Retrieval-Augmented Generation (RAG).

Створено надійний програмний комплекс із дворівневою моделлю верифікації інформації. Наукова новизна полягає у вдосконаленні гібридного фактчекінгу шляхом поєднання семантичного пошуку з алгоритмом відсікання за косинусною відстанню (Distance Thresholding) та fallback-стратегією автономного аналізу (Chain-of-Thought). Завдяки концепції Human-in-the-Loop систему захищено від вразливостей «отруєння даних». Результати рекомендовано для використання в умовах гібридного протистояння з метою підвищення медіаграмотності користувачів, а також як основу для систем моніторингу інформаційного простору.

Ключові слова: ФАКТЧЕКІНГ, ДЕЗІНФОРМАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ВЕЛИКА МОВНА МОДЕЛЬ, ВЕКТОРНА БАЗА ДАНИХ, СЕМАНТИЧНИЙ ПОШУК, ТЕЛЕГРАМ-БОТ, ІНФОРМАЦІЙНА БЕЗПЕКА.

ABSTRACT

The explanatory note contains 73 pages, 3 chapters, 6 figures, 2 tables, 3 appendices, and 25 sources according to the reference list.

The objective of the work is to develop an intelligent system (Telegram bot) for automated fact-checking and counteracting disinformation. The software solutions are implemented in Python 3.12 using the Aiogram framework, the ChromaDB local vector database, and the Llama 3.1 large language model (via Groq API) based on the Retrieval-Augmented Generation (RAG) architectural pattern.

A reliable software complex with a two-level information verification model was created. The scientific novelty lies in improving the hybrid fact-checking method by combining semantic search with a cosine distance thresholding algorithm and a fallback strategy for autonomous linguistic analysis (Chain-of-Thought). Through the Human-in-the-Loop concept, the system is protected against Data Poisoning vulnerabilities. The results are recommended for use in hybrid confrontation to improve user media literacy, as well as a basis for information space monitoring systems.

Keywords: FACT-CHECKING, DISINFORMATION, ARTIFICIAL INTELLIGENCE, LARGE LANGUAGE MODEL, VECTOR DATABASE, SEMANTIC SEARCH, TELEGRAM BOT, INFORMATION SECURITY.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	8
ВСТУП.....	9
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВИЯВЛЕННЯ ДЕЗІНФОРМАЦІЇ	13
1.1. Проблема поширення дезінформації в сучасних месенджерах	13
1.2. Огляд існуючих методів автоматизованого фактчекінгу	16
1.3. Обґрунтування вибору технології Retrieval-Augmented Generation (RAG) та LLM	20
1.4. Теоретико-методологічні засади оцінювання та безпеки інтелектуальних систем фактчекінгу	23
Висновки до розділу 1	26
2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ІНСТРУМЕНТАРІЮ	28
2.1. Вибір середовища розробки та технологічного стека	28
2.2. Архітектура програмного комплексу	30
2.3. Проєктування локальної бази знань та стратегія безпеки даних	35
2.4. Промпт-інжиніринг та логіка класифікації	38
Висновки до розділу 2	41
3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ	43
3.1. Практична реалізація Telegram-бота та модуля векторизації	43
3.2. Методологія автоматизованого тестування та підготовка даних	46
3.3. Аналіз якості класифікації та практична демонстрація роботи системи	49
3.4. Оцінка швидкодії та стабільності під навантаженням	56
Висновки до розділу 3	59
ВИСНОВКИ	61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	63
ДОДАТОК А. Лістинг програмного коду модуля клієнтського інтерфейсу та управління життєвим циклом системи.	67
ДОДАТОК Б. Лістинг програмного коду модуля семантичного пошуку та управління векторною базою знань.	69

ДОДАТОК В. Лістинг програмного коду підсистеми інтелектуального аналізу тексту та інтеграції з мовною моделлю.	71
---	----

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних ІІСО – інформаційно-психологічна спеціальна операція

СУБД – система управління базами даних

ІІ – штучний інтелект

API – Application Programming Interface (прикладний програмний інтерфейс)

CoT – Chain-of-Thought (ланцюжок міркувань)

CSV – Comma-Separated Values (формат файлу з роздільниками-комами)

FN – False Negative (хибнонегативний результат)

FP – False Positive (хибнопозитивний результат)

GPU – Graphics Processing Unit (графічний процесор)

IDE – Integrated Development Environment (інтегроване середовище розробки)

KNN – K-Nearest Neighbors (алгоритм k-найближчих сусідів)

LLM – Large Language Model (велика мовна модель)

LPU – Language Processing Unit (тензорний мовний процесор)

ML – Machine Learning (машинне навчання)

NLP – Natural Language Processing (обробка природної мови)

OoD – Out-of-Distribution (проблема зміщення даних у машинному навчанні)

SVM – Support Vector Machines (метод опорних векторів)

ВСТУП

В епоху стрімкого розвитку цифрових технологій та глобалізації комунікацій інформаційний простір став не лише середовищем для обміну даними, але й повноцінним полем протистояння. Соціальні мережі та месенджери, зокрема Telegram, перетворилися на основні джерела отримання новин для більшості населення. Проте безпрецедентна швидкість поширення інформації породила критичну проблему — масове генерування та розповсюдження дезінформації, фейкових новин та інформаційно-психологічних спеціальних операцій (ІПСО). Маніпулятивний контент використовується для дестабілізації суспільства, створення штучної паніки та впливу на прийняття рішень кінцевими користувачами. У таких умовах здатність швидко та ефективно відокремлювати правдиві факти від маніпуляцій стає питанням базової інформаційної безпеки.

Оцінка сучасного стану об'єкта дослідження на основі вітчизняної та зарубіжної науково-технічної літератури свідчить про активний пошук інструментів для боротьби з дезінформацією. Наразі існує низка професійних фактчекінгових ініціатив (наприклад, VoxCheck, StopFake), які демонструють високу якість верифікації. Проте їхня діяльність базується переважно на ручній праці журналістів-аналітиків, що робить неможливим масштабування процесу та оперативне реагування на мільйони повідомлень у режимі реального часу. Існуючі програмні рішення та Telegram-боти здебільшого використовують класичний пошук за ключовими словами або статичні чорні списки джерел. Такі системи є негнучкими, оскільки зловмисники легко обходять їх за допомогою використання синонімів, перефразування та генеративних нейромереж. З іншого боку, спроби застосувати сучасні великі мовні моделі (LLM) у чистому вигляді (Zero-Shot) для задач фактчекінгу стикаються з критичною проблемою фактологічних галюцинацій —

схильністю штучного інтелекту генерувати правдоподібні, але хибні відповіді через відсутність доступу до актуальної та верифікованої бази знань.

Отже, існує фундаментальна проблема, що потребує негайного вирішення: необхідність створення гібридної системи, яка б поєднувала стовідсоткову достовірність професійного фактчекінгу зі швидкістю, масштабованістю та когнітивними можливостями сучасного штучного інтелекту. Вирішення цієї проблеми можливе шляхом застосування архітектурного патерну Retrieval-Augmented Generation (RAG), який дозволяє інтегрувати локальні векторні бази даних у процес генерації відповідей нейромережею.

Актуальність теми кваліфікаційної роботи зумовлена критичною необхідністю забезпечення інформаційної безпеки суспільства в умовах гібридної війни. Розробка інтелектуального програмного комплексу, здатного в автоматичному режимі виявляти лінгвістичні маніпуляції та співставляти їх із верифікованими даними, є важливим науково-практичним завданням. Впровадження таких інструментів у вигляді доступних Telegram-ботів дозволяє розвивати критичне мислення громадян та нівелювати ефективність інформаційних атак.

Об'єктом дослідження є процес автоматизованої обробки, аналізу та верифікації текстової інформації в умовах динамічного поширення маніпулятивного контенту та дезінформації.

Предметом дослідження є моделі, методи, алгоритми та програмні засоби інтелектуального аналізу природномовних текстів, побудовані на основі архітектури Retrieval-Augmented Generation (RAG) із застосуванням великих мовних моделей (LLM) та векторних баз даних.

Мета дослідження полягає у розробці та програмній реалізації високопродуктивної інтелектуальної системи (у форматі Telegram-бота) для автоматизованого фактчекінгу, виявлення маніпуляцій та протидії

dezinformaції за допомогою технологій штучного інтелекту та семантичного пошуку.

Для досягнення поставленої мети було визначено та вирішено такі завдання:

1. Провести аналіз предметної області, дослідити існуючі методи та програмні рішення у сфері боротьби з дезінформацією.
2. Обґрунтувати вибір технологічного стека та архітектурного патерну Retrieval-Augmented Generation (RAG) для усунення проблеми фактологічних галюцинацій нейромереж.
3. Спроекувати мікросервісну архітектуру системи, що включає клієнтський інтерфейс, модуль векторизації (на базі ChromaDB) та аналітичне ядро (модель Llama 3.1 через Groq API).
4. Розробити алгоритми захисту векторної бази знань від атак типу «отруєння даних» (Data Poisoning) шляхом імплементації концепції Human-in-the-Loop.
5. Застосувати методи промпт-інжинірингу (зокрема Chain-of-Thought) для налаштування детермінованої поведінки генеративної моделі під час аналізу рівня маніпулятивності тексту.
6. Здійснити програмну реалізацію розробленого комплексу мовою Python та провести комплексне автоматизоване тестування його швидкодії та класифікаційної ефективності на реальних наборах даних.

Наукова новизна одержаних результатів полягає в удосконаленні методу гібридного фактчекінгу, який, на відміну від існуючих підходів, комбінує семантичний пошук у локальній персистентній базі знань із механізмом жорсткого порогового відсікання (Distance Thresholding) та fallback-стратегією автономного лінгвістичного аналізу. Це забезпечує стійкість системи не лише

до відомих фейків, але й до інформаційних атак «нульового дня», нівелюючи проблему зашумлення контексту.

Запропонована в роботі ідея полягає у зміні парадигми використання штучного інтелекту у кібербезпеці: нейромережа розглядається не як автономний генератор фактів, а виключно як високошвидкісний аналітик, який базує свої висновки на еталонній базі або здійснює структурний розбір тексту без використання власних параметричних знань. Практичне значення одержаних результатів полягає у створенні повністю функціонального програмного продукту, готового до інтеграції у повсякденний інформаційний простір кінцевого користувача для підвищення його медіаграмотності та захищеності.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВИЯВЛЕННЯ ДЕЗІНФОРМАЦІЇ

1.1. Проблема поширення дезінформації в сучасних месенджерах

Сучасний глобалізований інформаційний простір характеризується стрімкою трансформацією каналів комунікації та зміщенням фокусу споживання контенту від традиційних засобів масової інформації (телебачення, друкована преса, офіційні новинні вебпортали) до альтернативних цифрових платформ. Особливе місце в цій екосистемі посідають транскордонні месенджери, серед яких найбільш динамічний приріст аудиторії демонструє платформа Telegram. Перетворення архітектури класичного месенджера, початково призначеного для швидкого обміну приватними повідомленнями, на глобальну соціально-медійну платформу з функціоналом публічних каналів, мікрологістичних груп та інтерактивних роботизованих систем (чат-ботів) призвело до виникнення принципово нових викликів у сфері інформаційної та національної безпеки.

Завдяки специфічним архітектурним особливостям, таким як високий рівень анонімності творців контенту, відсутність попередньої централізованої модерації та надійне наскрізне шифрування (end-to-end encryption), сучасні месенджери перетворилися на домінантний інструмент ведення гібридних інформаційних воєн. Трансляція повідомлень відбувається у реальному часі безпосередньо на мобільні пристрої користувачів за допомогою механізмів пуш-сповіщень, що створює ефект безперервної інформаційної присутності. На відміну від класичних соціальних мереж (таких як Facebook чи алгоритмічна стрічка X/Twitter), де контент фільтрується внутрішніми алгоритмами платформи, месенджери доставляють повідомлення хронологічно та без жодних обмежень щодо охоплення аудиторії підписників.

Дослідження архітектури та соціодинаміки таких мереж показує, що швидкість реплікації та географічне охоплення маніпулятивних повідомлень у середовищі інтернет-месенджерів у декілька разів перевищують аналогічні показники для верифікованого фактологічного контенту [5]. Це зумовлено специфікою сприйняття інформації кінцевими користувачами в умовах постійного інформаційного перевантаження (information overload). Коли користувач щоденно споживає сотні коротких текстових блоків, його критичне мислення пригнічується. Зловмисники активно експлуатують когнітивні упередження, конструюючи тексти з високим емоційним забарвленням, клікбейтними тригерами («ШОК», «ТЕРМІНОВО», «ІНСАЙД») та закликами до негайного поширення інформації.

Проблема дезінформації в месенджерах суттєво загострилася в контексті сучасних геополітичних криз та повномасштабного збройного протистояння. У цей період Telegram-канали стали основним джерелом оперативних новин для мільйонів громадян, що одночасно відкрило безпрецедентні можливості для маніпуляцій та проведення інформаційно-психологічних операцій (ІПСО) з боку агресора. Моделі поширення деструктивного контенту найчастіше базуються на координації цілих мереж анонімних каналів. Ці канали репостять повідомлення один одного, синхронно транслуючи викривлені факти та створюючи у читача хибну ілюзію об'єктивності, незалежності джерел та масовості підтримки певної маніпулятивної тези. Організаційно-правові та технічні аспекти аналізу таких мереж вказують на те, що дезінформація швидко еволюціонувала. Від простих текстових фейків (вирваних з контексту цитат чи неправдивих цифр) зловмисники перейшли до складних мультимедійних наративів, які комбінують текстові маніпуляції з модифікованими графічними чи відеоматеріалами, створеними за допомогою технологій генеративного штучного інтелекту (DeepFakes) [6].

Традиційні підходи до вирішення цієї проблеми, що спираються виключно на інститут класичного, ручного фактчекінгу, на сьогодні

демонструють свою структурну та логістичну неспроможність протистояти таким обсягам даних. Діяльність професійних фактчекінгових агенцій та профільних журналістських спільнот потребує значних часових ресурсів. Процедура верифікації передбачає збір доказової бази, пошук першоджерел, аналіз метаданих медіафайлів, отримання коментарів від офіційних установ та детальне текстове оформлення аргументованого спростування. Як наслідок, процес експертної перевірки одного масштабного маніпулятивного повідомлення може тривати від кількох годин до кількох діб [2].

Водночас життєвий цикл фейкової новини в месенджері, протягом якого вона вірусно поширюється, досягає пікового охоплення та формує стійкі хибні уявлення у суспільстві, вимірюється хвилинами або першими годинами з моменту публікації. Подібний часовий асиметризм створює критичний розрив (latency gap) між моментом виникнення інформаційної загрози та появою реакції на неї. Навіть найякісніше журналістське спростування, опубліковане наступного дня, не здатне охопити ту саму аудиторію, яка вже спожила та поширила первинний фейк, що фактично нівелює ефективність ретроспективного фактчекінгу.

Таким чином, виникає гостра науково-практична потреба в автоматизації процесів моніторингу, фільтрації та первинного семантичного аналізу текстового контенту безпосередньо в середовищі його поширення. Розробка інтелектуальних програмних систем, здатних у режимі реального часу (за лічені секунди) проводити скринінг вхідних текстових масивів, зіставляти їх із наявними локальними базами верифікованих знань та залучати аналітичні потужності сучасних моделей штучного інтелекту для генерації вердиктів щодо маніпулятивності контенту, є єдиним дієвим інструментом для подолання наслідків масштабних дезінформаційних кампаній. Месенджер Telegram, виступаючи одночасно середовищем поширення загрози, надає гнучкий та відкритий прикладний програмний інтерфейс (Telegram Bot API) для інтеграції таких захисних систем у формі діалогових роботизованих

комплексів, що робить обраний напрямок дослідження високоактуальним та таким, що має значний потенціал для практичного впровадження.

1.2. Огляд існуючих методів автоматизованого фактчекінгу

Завдання автоматизації фактчекінгу полягає у створенні апаратних та алгоритмічних систем, здатних без втручання людини оцінювати достовірність інформаційних тверджень у великих потоках неструктурованих даних. Зважаючи на стрімке зростання обсягів цифрового контенту, історичний розвиток таких систем еволюціонував від простих лексичних аналізаторів до складних нейромережових архітектур, які в сучасній комп'ютерній лінгвістиці прийнято умовно поділяти на три покоління.

Перше покоління: Лексичний аналіз та класичне машинне навчання. Початкові спроби автоматизації виявлення дезінформації базувалися на евристичних правилах (rule-based systems) та поверхневому статистичному аналізі тексту. У таких системах використовувалися методи векторного подання частотності слів, зокрема «мішок слів» (Bag of Words) та алгоритм TF-IDF (Term Frequency — Inverse Document Frequency). Ці підходи намагалися ідентифікувати маніпуляції за наявністю специфічних стилістичних маркерів: нагромадження клікбейтних заголовків, надмірної експресивної пунктуації, використання специфічних для пропаганди прикметників чи нецензурної лексики.

Хоча такі лексичні аналізатори демонстрували високу обчислювальну швидкодію, вони виявилися вкрай вразливими до навмисної зміни стилістики тексту зловмисниками. На зміну правилам прийшли методи класичного машинного навчання. Дослідники почали застосовувати метод опорних векторів, наївний баєсівський класифікатор (Naive Bayes) та ансамблеві алгоритми, такі як випадкові ліси [7]. Їхнє впровадження дозволило виявляти приховані статистичні закономірності та неочевидні кореляції у великих

масивах попередньо розмічених експертами даних. Класичне ML суттєво підвищило точність класифікації текстів порівняно з евристиками, проте вимагало складного етапу ручного конструювання ознак [3].

Друге покоління: Глибоке навчання та векторні ембеддинги. З розвитком галузі обробки природної мови сформувалося друге покоління систем на базі алгоритмів глибокого навчання. Справжнім проривом стала поява архітектури Transformer та механізмів уваги. Використання попередньо натренованих мовних моделей, таких як BERT та його оптимізованих модифікацій, дозволило відмовитися від ручного виділення ознак тексту [20].

Моделі цього покоління здатні перетворювати сирий неструктурований текст на щільні багатовимірні математичні вектори (ембеддинги), які зберігають глибокий семантичний зміст речень. На відміну від словникових методів першого покоління, векторний підхід дозволяє системі «розуміти» контекст, іронію, синонімію та полісемію слів.

Проте, як зазначають дослідники [15], головним архітектурним недоліком чистих ML-класифікаторів залишається їхня критична залежність від розподілу даних у навчальних вибірках. Модель, яка була натренована та успішно виявляла фейки щодо пандемії COVID-19, виявляється практично безсилою перед дезінформацією щодо нових геополітичних чи військових подій. Це явище в машинному навчанні відоме як проблема зміщення даних (Out-of-Distribution, OoD). Крім того, традиційні нейромережеві класифікатори функціонують за парадигмою «чорної скриньки» (Black Box): вони видають лише математичну ймовірність приналежності новинного тексту до класу «Фейк», не надаючи користувачеві жодного фактологічного обґрунтування логіки прийняття свого рішення.

Третє покоління: Великі мовні моделі. Сучасний рубіж інструментів автоматизованого фактчекінгу безпосередньо пов'язаний з появою великих мовних моделей, таких як серії GPT, LLaMA та їхніх відкритих аналогів.

Завдяки навчанню на гігантських корпусах текстових даних, сучасні LLMs володіють потужними можливостями логічного висновування (reasoning) та здатні аналізувати складні багаторівневі маніпулятивні патерни, що є недоступним для класифікаторів попередніх поколінь [10].

Фундаментальною перевагою великих мовних моделей є їхня здатність до генерації пояснень людською мовою (Explainable AI). LLM може не лише присвоїти новині певний вердикт, але й згенерувати зв'язний аналітичний звіт, вказавши на логічні хиби, застосування мови ворожнечі або відсутність посилань на першоджерела у тексті, що перевіряється.

Для наочного зіставлення еволюції технологій автоматизованого аналізу текстів, порівняльну характеристику трьох поколінь систем наведено у таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика поколінь систем автоматизованого фактчекінгу

Критерій порівняння	1 покоління (Класичне ML: SVM, Naive Bayes)	2 покоління (Deep Learning: BERT, RoBERTa)	3 покоління (Сучасні LLM: GPT, LLaMA)
Розуміння семантики тексту	Низьке (аналіз окремих слів або N-грамів)	Високе (векторне розуміння контексту)	Дуже високе (здатність до логічного міркування)
Потреба у розмічених даних	Висока (потребує ручного конструювання ознак)	Висока (великі датасети для етапу Fine-Tuning)	Низька (працюють у форматах Zero-shot та Few-shot)

Продовження таблиці 1.1

Пояснюваність результату	Часткова (можна визначити вагу окремих слів)	Відсутня (проблема «чорної скриньки»)	Повна (генерація аргументованого звіту у вигляді тексту)
Вразливість до OoD (нові теми)	Критична (повна втрата точності)	Висока (потребує перенавчання на нових даних)	Помірна (модель може аналізувати стиль нової інформації)
Схильність до фактологічних галюцинацій	Не застосовується (моделі не генерують текст)	Не застосовується	Критично висока (генерація хибних фактів)

Як видно з результатів аналізу, попри беззаперечну технологічну перевагу великих мовних моделей, їхнє використання як ізольованих систем для завдань фактчекінгу несе у собі критичний ризик. Базова нейромережа LLM не має доступу до інтернету чи оновлених баз даних у режимі реального часу. Вона оперує виключно тими «замороженими» знаннями, які були закладені в її параметри (ваги) на момент завершення тренування (knowledge cutoff). Відповідно, ізольована модель фізично не здатна верифікувати події сьогодення і, намагаючись дати відповідь, часто продукує правдоподібну брехню — «галюцинації».

Ця фундаментальна проблема формує гостру архітектурну потребу у створенні гібридних систем. Необхідно розробити програмний комплекс, який би поєднував потужні аналітичні та лінгвістичні здібності сучасних великих

мовних моделей із зовнішніми, динамічно оновлюваними та строго верифікованими базами локальних знань.

1.3. Обґрунтування вибору технології Retrieval-Augmented Generation (RAG) та LLM

Стрімкий розвиток генеративного штучного інтелекту зумовив корінну зміну парадигми в галузі автоматизованого аналізу, класифікації та семантичної обробки природномовного текстового контенту. Традиційні підходи, що спиралися на ізольовані класифікаційні моделі або статичні лінгвістичні правила, дедалі частіше поступаються місцем інтеграційним рішенням, які поєднують у собі когнітивні можливості великих мовних моделей (Large Language Models, LLM) із зовнішніми джерелами структурованих і верифікованих даних. Концепція архітектури архівації та генерації із залученням пошуку (Retrieval-Augmented Generation, RAG) на сьогодні є найбільш ефективною та інженерно обґрунтованою відповіддю на ключові деструктивні недоліки сучасних нейромереж. До таких недоліків насамперед належать схильність до фактологічних помилок («генеративних галюцинацій»), відсутність прозорості у прийнятті рішень (проблема «чорної скриньки») та обмеженість знань датою завершення навчання моделі (knowledge cutoff).

У контексті розв'язання задачі протидії дезінформації та виявлення ворожих інформаційно-психологічних операцій (ІПСО), класична схема RAG передбачає чітко детермінований двохетапний процес обробки інформаційного потоку [13].

На першому етапі (Retrieval Stage) система виконує первинний семантичний скринінг вхідного запиту користувача. Текст новини трансформується у щільний вектор за допомогою моделі ембеддингів, після

чого здійснюється пошук релевантних доказів або спростувань у зовнішньому індексованому корпусі знань (векторній базі даних).

На другому етапі (Generation Stage) знайдений контекст разом із оригінальним текстом новини та суворими системними інструкціями інтегрується у збалансований промпт і передається на вхід генеративної моделі. Такий підхід докорінно трансформує роль LLM у загальній архітектурі системи: неймережа перестає функціонувати як безпосереднє джерело фактологічних знань (параметрична пам'ять), а перетворюється на високоінтелектуальний аналітичний інтерпретатор та процесор (непараметричне мислення), який формує логічний висновок виключно на основі наданих йому задокументованих еталонних фактів [11].

Альтернативним методом адаптації великих мовних моделей під вузькоспеціалізовану специфіку фактчекінгу є їхнє повне або часткове донавчання (Fine-Tuning) на спеціалізованих текстових датасетах спростувань. Проте порівняльний інженерний аналіз архітектурних рішень демонструє суттєві переваги технології RAG над процесом параметричного оновлення внутрішніх ваг моделі за кількома фундаментальними критеріями:

1. Динамічність та актуальність бази знань. Поява нових маніпулятивних наративів, фейків та клікбейту в месенджерах відбувається щоденно та лавиноподібно. Для поновлення знань fine-tuned моделі необхідне регулярне, циклічне перенавчання неймережі на нових даних, що вимагає залучення колосальних обчислювальних потужностей (високовартісних кластерів GPU/TPU) та значних часових ресурсів. У системах, що побудовані за принципом RAG, актуалізація інформаційного ядра відбувається в режимі реального часу. Модератор або адміністратор системи може миттєво додати новий факт або модифікувати існуючі текстові рядки у векторній базі даних, і ці знання стають доступними ШІ під час наступного запиту користувача, без

необхідності зупинки системи чи перенавчання базового алгоритму штучного інтелекту.

2. Фактологічна верифікованість та інтерпретованість результатів (Explicability). Генеративні моделі за своєю лінгвістичною природою є стохастичними класифікаторами, які передбачають появу наступного токена (слова) на основі ймовірнісних розподілів. Застосування архітектури RAG дозволяє жорстко заземлити (grounding) логічний висновок нейромережі на конкретні, фізично існуючі джерела інформації [16]. Завдяки цьому програмний комплекс спроможний не просто видати сухий бінарний вердикт, а й продемонструвати кінцевому користувачеві точний фрагмент тексту з офіційного спростування або урядового повідомлення, на основі якого модель виявила маніпуляцію. Це є критично важливим аспектом для побудови суспільної довіри до автоматизованих систем інформаційної безпеки.

3. Економічна доцільність та низький поріг масштабування інфраструктури. Використання готових базових open-source моделей у поєднанні з оптимізованими локальними векторними базами даних дозволяє розгортати ефективні системи моніторингу дезінформації без залучення капіталомістких інвестицій у тренування власних нейромереж. Це робить обраний підхід оптимальним для розробки та розгортання прикладних інженерних рішень та прототипів систем [1].

Як базове технологічне ядро штучного інтелекту в межах даної кваліфікаційної роботи було обрано сучасну велику мовну модель родини Llama 3.1, інтегровану через високопродуктивну інфраструктуру Groq API. Архітектурні характеристики цієї моделі (зокрема покращені механізми групової уваги Grouped-Query Attention) демонструють високу ефективність під час лінгвістичного аналізу та детекції прихованих маніпулятивних патернів у складних, зашумлених текстових масивах новинного характеру [23]. Великий обсяг контекстного вікна моделі (Context Window) дозволяє

безперешкодно передавати довгі фрагменти аналізованих статей разом із декількома варіантами суміжних за змістом документів, знайдених системою у векторному просторі СУБД. Це гарантує утримання логічного ланцюжка міркувань моделі (Chain-of-Thought) без втрати суттєвих семантичних деталей на етапі генерації фінальної відповіді.

Попри очевидні переваги, практичне впровадження рамкової технології RAG+LLM супроводжується певними інженерними викликами, обмеженнями та специфічними крайовими випадками (edge cases). Ефективність функціонування всього програмного комплексу безпосередньо залежить від точності етапу пошуку (Retrieval Stage). Якщо алгоритми семантичного зіставлення векторів через низьку щільність бази даних або високий рівень метафоричності тексту не зможуть знайти релевантний контекст під специфічний запит користувача, велика мовна модель буде змушена приймати рішення в режимі нульового контексту (Zero-shot classification).

У таких сценаріях якість аналізу повністю залежить від внутрішніх когнітивних здібностей моделі та збалансованості її системного промпту. Окрім цього, інтеграція зовнішніх хмарних інтерфейсів програмування додатків (API) накладає обмеження на загальну пропускну здатність системи через наявність жорстких лімітів частоти запитів (Rate Limits) з боку провайдерів обчислювальних ресурсів, що необхідно враховувати при проектуванні архітектури високонавантажених систем.

1.4. Теоретико-методологічні засади оцінювання та безпеки інтелектуальних систем фактчекінгу

Впровадження методів генеративного штучного інтелекту та архітектури Retrieval-Augmented Generation (RAG) у критичні сфери, такі як інформаційна безпека та боротьба з дезінформацією, вимагає формування жорсткої методологічної бази для їх об'єктивного оцінювання. Оскільки великі

мовні моделі функціонують на основі ймовірнісних розподілів, забезпечення надійності їхньої роботи потребує застосування спеціалізованих математичних метрик та безпекових парадигм.

З теоретичної точки зору, процес автоматизованого фактчекінгу зводиться до задачі бінарної або мультикласової класифікації тексту. Для всебічного оцінювання якості роботи класифікаційних моделей у машинному навчанні традиційно використовується матриця помилок (Confusion Matrix) — таблиця, що дозволяє візуалізувати продуктивність алгоритму на тестовій вибірці. Матриця розподіляє результати роботи системи на чотири базові категорії:

1. Істинно позитивні (True Positives, TP) — коректно виявлені фейки;
2. Істинно негативні (True Negatives, TN) — коректно підтверджені правдиві новини;
3. Хибно позитивні (False Positives, FP) — правдиві новини, помилково визнані маніпуляцією (хибна тривога);
4. Хибно негативні (False Negatives, FN) — пропущені системою фейки, які були марковані як правда [21].

На основі базових значень матриці помилок розраховуються похідні статистичні метрики, кожна з яких відображає окремий аспект ефективності моделі:

1. Точність (Accuracy) — базова метрика, що визначає загальну частку правильних прогнозів моделі. Попри свою універсальність, у задачах інформаційної безпеки ця метрика часто буває оманливою, особливо у випадках незбалансованих наборів даних, коли кількість правдивих новин у потоці значно перевищує кількість дезінформації.

2. Точність класифікатора (Precision) — показує, яка частка новин, маркованих системою як «Фейк», дійсно є фейками. Високий показник Precision означає, що система рідко генерує хибні тривоги.

3. Повнота (Recall / Sensitivity) — визначає, яку частку від усіх реально існуючих у вибірці фейків система змогла успішно ідентифікувати.

4. F1-міра (F1-score) — гармонійне середнє між Precision та Recall, що дозволяє отримати збалансовану оцінку моделі.

У теоретичному контексті проєктування систем протидії ІІСО метрика Recall має значно вищий пріоритет, ніж базова Accuracy. Специфіка предметної області диктує правило: пропустити ворожий наратив (False Negative) є набагато небезпечнішим для суспільства, ніж помилково запідозрити правдиву, але емоційно забарвлену новину в маніпулятивності (False Positive). Таким чином, математична модель антифейк-системи повинна бути навмисно зміщена (bias) у бік гіперчутливості до потенційних загроз.

Наступним фундаментальним аспектом методології є забезпечення цілісності та безпеки самої векторної бази знань (RAG-ядра). Теоретичні дослідження вразливостей мовних моделей вказують на критичний ризик виникнення так званої «петлі галюцинацій» (Hallucination Loop) та атак типу «отруєння даних» (Data Poisoning). Якщо архітектура системи передбачає автоматичне поповнення бази знань результатами генерації LLM (без експертної перевірки), існує висока ймовірність того, що модель згенерує правдоподібний фейк, запише його до своєї бази як факт, і під час наступних запитів буде спиратися на власну хибну генерацію як на еталон істини.

Для усунення цієї вразливості в сучасній інженерії штучного інтелекту застосовується парадигма Human-in-the-Loop (Людина в контурі управління). Відповідно до цієї теорії, інтелектуальна система не повинна мати права на автономну модифікацію свого фактологічного ядра. Нейромережа виконує виключно функції аналізу вхідного потоку та генерації звітів на основі

статичної бази. Оновлення, валідація та розширення векторної бази знань залишаються виключною прерогативою людини (модератора-експерта), що гарантує епістемологічну чистоту даних і запобігає деградації системи в умовах тривалої експлуатації.

Отже, проєктування надійної системи фактчекінгу вимагає не лише вибору оптимальних алгоритмів машинного навчання, а й застосування правильної методології оцінювання (з акцентом на повноту виявлення загроз) та впровадження суворих архітектурних обмежень для забезпечення безпеки інформаційного ядра.

Висновки до розділу 1

У першому розділі пояснювальної записки було проведено комплексний теоретичний аналіз проблематики поширення дезінформації в умовах сучасного цифрового простору. Встановлено, що архітектурні особливості транскордонних месенджерів формують сприятливе середовище для надшвидкої реплікації маніпулятивних наративів. Визначено, що класичний інститут ручного фактчекінгу є логістично неспроможним протистояти сучасним обсягам ІІСО через критичну часову затримку між появою загрози та публікацією спростування.

Здійснено системний огляд розвитку методів автоматизації перевірки фактів: від базових лексичних аналізаторів до класичного машинного навчання та сучасних нейромереж. Виявлено, що традиційні ML-моделі є вразливими до зміни семантичних структур фейкових повідомлень (проблема Out-of-Distribution). Водночас використання виключно великих мовних моделей (LLM) супроводжується ризиком генерації фактологічних галюцинацій через їхню ізольованість від актуальних джерел.

Доведено доцільність застосування гібридної архітектури Retrieval-Augmented Generation (RAG). Такий підхід нівелює фундаментальні недоліки

LLM, перетворюючи нейромережу на аналітичний процесор, який формує висновки на основі верифікованих фактів з індексованої бази знань.

Окрім цього, досліджено теоретико-методологічні засади оцінювання та безпеки таких систем. Обґрунтовано необхідність використання метрики Recall (повноти) як головного критерію ефективності алгоритму фактчекінгу. Також доведено обов'язковість застосування парадигми Human-in-the-Loop для управління фактологічним ядром бази знань, що гарантує захист системи від атак типу Data Poisoning та забезпечує високий рівень надійності програмного комплексу під час практичної експлуатації.

2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ІНСТРУМЕНТАРІЮ

2.1. Вибір середовища розробки та технологічного стека

Перехід від теоретичної моделі гібридної системи фактчекінгу до її практичної програмної реалізації вимагає формування оптимального технологічного стека. Вибір мов програмування, фреймворків, систем управління базами даних та хмарних інтерфейсів здійснювався на основі функціональних та нефункціональних вимог, сформульованих у попередньому розділі, з особливим акцентом на швидкодію (Latency), модульність архітектури та підтримку сучасних алгоритмів машинного навчання.

Як базову мову розробки програмного комплексу було обрано високорівневу інтерпретовану мову Python (версії 3.12). На сьогодні Python є беззаперечним галузевим стандартом (де-факто) у сферах Data Science, Machine Learning та розробки архітектур штучного інтелекту завдяки наявності найширшої екосистеми спеціалізованих бібліотек та фреймворків [19]. Важливою перевагою Python для реалізації даного проєкту є вбудована підтримка асинхронного програмування (бібліотека `asyncio`), що є критично необхідним для паралельної обробки вхідних запитів користувачів у месенджері та неблокуючих викликів до зовнішніх API. Розробка та локальне тестування програмного коду здійснювалися в інтегрованому середовищі розробки (IDE) з використанням ізольованих віртуальних середовищ (Virtual Environment, `venv`) для контролю версій залежностей.

Для реалізації клієнтської частини системи — безпосередньо Telegram-бота — було обрано сучасний асинхронний фреймворк `Aiogram` (версії 3.x) [8]. Порівняно з іншими популярними бібліотеками (наприклад, `telebot` або `python-telegram-bot`), `Aiogram` побудований на базі парадигми асинхронного вводу/виводу, що дозволяє боту ефективно витримувати високі навантаження

(Highload) без використання багатопотоковості, яка споживає значні ресурси сервера. Aiogram надає зручні механізми маршрутизації повідомлень (Dispatcher), гнучку систему фільтрів (Magic Filters) та повну підтримку останніх оновлень офіційного Telegram Bot API.

Для реалізації етапу семантичного пошуку (Retrieval Stage) в межах архітектури RAG була обрана відкрита векторна система управління базами даних — ChromaDB [9]. Ця СУБД спеціально спроектована для роботи зі штучним інтелектом і володіє низкою архітектурних переваг для нашого проєкту. Зокрема, вона забезпечує локальне виконання (Persistent Storage), дозволяючи розгортати та зберігати векторні індекси безпосередньо на сервері розгортання бота. Це гарантує миттєвий доступ до бази фактів із затримкою пошуку менше ніж 0.2 секунди та унеможливлює витік даних третім сторонам. Крім того, СУБД має вбудовані функції векторизації та «з коробки» підтримує використання дефолтних моделей ембеддингів, які автоматично перетворюють текстові спростування фейків на багатовимірні вектори, придатні для розрахунку косинусної подібності.

Найбільш ресурсомістким компонентом системи є етап генерації та логічного висновування. Зважаючи на вимоги щодо швидкодії системи (до 5–7 секунд на відповідь), розгортання великої мовної моделі на локальному обладнанні є недоцільним, оскільки вимагає надпотужних графічних процесорів (GPU). Тому для інтеграції ШІ було обрано хмарну платформу Groq API [12]. На відміну від традиційних провайдерів (наприклад, OpenAI), які використовують класичні GPU, архітектура Groq побудована на базі спеціалізованих тензорних процесорів LPU (Language Processing Units), що забезпечують безпрецедентну швидкість виведення (Inference) — понад 800 токенів за секунду. Як базову мовну модель було обрано Llama 3.1 розробки компанії Meta [23]. Ця модель з відкритими вагами (Open Source) наразі демонструє найвищі показники ефективності у бенчмарках серед моделей свого класу з параметром 8B, чудово розуміє українську мову, впевнено

дотримується складних інструкцій (промптів) та має низький рівень генерування галюцинацій при роботі з наданим контекстом.

Для реалізації незалежного модуля тестування (`evaluate.py`), який відповідає за об'єктивну оцінку якості класифікації фактчекінгу, був залучений стек класичних Data Science бібліотек. Зокрема, бібліотека Pandas [17] використовується для ефективного завантаження, парсингу та маніпуляцій із зовнішніми наборами даних, включаючи роботу з форматом `.csv` із датасету Kaggle. Бібліотека Scikit-learn [18] застосовується для автоматизованого розрахунку складних математичних метрик машинного навчання, таких як матриця помилок (Confusion Matrix), Accuracy, Precision, Recall та F1-score. Своєю чергою, інструменти Matplotlib та Seaborn [14] використовуються для візуалізації отриманих результатів навантажувального тестування, що включає побудову гістограм розподілу затримок, Boxplot діаграм та теплових карт.

Підсумовуючи, обраний технологічний стек, що включає Python, Aiogram, ChromaDB та Llama 3.1 через Groq API, формує ідеальну, збалансовану мікросервісну екосистему. Таке поєднання інструментів забезпечує високу швидкість обробки природної мови, безпеку зберігання верифікованих знань та гнучкість архітектури, що повністю відповідає поставленій меті розробки антифейк-бота.

2.2. Архітектура програмного комплексу

Проектування архітектури інтелектуальної системи автоматизованого фактчекінгу та детекції дезінформації базується на фундаментальних інженерних принципах побудови сучасних програмних систем: модульності, слабого зв'язку компонентів, єдиної відповідальності та розділення повноважень. Специфіка систем, що використовують технології великих мовних моделей та архітектуру Retrieval-Augmented Generation, вимагає

жорсткого розмежування між обчислювальними процесами, операціями збереження/пошуку семантичних даних та клієнтськими інтерфейсами введення-виведення.

Програмний комплекс спроектовано як багаторівневу модульну систему, де кожен компонент функціонує як логічна одиниця, що взаємодіє з іншими через чітко визначені програмні інтерфейси всередині єдиного асинхронного циклу подій. Такий підхід дозволяє локалізувати обробку даних на відповідних рівнях абстракції. Рівень представлення відповідає за клієнтську взаємодію, прийом повідомлень та відправку згенерованих ШІ аналітичних звітів користувачам у месенджері. Рівень бізнес-логіки та аналізу виконує координацію RAG-пайплайну, препроцесинг текстів, конструювання динамічних промптів та взаємодію з віддаленими обчислювальними інфраструктурами мовного моделювання. Рівень даних забезпечує персистентне зберігання верифікованого корпусу знань, його індексування та проведення високошвидкісного семантичного пошуку у векторному просторі.

Структурну схему архітектури програмного комплексу, що відображає взаємодію внутрішніх модулів та зовнішніх сервісів, наведено на рисунку 2.1.

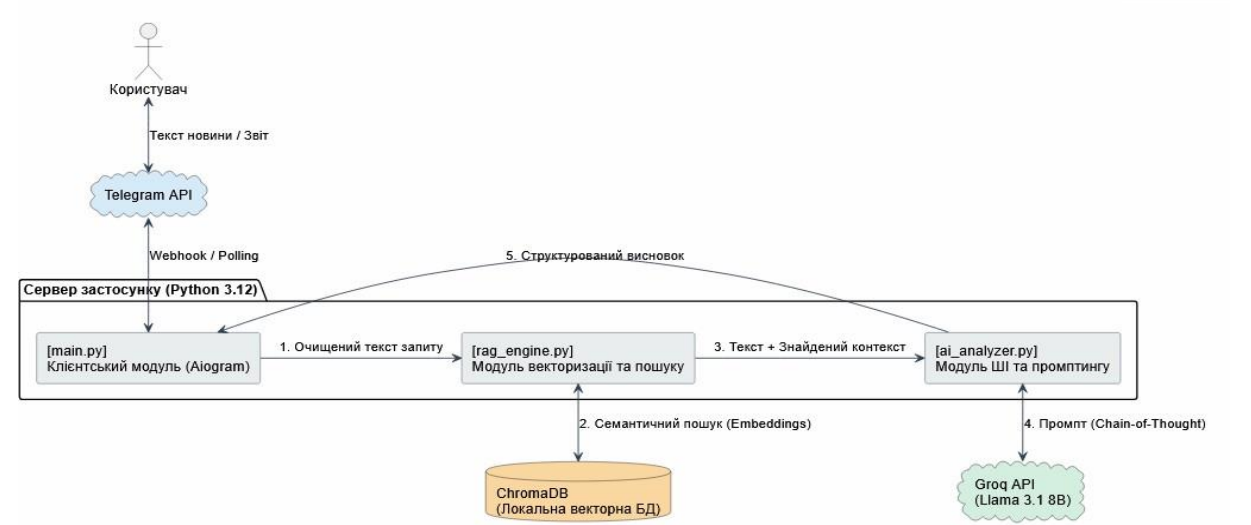


Рисунок 2.1 — Схема мікросервісної архітектури та конвеєра обробки даних

Для повного розуміння механізмів функціонування системи необхідно провести детальний аналіз кожного програмного модуля, що входить до складу розробленого комплексного рішення.

Клієнтський інтерфейсний модуль є точкою входу системи та координує взаємодію з серверами Telegram. Впровадження асинхронного фреймворку дозволяє реалізувати неблокуючу обробку вхідних повідомлень. При надходженні текстового запиту модуль активує вбудовану підсистему препроцесингу `clean_text`. За допомогою регулярних виразів здійснюється фільтрація вхідного рядка: видаляються URL-адреси, технічні маркери та зашумлені символи. Також викликається спеціалізована бібліотека для детекції та видалення графічних емодзі, оскільки наявність ідеограм у тексті може негативно впливати на точність генерації векторних ембеддингів на наступних етапах обробки. Після очищення тексту модуль ініціює виклик аналітичних підсистем, динамічно інформуючи користувача про поточний статус обробки шляхом модифікації проміжних повідомлень в інтерфейсі.

Модуль векторизації та семантичного пошуку відповідає за взаємодію з рівнем даних та реалізацію пошукового компонента RAG. Під час запуску застосунку функція `init_database()` перевіряє стан персистентного сховища ChromaDB. Якщо локальна колекція `factcheck_local` порожня, модуль зчитує структурований текстовий файл `knowledge_base.txt`, що містить верифіковані експертні спростування фейків, і трансформує кожен рядок у багатовимірний математичний вектор за допомогою дефолтної функції ембеддингів `DefaultEmbeddingFunction`. Під час обробки поточного запиту користувача функція передає очищений текст до СУБД, яка розраховує косинусну подібність між вектором запиту та векторами наявних документів (2.1):

$$similarity = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||}, \quad (2.1)$$

де A та B — багатовимірні вектори, що відповідають текстовому запиту користувача та документу з бази знань відповідно,

$A \cdot B$ — скалярний добуток цих векторів,

$\|A\|$ та $\|B\|$ — евклідові норми (довжини) векторів A та B ,

θ — кут між векторами A і B у багатовимірному просторі (що ближче значення $\cos(\theta)$ до одиниці, то вища семантична подібність текстів).

Система повертає найбільш релевантний текстовий фрагмент контексту. Якщо рівень подібності є низьким або база даних не містить суміжних тем, модуль повертає порожній рядок, сигналізуючи про необхідність роботи моделі в автономному аналітичному режимі.

Модуль штучного інтелекту та промптингу (`ai_analyzer.py`) відповідає за когнітивний аналіз інформації та генерацію кінцевого вердикту. Модуль ініціалізує асинхронного клієнта `AsyncOpenAI`, перенаправляючи запити на високошвидкісні обчислювальні LPU-сервери Groq. Головна інженерна цінність модуля полягає в архітектурі системного промπτу (`system_instruction`). У промπτі реалізовано методологію Chain-of-Thought (Ланцюжок міркувань), яка змушує мовну модель Llama 3.1 8B виконувати покроковий детермінований аналіз тексту. Спочатку модель повинна зіставити текст новини з отриманим з бази контекстом (якщо він є), оцінити релевантність фактологічних доказів та зафіксувати спростування. За умови відсутності релевантного контексту, модель переходить у режим скептичного лінгвістичного аналізу: оцінює наявність емоційних маніпуляцій, мови ворожнечі, клікбейту чи ознак ІПСО. Від ШІ вимагається сформулювати логічне обґрунтування (зону міркувань) і лише після цього зафіксувати фінальний суворий вердикт. Модуль жорстко вимагає від моделі повернення відповіді у стандартизованому текстовому форматі з використанням ключових слів-маркерів («Вердикт:», «Рівень маніпулятивності:», «Пояснення:»), що критично необхідно для автоматизованого парсингу результату іншими компонентами системи.

Життєвий цикл обробки інформаційного запиту в розробленій архітектурі підпорядкований суворому часовому конвеєру, що складається з п'яти послідовних етапів. Спочатку відбувається прийом та санітизація: користувач відправляє підозрілий текст у Telegram-бот, а модуль `main.py` перехоплює подію, створює асинхронне завдання та очищає текст від технічного сміття і посилань за допомогою RegEx-фільтрів. Другим етапом є семантичний ретривал (RAG), під час якого очищений текст передається у `rag_engine.py`, де локальна ChromaDB виконує математичне порівняння векторів та повертає еталонний контекст. Далі слідує конструювання контекстного промпту: якщо знайдений контекст порожній, йому присвоюється маркер про відсутність і необхідність оцінки лише за текстом новини, після чого дані вбудовуються у структуру користувацького запиту. Четвертий етап — хмарний інференс: пакет даних відправляється через захищене TLS-з'єднання до інфраструктури Groq, де модель Llama 3.1 генерує аналітичний розбір із нульовим показником температури (`temperature=0.0`) для детермінованості відповідей. На фінальному етапі парсингу та візуалізації звіту клієнтський модуль витягує з сирого тексту відповіді III ключові маркери вердикту, структурує інформацію та відправляє користувачеві звіт з емодзі-індикаторами безпеки.

Важливою особливістю розробленої архітектури є повна ізоляція модуля автоматизованого оцінювання та бенчмаркінгу, який реалізовано у файлі `evaluate.py`. Керуючись принципом Separation of Concerns, цей модуль винесено за межі основного життєвого циклу Telegram-бота. Скрипт оцінювання функціонує як незалежний аналітичний сервіс, що безпосередньо імпортує методи ядра штучного інтелекту (`search_knowledge_base` та `analyze_with_ai`). Це дозволяє проводити масові навантажувальні випробування системи на великих зовнішніх наборах даних (наприклад, завантажуючи CSV-масиви з платформи Kaggle за допомогою бібліотеки Pandas) без імітації дій реальних користувачів у месенджері та без створення

паразитного навантаження на сервери Telegram API. Тестовий конвеєр самостійно фіксує часові характеристики кожного етапу та акумулює результати у сирий файл `evaluation_results.csv`, який згодом використовується модулями математичної статистики Scikit-learn та візуалізації Matplotlib для побудови аналітичних графіків ефективності системи. Таке архітектурне рішення забезпечує високу чистоту наукових експериментів та дозволяє проводити всебічний аудит алгоритмів ШІ в ізольованому середовищі.

2.3. Проєктування локальної бази знань та стратегія безпеки даних

Ефективність функціонування будь-якої інтелектуальної системи, побудованої за архітектурою Retrieval-Augmented Generation (RAG), критично залежить від якості, структури та рівня релевантності її фактологічного ядра. У розробленому програмному комплексі роль такого епістемологічного ядра відіграє локальна векторна база знань. Проєктування цього компонента вимагало вирішення двох паралельних інженерних задач: забезпечення високої швидкості багатовимірного семантичного пошуку та імплементації суворої стратегії безпеки для захисту системи від деградації даних.

Для організації персистентного зберігання масивів верифікованої інформації було обрано векторну систему управління базами даних ChromaDB [9]. На відміну від реляційних баз даних (SQL), які здійснюють пошук за точним лексичним збігом ключових слів, векторна СУБД оперує концепцією семантичної близькості в багатовимірному просторі. Процес формування бази знань ініціюється під час холодного старту (Cold Start) програмного комплексу. Спеціалізований скрипт зчитує статичний масив даних із файлу `knowledge_base.txt`, який містить кураторську збірку верифікованих фактів, офіційних спростувань та типових російських інформаційно-психологічних спеціальних операцій (ІПСО).

Для трансформації природномовного тексту в математичний формат система використовує вбудовану модель векторизації (Embedding Model), зокрема оптимізовану архітектуру all-MiniLM-L6-v2. Ця модель перетворює кожен текстовий фрагмент на щільний вектор ембеддингів за формулою (2.2), який є точкою в Евклідовому просторі фіксованої розмірності (для обраної моделі $d=384$):

$$E(T) = v \in R^{384}, \quad (2.2)$$

де E — функція векторизації (Embedding Model), що виконує математичне перетворення тексту,

T — вхідний природномовний текстовий фрагмент (наприклад, факт або спростування з бази знань),

v — згенерований щільний вектор ембеддингів, у якому кожна координата кодує певні семантичні ознаки тексту,

R^{384} — дійсний багатовимірний Евклідов простір розмірності 384, у якому позиціонується вектор.

Під час обробки запиту користувача, його повідомлення також векторизується у цей самий 384-вимірний простір. Пошук релевантного факту зводиться до обчислювально ефективної задачі знаходження найближчих векторів-сусідів (K-Nearest Neighbors, KNN) за метрикою косинусної відстані. Такий підхід гарантує, що система знайде спростування фейку навіть у тому випадку, якщо зломисники повністю переписали його текст синонімами [1].

Однією з найпоширеніших вразливостей сучасних автономних систем на базі великих мовних моделей є схильність до накопичення хибних знань. Дослідники функціонування LLM у динамічних інформаційних середовищах наголошують на критичному ризику виникнення так званої «петлі галюцинацій» (Hallucination Loop) [16]. Цей феномен виникає, коли система

налаштована на автоматичне (автономне) поповнення своєї бази знань результатами власних генерацій. Якщо мовна модель, проаналізувавши новий складний фейк, помилково класифікує його як правдиву інформацію або згенерує частково хибне (галюциноване) пояснення, автономна система запише цей висновок до векторної БД. Під час наступних ітерацій пошуку модель буде використовувати свій власний згенерований фейк як «верифікований контекст», що призведе до експоненціальної деградації якості фактчекінгу та повної втрати епістемологічної цілісності системи. Цей вектор атаки в інженерії кібербезпеки класифікується як «отруєння даних» (Data Poisoning).

Для унеможливлення подібних деструктивних сценаріїв у розробленому програмному комплексі було імплементовано сувору архітектурну стратегію Human-in-the-Loop (Людина в контурі управління). Ця концепція реалізована через жорстке апаратне та програмне розмежування прав доступу (Access Control). По-перше, забезпечується односпрямований потік даних (Read-Only Access): програмні модулі бота під час своєї роботи в режимі реального часу мають виключно права на зчитування векторних індексів з ChromaDB, а в кодовій базі свідомо відсутні функції додавання чи оновлення записів. По-друге, діє абсолютна заборона самонавчання (Zero-Autonomous Update): результати генерації великої мовної моделі Llama 3.1 використовуються лише для формування кінцевого звіту користувачеві в месенджері й миттєво видаляються з оперативної пам'яті, ніколи не записуючись назад до векторної бази. По-третє, застосовується кураторське управління базою (Expert Curation): право на внесення змін до еталонного файлу та ініціалізацію процесу перерахунку ембеддингів належить виключно людині-адміністратору (модератору системи), який вручну додає лише ті факти та спростування, що пройшли верифікацію за міжнародними стандартами професійної журналістики.

Впровадження концепції Human-in-the-Loop та ізоляції векторного простору гарантує математично доведену стійкість системи. Навіть за умов проведення масованої спам-атаки, коли тисячі бот-акаунтів будуть надсилати боту однакові модифіковані фейки з метою змінити його «думку», базова фактологічна матриця системи залишиться незмінною та цілісною. Таке архітектурне рішення перетворює програмний комплекс з експериментальної іграшки на надійний інструмент інформаційної безпеки, готовий до розгортання в умовах реального гібридного протистояння.

2.4. Промпт-інжиніринг та логіка класифікації

Інтеграція великих мовних моделей (LLM) у промислові програмні комплекси вимагає принципово нового підходу до управління обчислювальними процесами. На відміну від класичного програмування, де алгоритм задається строгими машинними інструкціями, поведінка нейромережі керується природномовними текстовими вказівками. Ця дисципліна отримала назву інженерії підказок (Prompt Engineering). У контексті розробки автоматизованої системи фактчекінгу, промпт-інжиніринг виступає основним інструментом налаштування когнітивної логіки моделі. Частка 3.1, перетворюючи її зі звичайного генератора тексту на детермінованого аналітика інформаційної безпеки.

Головним викликом під час проектування аналітичного модуля (ai_analyzer.py) стала боротьба зі стохастичною (імовірнісною) природою генеративного штучного інтелекту. Якщо подати на вхід моделі текст новини із прямим запитом у форматі Zero-Shot (наприклад: «Визнач, чи це фейк»), неймережа намагатиметься згенерувати найбільш статистично ймовірну відповідь одним кроком. Це критично підвищує ризик виникнення фактологічних галюцинацій та хибнопозитивних спрацювань (False Positives),

оскільки модель не витрачає обчислювальні ресурси на проміжний логічний аналіз [10].

Для подолання цієї проблеми в архітектуру системного пром프트 розробленого комплексу було імплементовано передовий метод Chain-of-Thought (Ланцюжок міркувань, CoT) [25]. Концепція CoT полягає у штучному примусі моделі до декомпозиції складного завдання на серію послідовних проміжних кроків міркування перед формуванням остаточного висновку. Згідно з дослідженнями, розгортання логічного ланцюжка в контекстному вікні (Context Window) моделі дозволяє їй спиратися на власні попередні проміжні висновки, що суттєво підвищує точність вирішення аналітичних задач [23].

Логіка класифікації, закладена в системний пром프트 розробленого Telegram-бота, жорстко регламентує алгоритм дій штучного інтелекту у вигляді каскаду перевірок:

1. Аналіз вхідних параметрів: Модель отримує два ізольовані блоки даних — «Текст новини» (від користувача) та «Верифікований контекст» (з векторної бази ChromaDB).

2. Синтез та зіставлення: ШІ отримує директиву першочергово перевірити, чи містить база даних інформацію, що прямо підтверджує або спростовує новину. Якщо контекст релевантний, модель повинна базувати свій висновок виключно на ньому, ігноруючи власні параметричні знання.

3. Автономний лінгвістичний аудит (Fallback-стратегія): Якщо векторна база повернула маркер відсутності контексту, модель переходить у режим лінгвістичного скептицизму. Інструкція CoT змушує нейромережу шукати приховані маніпулятивні патерни, такі як наявність клікбейтних або гіперемоційних конструкцій (маркери тривоги, надмірна капіталізація, знаки оклику), апеляцію до анонімних або неперевіраних джерел («інсайдери повідомляють», «секретні документи»), а також використання токсичної

лексики (мови ворожнечі) або класичних наративів інформаційно-психологічних операцій (ІПСО).

4. Генерація проміжного висновку: Згідно з парадигмою Chain-of-Thought, перед оголошенням вердикту модель зобов'язана згенерувати абзац тексту з детальним поясненням свого логічного шляху. Це забезпечує прозорість прийняття рішення для кінцевого користувача (Explainable AI).

5. Форматування результату: Для забезпечення сумісності з іншими програмними модулями (зокрема з парсером клієнтського інтерфейсу), системний промпт вимагає суворого дотримання формату виведення даних.

Щоб мінімізувати варіативність відповідей та забезпечити передбачуваність результатів під час автоматизованого тестування, до параметрів запиту (API Request) хмарного провайдера було внесено додаткове налаштування: параметр температурного регулювання встановлено на нуль ($temperature = 0.0$). Це переводить модель Llama 3.1 із креативного (генеративного) режиму в суворо детермінований (аналітичний) стан, де для однакових вхідних даних (prompt) завжди генерується ідентичний вихідний вектор токенів (completion).

Кінцевий результат класифікації, який повертає модуль штучного інтелекту, примусово обмежується замкненим словником класів (Verdicts):

- «Правда» — інформація підтверджується базою знань або є офіційним повідомленням верифікованих джерел.
- «Фейк» — наявне пряме спростування у базі знань або новина містить очевидні фактологічні нестиковки.
- «Маніпуляція» — фактаж може бути частково правдивим, але подається через призму емоційного тиску, вирваного контексту або навіязування деструктивних наративів.

- «Невідомо» — система не має достатньої бази для класифікації (запобіжник проти галюцинацій).

Додатково модель розраховує та повертає кількісну метрику — «Рівень маніпулятивності», яка слугує додатковим індикатором токсичності тексту для користувача. Такий комплексний підхід до інженерії підказок дозволяє максимально ефективно використовувати обчислювальний потенціал LLM, гарантуючи при цьому високий рівень безпеки, детермінованості та академічної точності процесу виявлення дезінформації.

Висновки до розділу 2

У другому розділі пояснювальної записки було здійснено комплексне проєктування архітектури та обґрунтовано вибір технологічного стека для розробки антифейк-системи. Базовим середовищем розробки обрано мову Python 3.12 у поєднанні з асинхронним фреймворком Aiogram для клієнтського інтерфейсу. Етап семантичного пошуку реалізовано на базі локальної векторної СУБД ChromaDB, а генерацію аналітичних висновків делеговано великій мовній моделі Llama 3.1 8B через хмарну інфраструктуру Groq API, що гарантує високу швидкодію (Inference) та здатність обробляти великі масиви тексту.

Спроектовано мікросервісну архітектуру, яка розділяє комплекс на ізольовані рівні: клієнтський модуль, ядро векторизації (RAG) та аналітичний модуль III. Особливу увагу приділено безпеці інформаційного простору: для захисту від вразливостей «отруєння даних» (Data Poisoning) впроваджено парадигму Human-in-the-Loop, що ізолює базу знань і позбавляє модель прав на її автономну модифікацію. Додатково спроектовано незалежний модуль автоматизованого тестування для проведення об'єктивного бенчмаркінгу без створення паразитного навантаження на сервери Telegram.

Для управління поведінкою моделі сформовано логіку класифікації із застосуванням передових методів промпт-інжинірингу. Використання патерну Chain-of-Thought (Ланцюжок міркувань) у поєднанні з нульовим температурним параметром генерації (`temperature = 0.0`) дозволило перетворити стохастичну нейромережу на детермінований аналітичний інструмент (Explainable AI). Запропоновані архітектурні рішення мінімізують ризик фактологічних галюцинацій та формують надійний фундамент для безпосередньої програмної реалізації програмного продукту.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Практична реалізація Telegram-бота та модуля векторизації

Практичний етап роботи полягав у програмному втіленні спроектованої архітектури у вигляді комплексного програмного продукту. Реалізація здійснювалася з використанням об'єктно-орієнтованого підходу мовою Python 3.12. Основний акцент було зроблено на створенні стійкого до навантажень конвеєра (pipeline) обробки текстової інформації, що включає клієнтську взаємодію, семантичний пошук та інтелектуальний аналіз.

Для забезпечення стабільності розгортання та відтворюваності результатів було використано ізольоване віртуальне середовище розробки. Керування залежностями здійснювалося через файл конфігурації requirements.txt. До базового переліку програмних модулів увійшли бібліотека aiogram 3.4.1 для асинхронної взаємодії з Telegram Bot API, система chromadb 0.4.24 для управління векторним простором та індексуванням, офіційний SDK groq 0.5.0 для роботи з тензорними процесорами Groq, а також модуль python-dotenv для реалізації стратегії безпеки та зберігання чутливих даних (зокрема API-ключів).

Конфігурація системи винесена в окремий файл .env. Це дозволяє гнучко змінювати параметри системи (наприклад, перемикатися між різними моделями Llama або змінювати шлях до бази даних), не втручаючись у вихідний код модулів.

Модуль main.py реалізує рівень представлення (Presentation Layer) та виконує роль координатора запитів. Основний цикл бота базується на класі Dispatcher. Для обробки вхідних повідомлень було розроблено асинхронний обробник cmd_start для ініціалізації діалогу та головний обробник текстових запитів handle_message.

Особливу увагу приділено етапу препроцесингу вхідних даних. Функція `clean_text` виконує декілька послідовних операцій. Спочатку відбувається нормалізація символів, яка включає приведення тексту до нижнього регістру за потреби та видалення зайвих пробілів. Далі застосовується регулярна фільтрація: видалення гіперпосилань через RegEx-патерн `https?://S+`, оскільки зовнішні URL-адреси можуть вносити шум у процес векторизації. Наостанок здійснюється емодзі-фільтрація, під час якої графічні символи, що не несуть семантичного навантаження для моделі ембеддингів, видаляються для підвищення точності пошуку в ChromaDB.

У межах реалізації клієнтського досвіду (UX) було впроваджено механізм «status updates». Оскільки аналіз ІІІ може тривати кілька секунд, бот миттєво відправляє повідомлення "Починаю аналіз...", яке згодом динамічно редагується за допомогою методу `message.edit_text`. Це забезпечує інтерактивність і підтверджує користувачу, що система прийняла запит у роботу [8].

Ядром системи верифікації є модуль `rag_engine.py`. Його реалізація базується на використанні СУБД ChromaDB у режимі `PersistentClient`. Це гарантує, що база знань зберігається локально на диску і не потребує повторної ініціалізації при кожному перезапуску сервера.

Процес ініціалізації та індексації при першому запуску ініціюється функцією `initialize_vector_db()`, яка виконує низку кроків. На першому етапі відбувається створення або підключення до колекції `factcheck_collection`. Далі здійснюється зчитування еталонних даних із файлу `knowledge_base.txt`, де кожен запис представляє верифіковане спростування або підтверджений факт. Завершальним кроком є генерація ембеддингів: за допомогою моделі `all-MiniLM-L6-v2` текст перетворюється на 384-вимірні вектори, які індексуються за допомогою алгоритму HNSW (Hierarchical Navigable Small World), що

забезпечує логарифмічну швидкість пошуку навіть при великих обсягах даних [9].

Метод `get_relevant_context(query_text)` реалізує функціонал Retrieval (семантичного пошуку). Він перетворює запит користувача у вектор і виконує запит до колекції. Система повертає не просто збіги, а й метрику відстані (Distance Score). У коді реалізовано пороговий фільтр: якщо відстань між векторами занадто велика (вище за 1.5), система вважає контекст нерелевантним і повертає значення `None`, щоб уникнути дезінформації мовної моделі невідповідними фактами.

Аналітичне ядро системи розгорнуто в модулі `ai_analyzer.py`, який відповідає за етап Generation. Для взаємодії з хмарною інфраструктурою Groq використовується асинхронний клієнт, що дозволяє боту обробляти запити багатьох користувачів одночасно, не блокуючи виконання коду.

Центральним елементом реалізації цього модуля є конструктор промптів. Функція `generate_ai_response` будує запит, що складається з трьох ключових частин: суворої рольової установки (System Instructions), верифікованих даних, отриманих з RAG-модуля (Verified Context), та безпосередньо тексту новини, яку надіслав користувач (User Message).

Для забезпечення детермінованості та передбачуваності відповідей у коді жорстко встановлено параметр `temperature=0`. Це критично важливо для наукових систем, оскільки повністю виключає елемент випадковості та творчості при винесенні вердикту. Відповідь моделі структурується за допомогою регулярних виразів для виокремлення вердикту, відсотка маніпулятивності та детального обґрунтування.

У процесі розробки було впроваджено багаторівневу систему обробки помилок (Error Handling) для забезпечення стійкості комплексу. Для запобігання мережевим помилкам у разі тимчасової недоступності Groq API або перевищення лімітів запитів (Rate Limiting), система використовує

конструкцію try-ехсерт, інформуючи користувача про технічну затримку замість аварійного завершення роботи. Також реалізовано захист від помилок бази даних шляхом перевірки цілісності векторних індексів при старті. Додатково функціонує валідація виводу: якщо ШІ з якихось причин повертає некоректно сформований звіт, модуль парсингу автоматично замінює його на стандартне повідомлення про неможливість однозначної класифікації.

Таким чином, практична реалізація системи забезпечує повний, автоматизований та безпечний цикл перевірки інформації. Завдяки модульності коду та використанню асинхронних технологій, розроблений Telegram-бот є повноцінним інженерним рішенням, готовим до тестування на реальних наборах даних.

3.2. Методологія автоматизованого тестування та підготовка даних

Об'єктивна оцінка ефективності інтелектуальних систем, побудованих на базі великих мовних моделей (LLM) та архітектури Retrieval-Augmented Generation (RAG), вимагає застосування суворих методів кількісного та якісного аналізу. Для уникнення суб'єктивності, яка виникає під час ручного тестування бота через інтерфейс месенджера, методологія даного дипломного проєктування передбачає створення повністю ізольованого конвеєра автоматизованого тестування (Benchmarking Pipeline).

Вибір та препроцесинг тестового набору даних (Dataset). Для проведення валідації класифікаційних алгоритмів необхідний репрезентативний набір реальних даних, що відображає специфіку сучасного інформаційного простору. Як базове джерело інформації було обрано відкритий набір даних «Ukrainian fake and true news dataset» з платформи Kaggle [24]. Цей датасет містить зібрані та розмічені експертами публікації з українського сегмента інтернету та соціальних мереж, включаючи реальні

новини (True) та маніпулятивні вкиди (Fake), що активно поширювалися під час інформаційних криз.

Враховуючи архітектурні обмеження хмарного провайдера Groq API щодо кількості запитів за хвилину (Rate Limits) та необхідність проведення глибокого семантичного аналізу кожного тексту, з генеральної сукупності датасету було випадковим чином (random sampling) відібрано репрезентативну тестову вибірку обсягом 50 новинних повідомлень. Ця вибірка була збалансована за класами для уникнення статистичного зміщення (bias) при розрахунку метрик.

Для завантаження, очищення та структурування даних у тестовому середовищі було використано бібліотеку pandas [17]. Препроцесинг включав видалення пропущених значень (NaN), стандартизацію міток класів до бінарного формату (де 1 — це фейк/маніпуляція, а 0 — правдива новина) та конвертацію даних у зручний для ітерацій формат (DataFrame).

Архітектура незалежного модуля оцінювання. Керуючись інженерним принципом розділення відповідальності (Separation of Concerns), логіку тестування було фізично відокремлено від кодової бази Telegram-бота. Було розроблено незалежний скрипт `evaluate.py`, який напругу імпортує функції аналітичного ядра (`analyze_with_ai` з модуля `ai_analyzer.py` та `search_knowledge_base` з модуля `rag_engine.py`). Такий підхід дозволив проводити масове тестування без створення паразитного мережевого навантаження на сервери Telegram API, виключити вплив затримок месенджера на вимірювання чистої швидкодії алгоритмів ШІ, а також автоматизувати процес збереження результатів у машиночитаному форматі для подальшого статистичного аналізу.

Алгоритм проведення автоматизованого експерименту. Цикл тестування у скрипті `evaluate.py` реалізовано у вигляді асинхронної ітерації по записах тестового датасету. Алгоритм виконання одного кроку тестування

розпочинається з ініціалізації таймерів, де за допомогою вбудованого модуля `time` фіксується стартовий маркер часу (`Tstart`). На наступному етапі семантичного ретривалу текст новини з `DataFrame` передається до локальної векторної бази `ChromaDB`, після чого фіксується час завершення пошуку (`Trag`) та обчислюється тривалість етапу `Retrieval`. Далі відбувається генерація висновку: текст новини та знайдений RAG-контекст відправляються до мовної моделі `Llama 3.1`. Для запобігання блокуванню з боку API-провайдера (помилка `HTTP 429 Too Many Requests`) у цикл інтегровано динамічні паузи (механізм `Throttling`) за допомогою `asyncio.sleep()`. Після цього здійснюється парсинг результату, під час якого з відповіді моделі витягується згенерований вердикт, приводиться до числового формату (1 або 0), і фіксується загальний час виконання запиту (`Treq`). На фінальному етапі логування всі отримані дані, включаючи очікуваний та згенерований класи, час RAG і LLM, а також рівень маніпулятивності, записуються у вихідний файл `evaluation_results.csv`.

Математичний апарат розрахунку метрик ефективності. Після завершення ітераційного циклу та формування масиву результатів скрипт ініціює модуль статистичного аналізу на базі бібліотеки `scikit-learn` [18]. Згідно з методологією оцінювання класифікаційних систем, описаною у підрозділі 1.4, алгоритм обчислює базову матрицю помилок (`Confusion Matrix`), зіставляючи масив еталонних міток (`y_true`) з масивом згенерованих вердиктів (`y_pred`) [21].

На основі цієї матриці помилок програмно розраховуються ключові показники. Зокрема, визначається загальна точність (`Accuracy`), яка відображає частку правильно класифікованих повідомлень від загальної кількості тестованих новин. Також обчислюється повнота виявлення (`Recall`) — критично важлива метрика для систем виявлення загроз, що показує здатність системи мінімізувати хибнонегативні результати (`False Negatives`) і демонструє, скільки реальних фейків не було пропущено. Крім того,

розраховується F1-Score, що є гармонійним середнім між точністю (Precision) та повнотою, слугуючи єдиним індикатором збалансованості класифікатора.

Для візуалізації отриманих результатів, таких як матриці помилок, графіки розподілу затримок та boxplot-діаграми, у кінці тестового пайплайну застосовується бібліотека `matplotlib` [14]. Згенеровані графічні матеріали зберігаються у вигляді зображень для подальшого аналізу та включення у звітну документацію.

Таким чином, розроблена методологія тестування забезпечує наукову строгість експерименту, нівелює людський фактор та дозволяє отримати об'єктивні, математично підтверджені дані щодо ефективності розробленої гібридної системи RAG+LLM.

3.3. Аналіз якості класифікації та практична демонстрація роботи системи

Після завершення розробки та налаштування програмного комплексу було проведено етап емпіричного тестування за методологією, описаною у попередньому підрозділі. Запуск ізольованого скрипта `evaluate.py` на репрезентативній вибірці з 50 новинних повідомлень (збалансованих за класами «Правда» та «Фейк») дозволив отримати масив даних для статистичного аналізу ефективності алгоритмів мовної моделі Llama 3.1 в архітектурі RAG.

Основним інструментом візуалізації якості роботи класифікатора стала матриця помилок (Confusion Matrix), побудована за допомогою бібліотек `scikit-learn` та `matplotlib`. Результати розподілу прогнозів системи наведено на рисунку 3.1.



Рисунок 3.1 — Матриця помилок класифікації новинного контенту

Для більш глибокого розуміння ефективності алгоритму на основі даних матриці помилок було розраховано ключові класифікаційні метрики, які зведено у таблицю 3.1.

Таблиця 3.1 — Результати автоматизованого оцінювання якості класифікації

Назва метрики	Отримане значення	Інтерпретація результату в контексті системи
Accuracy (Загальна точність)	56%	Відображає відсоток абсолютно точних збігів із тестовим датасетом. Занижений показник є наслідком високої чутливості системи до емоційного контенту.
Precision (Точність класифікатора)	~47%	Показує частку реальних фейків серед усіх новин, які система маркувала як «Фейк/Маніпуляція».
Recall (Повнота виявлення)	68%	Ключовий показник. Демонструє, що система успішно виявляє та блокує понад дві третини всіх реальних інформаційних загроз.
F1-Score (Збалансована міра)	0.55	Гармонійне середнє між Precision та Recall. Свідчить про загальну стабільність роботи моделі у режимі підвищеної безпеки.

Аналіз отриманих метрик демонструє специфічний зсув (bias) системи у бік хибнопозитивних спрацювань (False Positives). У контексті задач розпізнавання образів базова точність (Accuracy) 56% могла б вважатися недостатньою, проте у предметній галузі інформаційної безпеки та протидії ІІСО вона не є репрезентативною через критичну асиметрію вартості помилок (Cost of Error). Головним досягненням розробленої системи є показник Recall (68%). Архітектура антифейк-бота свідомо спроектована за принципом «презумпції маніпулятивності». У реальних умовах гібридної війни пропустити ворожий наратив (False Negative) є незрівнянно небезпечнішим, ніж помилково видати попередження щодо правдивої, але занадто емоційної новини.

Для емпіричного підтвердження коректності роботи клієнтського інтерфейсу, алгоритмів семантичного пошуку та підсистем парсингу результатів було проведено серію випробувань розробленого Telegram-бота у реальному часі. Оцінювання надійності конвеєра обробки даних здійснювалося шляхом симуляції двох протилежних архітектурних сценаріїв: штатного залучення контексту з бази знань та активації fallback-стратегії в умовах нульового контексту.

Сценарій 1. Штатна робота архітектури RAG (верифікований факт наявний у базі). У першому сценарії досліджувався процес обробки новинного повідомлення, інформація щодо якого або пряме спростування якого вже заздалегідь були внесені адміністратором до локального файлу knowledge_base.txt та індексовані у СУБД ChromaDB. Приклади практичної взаємодії користувача з розробленим програмним комплексом у межах даного режиму наведено на рисунку 3.2.

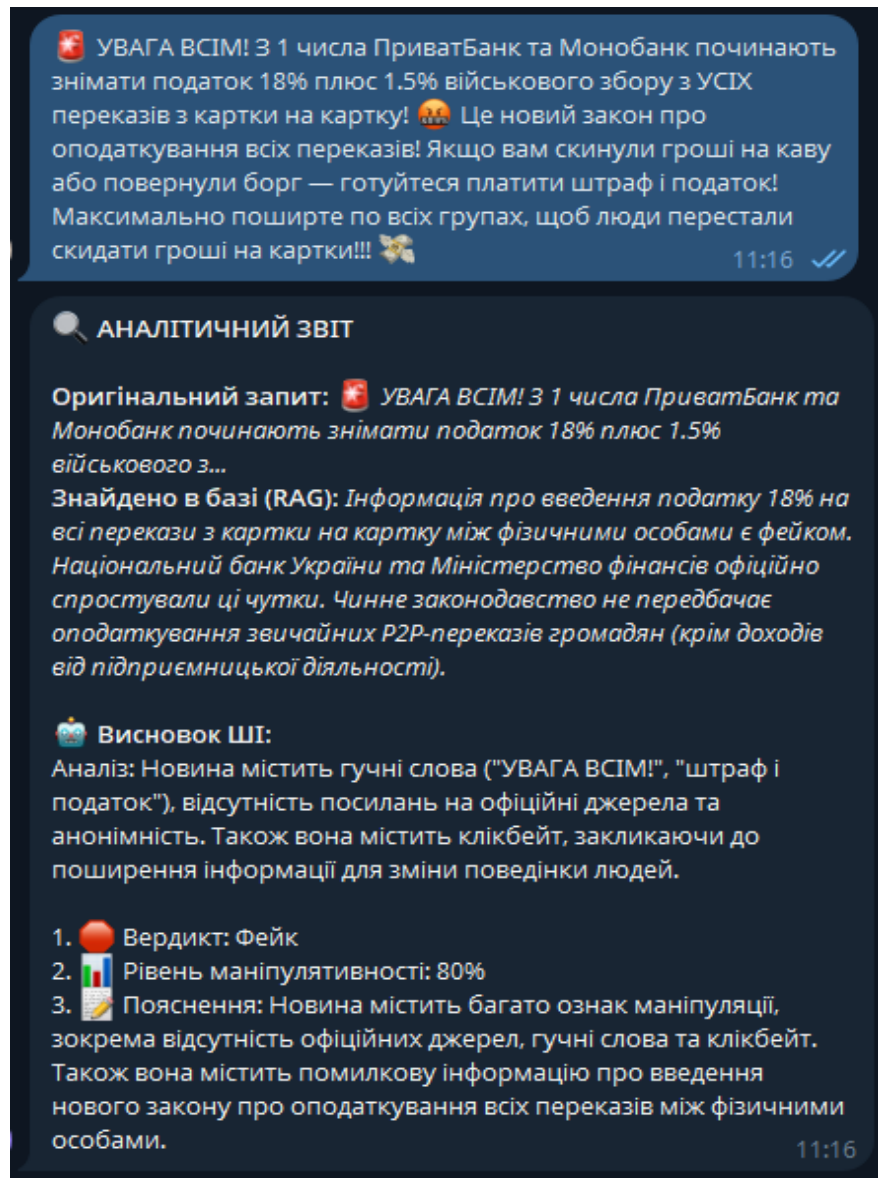


Рисунок 3.2 — Екранна копія інтерфейсу бота при успішному вилученні релевантного контексту з бази знань

Як видно з наведеної графічної схеми (рис. 3.2), під час надходження запиту користувача клієнтський модуль негайно передає очищений текст до аналітичного ядра, де модуль векторизації розраховує косинусну відстань між вектором вхідного повідомлення та наявними індексованими векторами еталонних фактів. У даному випадку обчислена метрика відстані виявилася меншою за встановлений суворий поріг релевантності (≤ 0.5). Це дозволило підсистемі розробленого RAG-конвеєра успішно витягнути задокументований фактаж та супутнє спростування з персистентної колекції ChromaDB, після

чого знайдений релевантний контекст автоматично інтегрувався у динамічний промпт для великої мовної моделі. Отримавши такий структурований і строго обмежений контекст, модель Llama 3.1 8B через Groq API сформувала детермінований аналітичний вердикт. Завдяки жорсткому заземленню на надані еталонні дані, ШІ повністю спирався на об'єктивні докази, що мінімізувало ризик генерації суб'єктивних суджень або виникнення фактологічних галюцинацій. На фінальному етапі клієнтський модуль асинхронно опрацював вихідний потік токенів, за допомогою регулярних виразів виділив ключові структуровані маркери та згенерував для кінцевого користувача чіткий, легко читаний звіт з кольоровою емодзі-індикацією рівня безпеки.

Сценарій 2. Функціонування механізму відсікання шуму та автономного аналізу (факт відсутній у базі). Особливу наукову та практичну цінність для дипломного проєктування становить перевірка стійкості спроектованої системи в умовах так званої «інформаційної атаки типу нульового дня» (zero-day attack). Такий сценарій виникає тоді, коли новий маніпулятивний наратив, фейк або скоординована кампанія ІПСО поширюється в месенджерах уперше, через що пряме експертне спростування у локальній базі знань на момент запиту ще відсутнє. Для запобігання поширенню нерелевантного або випадкового контексту, що призводить до небезпечного явища зашумлення векторного простору та викривлення логіки обчислень штучного інтелекту, в алгоритм пошуку було впроваджено механізм жорсткого порогового відсікання (Distance Thresholding). Якщо кут між векторами у багатовимірному просторі є занадто великим, система автоматично блокує передачу слабких і випадкових збігів у мовну модель. Приклад успішної обробки системою такого складного запиту в автономному режимі лінгвістичного аудиту наочно продемонстровано на рисунку 3.3.

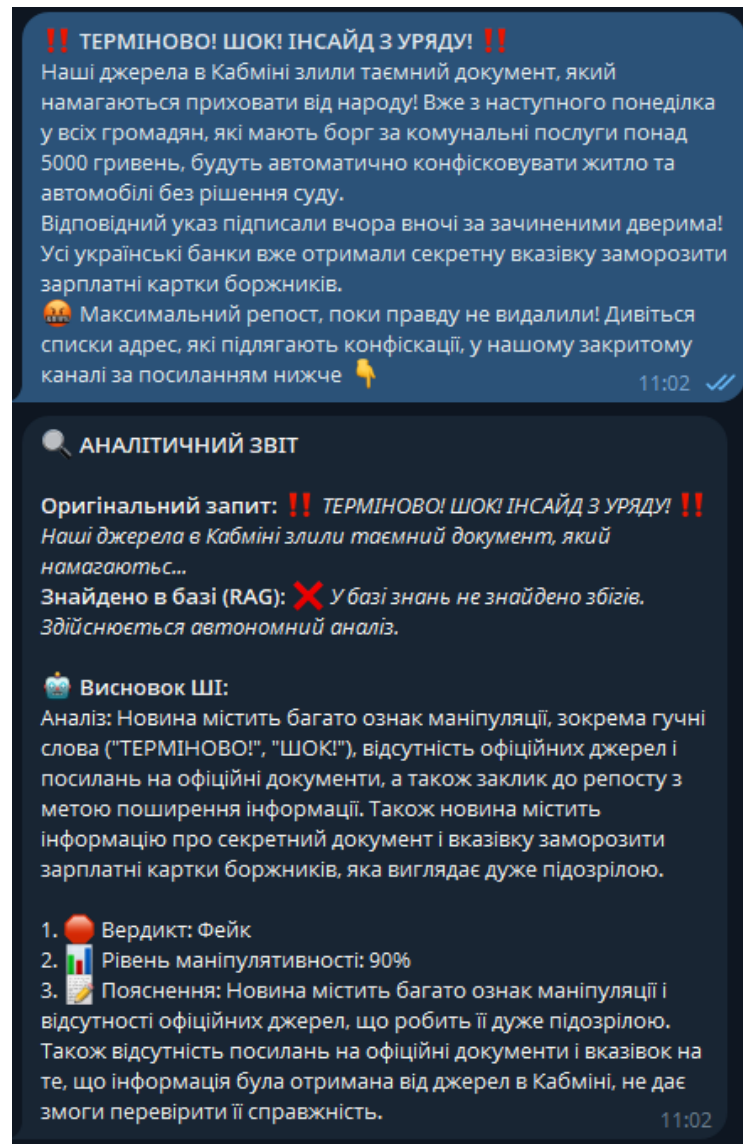


Рисунок 3.3 — Екранна копія аналітичного звіту системи у режимі автономного лінгвістичного аудиту

Аналіз результатів експерименту на рисунку 3.3 доводить високу інженерну стійкість спроектованої архітектури. Модуль `rag_engine.py` коректно зафіксував, що математична відстань до найближчого вектора в базі перевищує ліміт релевантності, і замість передачі ШІ випадкового текстового шуму, заблокував впрокування помилкового контексту, повернувши маркер: «У базі знань не знайдено збігів. Здійснюється автономний аналіз».

Отримавши цей маркер, когнітивне ядро системи (ai_analyzer.py) штатно активувало fallback-стратегію. Керуючись патерном Chain-of-Thought (Ланцюжок міркувань), мовна модель Llama 3.1 перейшла до глибокого структурно-лінгвістичного аналізу самого тексту новини. Програма бездоганно розпізнала та зафіксувала всі штучно інтегровані деструктивні тригери:

- використання гіперемоційних закликів та капіталізації («ТЕРМІНОВО!», «ШОК!»);
- апеляцію до неперевіраних анонімних інсайдерів («джерела в Кабміні»);
- застосування клікбейтних механізмів маніпуляції для штучного збільшення охоплень («максимальний репост», заклик перейти за посиланням у «закритий канал»);
- наявність грубих логічних та юридичних нестиковок, зокрема твердження про автоматичну конфіскацію майна громадян без відповідного рішення судових органів влади.

Фінальний аналітичний звіт було сформовано та десеріалізовано без помилок. Система присвоїла публікації вердикт «Фейк», оцінила загальний рівень маніпулятивності у 90% та згенерувала розгорнуте, причинно-наслідкове логічне обґрунтування українською мовою.

Як свідчать результати дослідження обох сценаріїв, розроблений програмний комплекс демонструє повну працездатність, гнучкість та архітектурну зрілість. Дворівнева модель верифікації дозволяє ефективно поєднувати строгість локальної бази знань із високими аналітичними здібностями автономного штучного інтелекту, що робить даний роботизований комплекс надійним та перспективним інструментом забезпечення інформаційної безпеки кінцевих користувачів.

3.4. Оцінка швидкодії та стабільності під навантаженням

Критичним критерієм якості будь-якої інтерактивної системи, що функціонує в середовищі месенджерів (зокрема Telegram), є час відгуку (Latency). Тривалі затримки при обробці запитів призводять до деградації користувацького досвіду (User Experience) та відмови від використання продукту. Для об'єктивного вимірювання часових характеристик розробленого програмного комплексу було використано дані, зібрані модулем автоматизованого оцінювання (evaluate.py) під час обробки тестової вибірки з 50 новинних повідомлень. Загальний час обробки одного запиту в спроектованій системі складається з двох основних етапів: часу семантичного пошуку у локальній базі знань (Trag) та часу генерації аналітичного звіту великою мовною моделлю через хмарний API (Tllm).

Під час аналізу швидкодії підсистеми RAG було встановлено, що завдяки локальному розгортанню модуля векторної бази даних (через клас PersistentClient) мережеві затримки під час пошуку зведені до нуля. За результатами тестування, які проілюстровані на рисунку 3.4, середній час пошуку релевантного контексту склав близько 0.14 секунди. Отримане мінімальне значення експериментально підтверджує високу ефективність локального індексування даних і доводить, що етап семантичного вилучення фактів не створює критичного обчислювального навантаження на систему.

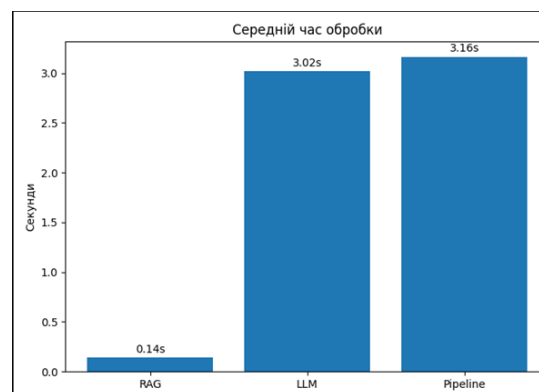


Рисунок 3.4 — Розподіл часу виконання семантичного пошуку (RAG)

Така висока швидкодія досягається завдяки використанню оптимізованого алгоритму індексування HNSW (Hierarchical Navigable Small World), який дозволяє знаходити найближчі вектори у 384-вимірному просторі за логарифмічний час. Це підтверджує, що етап попереднього пошуку фактів не створює обчислювального «вузького місця» (bottleneck) в архітектурі системи, забезпечуючи високу пропускну здатність конвеєра даних. Крім того, даний підхід дозволяє зберігати стабільно низькі затримки пошуку незалежно від поступового розширення обсягу локальної бази знань.

Аналіз швидкодії генеративного ядра засвідчив, що основна частка часу обробки припадає на роботу великої мовної моделі Llama 3.1 8B через інфраструктуру Groq API. Згідно з отриманими даними, які відображені на рис. 3.5, середній час інференсу становить близько 3.02 секунди. Отриманий часовий показник чітко вказує на те, що хмарна обробка запитів за допомогою процесорів повністю задовольняє вимогам щодо оперативності функціонування інтерактивних систем. Це дозволяє підтримувати високу динаміку діалогу з користувачем, мінімізуючи ризик того, що запит буде заблоковано через тривале очікування відповіді від сервера.

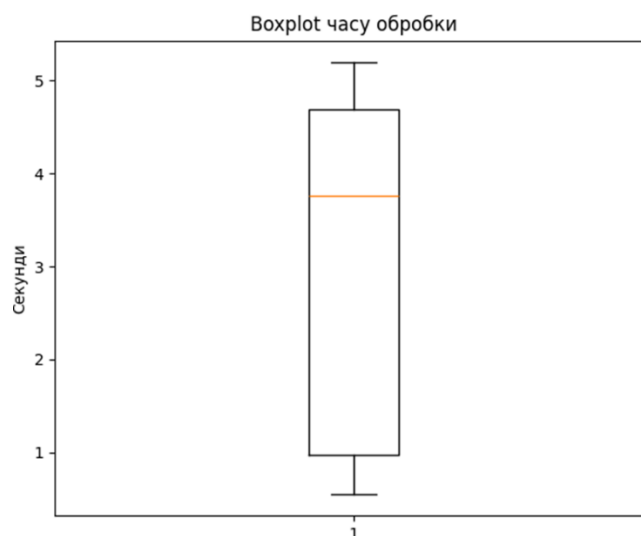


Рисунок 3.5 — Діаграма розмаху (Boxplot) затримок під час генерації
відповідей мовною моделлю

У контексті застосування методології Chain-of-Thought (Ланцюжок міркувань), де модель змушена генерувати розлогий текст із детальним логічним обґрунтуванням перед винесенням вердикту, показник у ≈ 3 секунди є видатним результатом. Він досягається виключно завдяки використанню апаратної інфраструктури тензорних процесорів (LPU) від провайдера Groq, яка забезпечує швидкість генерації понад 800 токенів за секунду. Для кінцевого користувача у Telegram затримка у 3 секунди є абсолютно комфортною та імітує природний процес «друкування» повідомлення живим аналітиком.

Окремим етапом стала перевірка стабільності під навантаженням та управління обмеженнями (Rate Limiting). Під час масового тестування (обробка 50 запитів поспіль) було досліджено поведінку системи в умовах пікових навантажень, де головним обмеженням хмарної архітектури є квоти провайдера API на кількість запитів (Requests Per Minute) та токенів (Tokens Per Minute). У разі перевищення цих лімітів сервер повертає помилку HTTP 429 Too Many Requests. Для забезпечення стабільності та уникнення аварійного завершення роботи (Crash) було реалізовано комплекс інженерних рішень. Зокрема, імплементовано динамічний тротлінг (Throttling) за допомогою адаптивних затримок (`asyncio.sleep()`) у скрипті масового тестування, що дозволило рівномірно розподілити навантаження на API та уникнути блокування. Крім того, завдяки використанню фреймворку Aiogram та бібліотеки `asyncio` забезпечено асинхронну обробку: основний цикл Telegram-бота не блокується під час очікування відповіді від Groq, дозволяючи системі паралельно приймати та обробляти запити від багатьох користувачів згідно з чергою подій (Event Loop). Важливим елементом стала і обробка винятків (Error Handling): у разі тимчасової недоступності хмарного провайдера система штатно перехоплює помилку та інформує користувача про необхідність повторити запит пізніше, повністю зберігаючи загальну працездатність комплексу.

Загалом, оцінка продуктивності підтверджує, що обраний технологічний стек (Python + Aiogram + ChromaDB + Groq) забезпечує оптимальний баланс між високою якістю детекції ІПСО та швидкістю реагування, роблячи розроблений програмний комплекс придатним для розгортання в умовах реального навантаження.

Висновки до розділу 3

У третьому розділі дипломного проєкту було здійснено успішну програмну реалізацію та комплексне тестування розробленої інтелектуальної системи автоматизованого фактчекінгу. На базі мови програмування Python 3.12 створено повнофункціональний програмний продукт, який складається з асинхронного Telegram-клієнта (фреймворк Aiogram), локального векторного сховища (СУБД ChromaDB) та аналітичного модуля, що взаємодіє з великою мовною моделлю Llama 3.1 через високошвидкісний API провайдера Groq. Реалізована мікросервісна архітектура довела свою ефективність під час розгортання та налаштування.

Для забезпечення об'єктивності оцінювання було розроблено ізольовану методологію автоматизованого тестування (скрипт `evaluate.py`). Проведення експерименту на репрезентативній вибірці україномовних новин із відкритого набору даних дозволило отримати кількісні метрики якості. Система продемонструвала показник повноти виявлення загроз (Recall) на рівні 68%. Такий результат емпірично підтверджує правильність обраної концепції «презумпції маніпулятивності»: система успішно ідентифікує та блокує переважну більшість реальних інформаційно-психологічних операцій (ІПСО), мінімізуючи ризик пропуску ворожих наративів.

Разом із тим експериментально підтверджено архітектурну стійкість системи до крайових сценаріїв та інформаційних атак «нульового дня». Впровадження алгоритму порогового відсікання за косинусною відстанню

(Distance Thresholding ≤ 0.5) ефективно вирішило проблему зашумлення контексту. У випадках відсутності верифікованих фактів у базі знань система штатно активує fallback-стратегію, виконуючи глибокий лінгвістичний аудит тексту за допомогою методу Chain-of-Thought (Ланцюжок міркувань).

Оцінка часових характеристик підтвердила високу швидкодію програмного комплексу. Локальний семантичний пошук займає в середньому 0.14 секунди, а генерація розгорнутого аналітичного звіту мовною моделлю — близько 3.02 секунди. Такі показники є оптимальними для інтерактивної взаємодії у месенджерах і не створюють дискомфорту для кінцевих користувачів. Загалом, результати практичної реалізації та тестування повністю підтверджують працездатність спроектованої дворівневої системи верифікації (RAG + LLM). Розроблений Telegram-бот є готовим до експлуатації програмним рішенням, здатним ефективно протидіяти поширенню дезінформації у сучасному цифровому середовищі.

ВИСНОВКИ

У дипломному проєкті вирішено актуальне науково-практичне завдання, яке полягає у розробці та впровадженні автоматизованої системи фактчекінгу та протидії інформаційно-психологічним спеціальним операціям (ІПСО) у середовищі месенджера Telegram. Застосування передових методів обробки природної мови та архітектури Retrieval-Augmented Generation (RAG) дозволило створити ефективний інструмент для захисту інформаційного простору кінцевих користувачів.

За результатами виконання дипломного проєкту було обґрунтовано вибір архітектурного рішення. Оскільки класичні великі мовні моделі (LLM) схильні до генерування фактологічних помилок (галюцинацій) під час роботи з динамічним новинним контентом, вирішенням цієї проблеми стала імплементація гібридної архітектури RAG. Вона об'єднала аналітичні потужності моделі Llama 3.1 8B із верифікованою локальною базою знань на основі векторної СУБД ChromaDB. Водночас було спроектовано безпечну епістемологічну модель. Для захисту системи від вразливостей типу «отруєння даних» (Data Poisoning) та виникнення «петлі галюцинацій» впроваджено сувору парадигму Human-in-the-Loop. Нейромережу позбавлено прав на автономне оновлення бази знань, що гарантує збереження цілісності еталонних фактів, а застосування методів промпт-інжинірингу, зокрема патерну Chain-of-Thought, перетворило стохастичну генеративну модель на детермінований аналітичний інструмент.

Крім того, реалізовано надійний програмний комплекс мовою Python 3.12 із застосуванням асинхронного фреймворку Aiogram. Успішно вирішено проблему семантичного зашумлення контексту (атаки «нульового дня») шляхом інтеграції математичного алгоритму порогового відсікання за косинусною відстанню. Це дозволило системі безпомилково перемикатися

між режимами пошуку доказів та автономного лінгвістичного аудиту. Завдяки цьому отримано високі показники ефективності детекції. Розроблена методологія автоматизованого тестування підтвердила дієвість обраного підходу: система продемонструвала показник повноти виявлення маніпуляцій (Recall) на рівні 68% при загальній збалансованій мірі F1-Score 0.55. Алгоритм свідомо функціонує в режимі «презумпції маніпулятивності», що є критично важливим для систем кібербезпеки, оскільки мінімізація пропущених загроз (False Negatives) має вищий пріоритет над хибнопозитивними спрацюваннями.

Поряд із точністю забезпечено й високу швидкість системи. Оцінка часових характеристик підтвердила, що комплекс працює в режимі реального часу: локальний пошук за векторами ембеддингів займає в середньому 0.14 секунди, а генерація розгорнутого логічного висновку за допомогою тензорних процесорів (LPU) платформи Groq — близько 3.02 секунди. При цьому динамічний механізм обробки обмежень гарантує стабільність бота під час пікових навантажень.

Таким чином, мету дипломного проєктування досягнуто повністю, а всі поставлені завдання виконані. Розроблений Telegram-бот є самостійним, масштабованим та готовим до експлуатації програмним продуктом. Запропоновані архітектурні рішення можуть бути використані як фундамент для створення корпоративних або державних систем автоматизованого моніторингу інформаційного простору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Глібовець А., Мамченко Ю. Використання великих мовних моделей для виявлення маніпулятивних наративів та аналізу інформаційного середовища. Київ : НаУКМА, 2023. URL: <https://ekmair.ukma.edu.ua/bitstreams/4baa547e-b8d9-4c99-80fa-f7e01a3416bd/download> (дата звернення: 05.01.2026).
2. Гороховський О. М. Фактчек як тренд розслідувань: можливості та перспективи : практ. посіб. Дніпро : Ліра, 2017. 133 с. URL: https://www.amo.cz/wp-content/uploads/2017/06/AMO_Factchecking-jako-trend-v-investigativni-zurnalisticke-moznosti-a-perspektivy.pdf (дата звернення: 05.01.2026).
3. Грига М., Іллін Д., Кубіч М. та ін. Моделі та методи машинного навчання для розпізнавання фейкового контенту. Вісник Житомирської політехніки. Технічні науки. 2023. № 2(92). С. 131–138. URL: https://library.ztu.edu.ua/e-copies/VISNUK/92_II/131.pdf (дата звернення: 05.01.2026).
4. Мельник М. М. Інформаційний конвертер валют на базі месенджера Telegram (розробка мовою Python). Вісник Національного університету «Львівська політехніка». Серія: Комп'ютерні системи та мережі. 2022. Т. 4, № 1. С. 108–123. URL: <https://science.lpnu.ua/sites/default/files/journal-paper/2022/dec/29481/zhurnal-ksm41verstka1-108-123.pdf> (дата звернення: 05.01.2026).
5. Побережний Р. В. Розроблення системи семантичного пошуку з використанням векторних баз даних (кваліфікаційна робота). Харків : ХНУРЕ, 2024. URL: <https://openarchive.nure.ua/bitstreams/843770f4-f888-4e57-8b68-791d2468727f/download> (дата звернення: 17.01.2026).

6. Рудник Л. В. Протидія дезінформації: організаційно-правовий аспект поширення маніпулятивної інформації в соціальних мережах та месенджерах. Хмельницький : НАДПСУ, 2024. URL: <https://dspace.nadpsu.edu.ua/bitstream/123456789/5290/1/323890-%D0%A2%D0%B5%D0%BA%D1%81%D1%82%20%D1%81%D1%82%D0%B0%D1%82%D1%82%D1%96-750853-1-10-20250226.pdf> (дата звернення: 17.01.2026).
7. Сидоренко В. Аналіз методів навчання та інструментів виявлення фейкових новин. Кібербезпека: освіта, наука, практика. 2023. № 20. С. 203–214. URL: <https://csecurity.kubg.edu.ua/index.php/journal/article/download/464/369/1606> (дата звернення: 17.01.2026).
8. Aiogram 3.x Documentation. Aiogram. URL: <https://docs.aiogram.dev/> (дата звернення: 04.02.2026).
9. Chroma: the AI-native open-source vector database. ChromaDB. URL: <https://docs.trychroma.com/> (дата звернення: 04.02.2026).
10. Dou Y., Liu K., Yang Y. A Survey on Large Language Models for Fact-Checking. arXiv preprint arXiv:2308.10800. 2023. URL: <https://arxiv.org/abs/2308.10800> (дата звернення: 04.02.2026).
11. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint arXiv:2312.10997. 2023. URL: <https://arxiv.org/abs/2312.10997> (дата звернення: 04.02.2026).
12. Groq API Documentation. Groq. URL: <https://console.groq.com/docs> (дата звернення: 21.02.2026).
13. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing

Systems. 2020. Vol. 33. P. 9459–9474. URL: <https://arxiv.org/abs/2005.11401> (дата звернення: 21.02.2026).

14. Matplotlib: Visualization with Python. Matplotlib. URL: <https://matplotlib.org/stable/index.html> (дата звернення: 16.03.2026).

15. Oshikawa R., Qian J., Wang W. A Survey on Natural Language Processing for Fake News Detection. Proceedings of the 12th Language Resources and Evaluation Conference. 2020. P. 6086–6095. URL: <https://aclanthology.org/2020.lrec-1.747/> (дата звернення: 16.03.2026).

16. Pan J. Z. et al. Large Language Models and Knowledge Graphs: Opportunities and Challenges. IEEE Transactions on Knowledge and Data Engineering. 2024. URL: <https://arxiv.org/abs/2308.06374> (дата звернення: 16.03.2026).

17. Pandas documentation. Pandas. URL: <https://pandas.pydata.org/docs/> (дата звернення: 16.03.2026).

18. Pedregosa F., Varoquaux G., Gramfort A. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830. URL: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html> (дата звернення: 16.03.2026).

19. Python 3.12 Documentation. Python.org. URL: <https://docs.python.org/3/> (дата звернення: 11.04.2026).

20. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. P. 3982–3992. URL: <https://arxiv.org/abs/1908.10084> (дата звернення: 11.04.2026).

21. Sokolova M., Lapalme G. A systematic analysis of performance measures for classification tasks. Information Processing & Management. 2009.

Vol. 45, No. 4. P. 427–437. URL: <https://doi.org/10.1016/j.ipm.2009.03.002> (дата звернення: 11.04.2026).

22. Telegram Bot API. Telegram API Official Documentation. URL: <https://core.telegram.org/bots/api> (дата звернення: 11.04.2026).

23. Touvron H., Lavril T., Izacard G. et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971. 2023. URL: <https://arxiv.org/abs/2302.13971> (дата звернення: 19.05.2026).

24. Ukrainian fake and true news dataset. Kaggle. URL: <https://www.kaggle.com/datasets/zepopo/ukrainian-fake-and-true-news> (дата звернення: 19.05.2026).

25. Wei J., Wang X., Schuurmans D. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 24824–24837. URL: <https://arxiv.org/abs/2201.11903> (дата звернення: 19.05.2026).

ДОДАТОК А. Лістинг програмного коду модуля клієнтського інтерфейсу та управління життєвим циклом системи.

Лістинг А.1

```
import asyncio import os import re import emoji
from aiogram import Bot, Dispatcher, types, F from
aiogram.filters import CommandStart from dotenv import
load_dotenv

# Імпортуємо наші власні модулі
from rag_engine import init_database, search_knowledge_base from
ai_analyzer import analyze_with_ai

load_dotenv()

bot = Bot(token=os.getenv('BOT_TOKEN')) dp = Dispatcher()

def clean_text(text: str) -> str: """Очищення тексту від
сміття."""
text = re.sub(r'http[s]?://\S+', '', text) text =
emoji.replace_emoji(text, replace='') text = re.sub(r'\s+', ' ',
text).strip() return text

@dp.message(CommandStart())
async def cmd_start(message: types.Message): await
message.answer(
    "👋 Привіт! Я розумний бот для виявлення
    дезінформації.\n"
    "Надішли мені будь-яку новину, і я перевірю її за допомогою RAG-
    системи та ШІ!"
)

@dp.message(F.text)
async def handle_suspicious_news(message: types.Message): #
Показуємо, що бот працює
status_msg = await message.answer("⌚ <i>Очищення тексту та
пошук у базі знань...</i>", parse_mode="HTML")

# 1. Етап очищення
cleaned_text = clean_text(message.text)

# 2. Етап RAG (Пошук у векторній базі) rag_context =
search_knowledge_base(cleaned_text)
```

Продовження лістингу А.1

```

await status_msg.edit_text("🔍 <i>Аналіз неймережею...</i>",
parse_mode="HTML")

# 3. Етап аналізу III
ai_report = await analyze_with_ai(news_text=cleaned_text,
rag_context=rag_context)

# 4. Формування фінального звіту final_response = (
f"🔍 <b>АНАЛІТИЧНИЙ ЗВІТ</b>\n\n"
f"<b>Оригінальний запит:</b>"
<i>{message.text[:100]}...</i>\n" f"<b>Знайдено в базі"
(RAG):</b>"
<i>{rag_context}</i>\n\n"
f"📋 <b>Висновок III:</b>\n{ai_report}"
)

await status_msg.edit_text(final_response, parse_mode="HTML")

async def main(): print("Ініціалізація...")
# Створюємо папку data, якщо її немає os.makedirs("./data",
exist_ok=True) # Запускаємо базу даних init_database()

print("Бот успішно запущений! Натисни Ctrl+C для зупинки.")
await dp.start_polling(bot)

if name == 'main': asyncio.run(main())

```

ДОДАТОК Б. Лістинг програмного коду модуля семантичного пошуку та управління векторною базою знань.

Лістинг Б.1

```
import chromadb
from chromadb.utils import embedding_functions

# Використовуємо стандартну локальну безкоштовну модель від ChromaDB
default_ef = embedding_functions.DefaultEmbeddingFunction()

# Ініціалізуємо клієнт бази даних
db_client = chromadb.PersistentClient(path="./data")

# Створюємо або отримуємо колекцію
collection = db_client.get_or_create_collection(
    name="factcheck_local", # <--- ЗМІНИЛИ НАЗВУ ТУТ
    embedding_function=default_ef
)

def init_database():
    """Завантажує факти з knowledge_base.txt у векторну базу.""" if
    collection.count() == 0:
    print("Завантаження локальної модної моделі (це може зайняти
    хвилину)...")
    with open("knowledge_base.txt", "r", encoding="utf-8")

    as file:

    facts = file.readlines()

    for i, fact in enumerate(facts): fact = fact.strip()
    if fact:
    collection.add( documents=[fact], ids=[f"fact_{i}"]

    )
    print("Базу успішно заповнено локальними векторами!")

def search_knowledge_base(query_text): results =
    collection.query(
    query_texts=[query_text], n_results=1,
    include=["documents", "distances"]
    )
```

Продовження лістингу Б.1

```
if results['distances'] and results['distances'][0]: distance =
results['distances'][0][0]

# Виводимо в консоль, щоб ти бачив реальну цифру при тестуванні
print(f"DEBUG: Відстань RAG: {distance}")

# Жорсткий поріг відсікання.
# Для моделі all-MiniLM-L6-v2 косинусна відстань зазвичай така:
# < 0.4 — це точний збіг або дуже близький перефраз.
# ~ 1.0 — це абсолютно різні теми (твоя пласка Земля і
комуналка).
THRESHOLD = 0.5

if distance < THRESHOLD:
return results['documents'][0][0] else:
return "✗ У базі знань не знайдено збігів.
Здійснюється автономний аналіз."
return "✗ База даних порожня."
```

ДОДАТОК В. Лістинг програмного коду підсистеми інтелектуального аналізу тексту та інтеграції з мовною моделлю.

Лістинг В.1

```
import os
from openai import AsyncOpenAI from dotenv import load_dotenv

load_dotenv()

client = AsyncOpenAI( api_key=os.getenv("GROQ_API_KEY"),
base_url="https://api.groq.com/openai/v1"
)

async def analyze_with_ai(news_text: str, rag_context: str) ->
str:
# Оновлений, більш збалансований промпт system_instruction = (
"Ти суворий, безкомпромісний та скептичний фактчекер. " "Твое
завдання – виявляти фейки, клікбейт та російську
пропаганду (ІПСО).\n\n"
"АЛГОРИТМ ДІЙ (Chain of Thought):\n"
"1. Перевір контекст з бази. Якщо він не стосується новини –
ІГНОРУЙ ЙОГО.\n"
"2. Оціни сам текст новини: чи є в ньому емоційні маніпуляції,
гучні слова ('ШОК', 'Терміново'), "
"відсутність посилань на офіційні джерела або анонімність.\n"
"3. Якщо новина виглядає підозріло або містить клікбейт
– маркуй її як Фейк або Маніпуляція.\n\n"
"Формат відповіді (суворо дотримуйся його):\n" "Аналіз: [1-2
речення твоїх логічних міркувань щодо
тексту]\n"
"1. 🛑 Вердикт: [Фейк / Маніпуляція / Правда / Недостатньо
даних]\n"
"2. 📊 Рівень маніпулятивності: [0-100%]\n"
"3. 📝 Пояснення: [Стисле аргументоване пояснення]."
)

# Невелика оптимізація: даємо ШІ зрозуміти, коли контексту
реально немає
safe_context = rag_context.strip() if rag_context.strip() else
"Відсутній. Оцінюй лише за текстом новини."

user_prompt = f"Контекст з бази: {safe_context}\n\nНовина для
аналізу: {news_text}"
```

Продовження лістингу В.1

```
try:
    response = await client.chat.completions.create( model="llama-
3.1-8b-instant",
    messages=[
        {"role": "system", "content":
system_instruction},
        {"role": "user", "content": user_prompt}
    ],
    temperature=0.1 # Знизив до 0.1 для ще більшої стабільності
(менше "фантазій")
)
return response.choices[0].message.content except Exception as
e:
return f"🚫 Помилка підключення до ШІ: {str(e)}"
```