НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА БОТА ДЛЯ АНАЛІЗУ БЕЗПЕКИ ТЕКСТОВИХ ТА ГРАФІЧНИХ ДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ»

Виконав: здобувач 4 курсу

Рівень бакалавр

Галузь знань 12 «Інформаційні технології»,

спеціальність 125 «Кібербезпека»

Столярик Олександр Васильович

Керівник: д.т.н., професор

Соколов Артем Вікторович

Рецензент: к.т.н., доцент

Бойко Віктор Дмитрович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
професор С.А. Горбаченко
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Столярику Олександру Васильовичу

1. Тема роботи: «Розробка бота для аналізу безпеки текстових та графічних даних на основі нейронних мереж»
Керівник роботи: д.т.н., професор Соколов Артем Вікторович
затверджені наказом НУ ОЮА від «18» березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Отримання та узгодження завдання на кваліфікаційну роботу	Вересень-жовтень 2024	
2	Збір та аналіз інформації, огляд літературних джерел	Листопад 2024	
3	Підготовка теоретичної частини роботи (Розділ 1)	Січень – лютий 2025	
4	Проектування архітектури та інтерфейсів бота (Розділ 2)	Лютий – березень 2025	
5	Розробка, інтеграція та тестування програмних компонентів бота (Розділи 3, 4)	Березень – квітень 2025	
6	Оформлення кваліфікаційної роботи та додатків	Квітень – травень 2025	
7	Подання роботи на перевірку керівнику та рецензенту	Травень 2025	
8	Підготовка до захисту (презентація, доповідь)	10.06.2025	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розробка бота для аналізу безпеки текстових та графічних даних на основі нейронних мереж» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 «Кібербезпека», освітньою програмою: Кібербезпека, містить 7 рисунків, 3 додатків, 33 літературних джерел за переліком посилань. Робота виконана на 80 сторінках загального тексту і 51 сторінках основного тексту.

Робота присвячена створенню програмних інструментів для автоматизованого аналізу безпеки цифрового контенту в популярних месенджерах. Розглянуто загрози, пов'язані як з текстовими, так і з графічними даними. Метою є розробка Telegram-бота як системи аналізу безпеки текстових даних, що інтегрує нейромережеву модель.

У рамках роботи було проаналізовано вимоги до подібних систем, досліджено ШІ-підходи до аналізу текстових та (теоретично) графічних даних, спроектовано архітектуру та інтерфейс бота. Програмні модулі розроблено на Python (pyTelegramBotAPI). Ключовим аспектом стала інтеграція з зовнішнім ШІ-модулем аналізу текстів на предмет ІПСО (розробленим в рамках паралельного дослідження Долінко К. В.), що включає оцінку надійності джерела. Розглянуто локальне розгортання та тестування.

В результаті було створено функціонуючого Telegram-бота, здатного приймати текстові дані, передавати їх на аналіз відповідній нейромережевій моделі, враховувати та оновлювати надійність джерела і надавати користувачеві результат перевірки. Розроблений бот демонструє практичне застосування сучасних технологій для вирішення задач кібербезпеки в частині аналізу тексту та може слугувати основою для систем модерації контенту.

КЛЮЧОВІ СЛОВА: TELEGRAM БОТ, КІБЕРБЕЗПЕКА, НЕЙРОННІ МЕРЕЖІ, PYTHON, АНАЛІЗ ТЕКСТУ, ОБРОБКА ПРИРОДНОЇ МОВИ, ІНТЕГРАЦІЯ СИСТЕМ, API, РОЗРОБКА БОТІВ, ОЦІНКА НАДІЙНОСТІ ДЖЕРЕЛА.

ABSTRACT

The qualification work on the topic "Development of a bot for analyzing the security of text and graphic data based on neural networks" for obtaining the first (bachelor's) level of higher education in specialty 125 "Cybersecurity", educational program: Cybersecurity, contains 7 figures, 3 appendices, and 33 literary sources according to the list of references. The work was completed on 80 pages of total text and 51 pages of main text.

The work is devoted to creating software tools for automated analysis of digital content security in popular messengers. Threats associated with both text and graphic data are considered. The aim is the development of a Telegram bot as a system for analyzing the security of text data, integrating a neural network model.

Within the scope of the work, requirements for such systems were analyzed, AI approaches for the analysis of text and (theoretically) graphic data were studied, and the bot's architecture and user interface were designed. Program modules were developed in Python (pyTelegramBotAPI). A key aspect was the integration with an external AI module for text analysis for IPSO (developed within a parallel study by Dolinko K.V.), which includes a source reliability assessment mechanism. Local deployment and testing were considered.

As a result, a functioning Telegram bot was created, capable of receiving text data, transmitting it for analysis to the corresponding neural network model, considering and updating source reliability, and providing the user with the verification result. The developed bot demonstrates the practical application of modern technologies for solving cybersecurity tasks in the context of text analysis and can serve as a basis for content moderation systems.

KEYWORDS: TELEGRAM BOT, CYBERSECURITY, NEURAL NETWORKS, PYTHON, TEXT ANALYSIS, NATURAL LANGUAGE PROCESSING, SYSTEM INTEGRATION, API, BOT DEVELOPMENT, SOURCE RELIABILITY ASSESSMENT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, програмний інтерфейс додатків

БД – база даних

ІБ – інформаційна безпека

ІПСО – інформаційно-психологічна операція

ШІ – штучний інтелект

ML – Machine Learning (машинне навчання)

NLP – Natural Language Processing (обробка природної мови)

NSFW – Not Safe For Work (контент не для роботи)

RNN – Recurrent Neural Network (рекурентна нейронна мережа)

UX – User Experience (користувацький досвід)

UI – User Interface (інтерфейс користувача)

venv – віртуальне середовище Python

ВСТУП

Актуальність теми. Сучасний цифровий простір, зокрема популярні месенджери як Telegram, переповнений величезними обсягами текстової та графічної інформації. Поряд з корисним контентом, стрімко поширюються і небезпечні матеріали: дезінформація, пропаганда, мова ворожнечі, шахрайські схеми (фішинг), контент для дорослих (NSFW), екстремістські заклики та символіка. Ручна модерація та аналіз таких обсягів даних стають неефективними та надзвичайно ресурсозатратними. У цьому контексті, розробка автоматизованих систем аналізу безпеки контенту з використанням технологій штучного інтелекту, зокрема нейронних мереж, набуває особливої актуальності для забезпечення інформаційної та психологічної безпеки користувачів. Створення зручних інструментів, таких як Telegram-боти, що інтегрують потужні аналітичні можливості ШІ, є перспективним напрямком вирішення цієї проблеми.

Стан розробки проблеми. Існують різноманітні підходи та інструменти для аналізу контенту, від простих фільтрів за ключовими словами до складних комерційних платформ модерації та систем запобігання витокам даних (DLP). Наукові дослідження активно ведуться у напрямках застосування нейронних мереж для класифікації текстів (виявлення токсичності, фейків, ІПСО) та зображень (розпізнавання об'єктів, NSFW-фільтрація). Однак, створення доступних, інтегрованих рішень у форматі Telegram-ботів, що поєднують аналіз різних типів даних та легко розгортаються, залишається актуальним завданням.

Мета і завдання дослідження. Метою даної кваліфікаційної роботи є теоретичне дослідження підходів до аналізу безпеки текстових та графічних даних засобами нейронних мереж та практична розробка програмного бота для платформи Telegram, який забезпечує інтерфейс користувача для аналізу безпеки текстових даних та інтегрується з відповідною нейромережевою моделлю.

Для досягнення поставленої мети необхідно було вирішити такі завдання:

1) Проаналізувати актуальні загрози безпеці, пов'язані з текстовим та графічним контентом у Telegram.

2) Оглянути існуючі підходи до розробки Telegram-ботів та принципи роботи Telegram Bot API.

3) Проаналізувати загальні принципи використання нейронних мереж для аналізу текстів та зображень.

4) Обґрунтувати вибір програмних засобів та бібліотек для реалізації бота.

5) Сформулювати вимоги до функціональності та архітектури бота.

6) Спроектувати архітектуру програмного бота, інтерфейс користувача та інтерфейс взаємодії (API) з модулями аналізу ШІ.

7) Розробити програмні модулі бота на мові Python, включаючи обробники повідомлень та логіку взаємодії.

8) Реалізувати інтеграцію розробленого бота з зовнішнім модулем аналізу текстів через визначений програмний інтерфейс.

9) Дослідити та реалізувати підходи до розгортання бота на сервері для забезпечення його автономної роботи.

10) Провести тестування функціональності, інтерфейсу та стабільності роботи розробленого бота.

Розмежування завдань. Дана робота є частиною спільного проекту. Розробка та дослідження рекурентних нейронних мереж для аналізу тексту на предмет інформаційно-психологічних операцій виконувались Долінко К. В. в рамках кваліфікаційної роботи на тему «Використання рекурентних нейронних мереж для аналізу тексту на предмет інформаційно-психологічних операцій». Дана робота зосереджена на проектуванні, розробці, інтеграції та розгортанні програмного бота як системи, що використовує результати роботи нейромережевих моделей.

Об'єкт дослідження – процес розробки, інтеграції та розгортання програмного Telegram-бота для аналізу безпеки текстових даних.

Предмет дослідження – архітектура, програмні модулі, користувацький інтерфейс Telegram-бота, інтеграції з нейромережевою моделлю аналізу тексту, способи забезпечення автономної роботи бота на сервері.

Методи дослідження. У роботі використано методи системного аналізу (для визначення вимог та архітектури), об'єктно-орієнтованого програмування (при реалізації на Python), принципи розробки програмного забезпечення, методи проектування та використання API, технології розгортання веб-додатків та фонових процесів, методи тестування програмного забезпечення (функціональне, інтеграційне, тестування зручності використання).

Наукова новизна одержаних результатів.

1) Запропоновано архітектуру програмного Telegram-бота, що забезпечує модульну інтеграцію з зовнішнім ШІ-модулем аналізу тексту, який використовує динамічну оцінку надійності джерела, через визначений програмний інтерфейс (прямий виклик Python-функції).

2) Розроблено програмну реалізацію Telegram-бота, який реалізує користувацький інтерфейс для аналізу текстових повідомлень на предмет ІПСО, здійснює підготовку даних для ШІ-модуля, обробляє результати аналізу та забезпечує взаємодію з базою даних для оновлення показників надійності джерел.

3) Досліджено та обґрунтовано підходи до локального розгортання та тестування Python-бота, що використовує метод Polling та інтегрує ресурсоємний ШІ-модуль на базі TensorFlow, з визначенням перспектив для серверного розгортання.

Практичне значення одержаних результатів. Розроблений програмний бот є готовим до використання інструментом, який може бути застосований для:

1) Оперативної перевірки користувачами підозрілих текстових повідомлень у Telegram.

2) Використання як компонента у більших системах модерації контенту для Telegram-каналів та груп.

3) Демонстрації можливостей інтеграції месенджер-платформ та систем штучного інтелекту для вирішення задач кібербезпеки.

4) Подальшого розвитку та додавання нових модулів аналізу (наприклад, для аналізу зображень, що досліджувався теоретично).

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 96 сторінок, з них основного тексту 57 сторінок. Робота містить 7 рисунків. Перелік посилань включає 33 найменувань.

1 АНАЛІЗ ПРОБЛЕМИ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ БОТА ДЛЯ АНАЛІЗУ БЕЗПЕКИ КОНТЕНТУ

1.1 Актуальність задачі автоматизованого аналізу безпеки текстових та графічних даних

Сучасне інформаційне суспільство характеризується безпрецедентним зростанням обсягів цифрових даних, що генеруються та поширюються щомиті. Особливо стрімко цей процес відбувається у соціальних мережах та месенджерах, які стали невід'ємною частиною щоденної комунікації мільярдів людей. Платформи, такі як Telegram, завдяки своїм функціональним можливостям (канали, групи, боти) та політиці конфіденційності, перетворилися на потужні інструменти не лише для спілкування, але й для масового поширення інформації, включаючи текстові повідомлення, зображення, відео та інші типи файлів.

Значення Telegram для інформаційного простору України важко переоцінити. За даними на початок 2024 року, кількість активних користувачів месенджера в країні перевищувала 11 мільйонів [1], а близько 44% українців використовували його для отримання новин ще у 2023 році [2]. Під час повномасштабної війни Telegram став основним джерелом інформації для значної частини населення, при цьому близько 47% громадян отримували новини саме через цю платформу [3].

Однак така популярність та швидкість розповсюдження контенту несуть в собі й суттєві загрози. Особливої гостроти ця проблема набуває в умовах повномасштабної інформаційної війни, яку Російська Федерація веде проти України. Месенджер Telegram активно використовується агресором як один із ключових майданчиків для проведення інформаційно-психологічних операцій (ІПСО), поширення деструктивних наративів, маніпуляції інформацією та відвертої дезінформації [4]. Ці дії спрямовані на підрив національної безпеки, деморалізацію суспільства та дискредитацію державних інституцій України. За даними досліджень, значна частина контенту проросійських каналів спрямована на поширення неправдивої інформації про українську армію, політику та на створення

паніки серед населення [5]. Серйозність проблеми підтверджується тим, що 71% українців вважають поширення дезінформації в Telegram значною загрозою, а близько 35% називають месенджери основним джерелом фейкових новин у 2024 році [3]. Занепокоєння викликає і позиція керівництва українських спецслужб: голова ГУР МО України Кирило Буданов зазначав, що платформа становить загрозу національній безпеці та потребує регулювання [7].

Поряд із загрозами, пов'язаними з інформаційною агресією, актуальними залишаються й інші види небезпечного контенту, що циркулюють у месенджері:

1) Мова ворожнечі (Hate Speech): Висловлювання, що розпалюють ненависть, дискримінацію або ворожнечу щодо окремих осіб чи груп, що особливо посилюється на тлі війни.

2) Кібербулінг та переслідування: Образи, погрози, цькування користувачів.

3) Шахрайство та фішинг: Спроби виманювання конфіденційних даних або коштів, які можуть маскуватися під повідомлення від державних органів чи волонтерських організацій.

4) Неприйнятний контент (NSFW - Not Safe For Work): Матеріали еротичного, порнографічного чи насильницького характеру.

5) Заборонена символіка та екстремістські матеріали: Поширення символів, зображень чи текстів, пов'язаних з терористичними або екстремістськими організаціями, включаючи символіку російської агресії.

6) Витоки конфіденційної інформації: Ненавмисне або зловмисне поширення персональних даних, комерційної таємниці чи іншої чутливої інформації.

Незважаючи на те, що адміністрація Telegram декларує певні кроки з модерації контенту, зокрема у відповідь на вимоги європейського законодавства (Digital Services Act, DSA), дозволяючи користувачам надсилати скарги на неприйнятний контент [8], ефективність цих заходів залишається обмеженою. Наскрізне шифрування в "секретних чатах" та загальна політика платформи ускладнюють проактивний автоматизований аналіз контенту самою компанією. Масштаби генерації контенту, особливо в контексті цілеспрямованих

інформаційних атак, все ще значно перевищують можливості як вбудованих механізмів платформи, так і ручної модерації.

У зв'язку з цим, виникає гостра потреба в розробці та впровадженні додаткових автоматизованих систем для аналізу безпеки текстових та графічних даних, які могли б функціонувати на боці користувача або в рамках незалежних моніторингових ініціатив. Технології штучного інтелекту, зокрема нейронні мережі глибокого навчання, продемонстрували високу ефективність у вирішенні складних завдань класифікації та розпізнавання образів, включаючи обробку природної мови (NLP) та комп'ютерний зір (CV). Застосування нейронних мереж дозволяє створювати моделі, здатні з високою точністю виявляти ознаки небезпечного контенту, адаптуватися до нових загроз та працювати в режимі, близькому до реального часу.

Створення інструментів, які інтегрують ці можливості у звичне для користувачів середовище, як-от Telegram-боти, є важливим кроком. Такі боти можуть слугувати як персональними помічниками для перевірки підозрілого контенту, так і компонентами більших систем модерації, підвищуючи загальний рівень безпеки в цифровому просторі. Таким чином, розробка бота для аналізу безпеки текстових та графічних даних на основі нейронних мереж, з особливим акцентом на протидії російській дезінформації та ІПСО, є надзвичайно актуальною науково-практичною задачею на перетині кібербезпеки та штучного інтелекту в умовах гібридної війни.

1.2 Огляд існуючих бот-платформ та підходів до модерації контенту в месенджерах

Розробка програмних ботів стала можливою завдяки наданню розробникам доступу до програмних інтерфейсів (API) з боку популярних комунікаційних платформ. Такі API дозволяють створювати автоматизовані акаунти (боти), здатні взаємодіяти з користувачами, обробляти повідомлення, виконувати команди та інтегруватися із зовнішніми сервісами.

Бот-платформи та їх можливості:

Найбільш відомі платформи, що надають розвинені API для створення ботів, включають:

1) Telegram: Вирізняється гнучким та потужним Bot API, що підтримує широкий спектр типів повідомлень (текст, медіа, стікери, локації), інтерактивні елементи (клавіатури, інлайн-кнопки), отримання оновлень через polling або webhooks, а також можливість використання асинхронних бібліотек (наприклад, aiogram) [8]. Однак, існують обмеження на швидкість надсилання повідомлень (близько 30/сек різним користувачам, 20/хв в одній групі), розміри файлів (до 50 МБ через публічне API) та відсутність можливості для бота ініціювати розмову без взаємодії з боку користувача [9].

2) Facebook Messenger: Також надає API для розробки ботів, часто використовується для бізнес-комунікацій та підтримки клієнтів.

3) WhatsApp (через WhatsApp Business API): Орієнтований переважно на бізнес-взаємодію, має більш суворі правила та обмеження для вихідних повідомлень.

4) Viber, Slack, Discord: Мають власні API та екосистеми для створення ботів з різним ступенем функціональності, часто орієнтовані на корпоративне використання або специфічні спільноти.

5) Для розробки складних, кастомізованих рішень можуть використовуватися відкриті фреймворки, такі як Rasa [10] або Microsoft Bot Framework для інтеграції з екосистемою Microsoft.

Вибір Telegram як цільової платформи для даної роботи обґрунтований його високою популярністю в Україні, широкими можливостями Bot API для роботи з різними типами контенту та гнучкими способами отримання оновлень, що є достатнім для реалізації поставлених завдань. Для взаємодії з API в даній роботі використовується бібліотека pyTelegramBotAPI, хоча існують і інші популярні Python-бібліотеки зі схожим функціоналом, такі як python-telegram-bot чи асинхронний фреймворк aiogram [11].

Підходи до модерації контенту в месенджерах та соціальних мережах:

Зі збільшенням обсягів контенту та актуалізацією загроз (як описано в п. 1.1), постає питання ефективної модерації. Існують наступні основні підходи:

1) Ручна модерація: Модератори (люди) переглядають контент та приймають рішення про його відповідність правилам платформи чи спільноти.

а) Переваги: Здатність розуміти контекст, сарказм, культурні особливості, що складно для автоматичних систем.

б) Недоліки: Низька швидкість, висока вартість, суб'єктивність, психологічне вигорання модераторів, неможливість обробки великих обсягів даних у реальному часі.

2) Системи на основі правил та ключових слів: Використовують регулярні вирази, чорні/білі списки для фільтрації. Існує багато ботів-модераторів для Telegram, що реалізують цей підхід, наприклад, MissRose Bot (@MissRose_bot), Shieldy (@shieldy_bot) або Combot (@combot), які дозволяють блокувати спам, посилання, певні слова, а також використовувати CAPTCHA для нових учасників чатів або каналів [12].

а) Переваги: Простота реалізації, швидкість роботи.

б) Недоліки: Легко обходяться (заміна символів, використання синонімів, емодзі), не враховують контекст, велика кількість хибних спрацювань (false positives) або пропусків (false negatives). Неефективні для аналізу зображень чи складних маніпулятивних текстів.

3) Системи на основі репутації та довіри: Оцінюють користувачів або джерела, обмежуючи дії акаунтів з низькою репутацією (напр., система репутації на Stack Overflow)

а) Переваги: Може зменшити вплив зловмисних акаунтів.

б) Недоліки: Можливість маніпуляції рейтингами, не захищає від шкідливого контенту з довірених джерел (злам акаунта, зміна позиції).

4) Системи скарг користувачів (User Reporting): Дозволяють користувачам позначати неприйнятний контент, який потім перевіряється (напр., реалізовано в Telegram, Reddit [7]).

а) Переваги: Використовує "колективний розум", дозволяє реагувати на новий, неочікуваний контент.

б) Недоліки: Реактивний підхід (реагує вже після публікації), можливість зловживання (масові скоординовані скарги), залежить від активності та обізнаності користувачів.

5) Автоматизована модерація на основі ШІ/Машинного навчання: Найбільш прогресивний підхід, що використовує нейронні мережі для аналізу тексту та медіа. Існують як комерційні платформи (напр., Hive Moderation, WebPurify, Sightengine [13]), так і відкриті інструменти та моделі:

- Для тексту: API для оцінки токсичності (Jigsaw Perspective API), моделі для виявлення мови ворожнечі, дезінформації (доступні на Hugging Face Hub, часто на базі архітектур BERT, RoBERTa), бібліотеки NLP (NLTK, spaCy) для створення власних класифікаторів.

- Для зображень: Моделі та API для виявлення NSFW-контенту (open_nsfw від Yahoo, NSFW JS, Sightengine API, ModerateContent API [13]).

Ці системи значно ефективніші за фільтри на основі правил, але потребують навчання, ресурсів та можуть мати упередження.

а) Переваги: Здатність обробляти величезні обсяги даних, виявляти складні патерни, що не піддаються простим правилам, потенційно вища точність (для певних задач) порівняно з простими фільтрами, можливість самонавчання та адаптації.

б) Недоліки: Потребують великих якісних наборів даних для навчання, значних обчислювальних ресурсів для тренування та іноді для роботи, складність інтерпретації рішень моделі, можливість упередженості (bias) в даних, все ще можливі помилки (хоча й іншого типу, ніж у простих фільтрів).

6) Гібридний підхід (AI + Human): Комбінація ШІ для первинної фільтрації та людини-модератора для перевірки складних випадків. Цей підхід використовується багатьма великими платформами (Facebook, Twitter) та спеціалізованими сервісами (Khoros Communities, Glynk’s Platform [14]), забезпечуючи баланс між швидкістю, масштабованістю та точністю.

Контекст Telegram: У Telegram існує багато сторонніх ботів, які допомагають адміністраторам груп модерувати контент, зазвичай використовуючи комбінацію правил, ключових слів та дій адміністраторів. Сама платформа також має вбудовані механізми боротьби зі спамом. Однак, задача глибокого аналізу контенту на предмет дезінформації, ІПСО чи складних графічних загроз часто виходить за межі можливостей стандартних інструментів та політик самої платформи.

Висновки щодо існуючих підходів: Кожен з описаних підходів до модерації має свої переваги та недоліки. Однак, в умовах цілеспрямованої інформаційної агресії, яку веде Росія проти України, традиційні методи виявляються особливо вразливими. Ручна модерація нездатна впоратися з масованими "вкидами" дезінформації та пропаганди. Системи на основі правил та ключових слів легко обходяться противником, який постійно адаптує свою лексику та методи поширення наративів. Системи скарг користувачів, хоч і корисні, є реактивними і можуть бути перевантажені або навіть цілеспрямовано атаковані через масові скарги ботоферм.

Саме складність та адаптивність загроз, характерних для інформаційної війни (виявлення прихованих маніпуляцій, ІПСО, багатошарової дезінформації), вимагають застосування більш потужних інструментів аналізу. Автоматизована модерація на основі ШІ та нейронних мереж є найбільш перспективним напрямком, оскільки такі системи здатні виявляти складні семантичні зв'язки, аналізувати контекст та розпізнавати патерни, що не піддаються простим правилам. Це особливо важливо для ідентифікації російських деструктивних наративів, які часто маскуються під об'єктивні новини або думки експертів.

Розробка бота, що інтегрує ШІ-моделі, навчені саме на виявлення специфічних ознак російської дезінформації та пропаганди, є кроком до створення ефективного інструменту протидії інформаційній агресії в середовищі Telegram. Такий бот може не лише фільтрувати очевидно небезпечний контент, але й допомагати користувачам ідентифікувати маніпулятивні техніки та ворожі наративи, заповнюючи прогалину в доступних засобах захисту в умовах інформаційної війни.

1.3 Принципи роботи telegram bot api та методи взаємодії

Ключовим інструментом для створення програмних ботів на платформі Telegram є наданий розробникам програмний інтерфейс додатків – Telegram Bot API. Це HTTP-орієнтований інтерфейс, що дозволяє створювати спеціалізовані акаунти (боти), які можуть автоматично взаємодіяти з користувачами, обробляти повідомлення та команди, надсилати різноманітний контент (текст, медіа, стікери, локації), керувати інтерактивними елементами (клавіатури, кнопки) та інтегруватися із зовнішніми системами [15].

Взаємодія з API відбувається через надсилання HTTPS-запитів до серверів Telegram на адресу https://api.telegram.org/bot<token>/<METHOD_NAME>, де <token> – це унікальний авторизаційний токен бота, отриманий від сервісу @BotFather, а <METHOD_NAME> – назва методу API, що викликається. Цей токен є критично важливим для безпеки бота, оскільки він надає повний контроль над ним. Тому зберігання токена має відбуватися у захищеному вигляді (наприклад, через змінні середовища або захищені конфігураційні файли), а не у відкритому коді [15].

Для ефективної розробки бота необхідно розуміти ключові об'єкти (типи даних), якими оперує Telegram Bot API [15]:

1) Update: Центральний об'єкт, що інкапсулює будь-яку подію, яка надсилається боту (нове повідомлення, редагування, натискання кнопки, зміна статусу в чаті тощо). Містить необов'язкові поля, що відповідають типу події, такі як message, edited_message, callback_query, inline_query, chat_member та інші.

2) Message: Представляє повідомлення будь-якого типу. Містить поля: message_id (унікальний ID повідомлення), from (об'єкт User, що описує відправника), chat (об'єкт Chat, що описує чат), date (час відправлення, Unix timestamp), text (текст повідомлення для текстових повідомлень), photo (масив об'єктів PhotoSize для фотографій), caption (підпис до медіа), entities (масив об'єктів MessageEntity для форматування тексту, команд, URL), reply_to_message

(посилання на вихідне повідомлення, якщо це відповідь) та багато інших для різних типів контенту.

3) Chat: Містить інформацію про чат: id (унікальний ідентифікатор), type ('private', 'group', 'supergroup', 'channel'), title (для груп, каналів), username (для чатів з публічним іменем) та ін.

4) User: Містить інформацію про користувача або бота: id (унікальний ідентифікатор), is_bot (булеве значення), first_name, last_name (опціонально), username (опціонально).

5) Клавіатури та Кнопки: API надає гнучкі засоби для створення інтерактивних елементів:

а) ReplyKeyboardMarkup: Клавіатура зі стандартними кнопками під полем вводу тексту. Натискання надсилає текст кнопки як звичайне повідомлення.

б) InlineKeyboardMarkup: Клавіатура з кнопками, прикріпленими безпосередньо до повідомлення бота. Складається з об'єктів InlineKeyboardButton.

в) InlineKeyboardButton: Може містити текст (text) та один з типів дії: URL-посилання (url), дані для зворотного виклику (callback_data), перемикання в інлайн-режим (switch_inline_query) тощо.

г) CallbackQuery: Об'єкт, що надсилається в Update, коли користувач натискає інлайн-кнопку з callback_data. Містить id (унікальний ID запиту), from (користувач, що натиснув), message (повідомлення, до якого була прикріплена кнопка) та data (значення callback_data з кнопки). Бот повинен відповісти на цей запит методом answerCallbackQuery.

6) Методи API: Існує широкий набір методів для виконання дій від імені бота. Серед найбільш використовуваних у контексті даної роботи:

а) getMe: Перевірка токена та отримання базової інформації про бота.

б) sendMessage: Надсилання текстових повідомлень. Дозволяє вказувати chat_id, text, parse_mode (для форматування Markdown/HTML), reply_markup (для додавання клавіатур) та інші параметри.

в) sendPhoto: Надсилання фотографій.

г) editMessageText, deleteMessage: Редагування та видалення надісланих ботом повідомлень.

д) answerCallbackQuery: Надсилання відповіді на натискання інлайн-кнопки.

е) getFile: Отримання інформації про файл (його file_id, file_unique_id, file_size та file_path) для подальшого завантаження.

є) Методи отримання оновлень: getUpdates, setWebhook, deleteWebhook.

Методи отримання оновлень (Updates): Polling та Webhooks

Щоб бот міг реагувати на події, він повинен отримувати об'єкти Update від Telegram. Існує два основних механізми [15]:

1) Long Polling (Довгі Запити)

Це метод за замовчуванням, при якому бот активно запитує Telegram про нові оновлення. Він реалізується через періодичні HTTPS-запити до методу getUpdates. Ключові параметри цього методу:

- offset: Ідентифікатор наступного оновлення, яке потрібно отримати. Дозволяє уникнути повторної обробки тих самих подій. Зазвичай встановлюється в ID_останнього_отриманого_Update + 1.

- limit: Максимальна кількість оновлень, що повертаються за один існуючий запит (1-100).

- timeout: Час у секундах (0-50), протягом якого сервер Telegram утримує запит відкритим, очікуючи на нові оновлення. Якщо timeout > 0, реалізується саме Long Polling.

Принцип роботи Long Polling: бот надсилає запит getUpdates з певним timeout. Якщо оновлення є, сервер повертає їх одразу. Якщо ні, сервер чекає до timeout секунд. Як тільки з'являється оновлення, воно повертається боту. Якщо за час timeout нічого не відбулося, повертається порожня відповідь. Після отримання відповіді (або таймауту) бот одразу робить наступний запит getUpdates, оновлюючи offset. Бібліотеки, такі як pyTelegramBotAPI, автоматизують цей процес у методі bot.polling().

Переваги Polling:

а) Простота налаштування: Не потребує веб-сервера, домену, SSL. Ідеально для локальної розробки та тестування.

б) Робота за NAT/Firewall: Може працювати з комп'ютерів без публічної IP-адреси.

Недоліки Polling:

а) Ресурсоємність: Постійні запити та утримання з'єднання.

б) Затримка: Хоч і невелика завдяки Long Polling, але все ж існує.

в) Проблеми з PaaS: Не підходить для безкоштовних або обмежених планів PaaS-платформ, які "присипляють" процеси або очікують відкриття порту.

г) Необхідність постійної роботи: Процес бота має бути запущений безперервно.

2) Webhooks (Зворотні Веб-виклики)

При використанні вебхуків Telegram сам надсилає оновлення боту. Для цього розробник повинен:

- Мати веб-сервер, доступний з Інтернету за HTTPS URL з дійсним SSL-сертифікатом.

- Зареєструвати цей URL в Telegram за допомогою методу setWebhook, вказавши url.

- Реалізувати на своєму сервері обробник (ендпоінт), який приймає POST-запити за вказаним URL. Telegram надсилатиме на цей ендпоінт JSON-об'єкт Update у тілі запиту при кожній новій події.

Переваги Webhooks:

а) Миттєва доставка: Оновлення приходять одразу.

б) Ефективність: Ресурси сервера використовуються лише при отриманні оновлення.

в) Масштабованість та сумісність з хмарами: Добре працює на PaaS (Render, Heroku), Serverless (AWS Lambda) платформах, підтримує балансування навантаження.

Недоліки Webhooks:

а) Складність налаштування: Вимагає веб-сервера, домену/IP, SSL-сертифіката.

б) Складність локального тестування: Необхідні інструменти тунелювання (напр., ngrok) або розгортання на тестовому сервері.

в) Залежність від сервера: Стабільність бота залежить від доступності та надійності веб-сервера.

Обґрунтування Вибору Методу у Даній Роботі:

На етапі розробки та первинного тестування даного програмного бота було обрано метод Polling завдяки його значній перевазі у простоті налаштування та можливості швидкого запуску, тестування і налагодження на локальному комп'ютері без додаткової інфраструктури. Використовувана бібліотека pyTelegramBotAPI надає зручний високо-рівневий метод bot.polling() для реалізації цього підходу.

Водночас, аналіз переваг та недоліків обох методів [16] свідчить, що для розгортання бота в продуктивному середовищі, яке вимагає високої доступності, миттєвої реакції та ефективного використання ресурсів, особливо на хмарних PaaS-платформах, перевагу слід віддавати Webhooks. Цей метод краще відповідає архітектурі сучасних веб-додатків та дозволяє використовувати переваги безкоштовних або дешевих веб-хостингів. Хоча повна реалізація Webhooks не входила до основних завдань даної роботи, розуміння цього механізму є важливим для подальшого розвитку системи. Варто зазначити, що при розгортанні на виділеному сервері (VPS), де розробник має повний контроль над середовищем, метод Polling також може бути ефективно та надійно використаний за умови запуску бота як постійно діючого сервісу за допомогою менеджерів процесів, таких як systemd або supervisor [17].

1.4 Загальний огляд підходів до аналізу контенту за допомогою нейронних мереж

Ефективний аналіз великих обсягів текстових та графічних даних на предмет загроз безпеки значною мірою спирається на сучасні технології штучного інтелекту. Серед них ключову роль відіграють нейронні мережі глибокого навчання (Deep Learning) – підгалузь машинного навчання, що використовує багатошарові архітектури для автоматичного виділення ознак з даних, поступово переходячи від простих до більш абстрактних рівнів подання [18]. Це дозволяє досягти високої точності у складних задачах розпізнавання образів, обробки мови та інших. У контексті даної роботи найбільш релевантними є підходи з обробки природної мови (Natural Language Processing, NLP) та комп'ютерного зору (Computer Vision, CV).

Обробка природної мови надає комп'ютерам можливість "розуміти" та інтерпретувати людську мову. Для задач аналізу безпеки текстів (виявлення дезінформації, ІПСО, мови ворожнечі тощо), які часто зводяться до класифікації тексту, застосовуються різні нейромережеві архітектури:

1) Рекурентні нейронні мережі (RNN): Ці архітектури, зокрема їх більш просунуті варіанти LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), добре підходять для обробки послідовних даних, якими є текст [19]. Вони використовують внутрішній стан ("пам'ять") для врахування контексту попередніх слів при аналізі поточного слова [19, 20]. Завдяки вентильним механізмам (gates), LSTM та GRU здатні ефективно навчатися на довгих послідовностях та виявляти довгострокові залежності, що важливо для розуміння нюансів тексту та виявлення маніпулятивних технік [20]. Саме архітектура на основі LSTM була обрана для розробки моделі аналізу тексту на предмет інформаційно-психологічних операцій в рамках паралельного дослідження Долінко К. В. Цікавою особливістю розробленої моделі є те, що вона враховує не лише семантику самого текстового повідомлення, але й додаткову ознаку – показник надійності джерела

повідомлення, що потенційно дозволяє підвищити точність виявлення ІПСО, інтегруючи метадані про походження інформації в процес класифікації.

2) Трансформерні архітектури: Моделі, що базуються на архітектурі Transformer (запропонованій у статті "Attention Is All You Need" [21]), такі як BERT, GPT, RoBERTa, здійснили значний прорив в NLP. Ключовою інновацією є механізм само-уваги (self-attention), який дозволяє моделі зважувати важливість різних слів у реченні незалежно від їхньої позиції, що ефективно фіксує контекстуальні зв'язки. Трансформери можуть обробляти послідовності паралельно, що прискорює навчання. Модель BERT, зокрема, обробляє текст двонаправлено, враховуючи контекст як зліва, так і справа від слова [22]. Такі моделі зазвичай попередньо навчаються на величезних текстових даних, а потім швидко донавчаються (fine-tuning) для конкретних завдань класифікації тексту з високою точністю [22]. Популярна бібліотека Hugging Face Transformers надає доступ до безлічі попередньо навчених моделей та інструментів для їх використання [23]. Хоча в даному проекті для аналізу тексту використовується LSTM, трансформерні моделі залишаються важливою альтернативою та можуть бути розглянуті для подальшого розвитку системи.

Для інтеграції розробленої LSTM-моделі (або будь-якої іншої NLP-моделі) у програмний бот, необхідний чіткий програмний інтерфейс (API). В рамках даного проекту, цей інтерфейс реалізовано у вигляді функції predict_single_message (як показано у коді моделі), яка приймає на вхід текст повідомлення та ID джерела (для визначення його надійності), а повертає результат класифікації (мітку категорії та ступінь впевненості) та оновлене значення надійності джерела. Це дозволяє боту взаємодіяти з моделлю аналізу тексту та використовувати її результати.

Комп'ютерний зір дозволяє системам "бачити" та інтерпретувати візуальну інформацію. Основою для аналізу зображень за допомогою глибокого навчання є згорткові нейронні мережі (Convolutional Neural Networks, CNN) [18].

1) Принцип роботи CNN: CNN використовують шари згортки (convolution), які застосовують фільтри (ядра) до вхідного зображення для виявлення локальних ознак (країв, текстур, кутів). Шари підвибірки (pooling) зменшують просторову

розмірність карт ознак, зберігаючи найважливішу інформацію та роблячи модель більш стійкою до невеликих зсувів об'єктів. Функції активації, як ReLU, вводять нелінійність. Наприкінці мережі розташовані повнозв'язні шари (fully connected), які приймають оброблені ознаки та виконують фінальну класифікацію [18]. Перевагами CNN є спільне використання параметрів (parameter sharing) у фільтрах та певна інваріантність до зсуву (translation invariance).

2) Популярні архітектури: З часом було розроблено багато ефективних архітектур CNN, серед яких:

а) VGG: Характеризується простою уніфікованою структурою з повторюваними блоками згорткових шарів.

б) ResNet (Residual Networks): Вирішує проблему деградації глибоких мереж за допомогою залишкових блоків (residual blocks) та пропусків з'єднань (skip connections), що дозволяє ефективно навчати дуже глибокі мережі [24].

в) Inception (GoogLeNet): Використовує модулі з паралельними згортковими фільтрами різних розмірів для ефективного захоплення ознак на різних масштабах [25].

г) EfficientNet: Застосовує метод комбінованого масштабування (compound scaling), одночасно збільшуючи глибину, ширину та роздільну здатність мережі, що забезпечує високу точність із меншою кількістю параметрів [25].

3) Трансферне навчання (Transfer Learning): Цей підхід є надзвичайно популярним у комп'ютерному зорі. Моделі, попередньо навчені на великих датасетах (як ImageNet), вже містять знання про загальні візуальні ознаки. Їх можна використовувати як:

а) Екстрактор ознак: "Заморозити" згорткові шари попередньо навченої моделі та навчити лише нові повнозв'язні шари для цільової задачі.

б) Основу для донавчання (fine-tuning): "Розморозити" частину або всі шари попередньо навченої моделі та донавчити їх на цільовому датасеті з меншою швидкістю навчання. Це значно скорочує час навчання та потребу в даних для нової задачі [26]. Популярні фреймворки, як TensorFlow/Keras та PyTorch, надають легкий доступ до попередньо навчених моделей.

4) Готові рішення для аналізу безпеки: Для стандартних задач, таких як виявлення неприйнятного контенту (NSFW), існують готові рішення:

а) Відкриті моделі: Наприклад, OpenNSFW2, Falconsai/nsfw_image_detection, LAION-AI/CLIP-based-NSFW-Detector [27].

б) Хмарні API: Сервіси Google Cloud Vision AI, AWS Rekognition, Azure Computer Vision надають готові API для детекції відвертого контенту, аналізу облич, об'єктів тощо, але зазвичай є платними [28].

Отже, сучасні методи глибокого навчання надають широкий спектр інструментів для автоматизованого аналізу текстів та зображень. Розуміння основних принципів їх роботи є необхідним для проектування системи (бота), яка буде ефективно інтегрувати ці можливості для вирішення задач безпеки контенту.

1.5 Обґрунтування вибору інструментів для розробки бота

Вибір програмних засобів та технологій є важливим етапом проектування будь-якої програмної системи, оскільки він безпосередньо впливає на швидкість розробки, надійність, можливості масштабування та подальшу підтримку продукту. Для розробки програмного бота для аналізу безпеки текстових та графічних даних в рамках даної кваліфікаційної роботи було обрано наступний стек технологій, виходячи з їхніх переваг та відповідності поставленим завданням:

1) Мова програмування: Python. Python є де-факто стандартом у галузі машинного навчання (ML), штучного інтелекту (AI) та обробки даних, що робить його оптимальним вибором для даного проекту. Ключові переваги:

а) Простота та читабельність: Відносно простий синтаксис Python дозволяє розробникам зосередитися на вирішенні проблем, прискорює розробку та полегшує читання і підтримку коду [29].

б) Розгалужена екосистема бібліотек: Python має надзвичайно багату екосистему бібліотек, критично важливих для даного проекту: для машинного та глибокого навчання (TensorFlow, PyTorch, Keras), NLP (Hugging Face Transformers, NLTK, spaCy), комп'ютерного зору (OpenCV, Pillow), обробки даних (Pandas, NumPy) та візуалізації (Matplotlib, Seaborn) [30].

в) Платформонезалежність: Код на Python може працювати на різних операційних системах [30].

г) Активна спільнота: Забезпечує доступ до численних навчальних матеріалів, форумів та готових рішень [30].

д) Інтеграційні можливості: Легко інтегрується з іншими мовами та технологіями, зокрема з веб-фреймворками (Flask, FastAPI) для створення API [29].

2) Бібліотека для Telegram Bot API: pyTelegramBotAPI. Для взаємодії з Telegram Bot API було обрано бібліотеку pyTelegramBotAPI [9]. Цей вибір зумовлений її відносною простотою використання, достатньою функціональністю для реалізації поставлених завдань (обробка повідомлень, клавіатур, Polling) та активною підтримкою на момент початку розробки.

3) Середовище розробки: Visual Studio Code (VS Code). VS Code було обрано як основне IDE завдяки поєднанню легкості, гнучкості та потужних функцій: відмінна підтримка Python (IntelliSense, дебагер, віртуальні середовища, лінтери), вбудовані інструменти (термінал, Git), величезний магазин розширень та кросплатформеність [31].

4) Система контролю версій: Git та GitHub. Використання Git [32] та хмарної платформи GitHub є необхідним для сучасного процесу розробки, забезпечуючи відстеження змін, гілкування, ефективну колаборацію та безпечне зберігання коду.

5) Інструменти для машинного навчання (для інтеграції): Архітектура бота проектувалася з урахуванням необхідності взаємодії з нейромережевою моделлю аналізу тексту, розробленою в рамках паралельного дослідження. Ця модель була реалізована за допомогою стандартного для індустрії фреймворку TensorFlow та його високорівневого API Keras [33]. Передбачається, що взаємодія бота з моделлю відбуватиметься через чітко визначений програмний інтерфейс (у даному випадку – через виклик відповідної Python-функції, що інкапсулює логіку передбачення).

Таким чином, обраний стек технологій є збалансованим рішенням, що поєднує потужні можливості Python для роботи зі штучним інтелектом та API, зручні інструменти розробки та стандартні практики управління кодом, що дозволяє ефективно реалізувати поставлені в кваліфікаційній роботі завдання.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА ІНТЕРФЕЙСУ ПРОГРАМНОГО БОТА ДЛЯ АНАЛІЗУ ТЕКСТУ

2.1 Формулювання функціональних та нефункціональних вимог до бота

Функціональні вимоги описують конкретні дії та операції, які повинен виконувати програмний бот для досягнення поставленої мети – аналізу безпеки текстових даних.

1) ФВ.1: Запуск та початкова взаємодія:

а) ФВ.1.1: Бот повинен реагувати на команду /start, надсилаючи вітальне повідомлення з описом своїх можливостей (аналіз тексту) та основних команд.

б) ФВ.1.2: Бот повинен реагувати на команду /help, надаючи користувачеві довідкову інформацію про аналіз тексту та способи використання.

2) ФВ.2: Прийом текстових даних від користувача:

а) ФВ.2.1: Бот повинен приймати текстові повідомлення, надіслані користувачем безпосередньо в чат з ботом.

б) ФВ.2.2: Бот повинен приймати текстові повідомлення, переслані (forwarded) користувачем з інших чатів (приватних, групових, каналів), ідентифікуючи джерело пересилання (source_id).

3) ФВ.3: Аналіз текстових даних:

а) ФВ.3.1: При отриманні текстового повідомлення (прямого або пересланого), бот повинен передати текст повідомлення та ідентифікатор джерела (source_id, якщо повідомлення переслане, інакше None) на аналіз до інтегрованого модуля аналізу тексту на предмет ІПСО (шляхом виклику відповідної функції).

б) ФВ.3.2: Бот повинен отримати результат аналізу від модуля ШІ, що включає вердикт (is_ipso), категорію ІПСО (ipso_category, якщо є) та показник надійності джерела (final_reliability).

в) ФВ.3.3: Бот повинен сформувати та надіслати користувачеві зрозумілу відповідь, що відображає результат аналізу тексту (наприклад, вказуючи

на виявлену категорію ІПСО або підтверджуючи відсутність ознак, можливо, згадуючи надійність джерела).

4) ФВ.4: Інтерфейс користувача:

а) ФВ.4.1: Взаємодія має бути простою: користувач надсилає або пересилає текст, бот аналізує його та надсилає результат. Додаткові кнопки для запуску аналізу не передбачаються для спрощення інтерфейсу.

б) ФВ.4.2: Повідомлення бота мають бути інформативними, стилістично нейтральними та зрозумілими для користувача.

5) ФВ.5: Обробка некоректних даних:

а) ФВ.5.1: Бот повинен коректно реагувати на отримання даних нетекстового типу (наприклад, зображення, відео, аудіо, стікери), інформуючи користувача, що він аналізує лише текст.

б) ФВ.5.2: Бот повинен обробляти можливі помилки при взаємодії з модулем аналізу ШІ або при роботі з базою даних надійності, повідомляючи користувача про неможливість виконати аналіз у даний момент (без технічних деталей).

Нефункціональні вимоги визначають атрибути якості та обмеження системи.

1) НФВ.1: Продуктивність:

а) НФВ.1.1: Час відповіді бота на аналіз текстового повідомлення має бути прийнятним для користувача і не перевищувати, наприклад, 10 секунд (з урахуванням часу на обробку LSTM-моделлю).

б) НФВ.1.2: Бот повинен бути здатним обробляти одночасні запити від кількох користувачів (в рамках можливостей сервера).

2) НФВ.2: Надійність:

а) НФВ.2.1: Бот повинен працювати стабільно в режимі 24/7 при розгортанні на сервері.

б) НФВ.2.2: Бот повинен мати механізми обробки винятків для уникнення повних збоїв через помилки API Telegram, помилки в ШІ-модулі або проблеми з БД.

в) НФВ.2.3: Операції з базою даних надійності мають бути атомарними або стійкими до збоїв, щоб уникнути некоректних значень.

3) НФВ.3: Безпека:

а) НФВ.3.1: Токен бота та, можливо, інші конфігураційні дані мають зберігатися безпечно (напр., у змінних середовища).

б) НФВ.3.2: Бот не повинен логувати або зберігати тексти повідомлень користувачів без явної необхідності та згоди (або знеособлено).

в) НФВ.3.3: Доступ до файлу бази даних SQLite має бути обмежений лише процесом бота.

4) НФВ.4: Масштабованість:

а) НФВ.4.1: Архітектура бота повинна дозволяти відносно легко оновлювати або замінювати модель аналізу тексту в майбутньому.

5) НФВ.5: Простота використання (Usability):

а) НФВ.5.1: Взаємодія з ботом має бути інтуїтивною для користувачів Telegram.

б) НФВ.5.2: Результати аналізу мають бути представлені чітко та зрозуміло.

6) НФВ.6: Супроводжуваність (Maintainability):

а) НФВ.6.1: Код бота має бути структурованим (розбитим на логічні модулі), документованим (коментарі, docstrings) та відповідати PEP 8.

б) НФВ.6.2: Має бути реалізовано логування основних подій та помилок для полегшення діагностики.

2.2 Проектування архітектури програмного бота

Для забезпечення супроводжуваності, масштабованості та логічної організації коду програмного бота було обрано модульну архітектуру. Проект структуровано таким чином, щоб розділити відповідальність між різними компонентами системи: конфігурацією, обробкою запитів від Telegram, взаємодією з модулем аналізу тексту та допоміжними функціями.

Проект має наступну основну структуру каталогів та файлів:

```
TelegramSecurityBot/
├── .env    # Файл для зберігання секретів (токен бота)
├── .gitignore     # Файл для ігнорування Git
├── requirements.txt     # Список залежностей Python
├── main.py       # Головний скрипт запуску бота
├── bot/    # Основний пакет з кодом логіки бота
│   ├── __init__.py
│   ├── config.py         # Модуль завантаження конфігурації
│   ├── handlers/         # Пакет з обробниками повідомлень Telegram
│       ├── __init__.py
│       ├── common.py         # Обробники спільних команд (/start, /help)
│       └── text.py       # Обробник текстових повідомлень
│
└── ai_model/     # Окрема директорія для коду та ресурсів ШІ
    ├── __init__.py
    ├── ipso_model.py   # Файл з функціями ШІ (завантаження, передбачення, БД)
    ├── ipso_database_binary.db       # Файл бази даних SQLite
    ├── tokenizer_binary_30k.json     # Файл збереженого токенізатора
    ├── label_encoder_binary.pkl      # Файл збереженого кодувальника міток
    └── model_weights_binary/         # Директорія з вагами моделі
        └── ipso_rnn_binary_lstm_30k.weights.h5  # Файл ваг моделі
```

Опис основних модулів:

1) main.py (Точка входу):

а) Відповідає за ініціалізацію системи: завантажує конфігурацію з bot.config, ініціалізує з'єднання з базою даних (ipso_database_binary.db) за допомогою функції з ai_model.ipso_model, завантажує ресурси ШІ (модель, токенізатор, кодувальник міток) також через виклик відповідної функції з ai_model.ipso_model, та створює екземпляр бота (telebot.TeleBot).

б) Реєструє обробники повідомлень з пакета bot/handlers, передаючи їм необхідні залежності (екземпляр бота, об'єкти ШІ-модуля, з'єднання з БД).

в) Запускає основний цикл роботи бота (метод bot.polling()) та забезпечує коректне закриття з'єднання з БД при завершенні роботи.

2) bot/config.py (Модуль конфігурації):

а) Забезпечує завантаження конфігураційних параметрів (зокрема, BOT_TOKEN) з файлу .env за допомогою бібліотеки python-dotenv та визначає шляхи до ресурсів ШІ (DB_PATH, MODEL_WEIGHTS_PATH і т.д.), необхідних для роботи main.py та ai_model.ipso_model.

3) Пакет bot/handlers/ (Обробники повідомлень):

а) common.py: Містить логіку для обробки стандартних команд /start та /help, формуючи відповідні вітальні та довідкові повідомлення.

б) text.py: Реалізує основну логіку обробки вхідних текстових повідомлень (прямих та пересланих). Безпосередньо викликає функцію аналізу тексту (predict_single_message) з модуля ai_model.ipso_model, отримує результат (висновок про наявність ІПСО та надійність джерела), формує відповідь користувачеві та надсилає її. Також відповідає за визначення source_id для пересланих повідомлень.

4) Пакет ai_model/ (Модуль аналізу тексту):

а) Розроблений в рамках паралельного дослідження Долінко К.В., інкапсулює всю логіку, пов'язану зі штучним інтелектом.

б) ipso_model.py: Містить функції для ініціалізації БД (init_db), побудови архітектури моделі (build_ipso_rnn_model), завантаження всіх ресурсів ШІ (load_ai_resources), очистки тексту (clean_text) та безпосередньо для аналізу повідомлення і оновлення надійності (predict_single_message).

в) Інші файли в цій папці (.db, .json, .pkl, .h5 у підпапці model_weights_binary/) є необхідними ресурсами для роботи ШІ-модуля (база даних, токенізатор, кодувальник, ваги моделі).

Взаємодію основних компонентів системи можна представити наступною схемою

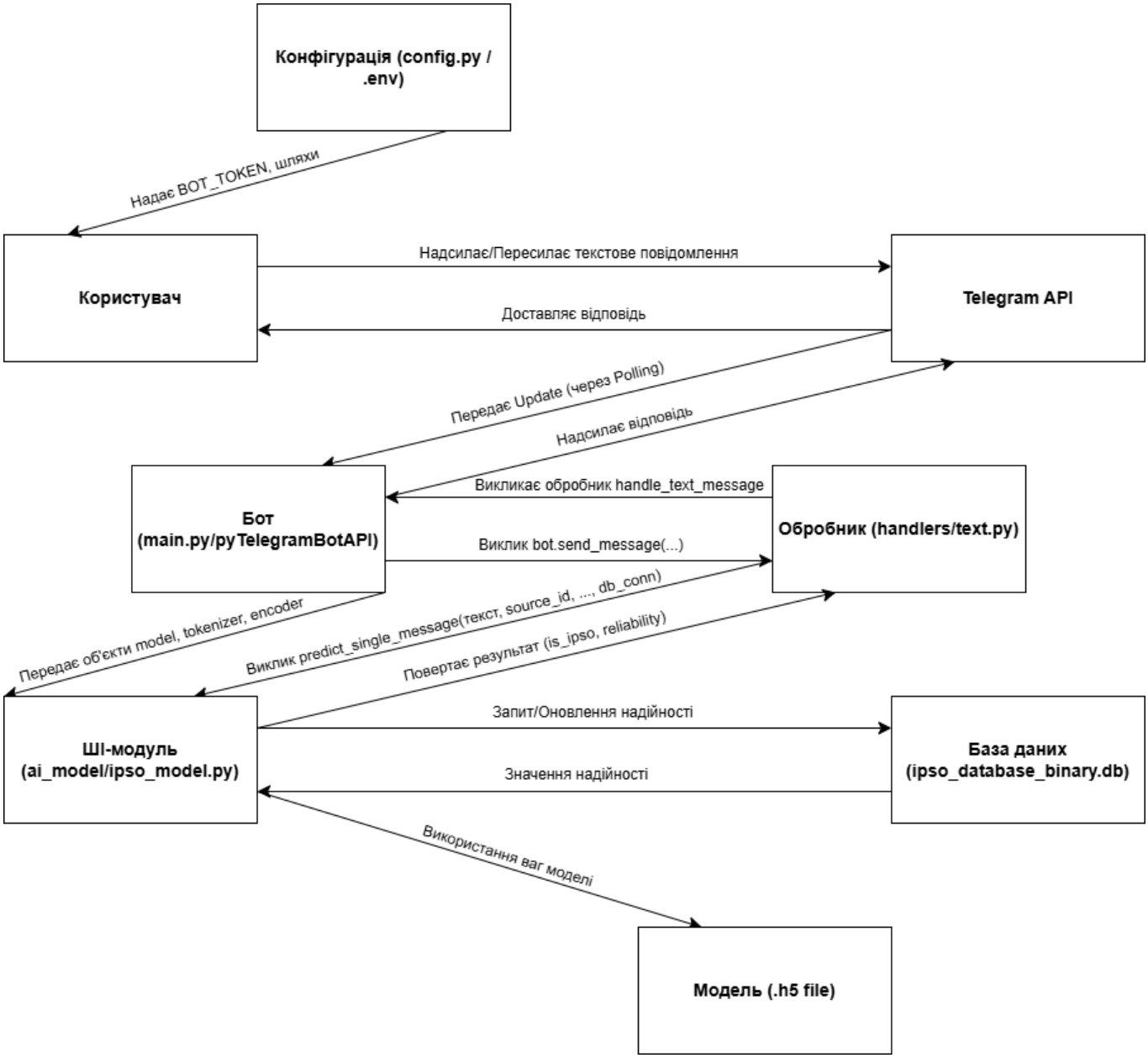

Рисунок 2.1 – Схема взаємодії модулів програмного бота

Наведена на рис. 2.1 схема ілюструє послідовність взаємодії основних компонентів розробленої системи при обробці текстового повідомлення користувача. Процес починається з отримання повідомлення через Telegram API, яке передається на обробку основному модулю бота. Відповідний обробник (handlers/text.py) готує дані (текст та ідентифікатор джерела) і викликає функцію аналізу з ШІ-модуля (ai_model/ipso_model.py), передаючи їй також необхідні об'єкти моделі, токенізатора, кодувальника та з'єднання з базою даних. ШІ-модуль виконує передбачення, використовуючи ваги моделі та звертаючись до бази даних

для отримання/оновлення показника надійності джерела. Результат аналізу повертається обробнику, який формує відповідь та надсилає її користувачеві через Telegram API. Окремо показано отримання ботом конфігураційних параметрів (зокрема, токена) при ініціалізації.

2.3 Проектування інтерфейсу взаємодії з користувачем у Telegram

Ефективність та зручність використання Telegram-бота значною мірою залежать від якості проектування його інтерфейсу користувача (User Interface, UI) та досвіду взаємодії (User Experience, UX). Інтерфейс має бути інтуїтивно зрозумілим, не перевантаженим та надавати користувачеві чіткий зворотний зв'язок.

Для взаємодії з ботом передбачено наступні елементи інтерфейсу:

1) Стандартні команди:

а) /start: Ініціює діалог з ботом. У відповідь бот надсилає вітальне повідомлення, коротко описує свою функцію (аналіз тексту на предмет ІПСО, враховуючи надійність джерела) та пояснює, як ним користуватися (просто надіслати або переслати текст).

б) /help: Надає розгорнуту довідкову інформацію, включаючи опис того, що бот аналізує, як інтерпретувати результати (виявлення ІПСО, значення надійності джерела), та можливі обмеження.

2) Кнопки:

Для спрощення взаємодії та уникнення необхідності користувачеві вводити додаткові команди, основний сценарій використання не передбачає наявності спеціальних кнопок ReplyKeyboardMarkup для запуску аналізу. Бот автоматично аналізує будь-яке отримане текстове повідомлення.

3) Повідомлення бота:

а) Вітальне повідомлення (при команді /start): Має бути коротким, привітним та інформативним. Приклад: "Привіт! Я DisRaze - бот на основі штучного інтелекту, який допоможе тобі виявити дезінформацію та ознаки ІПСО у текстових повідомленнях. Просто надішли мені будь-яке текстове повідомлення

або перешли його з іншого чату/каналу, і я спробую його проаналізувати, враховуючи також надійність джерела. Для довідки використовуй /help."

б) Довідкове повідомлення (при команді /help): Має містити детальніший опис функціоналу, пояснення концепції "надійності джерела" та поточних обмежень (аналіз лише тексту, рекомендаційний характер висновків). Приклад тексту наведено у п. 4.2.1. (Можна послатися на текст з Розділу 4, де ти тестував, або навести тут актуальний текст з common.py).

в) Повідомлення з результатом аналізу: Це ключове повідомлення, що чітко вказує результат перевірки. Формат використовує HTML-розмітку для кращого сприйняття:

- Якщо ознак ІПСО не виявлено: "✅ Аналіз завершено. Ознак ІПСО не виявлено. Джерело: [ID джерела або "Пряме повідомлення"] ([Контекст пересилання]). Надійність джерела: [Значення]%."

- Якщо виявлено ознаки ІПСО: "⚠️ Виявлено ознаки ІПСО! Джерело: [ID джерела або "Пряме повідомлення"] ([Контекст пересилання]). Поточна надійність джерела: [Значення]%. Будьте обережні з інформацією з цього джерела."

г) Повідомлення про непідтримуваний тип контенту: "Вибачте, я призначений для аналізу лише текстових повідомлень. Будь ласка, надішліть текст."

д) Повідомлення про помилку аналізу: "На жаль, сталася помилка під час аналізу вашого повідомлення. Спробуйте пізніше."

Основний сценарій використання передбачає наступну логіку обробки:

1) Користувач надсилає текстове повідомлення боту або пересилає його з іншого чату.

2) Бот отримує об'єкт Update з інформацією про Message.

3) Бот перевіряє тип контенту. Якщо це текст:

а) Визначає текст повідомлення (message.text).

б) Визначає ідентифікатор джерела (source_id):

- Якщо повідомлення переслане (наявні поля message.forward_from_chat або message.forward_from), source_id встановлюється як ID відповідного чату або користувача.

- Якщо повідомлення пряме (не переслане), source_id встановлюється як None.

в) Передає підготовлений текст та source_id (разом з іншими необхідними об'єктами: моделлю, токенізатором, кодувальником міток, з'єднанням з БД, max_length, threshold) до функції predict_single_message ШІ-модуля.

г) Отримує результат аналізу у вигляді кортежа (is_ipso, final_reliability).

д) Формує відповідь користувачеві з результатом аналізу, використовуючи формати повідомлень, описані у п. 2.3.

е) Надсилає сформовану відповідь користувачеві.

4) Якщо тип контенту не текстовий:

а) Надсилає користувачеві повідомлення про підтримку лише текстових даних.

Такий підхід забезпечує простий та інтуїтивно зрозумілий спосіб взаємодії для користувача, мінімізуючи необхідність вивчення складних команд чи інтерфейсів.

2.4. Проектування інтерфейсу взаємодії з модулем аналізу тексту

Для забезпечення чіткої та надійної взаємодії між програмним ботом та нейромережевим модулем аналізу тексту (ai_model.ipso_model.py), розробленим в рамках паралельного дослідження Долінко К. В. , було визначено програмний інтерфейс (Application Programming Interface, API). Враховуючи, що обидва компоненти (бот та ШІ-модуль з його залежностями та базою даних) плануються до розгортання в єдиному середовищі виконання, найпростішим та найефективнішим рішенням є реалізація цього інтерфейсу у вигляді прямого виклику Python-функції.

Було спроектовано та реалізовано функцію predict_single_message у модулі ai_model.ipso_model.py, яка інкапсулює логіку роботи з нейронною мережею та базою даних надійності джерел і має наступний інтерфейс:

Назва функції: predict_single_message

Вхідні параметри:

1) model: Об'єкт навченої моделі TensorFlow/Keras (tensorflow.keras.models.Model), попередньо завантажений при старті бота.

2) tokenizer: Об'єкт токенізатора Keras (tensorflow.keras.preprocessing.text.Tokenizer), попередньо завантажений та навчений на відповідному корпусі текстів.

3) label_encoder: Об'єкт кодувальника міток Scikit-learn (sklearn.preprocessing.LabelEncoder), попередньо завантажений та навчений на повному списку можливих класів.

4) db_conn: Об'єкт з'єднання з базою даних SQLite (sqlite3.Connection), ініційований при старті бота.

5) text: Рядок (str), що містить текст повідомлення, отриманий від користувача, для аналізу.

6) source_id: Рядок (str) або None, що представляє ідентифікатор джерела повідомлення (каналу, групи або користувача для пересланих повідомлень).

7) max_length: Ціле число (int), максимальна довжина послідовності токенів, до якої буде доповнено або обрізано вхідний текст (відповідає параметру, використаному при навчанні моделі, та передається з конфігурації).

8) threshold: Число з плаваючою комою (float), поріг для бінарної класифікації, що визначає межу для віднесення повідомлення до класу ІПСО (значення за замовчуванням у функції 0.5).

Вихідні дані (функція повертає кортеж):

1) is_ipso: Булеве значення (bool). True, якщо передбачена моделью ймовірність належності тексту до класу ІПСО (prediction_prob_ipso) більша або дорівнює значенню threshold; False в іншому випадку.

2) final_reliability: Число з плаваючою комою (float). Поточне (або оновлене, враховуючи результат is_ipso та наявність source_id) значення надійності джерела повідомлення.

Процес взаємодії:

1) Програмний бот (модуль bot/handlers/text.py) при отриманні текстового повідомлення готує вхідні дані: виділяє текст (text) та визначає ідентифікатор джерела (source_id).

2) Бот викликає функцію predict_single_message, передаючи їй підготовлені дані, а також попередньо ініціалізовані при старті бота об'єкти model, tokenizer, label_encoder, db_conn та значення max_length і threshold.

3) Функція predict_single_message всередині себе виконує всю необхідну логіку: очистку тексту, токенізацію, доповнення/обрізання послідовності, отримання поточної надійності з бази даних, перевірку хешу (для уникнення повторної обробки вже проаналізованого повідомлення з того ж джерела), формування вхідних тензорів для моделі, отримання передбачення (ймовірності ІПСО), визначення класу is_ipso на основі заданого порогу, оновлення показника надійності в базі даних (якщо повідомлення нове та джерело відоме) та збереження хешу обробленого повідомлення.

4) Функція повертає боту кортеж з двох значень: (is_ipso, final_reliability).

5) Бот (модуль bot/handlers/text.py) отримує цей результат та використовує його для формування та надсилання фінальної відповіді користувачеві.

Такий підхід прямого виклику функції забезпечує максимальну продуктивність взаємодії (завдяки відсутності мережевих затримок), простоту реалізації та дозволяє легко передавати складні об'єкти (модель, токенізатор) і керувати спільними ресурсами (такими як з'єднання з базою даних).

3 РОЗРОБКА ТА ІНТЕГРАЦІЯ КОМПОНЕНТІВ ПРОГРАМНОГО БОТА

3.1 Налаштування середовища розробки та системи контролю версій (git)

Для забезпечення ефективного та організованого процесу розробки було виконано наступні підготовчі кроки:

1) Створення структури проекту: На локальному комп'ютері було створено основну директорію проекту (TelegramSecurityBot) та внутрішню структуру каталогів (bot, bot/handlers, bot/analysis, ai_model, ai_model/model_weights_binary) відповідно до архітектури, описаної в п. 2.2.

2) Налаштування віртуального середовища: Для ізоляції залежностей проекту та уникнення конфліктів з іншими Python-пакетами в системі було створено віртуальне середовище за допомогою вбудованого модуля venv: python -m venv venv

Подальша робота з проектом, включаючи встановлення бібліотек та запуск скриптів, здійснювалася в активованому віртуальному середовищі (командою venv\Scripts\activate для Windows cmd). Папка віртуального середовища (venv/) була додана до файлу .gitignore для виключення її з системи контролю версій.

3) Встановлення залежностей: За допомогою менеджера пакетів pip було встановлено основні бібліотеки, необхідні для роботи бота та взаємодії з ШІ-модулем. Точні версії зафіксовано у файлі requirements.txt. Ключові залежності включають:

а) pyTelegramBotAPI: Для взаємодії з Telegram Bot API.

б) python-dotenv: Для безпечного завантаження конфігураційних параметрів (токену) з .env файлу.

в) tensorflow: Фреймворк глибокого навчання, необхідний для завантаження та використання моделі аналізу тексту.

г) pandas, numpy, scikit-learn: Бібліотеки, що використовуються ШІ-модулем для обробки даних та підготовки тексту.

д) nltk: Бібліотека для обробки природної мови.

Всі залежності були встановлені командою pip install -r requirements.txt.

4) Налаштування системи контролю версій:

а) Локальний Git-репозиторій було ініціалізовано (git init).

б) Було створено та налаштовано файл .gitignore для виключення з відстеження файлів конфігурації (.env), папки віртуального середовища (venv/), кешу Python (pycache/, *.pyc), файлів бази даних (ai_model/ipso_database*.db), ваг моделі (ai_model/model_weights_binary/), файлів токенізатора та кодувальника (ai_model/*.json, ai_model/*.pkl), а також даних для навчання (ai_model/data/).

в) Було налаштовано ідентифікаційні дані користувача Git (git config).

г) Було створено віддалений репозиторій на платформі GitHub та пов'язано з локальним (git remote add origin).

д) Початкова структура проекту та код були додані, зафіксовані та відправлені (git add, git commit, git push) до віддаленого репозиторію.

5) Налаштування середовища розробки (VS Code): Було встановлено та налаштовано IDE Visual Studio Code, включаючи розширення для Python, що забезпечує зручні інструменти для написання, тестування та налагодження коду, а також інтеграцію з Git. Було налаштовано використання інтерпретатора Python з віртуального середовища проекту.

Виконання цих кроків дозволило створити організоване, контрольоване та відтворюване середовище для подальшої розробки програмних модулів бота та їх інтеграції.

3.2 Програмна реалізація основних модулів бота

Після налаштування середовища розробки було здійснено програмну реалізацію основних модулів бота відповідно до спроектованої архітектури (п. 2.2) та функціональних вимог (п. 2.1). Розробка велася на мові програмування Python з використанням бібліотеки pyTelegramBotAPI.

Завданням цього блоку є безпечне завантаження конфігураційних параметрів та ініціалізація основних об'єктів системи перед запуском основного циклу бота.

1) Модуль config.py: Відповідає за завантаження конфігурації з файлу .env за допомогою бібліотеки python-dotenv. Основним параметром, що зчитується, є авторизаційний токен бота (BOT_TOKEN). Також у цьому модулі централізовано визначені шляхи до файлових ресурсів ШІ-модуля (DB_PATH, MODEL_WEIGHTS_PATH, TOKENIZER_PATH, LABEL_ENCODER_PATH) та параметр максимальної довжини послідовності MAX_LENGTH для моделі. Реалізовано перевірку наявності BOT_TOKEN. Фрагмент коду, що ілюструє визначення шляхів у модулі bot/config.py наведено нижче.

```
# bot/config.py
import os
from dotenv import load_dotenv

PROJECT_ROOT = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
# ... (завантаження .env) ...
AI_MODEL_DIR = os.path.join(PROJECT_ROOT, 'ai_model')
DB_PATH = os.path.join(AI_MODEL_DIR, 'ipso_database_binary.db')
# ... (інші шляхи з файлу config.py) ...
MAX_LENGTH = 200
```

2) Модуль main.py (Ініціалізація): Головний скрипт системи, що виконує наступні ключові дії при запуску:

а) Імпортує необхідні конфігураційні параметри та функції з інших модулів проекту;

б) Викликає функцію init_db з ШІ-модуля ai_model.ipso_model для ініціалізації або перевірки бази даних SQLite та отримує об'єкт з'єднання db_conn;

в) Викликає функцію load_ai_resources з ШІ-модуля для завантаження навченої нейромережевої моделі, токенізатора та кодувальника міток;

г) Створює екземпляр бота telebot.TeleBot, передаючи йому токен та вказуючи parse_mode='HTML' для відповідей за замовчуванням;

д) Реєструє обробники команд та повідомлень з пакета bot.handlers, передаючи їм усі необхідні залежності (екземпляр бота, завантажені об'єкти ШІ, об'єкт з'єднання з БД, параметр MAX_LENGTH та поріг класифікації);

е) Запускає основний цикл роботи бота методом bot.polling() та забезпечує коректне закриття з'єднання з БД (db_conn.close()) у блоці finally при завершенні роботи програми.

Повний лістинг файлу main.py наведено у Додатку Б.1.

Обробники подій, що надходять від Telegram, реалізовані як Python-функції, які реєструються в main.py для відповідних команд або типів повідомлень.

1) Модуль handlers/common.py: Містить обробники стандартних команд, що забезпечують базову взаємодію з користувачем:

а) Функція start_handler: зареєстрована для обробки команди /start. При її виклику бот надсилає користувачеві вітальне повідомлення з описом основного функціоналу та інструкціями щодо використання (приклад тексту наведено в п. 2.3);

б) Функція help_handler: зареєстрована для команди /help. Надає користувачеві розгорнуту довідкову інформацію про можливості бота, інтерпретацію результатів аналізу та наявні обмеження.

Реєстрація цих обробників здійснюється за допомогою методу bot_instance.register_message_handler() з передачею відповідних команд.

2) Модуль handlers/text.py: Реалізує ключову логіку для аналізу текстових повідомлень:

а) Функція handle_text_message: зареєстрована для обробки всіх вхідних повідомлень типу content_types=['text']. Вона отримує на вхід об'єкт message від Telegram та набір залежностей (екземпляр бота, об'єкти моделі, токенізатора, кодувальника, з'єднання з БД, MAX_LENGTH, threshold), переданих з main.py за допомогою functools.partial;

б) Основна логіка функції handle_text_message включає: визначення тексту повідомлення та ідентифікатора джерела source_id; виклик функції predict_single_message з ШІ-модуля ai_model.ipso_model для отримання результату аналізу (is_ipso, final_reliability); формування текстової відповіді користувачеві з використанням HTML-розмітки для візуального виділення результату та показника надійності джерела; надсилання сформованої відповіді методом reply_to;

в) Функція handle_other_messages: зареєстрована для обробки всіх типів контенту, окрім текстового. При отриманні такого повідомлення бот інформує користувача, що він призначений лише для аналізу тексту.

Реєстрація цих обробників у main.py здійснюється за допомогою функції register_text_handlers.

Взаємодія з користувачем побудована на обміні текстовими повідомленнями та використанні стандартних команд /start та /help, що відповідає проектним рішенням (п. 2.3.1).

1) Надсилання повідомлень користувачеві реалізовано за допомогою методів send_message та reply_to бібліотеки pyTelegramBotAPI.

2) Для покращення візуального сприйняття та читабельності результатів аналізу у відповідях бота активно використовується HTML-форматування (parse_mode='HTML').

3) Згідно з проектним рішенням щодо спрощення інтерфейсу, додаткові кнопки ReplyKeyboardMarkup для вибору режиму аналізу не використовуються; бот автоматично аналізує кожне отримане текстове повідомлення.

3.3 Реалізація механізму інтеграції з модулем аналізу тексту

Одним із центральних завдань розробки була інтеграція програмного Telegram-бота з нейромережевим модулем аналізу тексту (ai_model/ipso_model.py), розробленим в рамках паралельного дослідження. Як було визначено на етапі проектування (п. 2.4), інтеграція реалізована шляхом прямого виклику Python-функції predict_single_message з коду обробника текстових повідомлень.

Клієнтська частина інтеграції реалізована всередині функції handle_text_message модуля bot/handlers/text.py. Процес виклику функції ШІ включає наступні кроки:

1) Отримання Залежностей: Функція handle_text_message отримує всі необхідні для виклику ШІ об'єкти як аргументи під час її реєстрації в main.py за допомогою functools.partial. Це включає екземпляр моделі (model), токенізатор

(tokenizer), кодувальник міток (label_encoder), активне з'єднання з базою даних (db_conn), а також параметри max_length і threshold.

2) Підготовка Вхідних Даних для Функції ШІ:

а) З об'єкта message вилучається текст повідомлення (text = message.text).

б) Визначається ідентифікатор джерела (source_id). Перевіряються поля message.forward_from_chat та message.forward_from. Якщо одне з них існує, відповідний id (чату або користувача) перетворюється на рядок і присвоюється source_id. В іншому випадку source_id залишається None.

3) Безпосередній Виклик Функції: Здійснюється виклик функції predict_single_message, імпортованої з ai_model.ipso_model, з передачею всіх підготовлених параметрів:

```
# bot/handlers/text.py
# ...
is_ipso, final_reliability = predict_single_message(
model=model,
tokenizer=tokenizer,
label_encoder=label_encoder,
db_conn=db_conn,
text=text_to_analyze,
source_id=source_id,
max_length=max_length,
threshold=threshold
)
# ...
```

Обробка Винятків: Виклик обгорнутий у блок try...except для перехоплення можливих помилок, що можуть виникнути під час виконання функції predict_single_message (наприклад, помилки передбачення, проблеми з БД, помилки обробки даних всередині функції), забезпечуючи стабільність роботи бота.

Функція predict_single_message повертає кортеж (is_ipso, final_reliability). Обробник handle_text_message використовує ці дані для генерації відповіді:

1) Інтерпретація is_ipso: Булеве значення is_ipso безпосередньо визначає, чи містить повідомлення ознаки ІПСО.

2) Використання final_reliability: Значення надійності джерела форматується у відсотковий вигляд.

3) Формування Тексту Відповіді: Залежно від значення is_ipso, формується один з двох варіантів повідомлення (як описано в п. 2.3), куди підставляється інформація про джерело та розрахований відсоток надійності. Використовується HTML-розмітка для виділення важливих частин. Приклад формування наведено нижче.

```
    # bot/handlers/text.py
    # ...
    reliability_percent = final_reliability * 100
    source_display = f"<code>{source_id or 'Пряме повідомлення'}</code>{forward_info}"
    if is_ipso:
        response = (f"⚠ <b>Виявлено ознаки ІПСО!</b>\n" # і т.д.
                    # ...
        )
    else:
        response = (f"✅ Аналіз завершено. Ознак ІПСО не виявлено.\n" # і т.д.
                    # ...
        )
    # ...
```

4) Надсилання Користувачеві: Сформована відповідь надсилається як відповідь на оригінальне повідомлення користувача (reply_to) з увімкненим парсингом HTML.

Обраний механізм прямого виклику функції забезпечує тісну та ефективну інтеграцію між ботом та ШІ-модулем, дозволяючи використовувати складну логіку аналізу, включаючи роботу з базою даних надійності, безпосередньо під час обробки повідомлення користувача.

3.4 Розробка механізмів обробки помилок та логування роботи бота

Для забезпечення стабільної та надійної роботи програмного бота, а також для полегшення процесу налагодження та подальшої підтримки, було реалізовано механізми обробки помилок та базового логування подій.

Обробка помилок (Винятків):

Для запобігання неочікуваним зупинкам бота та коректної реакції на позаштатні ситуації використано механізм обробки винятків Python (try...except...finally):

1) Глобальна обробка в main.py: Весь основний код в блоці if __name__ == '__main__': обгорнутий у конструкцію try...except Exception as e...finally. Це дозволяє перехоплювати будь-які не оброблені на нижчих рівнях помилки, що можуть виникнути під час ініціалізації (завантаження конфігурації, ресурсів ШІ, підключення до БД) або під час тривалого процесу полінгу. При виникненні такої помилки її детальний трейсбек виводиться в консоль за допомогою модуля traceback, що критично важливо для діагностики проблем на сервері. Блок finally гарантує виконання завершальних дій, таких як закриття з'єднання з базою даних (db_conn.close()), навіть якщо сталася помилка.

2) Локальна обробка в обробниках (handlers/text.py): Логіка обробки кожного окремого повідомлення користувача у функції handle_text_message також обгорнута у власний блок try...except. Це дозволяє ізолювати помилки, що виникають при аналізі конкретного повідомлення (наприклад, помилка під час передбачення моделлю, помилка запису/читання в SQLite через блокування або інші проблеми), від основної роботи бота. Якщо виникає помилка, бот не "падає" повністю, а лише надсилає користувачеві повідомлення про неможливість обробити саме цей запит, а деталі помилки логуються для розробника. Реалізовано перехоплення як загальних Exception, так і специфічних помилок, як sqlite3.Error, для більш точної діагностики.

Логування:

На даному етапі розробки використовується базове логування подій та помилок за допомогою функції print() для виведення інформації безпосередньо в консоль, де запускається бот. Це включає:

1) Повідомлення про основні етапи ініціалізації в main.py.

2) Інформацію про отримання нового текстового повідомлення в handlers/text.py (з chat_id, message_id).

3) Виведення тексту повідомлення (або його частини) та визначеного source_id перед передачею до ШІ-модуля.

4) Детальне логування результатів аналізу (передбачення, ймовірність, зміна надійності), яке відбувається всередині функції predict_single_message модуля ai_model.ipso_model.

5) Виведення повного трейсбеку помилок у блоках except.

Таке логування є достатнім для контролю роботи бота під час розробки, тестування та пошуку несправностей. Для розгортання в продуктивному середовищі рекомендується перехід на використання стандартного модуля logging, який надає значно більші можливості щодо керування рівнями логування, форматуванням повідомлень та спрямуванням виводу у файли чи зовнішні системи моніторингу.

4 РОЗГОРТАННЯ, ТЕСТУВАННЯ ТА ОЦІНКА РОБОТИ БОТА ДЛЯ АНАЛІЗУ ТЕКСТУ

4.1 Вибір та налаштування платформи для розгортання та тестування бота

Для забезпечення функціонування та тестування розробленого Telegram-бота необхідно було налаштувати відповідне середовище виконання. Були розглянуті різні варіанти розгортання, що впливають на доступність та стабільність роботи бота.

Були розглянуті наступні підходи:

1) Серверне розгортання (PaaS/VPS): Для забезпечення цілодобової автономної роботи бота в продуктивному середовищі оптимальним є розгортання на сервері. Були проаналізовані хмарні PaaS-платформи (Render, Heroku, PythonAnywhere) та Віртуальні Приватні Сервери (VPS). Встановлено, що для Polling-ботів, особливо тих, що використовують ресурсоємні бібліотеки як TensorFlow, безкоштовні плани PaaS часто мають суттєві обмеження (нестабільність фонових задач, ліміти ресурсів). Більш надійними є платні тарифи PaaS (типу "Background Worker", як, наприклад, на Render) або VPS з налаштуванням запуску бота як системного сервісу (наприклад, через systemd). Ці варіанти забезпечують необхідні ресурси та стабільність для постійної роботи.

2) Локальний запуск: Запуск бота на персональному комп'ютері розробника є найпростішим варіантом для розробки, налагодження, проведення функціонального та інтеграційного тестування. Він дозволяє швидко вносити зміни в код, тестувати їх та безпосередньо моніторити роботу всіх компонентів системи.

Обґрунтування вибору для даного етапу: Враховуючи основну мету даної кваліфікаційної роботи – розробку функціонального прототипу бота та демонстрацію інтеграції з ШІ-модулем – основне тестування та демонстрація працездатності системи проводилися шляхом запуску бота на локальному комп'ютері розробника. Цей підхід дозволив повною мірою перевірити всі функціональні вимоги та взаємодію компонентів без необхідності додаткових

витрат на серверні ресурси. Водночас, розгортання на сервері (VPS або платний PaaS, наприклад, Render Background Worker) розглядається як необхідний крок для подальшого розвитку проекту та переведення його у режим експлуатації з цілодобовою доступністю.

Процес підготовки локального середовища для розробки та запуску бота детально описаний у п. 3.1 даної роботи. Ключові етапи включали:

1) Створення структури проекту з окремими директоріями для коду бота (bot/) та ШІ-модуля (ai_model/).

2) Ініціалізація віртуального середовища Python (venv) з використанням інтерпретатора Python версії 3.11.

3) Встановлення всіх необхідних бібліотек (включаючи pyTelegramBotAPI, tensorflow, pandas, scikit-learn, numpy, python-dotenv, nltk) з файлу requirements.txt.

4) Розміщення файлів ШІ-модуля (код .py, ваги моделі .h5, токенізатор .json, кодувальник міток .pkl, база даних .db) у відповідних піддиректоріях папки ai_model/.

5) Налаштування файлу .env з токеном Telegram-бота та файлу .gitignore. Запуск бота для тестування здійснювався виконанням головного скрипта main.py в активованому віртуальному середовищі з кореневої директорії проекту. Таке налаштування дозволило ефективно проводити розробку, налагодження та всебічне тестування функціональності бота в контрольованому локальному середовищі.

4.2 Тестування функціональності та інтерфейсу бота

Метою функціонального тестування була перевірка відповідності реалізованого функціоналу визначеним вимогам (п. 2.1), коректності взаємодії з ШІ-модулем аналізу тексту та загальної зручності інтерфейсу користувача. Тестування проводилося вручну шляхом безпосередньої взаємодії з запущеним ботом у месенджері Telegram.

На цьому етапі перевірялася базова працездатність бота та його реакція на стандартні команди і різні типи вхідних даних.

1) Команди /start та /help: Бот успішно відповідав вітальним та довідковим повідомленнями, що відповідають оновленому дизайну (п. 2.3). Попередні кнопки ReplyKeyboardMarkup були коректно видалені після оновлення обробника команди /start. (Додаток А, рис. А.1).

2) Прямі текстові повідомлення: Бот коректно обробляв повідомлення, надіслані безпосередньо в чат, передавав їх на аналіз та повертав результат.

3) Переслані текстові повідомлення: Бот успішно обробляв переслані повідомлення, коректно ідентифікуючи source_id (ID оригінального відправника або каналу/групи) для подальшого аналізу надійності джерела.

4) Нетекстові повідомлення: При надсиланні повідомлень типу "фото", "стікер", "відео" бот коректно відповідав повідомленням про те, що він аналізує лише текстові дані.

За результатами тестування встановлено, що бот коректно обробляє команди, приймає різні типи текстових повідомлень та належним чином реагує на непідтримувані типи контенту, що відповідає функціональним вимогам ФВ.1, ФВ.2 та ФВ.5.1.

Цей етап був спрямований на перевірку коректності інтеграції з ШІ-модулем та роботи механізму оновлення надійності джерел.

1) Класифікація тексту: В ході тестування бот передавав текстові повідомлення на аналіз ШІ-модулю, який повертав булеве значення is_ipso та показник надійності final_reliability. Було перевірено класифікацію як завідомо нейтральних текстів (де is_ipso очікувано було False), так і текстів, що містять потенційні ознаки ІПСО.

2) Оновлення надійності: Механізм динамічного оновлення надійності джерел продемонстрував свою працездатність. При отриманні повідомлень, класифікованих як ІПСО (is_ipso=True), надійність відповідного джерела (source_id) коректно знижувалася на значення RELIABILITY_DECREMENT (0.02). При отриманні повідомлень, класифікованих як безпечні (is_ipso=False), надійність джерела підвищувалася на RELIABILITY_INCREMENT (0.01), залишаючись в межах визначених константами MIN_RELIABILITY та INITIAL_RELIABILITY.

3) Уникнення повторної обробки: Механізм перевірки хешу повідомлень (реалізований у ШІ-модулі) ефективно запобігав повторному оновленню надійності для ідентичних повідомлень, що надходили з одного й того ж джерела, що було підтверджено аналізом логів роботи бота.

4) Якість класифікації ШІ-моделі: Було відзначено, що поточна версія ШІ-моделі не завжди точно класифікує деякі типи текстів (наприклад, повідомлення з ненормативною лексикою могли бути визначені як безпечні). Це питання відноситься до якості навчання самої моделі та потребує подальшого доопрацювання в рамках відповідного дослідження. Однак, сам механізм інтеграції та передачі даних між ботом та ШІ-модулем функціонував коректно.

Результати тестування підтвердили, що взаємодія бота з ШІ-модулем реалізована правильно, і логіка оновлення надійності джерел працює відповідно до закладених алгоритмів.

4.3 Оцінка продуктивності та стабільності роботи бота

Для оцінки швидкодії бота було проведено вимірювання часу від моменту надсилання текстового повідомлення до моменту отримання відповіді. Тестування проводилося на локальному комп'ютері розробника (Процесор: AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz, Оперативна пам'ять: 16,0 ГБ) в умовах типового навантаження.

1) Результати: Для переважної більшості тестових повідомлень час відповіді бота становив менше 1 секунди. Цей час включає всі етапи: отримання повідомлення, його обробку, виклик функції ШІ-модуля (очистка тексту, токенізація, паддінг, передбачення LSTM-моделлю, взаємодія з БД SQLite) та надсилання відповіді користувачеві.

2) Відповідність вимогам: Отриманий час відповіді повністю задовольняє нефункціональну вимогу НФВ.1.1 (час відповіді не більше 10 секунд) та забезпечує високий рівень комфорту при взаємодії з ботом.

Стабільність роботи бота оцінювалася під час тривалих тестових сесій на локальному комп'ютері загальною тривалістю близько 20 годин протягом 5 днів активної розробки та тестування.

1) Результати: За час тестування критичних збоїв, що призводили б до повної зупинки процесу бота, не спостерігалося. Реалізовані механізми обробки винятків дозволяли коректно обробляти потенційні помилки (наприклад, мережеві помилки Telegram API) без переривання основної роботи. Взаємодія з базою даних SQLite, навіть в умовах багатопоточного виконання обробників pyTelegramBotAPI (з використанням єдиного об'єкта з'єднання та check_same_thread=False), не призводила до помилок блокування.

2) Висновок щодо стабільності: В умовах локального тестування розроблений бот продемонстрував високу стабільність. Для оцінки поведінки під довготривалим навантаженням від багатьох користувачів необхідне розгортання у серверному середовищі.

4.4 Характеристика зручності використання та якості користувацького досвіду при взаємодії з ботом

Оцінка зручності використання та загального користувацького досвіду є важливим етапом аналізу розробленого програмного продукту. Для Telegram-бота, призначеного для широкого кола користувачів, ці аспекти мають ключове значення. Оцінка проводилася на основі власного досвіду взаємодії з ботом під час тестування.

1) Простота та Інтуїтивність Інтерфейсу: Основна взаємодія з ботом реалізована за принципом "надіслав текст – отримав результат". Такий підхід є максимально простим та інтуїтивно зрозумілим для будь-якого користувача Telegram, оскільки не вимагає вивчення спеціальних команд чи навігації по складних меню. Після оновлення логіки та видалення ReplyKeyboardMarkup кнопок для вибору режиму аналізу, інтерфейс став ще чистішим, оскільки бот автоматично обробляє будь-яке отримане текстове повідомлення. Стандартні

команди /start та /help надають всю необхідну початкову та довідкову інформацію, що відповідає усталеній практиці для Telegram-ботів.

2) Зрозумілість та Інформативність Повідомлень Бота: Тексти повідомлень, що надсилаються ботом, були розроблені з урахуванням їх інформативності та легкості сприйняття. Вітальне повідомлення (/start) чітко окреслює основну функцію бота та спосіб взаємодії. Довідкове повідомлення (/help) надає розгорнуте пояснення щодо процесу аналізу, концепції "надійності джерела" та поточних обмежень (аналіз лише тексту, рекомендаційний характер висновків). Повідомлення з результатами аналізу структуровані для швидкого розуміння: використовуються емодзі (✅, ⚠) для візуального індикатора, чітко вказується вердикт щодо наявності ознак ІПСО, а також надається інформація про джерело та його поточний показник надійності. Використання HTML-форматування для виділення ключових моментів (наприклад, жирним шрифтом) сприяє кращій читабельності. Повідомлення про обробку непідтримуваних типів даних та про можливі помилки аналізу є короткими та зрозумілими.

3) Зворотний Зв'язок та Швидкість Реакції: Як було зазначено в п. 4.3, бот демонструє високу швидкість реакції на повідомлення користувача (менше 1 секунди на тестовому локальному середовищі). Це забезпечує комфортний користувацький досвід, оскільки користувач отримує результат аналізу практично миттєво, що важливо для інтерактивного інструменту.

4) Загальні Враження та Потенційні Покращення: Загальний досвід використання бота оцінюється як позитивний. Простота взаємодії, швидкість відповіді та чіткість представлення результатів роблять його зручним інструментом для швидкої перевірки текстових повідомлень. Серед потенційних напрямків для покращення користувацького досвіду в майбутньому можна розглянути:

а) Впровадження механізму зворотного зв'язку від користувача щодо точності аналізу (наприклад, кнопки "Правильно" / "Помилка"), що могло б сприяти збору даних для подальшого вдосконалення ШІ-моделі.

б) Надання користувачеві можливості переглядати історію аналізу або статистику по певних джерелах (якщо це буде визнано доцільним з точки зору приватності та функціоналу).

в) Додавання можливості налаштування чутливості (порогу класифікації) для досвідчених користувачів, хоча це може ускладнити інтерфейс для пересічного користувача.

Висновок щодо зручності використання: Розроблений Telegram-бот має простий, інтуїтивно зрозумілий та ефективний інтерфейс користувача, що відповідає основним принципам хорошого UX. Він не вимагає спеціальних навичок для використання та надає результати аналізу у доступній формі.

ВИСНОВКИ

У даній кваліфікаційній роботі було вирішено актуальну науково-практичну задачу розробки програмного Telegram-бота для аналізу безпеки текстових даних на основі нейронних мереж, з теоретичним розглядом можливостей аналізу графічних даних. Метою роботи було створення функціонального бота, здатного інтегрувати ШІ-моделі для виявлення потенційних загроз у текстовому контенті, зокрема ознак інформаційно-психологічних операцій (ІПСО) в контексті протидії російській дезінформації.

За результатами виконаної роботи можна зробити наступні висновки:

1) Проаналізовано актуальність проблеми та теоретичні основи. Встановлено, що в умовах стрімкого зростання обсягів цифрової інформації та активної інформаційної війни, особливо в популярних месенджерах як Telegram, автоматизований аналіз текстового контенту на предмет дезінформації, ІПСО та інших загроз є надзвичайно важливим завданням. Досліджено принципи роботи Telegram Bot API, що є основою для створення інтерактивних ботів, та розглянуто два основні методи отримання оновлень – Polling та Webhooks. Проведено оглядовий аналіз нейромережевих підходів до обробки природної мови (NLP), зокрема архітектур на базі LSTM та Трансформерів, а також методів комп'ютерного зору (CV) для аналізу зображень, що підкреслило потенціал глибокого навчання для вирішення поставлених завдань. На основі аналізу було обґрунтовано вибір стеку технологій для розробки: мова Python, бібліотека pyTelegramBotAPI, фреймворк TensorFlow/Keras для інтеграції ШІ.

2) Спроектовано архітектуру та інтерфейси програмного бота. Сформульовано функціональні та нефункціональні вимоги до системи, орієнтовані на аналіз текстових повідомлень та врахування надійності джерела. Розроблено модульну архітектуру програмного бота, що включає компоненти для конфігурації, обробки команд та повідомлень Telegram, а також взаємодії з зовнішнім ШІ-модулем аналізу тексту. Спроектовано користувацький інтерфейс, що забезпечує просту та інтуїтивну взаємодію через стандартні команди (/start, /help) та

автоматичну обробку надісланого тексту. Визначено програмний інтерфейс (у вигляді Python-функції predict_single_message) для взаємодії бота з ШІ-модулем, що інкапсулює логіку аналізу та роботу з базою даних надійності джерел.

3) Здійснено розробку та інтеграцію компонентів бота. Налаштовано середовище розробки, включаючи віртуальне середовище Python, встановлення необхідних залежностей та систему контролю версій Git. Реалізовано програмні модулі бота: модуль конфігурації (config.py), обробники команд (handlers/common.py) та текстових повідомлень (handlers/text.py), а також головний скрипт ініціалізації та запуску (main.py). Ключовим етапом стала успішна інтеграція розробленого бота з ШІ-модулем аналізу тексту на основі LSTM-мережі (розробленим в рамках паралельного дослідження), що включає виклик функції передбачення, передачу тексту та ідентифікатора джерела, отримання результату класифікації (ІПСО/не ІПСО) та оновленого показника надійності джерела. Реалізовано базові механізми обробки помилок та логування.

4) Проведено тестування та оцінку роботи бота. Тестування проводилося шляхом запуску бота на локальному комп'ютері розробника. Функціональне тестування підтвердило коректну обробку команд, прийом прямих та пересланих текстових повідомлень, правильну ідентифікацію джерела та адекватну реакцію на непідтримувані типи контенту. Тестування взаємодії з ШІ-модулем продемонструвало коректну передачу даних, отримання результатів та функціонування механізму динамічного оновлення надійності джерел (зменшення при ІПСО, збільшення при безпечному контенті, уникнення повторної обробки). Оцінка продуктивності показала високу швидкодію (час відповіді менше 1 секунди на тестовому обладнанні). Бот продемонстрував стабільну роботу під час тестових сесій. Аналіз зручності використання підтвердив простоту та інтуїтивність інтерфейсу.

Таким чином, мета кваліфікаційної роботи – розробка програмного Telegram-бота для аналізу безпеки текстових даних, що інтегрується з нейромережевою моделлю – була досягнута. Всі поставлені завдання виконані.

Наукова новизна отриманих результатів полягає у запропонованій архітектурі програмного Telegram-бота, орієнтованій на інтеграцію з зовнішнім ШІ-модулем аналізу тексту через прямий виклик функції, та програмній реалізації системи, що забезпечує взаємодію користувача з комплексним аналізом текстових даних, враховуючи динамічний показник надійності джерела.

Практичне значення одержаних результатів полягає у створенні готового до використання інструменту, який може бути застосований для оперативної перевірки користувачами підозрілих текстових повідомлень у Telegram, а також слугувати основою для більш комплексних систем модерації контенту та інструментом для протидії інформаційним загрозам, зокрема російській дезінформації та ІПСО.

Обмеженнями даної роботи є тестування бота переважно в локальному середовищі, а також залежність якості фінального аналізу від точності та повноти навчання зовнішньої ШІ-моделі. Практична реалізація обмежується аналізом лише текстових даних.

Напрямками подальшого розвитку проекту є: розгортання бота на сервері (VPS або платний PaaS) для забезпечення цілодобової доступності; подальше вдосконалення ШІ-моделі аналізу тексту (спільно з розробником моделі); реалізація та інтеграція модуля для аналізу графічних даних (на основі теоретичного огляду, проведеного в Розділі 1); розширення функціоналу бота (наприклад, додавання можливості для користувачів надавати зворотний зв'язок, налаштування порогу чутливості).

ПЕРЕЛІК ПОСИЛАНЬ

1. 2024 Q1 Unified Top 5 Communication Apps Units UA [Електронний ресурс] / Sensor Tower Blog. – 2024. – Режим доступу: https://sensortower.com/blog/2024-q1-unified-top-5-communication%20apps-units-ua-6070aae1241bc16eb81f5bab (дата звернення: 08.05.2025).
2. Denisova K. Telegram in Ukraine: Over 50% of Ukrainians oppose ban but support restrictions, poll shows [Електронний ресурс] / The Kyiv Independent. – Режим доступу: https://kyivindependent.com/telegram-in-ukraine/ (дата звернення: 08.05.2025).
3. Telegram has become the main source of information for Ukrainians: what about disinformation? [Електронний ресурс] / Інформаційне агентство "Українські Національні Новини" (УНН). – Режим доступу: https://unn.ua/en/news/telegram-has-become-the-main-source-of-information-for-ukrainians-what-about-disinformation (дата звернення: 08.05.2025).
4. Шлінчак В. Росія поширює дезінформацію через Telegram і деякі західні ЗМІ [Електронний ресурс] / Фонд Арсенія Яценюка "Відкрий Україну". – Режим доступу: https://ksf.openukraine.org/en/categories/news/rosiia-poshyriuie-dezinformatsiiu-cherez-telegram-i-deiaki-zakhidni-zmi-viktor-shlinchak (дата звернення: 08.05.2025).
5. Is Telegram a threat to Ukraine's national security? [Електронний ресурс] / Deutsche Welle (DW). – Режим доступу: https://www.dw.com/en/is-telegram-a-threat-to-ukraines-national-security/a-68732966 (дата звернення: 08.05.2025).
6. Telegram poses national security threat to Ukraine — Budanov [Електронний ресурс] / The New Voice of Ukraine (NV.ua). – Режим доступу: https://english.nv.ua/nation/telegram-poses-national-security-threat-to-ukraine-budanov-50449065.html (дата звернення: 08.05.2025).
7. Контент Telegram регулюватиметься за новими правилами ЄС [Електронний ресурс] / Телеканал "24 Канал". – Режим доступу: https://24tv.ua/tech/kontent-

telegram-regulyuvatimetsya-za-novimi-pravilami-yes_n2550093 (дата звернення: 08.05.2025).

8. Telegram API vs Telegram Bot API [Електронний ресурс] / StackShare. – Режим доступу: https://stackshare.io/stackups/telegram-api-vs-telegram-bot-api (дата звернення: 08.05.2025).
9. pyTelegramBotAPI [Електронний ресурс] / Python Package Index (PyPI). – Режим доступу: https://pypi.org/project/pyTelegramBotAPI/ (дата звернення: 08.05.2025).
10. Rasa Open Source Documentation [Електронний ресурс] / Rasa Technologies GmbH. – Режим доступу: https://rasa.com/docs/rasa/ (дата звернення: 08.05.2025).
11. python-telegram-bot: Alternatives [Електронний ресурс] / StackShare. – Режим доступу: https://stackshare.io/pypi-python-telegram-bot/alternatives (дата звернення: 08.05.2025).
12. Telegram Bot Examples for Marketing and Automation [Електронний ресурс] / SendPulse Blog. – Режим доступу: https://sendpulse.com/blog/telegram-bot-examples (дата звернення: 08.05.2025).
13. Content Moderation Tools for Social Media in 2024 [Електронний ресурс] / Influencer Marketing Hub. – Режим доступу: https://influencermarketinghub.com/content-moderation-tools/ (дата звернення: 08.05.2025).
14. Community Moderation: Best Practices & Essential Tools [Електронний ресурс] / Khoros Blog. – Режим доступу: https://khoros.com/blog/community-moderation (дата звернення: 08.05.2025).
15. Bots API [Електронний ресурс] / Telegram Core Documentation. – Режим доступу: https://core.telegram.org/bots/api (дата звернення: 08.05.2025).
16. Esubalew A. Mastering Telegram Bot Hosting: Webhook vs Long Polling Explained [Електронний ресурс] / Code Esubalew Blog. – Режим доступу: https://code.esubalew.et/mastering-telegram-bot-hosting-webhook-vs-long-polling-explained (дата звернення: 08.05.2025).

17. How To Use Systemctl to Manage Systemd Services and Units [Електронний ресурс] / DigitalOcean Community. – Режим доступу: https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units (дата звернення: 08.05.2025).
18. Liu W. et al. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI [Електронний ресурс] / Frontiers in Artificial Intelligence. – 2020. – Vol. 3, Art. No. 4. – DOI: 10.3389/frai.2020.00004. – Режим доступу: https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2020.00004/full (дата звернення: 08.05.2025).
19. Introduction to Recurrent Neural Network (RNN) [Електронний ресурс] / GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ (дата звернення: 08.05.2025).
20. Recurrent Neural Networks and LSTM [Електронний ресурс] / Built In. – Режим доступу: https://builtin.com/data-science/recurrent-neural-networks-and-lstm (дата звернення: 08.05.2025).
21. Vaswani A. et al. Attention Is All You Need [Електронний ресурс] / arXiv. – 2017. – arXiv:1706.03762. – Режим доступу: https://arxiv.org/abs/1706.03762 (дата звернення: 08.05.2025).
22. Sharma A. Building Text Classification Models Using BERT [Електронний ресурс] / DataDrivenScience. – Режим доступу: https://datadrivenscience.com/building-text-classification-models-using-bert/ (дата звернення: 08.05.2025).
23. Transformers Documentation [Електронний ресурс] / Hugging Face. – Режим доступу: https://huggingface.co/docs/transformers/index (дата звернення: 08.05.2025).
24. He K. et al. Deep Residual Learning for Image Recognition [Електронний ресурс] / arXiv. – 2015. – arXiv:1512.03385. – Режим доступу: https://arxiv.org/abs/1512.03385 (дата звернення: 08.05.2025).

25. Szegedy C. et al. Rethinking the Inception Architecture for Computer Vision [Електронний ресурс] / arXiv. – 2015. – arXiv:1512.00567. – Режим доступу: https://arxiv.org/abs/1512.00567 (дата звернення: 08.05.2025).
26. What is Transfer Learning? [Електронний ресурс] / IBM Think. – Режим доступу: https://www.ibm.com/think/topics/transfer-learning (дата звернення: 08.05.2025).
27. LAION-AI/CLIP-based-NSFW-Detector [Електронний ресурс] : програмний репозиторій / GitHub. – Режим доступу: https://github.com/LAION-AI/CLIP-based-NSFW-Detector (дата звернення: 08.05.2025).
28. What is Computer Vision? [Електронний ресурс] / Microsoft Azure Documentation. – Режим доступу: https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/ (дата звернення: 08.05.2025).
29. The Python Tutorial [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/3/tutorial/index.html (дата звернення: 08.05.2025).
30. Sheremetov D. Why Python is the Best Choice for AI, ML, and Deep Learning [Електронний ресурс] / Onix Systems Blog. – Режим доступу: https://onix-systems.com/blog/python-is-best-for-ai-ml-and-deep-learning (дата звернення: 08.05.2025).
31. Visual Studio Code Docs [Електронний ресурс] / Microsoft. – Режим доступу: https://code.visualstudio.com/docs (дата звернення: 08.05.2025).
32. What is version control? [Електронний ресурс] / GitLab. – Режим доступу: https://about.gitlab.com/topics/version-control/what-is-git/ (дата звернення: 08.05.2025).
33. TensorFlow Core: Quickstart for beginners [Електронний ресурс] / TensorFlow. – Режим доступу: https://www.tensorflow.org/tutorials/quickstart/beginner (дата звернення: 08.05.2025).

Додаток А. Приклади роботи програмного бота

Рисунок А.1 – Відповідь бота на команду /start

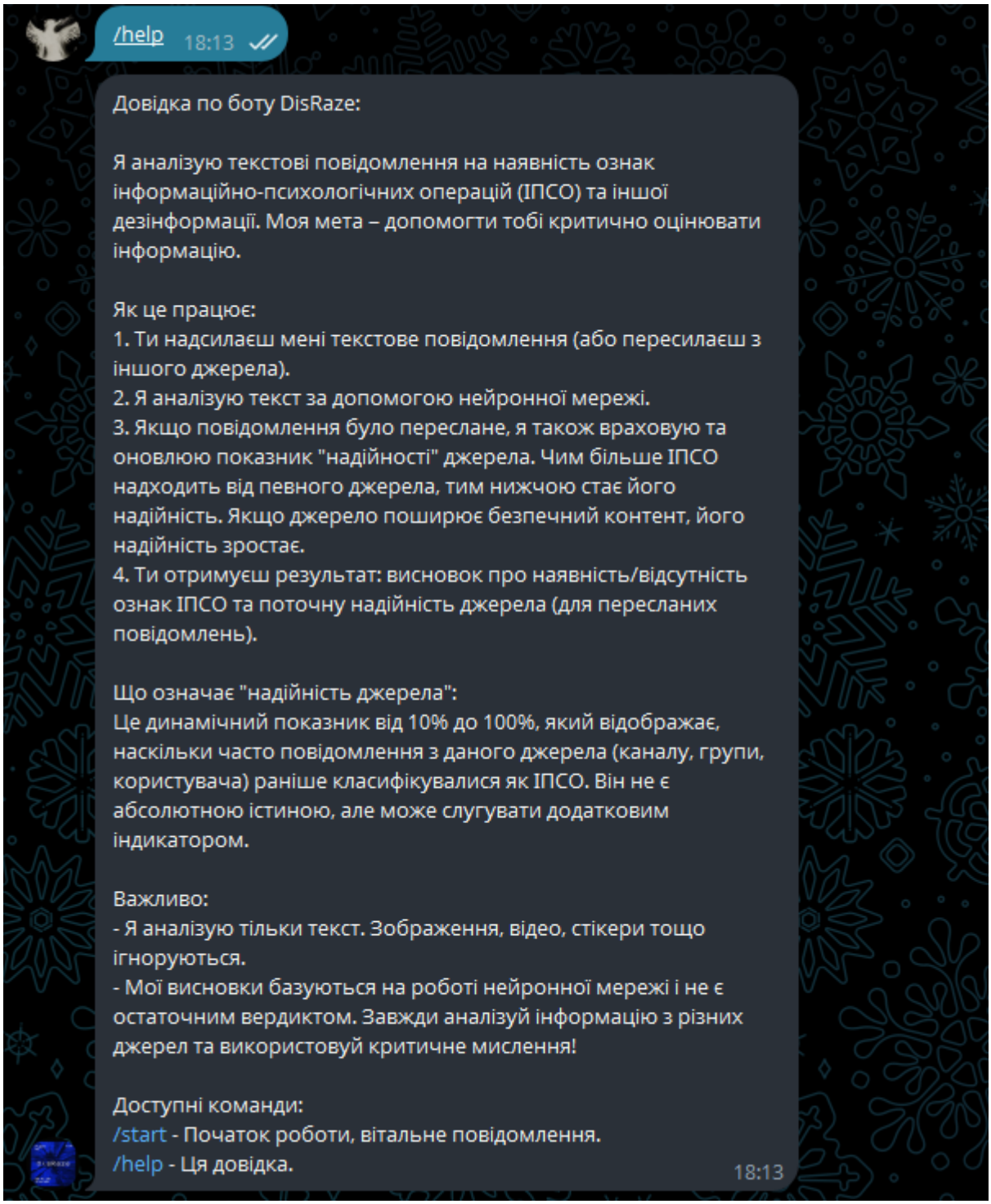

Рисунок А.2 – Відповідь бота на команду /help

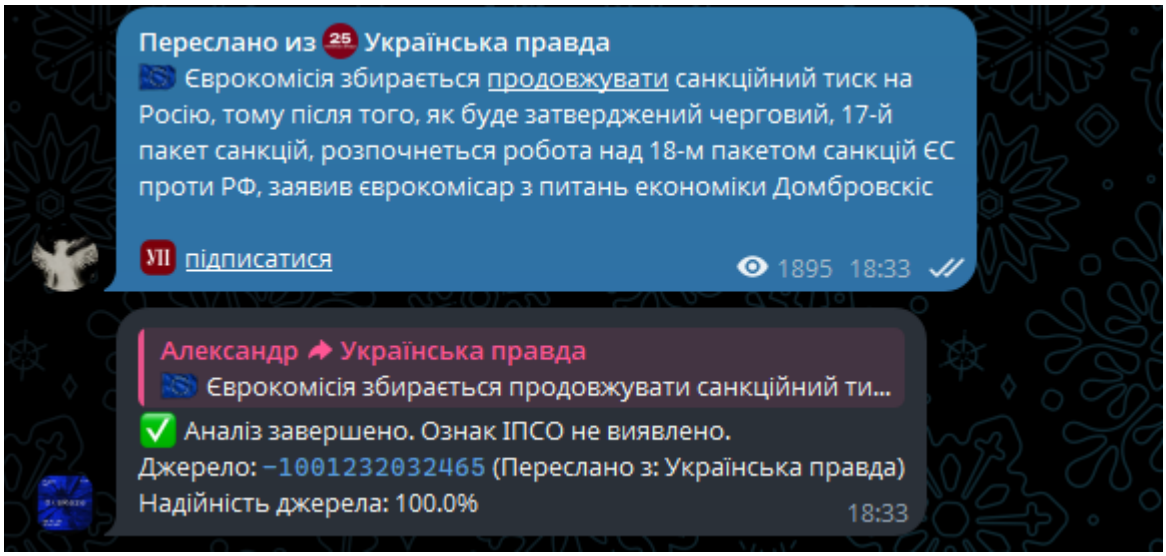

Рисунок А.3 – Приклад аналізу безпечного текстового повідомлення

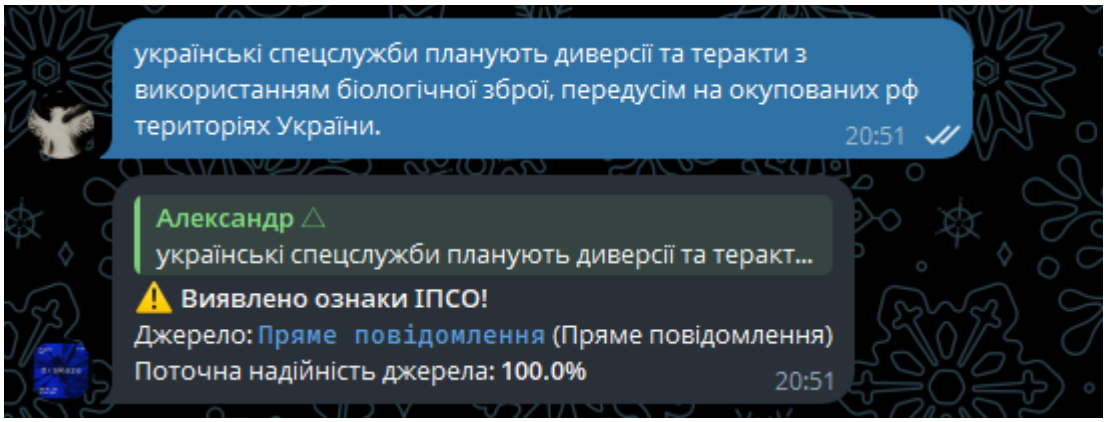

Рисунок А.4 – Приклад аналізу текстового повідомлення з ознаками ІПСО

Рисунок А.5 – Демонстрація зміни надійності джерела при послідовному аналізі

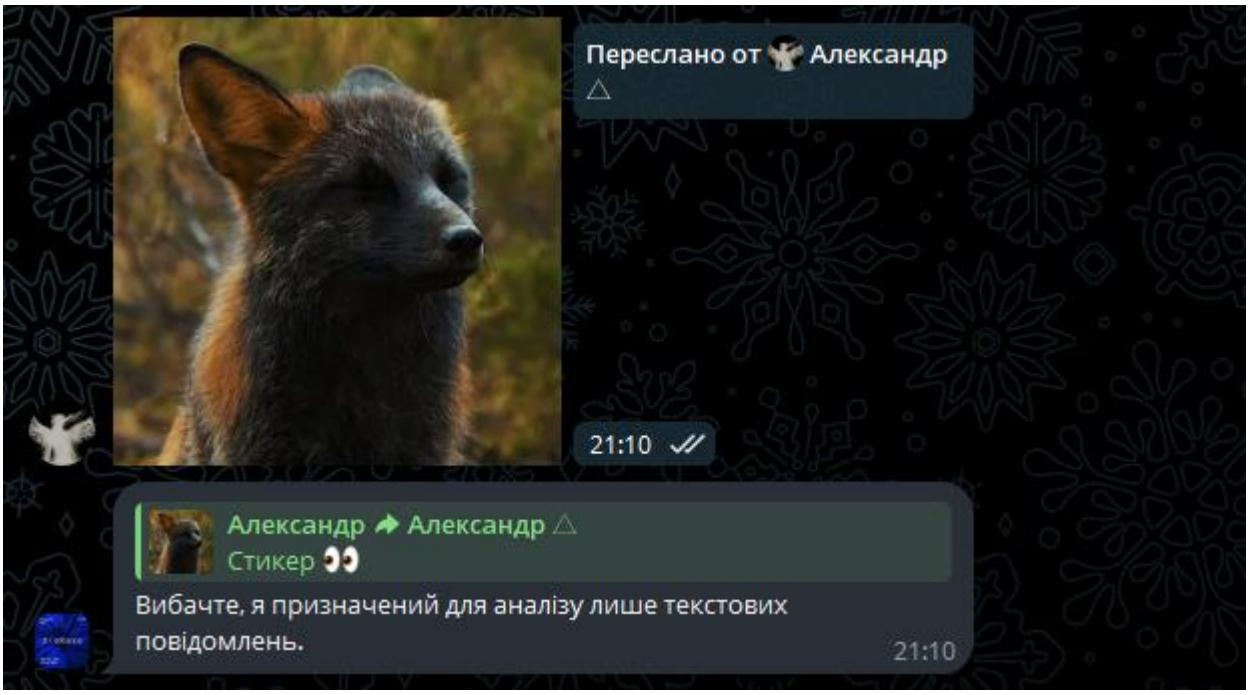

Рисунок А.6 – Реакція бота на нетекстове повідомлення (стікер)

Додаток Б. Лістинги ключових фрагментів програмного коду

Лістинг основного файлу запуску та ініціалізації (main.py)

```
import telebot
import os
import sys
import sqlite3 # Потрібен для визначення типу db_conn та обробки помилок
from functools import partial

try:
    from bot.config import (
        BOT_TOKEN, DB_PATH, MODEL_WEIGHTS_PATH, TOKENIZER_PATH,
        LABEL_ENCODER_PATH, MAX_LENGTH
    )
    print("Конфігурацію завантажено.")
except ImportError as e:
     print(f"КРИТИЧНА ПОМИЛКА: Помилка імпорту конфігурації з bot.config: {e}")
     sys.exit(1)

try:
    from bot.handlers.common import register_common_handlers
    from bot.handlers.text import register_text_handlers
    print("Обробники імпортовано.")
except ImportError as e:
     print(f"КРИТИЧНА ПОМИЛКА: Помилка імпорту обробників з bot.handlers: {e}")
     sys.exit(1)

try:
    from ai_model.ipso_model import load_ai_resources
    print("Функцію завантаження ШІ 'load_ai_resources' імпортовано успішно.")
except ImportError as e:
    print(f"КРИТИЧНА ПОМИЛКА: Помилка імпорту 'load_ai_resources' з ai_model.ipso_model: {e}")
    print("Перевірте наявність файлу __init__.py в папці ai_model та правильність імені файлу/функції.")
    sys.exit(1)
except Exception as e:
    print(f"КРИТИЧНА ПОМИЛКА: Інша помилка під час імпорту ШІ модуля: {e}")
    sys.exit(1)

# --- Основний блок запуску ---
if __name__ == '__main__':
    print("Запуск main.py...")
    model = None
    tokenizer = None
    label_encoder = None
```

```
    db_conn = None # Важливо ініціалізувати як None для блоку finally
    optimal_threshold = 0.5

    try:
        # 1. Завантаження ВСІХ ресурсів ШІ + ініціалізація БД через
єдину функцію
        print("Завантаження ресурсів ШІ та підключення до БД...")
        model, tokenizer, label_encoder, db_conn = load_ai_resources(
            model_weights_path=MODEL_WEIGHTS_PATH,
            tokenizer_path=TOKENIZER_PATH,
            encoder_path=LABEL_ENCODER_PATH,
            db_path=DB_PATH
        )

        # Перевірка, чи все завантажено
        if not all([model, tokenizer, label_encoder, db_conn]):
             raise RuntimeError("Не вдалося завантажити всі компоненти
ШІ або підключитися до БД.")
        print("Ресурси ШІ та з'єднання з БД готові.")

        # 2. Встановлення порогу
        print(f"Встановлено поріг класифікації: {optimal_threshold}")

        # 3. Ініціалізація бота
        print("Ініціалізація екземпляру бота...")
        bot = telebot.TeleBot(BOT_TOKEN, parse_mode='HTML')

        # 4. Реєстрація обробників
        print("Реєстрація обробників...")
        register_common_handlers(bot)
        register_text_handlers(
            bot_instance=bot,
            model=model,
            tokenizer=tokenizer,
            label_encoder=label_encoder,
            db_conn=db_conn, # Передаємо об'єкт з'єднання
            max_length=MAX_LENGTH, # MAX_LENGTH з config.py
            threshold=optimal_threshold
        )
        print("Обробники зареєстровано.")

        # 5. Запуск бота
        print("-" * 30)
        print("Запуск бота (полінг)... Натисніть Ctrl+C для зупинки.")
        print("-" * 30)
        bot.polling(none_stop=True, interval=0, timeout=30)

    except FileNotFoundError as e:
        print(f"КРИТИЧНА ПОМИЛКА ЗАПУСКУ: Не знайдено необхідний файл
ресурсів ШІ або БД. {e}")
    except ConnectionError as e:
        print(f"КРИТИЧНА ПОМИЛКА ЗАПУСКУ: Проблема з базою даних.
{e}")
```

```
    except ImportError as e:
         print(f"КРИТИЧНА ПОМИЛКА ЗАПУСКУ: Не вдалося імпортувати
необхідний модуль. {e}")
    except RuntimeError as e:
         print(f"КРИТИЧНА ПОМИЛКА ЗАПУСКУ: Проблема ініціалізації
ресурсів. {e}")
    except Exception as e:
        print(f"КРИТИЧНА ПОМИЛКА під час виконання main.py: {e}")
        import traceback
        traceback.print_exc() # Виводимо детальний трейсбек
    finally:
        if db_conn:
            try:
                db_conn.close()
                print("З'єднання з БД закрито.")
            except Exception as e_close:
                 print(f"Помилка закриття БД: {e_close}")
        print("Роботу бота завершено (або сталася критична помилка).")
```

Лістинг модуля конфігурації (bot/config.py)

```
import os
from dotenv import load_dotenv

# Визначаємо шлях до папки проекту
PROJECT_ROOT                                                    =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
dotenv_path = os.path.join(PROJECT_ROOT, '.env')
load_dotenv(dotenv_path)

# --- Основні Налаштування Бота ---
BOT_TOKEN = os.getenv('BOT_TOKEN')
if not BOT_TOKEN:
    print("КРИТИЧНА ПОМИЛКА: Токен бота ('BOT_TOKEN') не знайдено!
Перевір .env або змінні середовища.")
    exit(1) # Вихід з помилкою
print("Токен бота завантажено.")

# --- Шляхи до Ресурсів ШІ ---
AI_MODEL_DIR = os.path.join(PROJECT_ROOT, 'ai_model')
DB_PATH = os.path.join(AI_MODEL_DIR, 'ipso_database_binary.db')
MODEL_WEIGHTS_DIR            =            os.path.join(AI_MODEL_DIR,
'model_weights_binary')
MODEL_WEIGHTS_PATH          =          os.path.join(MODEL_WEIGHTS_DIR,
'ipso_rnn_binary_lstm_30k.weights.h5')
TOKENIZER_PATH              =              os.path.join(AI_MODEL_DIR,
'tokenizer_binary_30k.json')
LABEL_ENCODER_PATH           =           os.path.join(AI_MODEL_DIR,
'label_encoder_binary.pkl')

# --- Параметри Моделі ---
MAX_LENGTH = 200
```

Лістинг обробника текстових повідомлень (bot/handlers/text.py)

```
import telebot
from telebot import types
from functools import partial
import sqlite3
import os

# Імпортуємо функцію аналізу з модуля ШІ
try:
    from ai_model.ipso_model import predict_single_message
except ImportError:
    print("ПОМИЛКА: Не вдалося імпортувати predict_single_message.")
    def predict_single_message(*args, **kwargs):
        print("УВАГА: Використовується заглушка predict_single_message!")
        return False, 1.0

def handle_text_message(message: types.Message, bot_instance: telebot.TeleBot, model, tokenizer, label_encoder, db_conn: sqlite3.Connection, max_length, threshold):
    chat_id = message.chat.id
    text_to_analyze = message.text
    source_id = None
    forward_info = ""

    if message.forward_from_chat:
        source_id = str(message.forward_from_chat.id)
        forward_info = f" (Переслано з: {message.forward_from_chat.title or source_id})"
    elif message.forward_from:
        source_id = str(message.forward_from.id)
        forward_info = f" (Переслано від: {message.forward_from.first_name or source_id})"
    else:
        forward_info = " (Пряме повідомлення)"

    response = "На жаль, сталася помилка під час аналізу. Спробуйте пізніше."

    try:
        if not all([model, tokenizer, label_encoder]):
            raise ValueError("Ресурси ШІ не були належним чином ініціалізовані.")
        if db_conn is None:
            raise ConnectionError("Відсутнє з'єднання з базою даних.")

        is_ipso, final_reliability = predict_single_message(
            model=model, tokenizer=tokenizer, label_encoder=label_encoder,
            db_conn=db_conn, text=text_to_analyze, source_id=source_id,
            max_length=max_length, threshold=threshold
```

```
        )

        reliability_percent = final_reliability * 100
        source_display     =     f"<code>{source_id     or     'Пряме
повідомлення'}</code>{forward_info}"
        if is_ipso:
            response = (f"⚠ <b>Виявлено ознаки ІПСО!</b>\n"
                        f"Джерело: {source_display}\n"
                        f"Поточна         надійність         джерела:
<b>{reliability_percent:.1f}%</b>")
        else:
            response  =  (f"✅  Аналіз  завершено.  Ознак  ІПСО  не
виявлено.\n"
                        f"Джерело: {source_display}\n"
                        f"Надійність                              джерела:
{reliability_percent:.1f}%")
    except Exception as e:
        print(f"Помилка аналізу для chat_id={chat_id}: {e}")
        # import traceback # Розкоментуй для детального дебагу
        # traceback.print_exc()

    try:
        bot_instance.reply_to(message, response, parse_mode='HTML')
    except Exception as e_send:
        print(f"Помилка надсилання HTML відповіді: {e_send}")
        try:
            response_plain                                          =
response.replace('<b>','').replace('</b>','').replace('<code>','').r
eplace('</code>','')
            bot_instance.reply_to(message, response_plain)
        except Exception as e_send_plain:
            print(f"Повторна  помилка  надсилання  plain  відповіді:
{e_send_plain}")

def  handle_other_messages(message:  types.Message,  bot_instance:
telebot.TeleBot):
    """Інформує користувача, що бот обробляє лише текст."""
    try:
        bot_instance.reply_to(message,  "Вибачте,  я  призначений  для
аналізу лише текстових повідомлень.")
    except Exception as e:
        print(f"Помилка відповіді на нетекстове повідомлення: {e}")

def  register_text_handlers(bot_instance:  telebot.TeleBot,  model,
tokenizer,  label_encoder,  db_conn:  sqlite3.Connection,  max_length,
threshold):
    """Реєструє обробники для текстових та інших типів повідомлень."""
    text_processor = partial(handle_text_message,
                             bot_instance=bot_instance,  model=model,
tokenizer=tokenizer,
                             label_encoder=label_encoder,
db_conn=db_conn,
```

```
                                max_length=max_length,
threshold=threshold)
    bot_instance.register_message_handler(text_processor,
content_types=['text'], pass_bot=False)

    other_processor         =         partial(handle_other_messages,
bot_instance=bot_instance)
    bot_instance.register_message_handler(other_processor,
                                          content_types=['audio',
'photo', 'voice', 'video', 'document',
                                                         'location',
'contact', 'sticker', 'video_note',
                                                         'animation',
'game', 'poll', 'dice'], pass_bot=False)
```

Лістинг ключових функцій ШІ-модуля (ai_model/ipso_model.py)

```
import os
import sqlite3
import hashlib
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input,    Embedding,    LSTM,    Dense,    Dropout,    Concatenate,
BatchNormalization
)
from tensorflow.keras.preprocessing.text import tokenizer_from_json
from tensorflow.keras.preprocessing.sequence import pad_sequences
import re
import pickle

# --- Конфігурація шляхів та параметрів (використовуються функціями
нижче) ---
try:
    BASE_AI_DIR = os.path.dirname(os.path.abspath(__file__))
except NameError:
     BASE_AI_DIR = os.getcwd()

DB_FILE_DEFAULT              =              os.path.join(BASE_AI_DIR,
'ipso_database_binary.db')
MODEL_WEIGHTS_FILE_DEFAULT        =        os.path.join(BASE_AI_DIR,
'model_weights_binary', 'ipso_rnn_binary_lstm_30k.weights.h5')
TOKENIZER_FILE_DEFAULT          =          os.path.join(BASE_AI_DIR,
'tokenizer_binary_30k.json')
LABEL_ENCODER_FILE_DEFAULT        =        os.path.join(BASE_AI_DIR,
'label_encoder_binary.pkl')

INITIAL_RELIABILITY = 1.0
RELIABILITY_DECREMENT = 0.02
RELIABILITY_INCREMENT = 0.01
MIN_RELIABILITY = 0.1
```

```
VOCAB_SIZE = 30000
MAX_LENGTH = 200
EMBEDDING_DIM = 128
LSTM_UNITS = 64
DROPOUT_RATE = 0.4

SAFE_CONTENT_LABEL = "SAFE_CONTENT"
IPSO_LABEL = "IPSO"
BINARY_LABELS = [SAFE_CONTENT_LABEL, IPSO_LABEL]
# NUM_CLASSES = len(BINARY_LABELS) # Визначається в load_ai_resources

# --- Функції для роботи з БД---
def init_db(db_path=DB_FILE_DEFAULT):
    """Ініціалізує БД, створює таблиці sources та
processed_messages."""
    db_dir = os.path.dirname(db_path)
    if db_dir and not os.path.exists(db_dir):
        try:
            os.makedirs(db_dir)
        except OSError as e:
            print(f"[DB Init] Помилка створення директорії {db_dir}:
{e}")
            return None # Не можемо створити БД без папки
    conn = None
    try:
        conn = sqlite3.connect(db_path, check_same_thread=False,
timeout=10)
        cursor = conn.cursor()
        cursor.execute('CREATE TABLE IF NOT EXISTS sources (source_id
TEXT PRIMARY KEY, reliability REAL NOT NULL, last_updated TIMESTAMP
DEFAULT CURRENT_TIMESTAMP)')
        cursor.execute('CREATE INDEX IF NOT EXISTS idx_source_id ON
sources(source_id)')
        cursor.execute('CREATE TABLE IF NOT EXISTS processed_messages
(message_hash TEXT PRIMARY KEY, processed_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP)')
        conn.commit()
        print(f"[DB Init] БД '{db_path}' ініціалізована.")
        return conn
    except sqlite3.Error as e:
        print(f"[DB Init] Помилка ініціалізації БД: {e}")
        if conn: conn.close()
        return None

def get_source_reliability(conn, source_id):
    if source_id is None: return INITIAL_RELIABILITY
    if conn is None: print("[DB GetRel] Помилка: Немає з'єднання з
БД."); return INITIAL_RELIABILITY
    try:
        cursor = conn.cursor()
        cursor.execute('SELECT reliability FROM sources WHERE
source_id = ?', (str(source_id),))
```

```
        result = cursor.fetchone()
        if result: return result[0]
        else:
            cursor.execute('INSERT OR IGNORE INTO sources (source_id,
reliability) VALUES (?, ?)', (str(source_id), INITIAL_RELIABILITY))
            conn.commit()
            print(f"[DB GetRel] Додано/знайдено джерело {source_id} з
надійністю {INITIAL_RELIABILITY:.3f}")
            return INITIAL_RELIABILITY
    except sqlite3.Error as e:
        print(f"[DB GetRel] Помилка для {source_id}: {e}")
        return INITIAL_RELIABILITY

def update_source_reliability(conn, source_id, is_ipso):
    if source_id is None: return None # Немає чого оновлювати
    if conn is None: print("[DB UpdateRel] Помилка: Немає з'єднання з
БД."); return None

    current_reliability = get_source_reliability(conn, source_id) #
Отримуємо актуальну
    new_reliability = current_reliability

    if is_ipso:
        new_reliability = max(MIN_RELIABILITY, current_reliability -
RELIABILITY_DECREMENT)
        change_type = "зменшено"
    else:
        new_reliability          =          min(INITIAL_RELIABILITY,
current_reliability + RELIABILITY_INCREMENT)
        change_type = "збільшено"

    # Оновлюємо тільки якщо значення дійсно змінилося (враховуючи
float-порівняння)
    if abs(new_reliability - current_reliability) > 1e-9: # Маленький
епсилон для порівняння float
        try:
            cursor = conn.cursor()
            cursor.execute('UPDATE  sources  SET  reliability  =  ?,
last_updated   =   CURRENT_TIMESTAMP   WHERE   source_id   =   ?',
(new_reliability, str(source_id)))
            conn.commit()
            print(f"[DB     UpdateRel]     Надійність     {source_id}
{change_type} до {new_reliability:.3f}")
        except sqlite3.Error as e:
            print(f"[DB UpdateRel] Помилка для {source_id}: {e}")
            return current_reliability # Повертаємо попереднє значення
у разі помилки оновлення
    return new_reliability

# --- Функції для обробки тексту---
def clean_text(text):
    if not isinstance(text, str): return ""
    text = text.lower()
```

```
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#\w+', '', text) # Видаляємо згадки та хештеги
    text = re.sub(r'\d+', '', text) # Видаляємо всі цифри
    text = re.sub(r'[^\w\s\-_]', '', text) # Видаляємо пунктуацію, крім підкреслень та дефісів
    text = re.sub(r'\s+', ' ', text).strip() # Нормалізуємо пробіли
    return text

# --- Побудова моделі ---
def build_ipso_rnn_model(vocab_size_param, embedding_dim_param, max_length_param, lstm_units_param, dropout_rate_param):
    text_input = Input(shape=(max_length_param,), name='text_input')
    reliability_input = Input(shape=(1,), name='reliability_input')
    embedding_layer = Embedding(input_dim=vocab_size_param, output_dim=embedding_dim_param, name='embedding')(text_input)
    lstm_layer = LSTM(lstm_units_param, dropout=dropout_rate_param, recurrent_dropout=dropout_rate_param, name='lstm')(embedding_layer)
    concatenated = Concatenate(name='concatenate')([lstm_layer, reliability_input])
    x = Dense(128, activation='relu', name='dense_1')(concatenated)
    x = BatchNormalization()(x)
    x = Dropout(dropout_rate_param)(x)
    x = Dense(64, activation='relu', name='dense_2')(x)
    x = BatchNormalization()(x)
    x = Dropout(dropout_rate_param)(x)
    output = Dense(1, activation='sigmoid', name='output')(x)
    model = Model(inputs=[text_input, reliability_input], outputs=output, name='IPSO_RNN_LSTM_Binary_Model')
    optimizer = tf.keras.optimizers.Adam()
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    # model.summary(line_length=100) # Можна закоментувати для Додатку
    return model

# --- Функція завантаження ресурсів---
def load_ai_resources(model_weights_path=MODEL_WEIGHTS_FILE_DEFAULT,
                      tokenizer_path=TOKENIZER_FILE_DEFAULT,
                      encoder_path=LABEL_ENCODER_FILE_DEFAULT,
                      db_path=DB_FILE_DEFAULT):
    print("\n[AI Loader] Завантаження ресурсів ШІ...")
    loaded_model, loaded_tokenizer, loaded_encoder, db_conn_local = None, None, None, None
    try:
        required_files = {'Ваги': model_weights_path, 'Токенізатор': tokenizer_path, 'Енкодер': encoder_path}
        missing_files = [name for name, path in required_files.items() if not os.path.exists(path)]
        if missing_files:
            raise FileNotFoundError(f"Відсутні файли: {', '.join(missing_files)}")
        print("[AI Loader] Всі файли ресурсів ШІ знайдено.")
```

```
        db_conn_local = init_db(db_path)
        if db_conn_local is None:
            raise ConnectionError(f"Не вдалося підключитися до БД:
{db_path}")

        print(f"[AI    Loader]    Завантаження    токенізатора    з
{tokenizer_path}...")
        with open(tokenizer_path, 'r', encoding='utf-8') as f:
            loaded_tokenizer = tokenizer_from_json(f.read())

        print(f"[AI   Loader]   Завантаження   кодувальника   міток   з
{encoder_path}...")
        with open(encoder_path, 'rb') as f:
            loaded_encoder = pickle.load(f)
        print(f"[AI Loader] Токенізатор та енкодер завантажено. Класи
енкодера: {list(loaded_encoder.classes_)}")
        if list(loaded_encoder.classes_) != BINARY_LABELS:
             print(f"!!!   ПОПЕРЕДЖЕННЯ:   Класи   в   LabelEncoder
({list(loaded_encoder.classes_)})   не   відповідають   BINARY_LABELS
({BINARY_LABELS}) у коді!")

        # Використовуємо VOCAB_SIZE з констант, оскільки модель
навчалася з цим лімітом
        build_vocab_size = VOCAB_SIZE
        print(f"[AI          Loader]          Побудова          моделі
(словник={build_vocab_size},          embedding_dim={EMBEDDING_DIM},
max_length={MAX_LENGTH})...");
        loaded_model     =     build_ipso_rnn_model(build_vocab_size,
EMBEDDING_DIM, MAX_LENGTH, LSTM_UNITS, DROPOUT_RATE)

        print(f"[AI      Loader]      Завантаження      ваг      з
{model_weights_path}...")
        loaded_model.load_weights(model_weights_path)

        print("[AI Loader] Прогрів моделі (тестове передбачення)...")
        _     =    loaded_model.predict([np.zeros((1,    MAX_LENGTH),
dtype=np.int32), np.zeros((1, 1), dtype=np.float32)], verbose=0)
        print("[AI Loader] Модель завантажена та готова.")

        return   loaded_model,   loaded_tokenizer,   loaded_encoder,
db_conn_local
    except Exception as e:
        print(f"[AI Loader] КРИТИЧНА ПОМИЛКА завантаження ресурсів ШІ:
{e}")
        if db_conn_local:
            db_conn_local.close()
        return None, None, None, None

# --- Функція передбачення---
def predict_single_message(model, tokenizer, label_encoder, db_conn,
text, source_id, max_length=MAX_LENGTH, threshold=0.5):
    if not text or not isinstance(text, str):
```

```
        print("[Predict] Попередження: Порожній або некоректний текст.");
        reliability = get_source_reliability(db_conn, source_id) if db_conn else INITIAL_RELIABILITY
        return False, reliability # is_ipso, final_reliability

    cleaned_text = clean_text(text)
    if not cleaned_text:
        print("[Predict] Попередження: Текст порожній після очищення.")
        reliability = get_source_reliability(db_conn, source_id) if db_conn else INITIAL_RELIABILITY
        return False, reliability

    current_reliability = get_source_reliability(db_conn, source_id)
    message_processed = False
    message_hash = None

    if source_id and db_conn:
        try:
            hash_input = f"{str(source_id)}::{cleaned_text}"
            message_hash = hashlib.sha256(hash_input.encode('utf-8')).hexdigest()
            cursor = db_conn.cursor()
            cursor.execute("SELECT 1 FROM processed_messages WHERE message_hash = ?", (message_hash,))
            result = cursor.fetchone()
            if result:
                message_processed = True
                # print(f"[Predict] Повідомлення від {source_id} (хеш: {message_hash[:8]}...) вже оброблено.")
        except Exception as e_hash_check:
            print(f"[Predict] Помилка перевірки хешу: {e_hash_check}")
    try:
        sequence = tokenizer.texts_to_sequences([cleaned_text])
        padded_sequence = pad_sequences(sequence, maxlen=max_length, padding='post', truncating='post')
    except Exception as e_tok:
        print(f"[Predict] Помилка токенізації: {e_tok}")
        return False, current_reliability

    reliability_input = np.array([[current_reliability]], dtype='float32')
    is_ipso = False
    confidence = 0.0

    try:
        if padded_sequence.shape != (1, max_length): raise ValueError(f"Неправильна форма тексту для моделі: {padded_sequence.shape}, очікується (1, {max_length})")
```

```
        if reliability_input.shape != (1, 1): raise
ValueError(f"Неправильна форма надійності для моделі:
{reliability_input.shape}, очікується (1, 1)")

        prediction_prob_ipso = model.predict([padded_sequence,
reliability_input], verbose=0)[0][0]
        confidence = prediction_prob_ipso
        is_ipso = prediction_prob_ipso >= threshold
    except Exception as e_pred:
        print(f"[Predict] Помилка передбачення моделлю: {e_pred}")
        return False, current_reliability

    final_reliability = current_reliability
    if not message_processed and source_id is not None and db_conn:
        updated_reliability_val = update_source_reliability(db_conn,
source_id, is_ipso)
        final_reliability = updated_reliability_val if
updated_reliability_val is not None else current_reliability
        if message_hash: # Записуємо хеш тільки якщо повідомлення не
було оброблене раніше
            try:
                cursor = db_conn.cursor()
                cursor.execute("INSERT OR IGNORE INTO
processed_messages (message_hash) VALUES (?)", (message_hash,))
                db_conn.commit()
            except Exception as e_hash_insert:
                print(f"[Predict] Помилка збереження хешу:
{e_hash_insert}")
    elif message_processed:
        # Якщо повідомлення вже оброблене, надійність не змінюється і
береться поточна
        final_reliability = current_reliability

    print(f"[Predict] Аналіз: Текст='{cleaned_text[:50]}...',
Джерело={source_id}, Ймовірн_IPSO={confidence:.3f},
IS_IPSO={is_ipso}(Поріг={threshold}),
Надійність={final_reliability:.3f}")
    return is_ipso, final_reliability
```

Додаток В. Схема взаємодії модулів програмного бота

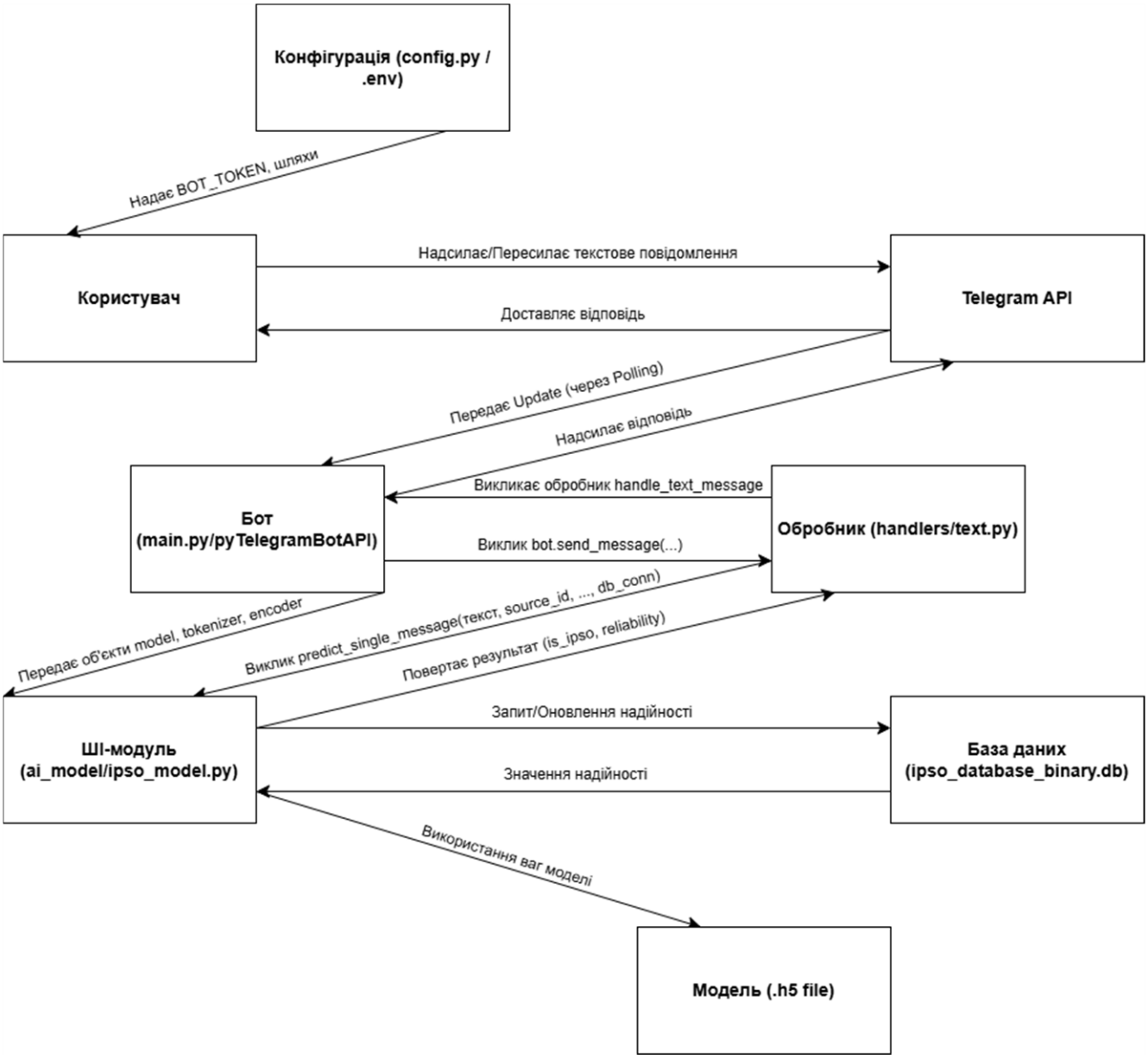

Рисунок В.1 – Схема взаємодії основних компонентів системи при обробці текстового повідомлення