Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання

Виконав: здобувач 2-го курсу,
групи 1КІТС-24м
спеціальності 125– Кібербезпека
та захист інформації
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Гуменчук Е.С.
(прізвище та ініціали)

Керівник:
Грицак А. В.
(прізвище та ініціали)
«10» грудня 2025 р.

Опонент:
Захарченко С. М.
(прізвище та ініціали)
«10» грудня 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

Юрій ЯРЕМЧУК
"10" грудня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформації
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

_____ **Юрій ЯРЕМЧУК**
"24" березня 2025 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Гуменчуку Едуарду Сергійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи: Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання.

Керівник роботи Грицак Анатолій Васильович к.т.н., доц. кафедри МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «24» вересня 2025 року № 313
2. Строк подання студентом роботи за тиждень до захисту
3. Вихідні дані до роботи: Наукові статті, книги, документи та електронні джерела. Існуюче програмне забезпечення, вимоги та обмеження до програмного забезпечення, технічна документація
4. Зміст текстової частини: Вступ. Розділ 1. Аналіз сучасних методів та технологій для протидії фішинговим загрозам. Розділ 2. Розробка удосконаленого методу виявлення фішингових загроз. Розділ 3. Практична реалізація та експериментальне дослідження гібридної системи виявлення фішингу. Розділ 4. Економічне обґрунтування та комплексний аналіз ефективності впровадження гібридної системи виявлення фішингових загроз. Висновки. Перелік посилань. Додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Зображення поетапної реалізації та налаштування вебдодатку

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина			
I	Грицак А. В. к.т.н., доц. кафедри МБІС		
II	Грицак А. В. к.т.н., доц. кафедри МБІС		
III	Грицак А. В. к.т.н., доц. кафедри МБІС		
Економічна частина			
IV	Ратушняк О. Г. к.т.н., доц кафедри ЕПВМ		

7.Дата видачі завдання 24 вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
		початок	закінчення	
1	Аналіз сучасних методів та технологій протидії фішингу	24.09.2025	28.09.2025	
2	Проєктування архітектури гібридної системи та вибір моделі DistilBERT	29.09.2025	08.10.2025	
3	Розробка алгоритмів генерації синтетичних даних та евристичних правил	09.10.2025	15.10.2025	
4	Програмна реалізація серверної частини та ядра аналізу	16.10.2025	25.10.2025	
5	Реалізація клієнтського інтерфейсу та модуля пояснень XAI	26.10.2025	28.10.2025	
6	Експериментальна перевірка, навчання моделі та аналіз результатів	29.10.2025	09.11.2025	
7	Економічне обґрунтування та аналіз ефективності впровадження	10.11.2025	15.11.2025	
8	Оформлення пояснювальної записки та графічних матеріалів	16.11.2025	20.11.2025	
9	Передзахист магістерської кваліфікаційної роботи	21.11.2025	21.11.2025	
10	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	22.11.2025	07.12.2025	
11	Захист магістерської кваліфікаційної роботи	08.12.2025	08.12.2025	

Студент (підпис) Гуменчук Е.С.

Керівник роботи (підпис) Грицак А. В.

АНОТАЦІЯ

УДК 004.056.5

Гуменчук Е. С. Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання. Магістерська кваліфікаційна робота зі спеціальності 125 – кібербезпека, освітня програма – кібербезпека інформаційних технологій та систем. Вінниця: ВНТУ, 2025. 171 с. На укр. мові. Бібліогр.: 53 назв; рис.: 19, табл.: 10.

Магістерська робота присвячена підвищенню захищеності веборієнтованих систем від сучасних фішингових атак, зокрема тих, що використовують соціальну інженерію та генеративний ШІ. Дослідження спрямоване на створення адаптивної гібридної системи для виявлення складних семантичних патернів у текстах та URL, долаючи обмеження традиційних методів.

У роботі реалізовано удосконалений метод виявлення загроз на основі доопрацьованої моделі DistilBERT, що балансує точність і швидкодію. Впроваджено модуль генерації синтетичних даних за допомогою великих мовних моделей у замкненому циклі для вирішення дефіциту даних та адаптації до дрейфу загроз. Також інтегровано механізм пояснюваного штучного інтелекту для забезпечення прозорості рішень.

Ефективність системи підтверджено результатами: точність класифікації – 98,2%, повнота – 97,9%, точність позитивного класу – 98,6%, що свідчить про мінімум хибних спрацювань. Економічний аналіз показує термін окупності 3,7 місяця та рентабельність 217% у перший рік експлуатації.

Ключові слова: фішингові загрози, трансформерні архітектури, глибинне навчання, DistilBERT, великі мовні моделі, генерація синтетичних даних, пояснюваний штучний інтелект , гібридна система виявлення, кібербезпека.

ABSTRACT

UDC 004.056.5

Humenchuk E. S. Improvement of the method of detection and prevention of phishing threats in web-oriented information systems based on transformer deep learning architectures. Master's thesis in the specialty 125 – cyber security, educational program – cyber security of information technologies and systems. Vinnytsia: VNTU, 2025. 171 p. In Ukrainian language. Bibliography: 53 titles; Fig.: 19, Table: 10.

The Master's thesis focuses on enhancing the security of web-based systems against modern phishing attacks, particularly those utilizing social engineering and generative AI. The study aims to develop an adaptive hybrid system capable of detecting complex semantic patterns in text and URLs, overcoming the limitations of traditional methods.

An improved threat detection method based on a fine-tuned DistilBERT model was implemented, balancing accuracy and performance. A closed-loop synthetic data generation module using Large Language Models was developed to address data scarcity and adapt to threat concept drift. Additionally, an Explainable AI mechanism based on Layer Integrated Gradients was integrated to ensure decision transparency.

The results demonstrate high system efficiency: 98.2% classification accuracy, 97.9% recall, and 98.6% precision, indicating minimal false positives. Economic analysis confirms feasibility with a payback period of 3.7 months and a 217% ROI in the first year of operation.

Keywords: phishing threats, transformer architectures, deep learning, DistilBERT, Large Language Models, synthetic data generation, Explainable AI, hybrid detection system, cybersecurity.

ЗМІСТ

ВСТУП

Актуальність. В умовах стрімкої цифровізації та поширення веборієнтованих сервісів безпека інформаційного простору стає критично важливою. Кількість кібератак невпинно зростає, причому фішинг залишається домінантним вектором початкової компрометації систем, призводячи до фінансових втрат та витоку конфіденційних даних. Зловмисники активно використовують новітні технології, зокрема генеративний штучний інтелект, для створення переконливих персоналізованих атак, які важко відрізнити від легітимної комунікації. Незважаючи на наявність засобів захисту, традиційні методи часто виявляються безсилими проти атак нульового дня та методів соціальної інженерії. Саме тому удосконалення системи виявлення фішингових сайтів та повідомлень є актуальним завданням сьогодення.

На сьогодні існують різноманітні методи виявлення фішингу, від чорних списків до евристичних правил, однак більшість з них мають свої обмеження, зокрема високий рівень хибних спрацювань та нездатність адаптуватися до швидкої зміни тактик зловмисників. Використання трансформерних архітектур глибинного навчання, великих мовних моделей та методів пояснюваного штучного інтелекту дозволяє значно підвищити точність аналізу семантики контенту та забезпечити прозорість рішень, що робить даний підхід перспективним для розв'язання цієї проблеми.

У роботі досліджено та порівняно різні підходи до виявлення фішингових загроз, оцінено їх ефективність та стійкість до сучасних методів обходу захисту. Основну увагу приділено трансформерним моделям обробки природної мови, які дозволяють виявляти приховані патерни соціальної інженерії в тексті та структурі URL. Проаналізовано обмеження класичних методів та запропоновано гібридний підхід, що поєднує швидкість евристичного аналізу з глибиною розуміння контексту, яку забезпечують нейронні мережі.

Крім того, у роботі розробляється власна адаптивна система для автоматичного виявлення фішингових загроз, що використовує легковагову

трансформерну модель DistilBERT для семантичного аналізу. Запропонована модель враховує проблему дефіциту якісних навчальних даних та вирішує її шляхом впровадження модуля генерації синтетичних зразків за допомогою великих мовних моделей. Особлива увага приділяється інтеграції механізмів пояснюваного штучного інтелекту, що дозволяє аналітикам безпеки розуміти причини класифікації та підвищує довіру до системи.

Метою даної роботи є розробка та удосконалення методу виявлення фішингових загроз у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання з використанням синтетичних даних для адаптації до нових типів атак. Для реалізації системи обрано стек технологій Python, PyTorch та бібліотеку Hugging Face Transformers, що є стандартом у сфері NLP. Серверна частина реалізована на Flask, а клієнтська – на React, що забезпечує зручну взаємодію користувача з системою захисту.

Для досягнення мети необхідно вирішити такі задачі: дослідити сучасні методи фішингу та існуючі засоби протидії; обґрунтувати вибір архітектури DistilBERT; розробити алгоритм генерації синтетичних даних через LLM для подолання концептуального дрейфу; реалізувати гібридну систему виявлення; впровадити модуль пояснення рішень; провести експериментальну перевірку ефективності системи.

Об'єкт дослідження – процес виявлення фішингових атак у веборієнтованих інформаційних системах.

Предмет дослідження – методи та моделі трансформерних нейронних мереж, методи генерації синтетичних даних та алгоритми пояснюваного штучного інтелекту для ідентифікації фішингового контенту.

Новизна роботи полягає у створенні адаптивної системи виявлення фішингу, яка поєднує доопрацьовану модель DistilBERT із замкненим циклом генерації навчальних даних за допомогою LLM. Запропонований підхід дозволяє системі автономно адаптуватися до нових видів загроз без необхідності ручного збору даних, а інтеграція методу Layer Integrated Gradients забезпечує інтерпретованість результатів аналізу.

Розділ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ ПРОТИДІЇ ФІШИНГОВИМ ЗАГРОЗАМ

1.1 Класифікація та життєвий цикл фішингових атак

Для створення ефективних механізмів протидії фішинговим атакам необхідно глибоко розуміти їхню природу, види та етапи реалізації. Фішинг не є однорідною загрозою; це багатогранний набір технік, адаптованих під різні цілі та типи жертв. Класифікація таких атак дозволяє систематизувати знання про них і визначити унікальні індикатори компрометації для кожного виду. Найпоширеніший тип – масовий фішинг, де зловмисники розсилають стандартизовані повідомлення великій аудиторії, розраховуючи на невеликий, але достатній відсоток успішності [1]. За даними Check Point за четвертий квартал 2024 року, у масових фішингових кампаніях найчастіше імітувалися бренди Microsoft, Apple і Google. Виявлено понад 5000 підроблених листів, що імітували повідомлення Microsoft, які вирізнялися високою якістю оформлення та відсутністю граматичних помилок [2].

Проте ще більшу небезпеку становлять цілеспрямовані атаки. Спір-фішинг спрямований на конкретних осіб або організації. У таких випадках нападники заздалегідь проводять розвідку, збираючи інформацію з відкритих джерел, соціальних мереж і професійних платформ. Це дозволяє створити персоналізовані й правдоподібні повідомлення. Завдяки стрімкому розвитку штучного інтелекту в 2025 році рівень персоналізації спір-фішингових атак сягнув безпрецедентних масштабів. За дослідженням Synchron, AI-генеровані фішингові листи мають показник кліків близько 44%, тоді як звичайні фішингові листи демонструють успішність на рівні 19-28%. Генеративні моделі дозволяють кіберзлочинцям створювати тексти, які важко відрізнити від реального листування, враховуючи професію, стиль спілкування, мову та поведінкові особливості жертви [3].

Ще більш вузькоспрямованим типом є вейлінг, де об'єктами атак стають високопоставлені особи в організаціях, такі як керівники, фінансові директори чи топменеджери. Мета таких атак – отримання доступу до стратегічно важливої

інформації чи фінансових активів. Ці атаки вирізняються високим рівнем підготовки й комбінують кілька комунікаційних каналів для підвищення довіри жертви. Наприклад, зловмисники можуть спочатку надіслати електронного листа, потім зателефонувати, імітуючи голос керівника за допомогою технологій синтезу мовлення, а на завершення створити діпфейк-відео для остаточного переконання.

Окрім електронної пошти, зловмисники активно використовують і інші канали для фішингових атак. Смішинг передбачає розсилку фішингових посилань через SMS-повідомлення, які зазвичай видають себе за повідомлення від банків, кур'єрських служб або державних установ. Вішинг, або голосовий фішинг, застосовує телефонні дзвінки для отримання конфіденційної інформації. У 2025 році популярність вішингу суттєво зросла завдяки впровадженню синтезованих голосів на основі технологій діпфейк, що робить такі дзвінки вкрай правдоподібними. Частота нападів із використанням голосового фішингу за останній період зросла більш ніж на 550% [4]. Іншим напрямом атак став квішинг, що використовує QR-коди для шахрайства. Цей метод швидко набирає обертів у США та Європі: шахраї розміщують QR-коди в електронних листах, рекламі або навіть фізичних листах. Вони ведуть користувачів на підроблені сайти або ініціюють завантаження шкідливого програмного забезпечення. За даними статті CNBC за липень 2025 року, десятки мільйонів американців вже постраждали від квішингу, що підкреслює серйозність цієї загрози [5].

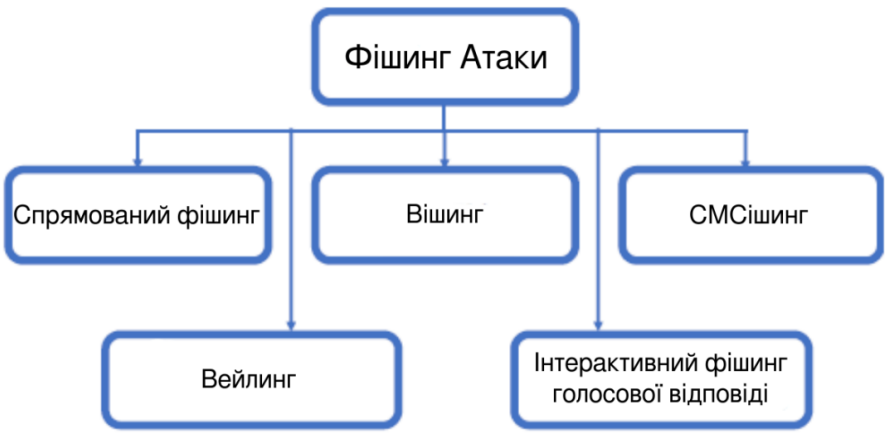


Рисунок 1.1 – Види фішингових атак

Розуміння життєвого циклу фішингової кампанії є критично важливим для знаходження вразливих точок захисту. Цей цикл умовно поділяється на кілька основних етапів зі своїми особливостями та індикаторами. Перший етап – підготовка та розвідка. Тут зловмисники визначають ціль та збирають про неї інформацію з відкритих джерел, таких як соціальні мережі, професійні платформи, корпоративні сайти і публічні бази даних. Також вони реєструють доменні імена, які нагадують легітимні адреси, використовуючи техніки тайпсквотингу або гомогліфи. На цьому етапі налаштовуються сервери для фішингових сайтів та інфраструктура для викрадення даних: анонімні поштові скриньки, проксі-сервери і платіжні системи.

Другий етап – створення та доставка фішингової приманки. Атакувальники формують електронні листи, SMS чи повідомлення у месенджерах, що стилістично нагадують офіційні повідомлення від авторитетних організацій. Такі повідомлення можуть містити шкідливі посилання, QR-коди або вкладення й розсилаються масово або цілеспрямовано потенційним жертвам. У 2025 році для генерації такого контенту активно використовуються мовні моделі, які створюють тексти з високою точністю й мінімумом граматичних помилок. Експерти з Positive Technologies зазначають, що межа між масовим і цільовим фішингом поступово стирається, оскільки персоналізовані елементи все частіше з’являються у масштабних кампаніях [6].

Третій етап передбачає експлуатацію, коли жертва, обдурена переконливим повідомленням, виконує дії в інтересах шахраїв. Людина переходить за посиланням або сканує QR-код, потрапляючи на фішинговий вебсайт, що зовні виглядає як справжній ресурс банку, соціальної мережі чи іншого сервісу. Там вона вводить конфіденційну інформацію: логін, пароль, номер кредитної картки чи інші особисті дані. Сучасні фішингові сайти часто використовують HTTPS-з'єднання та підроблені сертифікати безпеки для створення ілюзії легітимності. Багато з них також містять інтерактивні елементи, як-от CAPTCHA чи мімікрію двофакторної автентифікації, щоб виглядати більш правдоподібно.

Четвертий і завершальний етап – монетизація та приховування діяльності. Отримані дані жертв використовуються для фінансової вигоди через різноманітні схеми: незаконні банківські операції, покупки шляхом вкраденої платіжної інформації, доступ до корпоративних систем для крадіжки інтелектуальної власності або збут даних на чорному ринку. Як повідомляє IBM, на виявлення і ліквідацію порушень безпеки в середньому йде 277 днів, що дозволяє зловмисникам доволі довго використовувати викрадену інформацію [7]. За даними ФБР у США, щорічні збитки від фішингових атак перевищують 500 мільйонів доларів, при цьому приблизно 90% витоків даних безпосередньо пов'язані з цим видом атак. Досягнувши своїх цілей, зловмисники намагаються замести сліди: видаляють фішингові вебсайти, вимикають скомпрометовані домени та використовують інструменти анонімізації, як-от Tor або VPN, щоб приховати своє місцеперебування [8]. Схематично життєвий цикл фішингових атак показано на рисунку 1.2.

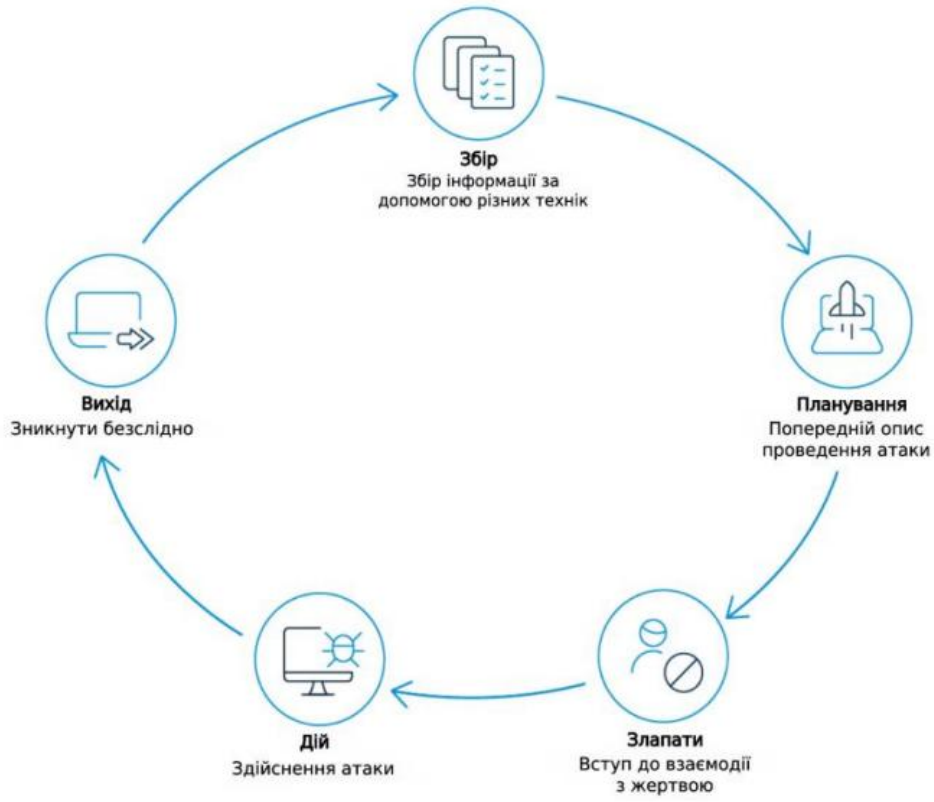


Рисунок 1.2 – Життєвий цикл фішингових атак

Наукові дослідження у сфері кібербезпеки доводять, що найбільш ефективними є механізми для виявлення фішингу, які виявляють загрози на етапах доставки та експлуатації. Такі системи аналізують URL-адреси і текстове наповнення повідомлень у реальному часі до того, як користувач стає жертвою атаки. Зокрема, Штонда та його колеги показали, що гібридні рішення, які об’єднують швидкі евристичні методи для виявлення очевидних ознак компрометації разом із глибоким аналізом семантики на основі машинного навчання для складніших патернів, демонструють найвищу ефективність у тестових дослідженнях [9]. Подібні висновки були представлені Стефанівим, який наголосив на важливості інтеграції декількох підходів до детекції для боротьби з фішинговими атаками ще на ранніх стадіях їх реалізації [10]. Дані свідчать, що багатошарові системи є здатними успішно виявляти як масові атаки зі звичними структурними аномаліями, так і складні індивідуалізовані кампанії, які включають контент, створений за допомогою штучного інтелекту, та методи соціальної інженерії.

1.2 Огляд наявних підходів до виявлення фішингу та їх обмеження.

Еволюція методів боротьби з фішингом демонструє безперервну взаємодію між кіберзлочинцями та експертами з кібербезпеки. Кожен новий підхід виникав у відповідь на недоліки попереднього, поступово призводячи до необхідності впровадження комплексних, багатошарових систем захисту. Згідно з дослідженнями Netskope Threat Labs, фішинг продовжує залишатися найпоширенішим методом обходу засобів безпеки для отримання доступу до корпоративних систем у Європі станом на 2025 рік [11]. Основна причина цього постійна здатність нападників адаптувати свої техніки, використовуючи сучасні технології, включно зі штучним інтелектом, щоб створювати складніші та персоналізовані атаки.

Традиційні механізми виявлення фішингу історично стали першою лінією захисту від таких загроз. Чорні списки – найпростіший спосіб протидії, який

полягає у веденні централізованих баз даних відомих фішингових URL-адрес та доменів. Під час переходу за посиланням система перевіряє його наявність у списку й блокує доступ в разі збігу. Інструменти на зразок Google Safe Browsing, PhishTank чи OpenPhish пропонують публічні бази даних фішингових сайтів, які інтегруються в браузери та антивірусне програмне забезпечення для захисту користувачів. Основною перевагою чорних списків є швидкість перевірки, яка займає мілісекунди, та низький рівень хибнопозитивних спрацьовувань для вже ідентифікованих загроз. Однак цей підхід має суттєвий недолік – він залишається реактивним. Чорний список поповнюється тільки після того, як атака вже відбулася, і хтось став її жертвою. Зважаючи на те, що середній час існування фішингового сайту часто не перевищує кілька годин, а часом навіть мене як годину, чорні списки безсилі проти атак нульового дня із використанням новостворених доменів або динамічно генерованих URL-адрес. За даними Anti-Phishing Working Group, понад 60% фішингових атак здійснюються через домени, зареєстровані менш ніж за тиждень до початку операції, залишаючи їх «невидимими» для систем на основі чорних списків [12].

Евристичні системи й засоби аналізу на основі правил стали наступним кроком у розвитку технологій виявлення фішингу. Вони розглядають URL не як ізольований об’єкт, а через аналіз його структурних і лексичних характеристик, щоб виявити типові ознаки фішингових ресурсів. Ці ознаки включають структурні аномалії, наприклад використання IP-адрес замість доменного імені – часта ознака тимчасової або незаконної інфраструктури, надмірно довгі URL з великою кількістю параметрів запиту, велика кількість піддоменів або використання символу @ для приховування справжнього домену. Лексичний аналіз виявляє ключові слова, такі як «login», «secure», «verify», «account» чи «update» у доменному імені чи адресному шляху URL; назви популярних брендів, наприклад «paypal», «amazon», «microsoft», у таких місцях адреси, де вони не мають бути; а також використання гомогліфів – візуально схожих символів з різних алфавітів для створення шахрайських доменів. Аналіз доменних даних охоплює також вік

домену (оскільки більшість фішингових ресурсів реєструються безпосередньо перед атакою), використання рідкісних або екзотичних доменів верхнього рівня, відсутність HTTPS або застосування само підписаних сертифікатів.

Перевага евристичних систем полягає в їхній здатності розпізнавати нові, раніше невідомі загрози без необхідності попереднього додавання цих загроз до бази даних. Вони також мають відносно низькі обчислювальні витрати, що дозволяє забезпечувати перевірку в режимі реального часу. Однак їхнім недоліком є високий рівень хибнопозитивних сигналів, адже легітимні вебсайти, особливо стартапи або невеликі компанії, іноді мають структури, які відповідають евристичним правилам. Крім того, досвідчені зловмисники швидко опанували створення посилань, здатних обходити прості евристичні методи. Наприклад, вони можуть реєструвати домени з правильною структурою, застосовувати легітимні HTTPS-сертифікати від сервісів типу Let's Encrypt і використовувати вкорочувачі URL або редиректи для маскування підозрілих частин.

Розвиток штучного інтелекту та доступність великих обсягів навчальних даних дали змогу запровадити підходи на основі класичного машинного навчання. Алгоритми, такі як Support Vector Machines, Random Forest, Decision Trees, Naive Bayes та Logistic Regression, почали активно застосовуватися для розробки бінарних або багатокласових класифікаторів, що дозволяють відрізняти фішингові URL від легітимних. Ці моделі проходять навчання на великих наборах даних, де кожен URL перетворюється на набір ознак. Ці ознаки включають як структурні характеристики, що використовуються для евристик, так і додаткові метрики – наприклад, ентропію домену, кількість цифр в адресі, наявність підозрілих ключових слів чи статистичні параметри довжини різних елементів URL. Дослідження показують, що алгоритми на кшталт Random Forest і Gradient Boosting досягають точності виявлення фішингових сайтів на рівні 95-97% у тестових вибірках. Проте цей підхід ускладнює процес ручного інжинірингу ознак, який потребує значних зусиль та глибокого розуміння предметної області.

Ефективність моделей машинного навчання тісно пов’язана з якістю та повнотою набору ознак, розробленого експертом із безпеки. Цей процес вимагає значних зусиль, адже потребує ретельного аналізу великої кількості фішингових і легітимних зразків для визначення атрибутів, здатних відрізняти одне від одного. Водночас він залишається недостатньо адаптивним до змін, спричинених еволюцією загроз. Моделі, створені на основі фіксованих наборів ознак, можуть втратити ефективність, якщо зловмисники почнуть використовувати інші методи обфускації, маскування чи нові технічні способи, що не були враховані в первинному наборі даних. Згідно зі звітом KnowBe4 про тенденції фішингових атак у 2025 році, більше ніж 76% фішингових листів містять щонайменше одну поліморфну ознаку, яка дозволяє їм обходити системи захисту, побудовані на сигнатурах і статичних правилах [13]. Такі зміни роблять неефективними застарілі підходи до виявлення загроз і наголошують на потребі у більш гнучких рішеннях.

Усі описані вище методи стикаються з базовою проблемою концептуального дрейфу, що є одним із найскладніших викликів машинного навчання у сфері кібербезпеки. Цей феномен стосується змін умовного розподілу ймовірності цільової змінної, які поступово знижують продуктивність моделей. У більш доступній формі це означає, що критерії для визначення «фішингу» та «легітимності» постійно змінюються і ці зміни не є лінійними або передбачуваними. Особливо у сфері кібербезпеки цей процес швидко прогресує через антагоністичний характер середовища, де нападники активно досліджують слабкі сторони захисту, тестують їх і модифікують свої стратегії, щоб уникнути виявлення. Згідно з дослідженнями, точність моделей для визначення фішингу може зменшитися на 15-30% уже через три-шість місяців після їхнього первинного розгортання без регулярного перенавчання. Зведені дані про порівняння таких підходів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз методів виявлення фішингу

Підхід	Принцип роботи	Переваги	Недоліки	Стійкість до атак нульового дня
Чорні списки	Перевірка URL за базою даних відомих шкідливих сайтів	Висока швидкість, низький рівень хибнопозитивних спрацьовувань	Реагує лише на відомі загрози, не може виявити нові атаки	Дуже низька
Евристичний аналіз	Аналіз структури та лексики URL на основі набору правил	Швидкий, не потребує зовнішніх баз даних, може виявляти нові загрози	Високий рівень хибнопозитивних спрацьовувань, правила легко обійти	Низька
Класичне машинне навчання	Класифікація URL на основі вектора вручну створених ознак	Автоматизує процес прийняття рішень, краща сумарна здатність	Залежність від ручного інжинірингу ознак, вразливість до нових тактик	Середня
Глибоке навчання	Автоматичне вивчення ознак з сирих даних (текст, URL) для класифікації	Не потребує ручного створення ознак, розуміння контексту та семантики	Високі обчислювальні вимоги, потреба у великих наборах даних	Високий

Статична модель, навчена в один момент часу на фіксованому наборі даних, поступово втрачає свою ефективність через швидку еволюцію патернів атак порівняно з оновленням систем захисту. Відповідно до звіту Всесвітнього економічного форуму «Глобальний огляд кібербезпеки 2025», спостерігається зростання продажів інструментів для створення діпфейків у даркнеті на 223% [14]. Це робить технологію дешевшою, якіснішою та доступною для ширшого кола зловмисників. Штучний інтелект не лише ускладнює методи фішингу, але й надає їх у розпорядження навіть тих нападників, у яких мінімальні технічні навички. Навіть новачки без досвіду програмування можуть здійснювати масові персоналізовані атаки, використовуючи готові фішингові набори та інструменти на

базі ШІ. Утворюється «сліпа зона» для атак нульового дня, що вимагає регулярного перенавчання моделей на актуальних даних, які включають новітні зразки загроз.

Розуміння слабких сторін окремих методів детекції фішингу сприяло розвитку гібридних підходів, що поєднують сильні сторони різних технік для формування більш ефективної та комплексної системи захисту. Сучасні дослідження вказують на переваги багатошарових систем безпеки, які інтегрують декілька методів виявлення, забезпечуючи вищу точність роботи, стійкість до спроб обходу і здатність адаптуватися до нових загроз.

Принцип гібридного підходу полягає у створенні багаторівневої системи захисту, де кожен рівень компенсує слабкі місця попереднього. Наприклад, перший рівень часто базується на чорних списках та репутаційних базах даних, що забезпечує швидке блокування відомих загроз із низькими обчислювальними витратами. Другий рівень включає евристичний аналіз для виявлення структурних аномалій і явних ознак компрометації. Третій рівень застосовує поведінковий аналіз та машинне навчання для здійснення глибокої перевірки контенту й контексту повідомлення. Останній, четвертий рівень, контролює дії користувача після взаємодії з потенційно небезпечним контентом. Такий підхід дозволяє максимально ефективно усувати більшість загроз на ранніх етапах, залишаючи ресурсомісткий аналіз лише для найскладніших випадків.

Сучасні комерційні платформи переконливо демонструють практичну ефективність гібридних рішень. Наприклад, SentinelOne безперешкодно інтегрується з поштовими шлюзами, такими як Mimecast, забезпечуючи автоматичне переміщення фішингових листів з небезпечними посиланнями до карантину [15]. Крім того, система використовує поведінковий аналіз для моніторингу активності в рамках усієї системи, виявляючи підозрілі тактики та методи, а також автоматично об’єднуючи взаємопов’язані інциденти у прозорі та зрозумілі оповіщення. Cynet діє подібно, застосовуючи вдосконалені алгоритми машинного навчання, які аналізують зміст і структуру електронних листів для визначення фішингових повідомлень та їх блокування [16]. Платформа також

враховує поведінку відправника, що дає змогу виявляти навіть найбільш витончені атаки. Система автоматично сканує всі посилання в листах і на вебресурсах через репутаційні бази даних і поведінковий аналіз, а також стежить за діями користувачів у мережі, вишукуючи аномалії, які можуть свідчити про компрометацію облікових записів. Завдяки застосуванню Extended Detection and Response (XDR) можлива інтеграція даних із різних джерел – кінцевих точок, мережі та хмари – з метою забезпечення повної видимості ситуації та координації зусиль для реагування на атаки, що можуть використовувати численні вектори одночасно.

Класифікація гібридної інтеграції виділяє кілька архітектурних підходів до організації детекторів. Паралельна інтеграція базується на синхронній роботі кількох незалежних детекторів із подальшим узагальненням результатів через механізми голосування, зважені середні або максимізацію ризику. Ця методика забезпечує високу швидкість обробки даних, однак вимагає впровадження складних алгоритмів для інтеграції результатів. У послідовній інтеграції детектори розташовані у вигляді конвеєра, де кожен наступний етап активується лише при необхідності додаткового аналізу. Це підвищує ефективність використання ресурсів шляхом раннього відсіювання менш складних випадків. Ієрархічна інтеграція передбачає делегування рішень: прості й швидкі методи обробки вирішують більшість завдань, а складніші запити передаються на більш потужні, хоча й менш оперативні елементи аналізу.

Статистичні дані свідчать про значну перевагу гібридних систем над монолітними у сфері виявлення загроз. В аналізі Datami відзначено, що комбіновані системи, які поєднують евристичний підхід і моделі глибокого навчання, досягають точності ідентифікації фішингових атак на рівні 97-99%, із часткою помилкових спрацьовувань менше ніж 2% [17]. У порівнянні, окремі методи демонструють набагато гірші результати. Сфера кібербезпеки підтверджує, що завдяки гібридним підходам кількість пропущених атак нульового дня знижується на 60-70% у порівнянні з традиційними системами, заснованими на сигнатурах чи

чорних списках. Ще однією суттєвою перевагою є краща стійкість до методів обходу: для зловмисника стає значно складніше обійти відразу кілька різнотипних детекторів, що ускладнює успішну атаку.

Сучасні системи для виявлення фішингових загроз повинні бути здатні адаптуватися до швидких змін або мати архітектуру, яка мінімізує ризик від явища концептуального дрейфу. Це обґрунтовує перехід від аналізу лише поверхневих атрибутів URL чи структурних особливостей до глибшого розуміння семантики контенту. Такі можливості стали доступними завдяки використанню моделей глибокого навчання, заснованих на трансформерних підходах. Розвиток технологій автентифікації, таких як багатофакторна верифікація, додає додатковий рівень безпеки, однак не повністю розв’язує питання оперативного виявлення спроб фішингу до моменту взаємодії користувача зі шкідливим контентом. Аналіз ProIT доводить, що із впровадженням технологій штучного інтелекту та машинного навчання методики протидії фішинговим атакам стають більш гнучкими та здатними обробляти великі обсяги даних, виявляючи аномальну поведінку користувачів та оперативно реагуючи на можливі загрози [18].

1.3 Трансформерні архітектури в задачах обробки природної мови для кібербезпеки

Впровадження трансформерних архітектур у 2017 році, яке почалося з революційної роботи Васвані та колег «Attention is All You Need», стало ключовим проривом у сфері обробки природної мови [19]. Ця архітектура суттєво змінила підходи до розв’язання завдань природної мовної обробки, запропонувавши новий механізм роботи з послідовностями, що виявився значно ефективнішим за попередні підходи на основі рекурентних мереж і згорткових нейронних мереж. Відповідно до дослідження Іосіфова, Соколова та Складанного, представленого у монографії «Безпечна обробка голосової інформації» за 2025 рік, більший інтерес до NLP-технологій з боку не лише зловмисників, але й державних структур висунув потребу у впровадженні передових методів обробки природної мови в системи

забезпечення безпеки [20]. Машинне навчання і глибокі нейронні мережі були визнані одними з найперспективніших підходів для покращення точності аналізу мовних даних у контексті кібербезпеки.

Основною інновацією, яка виділяє трансформери серед попередніх рекурентних та згорткових архітектур, є механізм само уваги, що радикально змінив спосіб обробки послідовних даних. Цей механізм дозволяє моделям одночасно оцінювати важливість кожного слова в реченні стосовно всіх інших слів у цьому ж реченні, а не виконувати таку операцію поступово, як це роблять рекурентні мережі. Математично увага розраховується як:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1.1)$$

де, Q, K та V є лінійними проєкціями вхідних даних, а dk – розмірність ключів. Завдяки використанню цього механізму модель здатна ефективно захоплювати складні контекстуальні взаємозв'язки навіть між словами, розташованими на великій відстані одне від одного в тексті. При цьому відсутні обмеження на довжину таких залежностей, які характерні для LSTM та GRU-мереж. Це забезпечує глибше та двостороннє розуміння контексту, на відміну від односпрямованих рекурентних мереж, що читають текст лише зліва направо і втрачають інформацію про подальший контекст. Архітектура моделі BERT показана на рисунку 1.3.

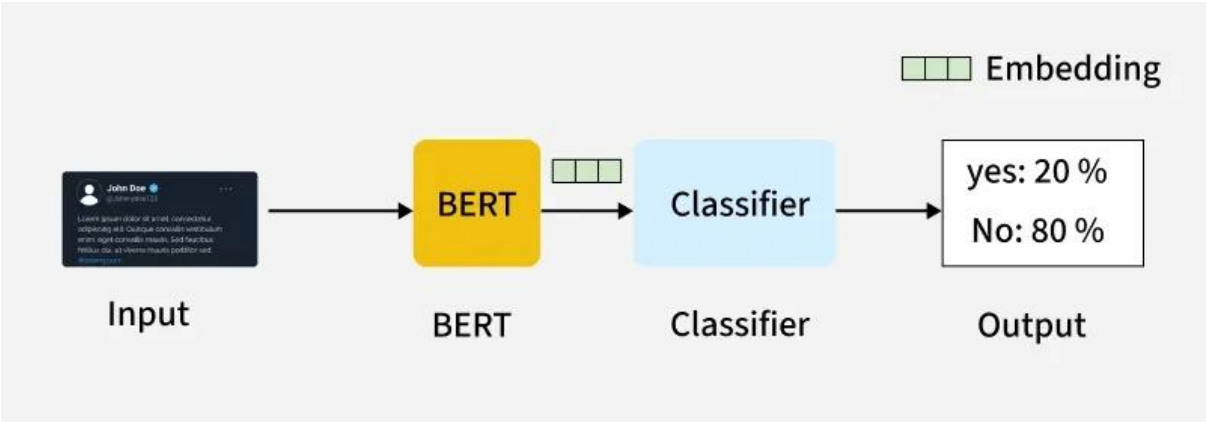


Рисунок 1.3 – Архітектура моделі BERT

Ця здатність до контекстуального аналізу є надзвичайно цінною в умовах кібербезпеки, де зловмисники часто вдаються до тонких мовних хитрощів, двозначності, гри слів та методів соціальної інженерії для створення переконливих фішингових повідомлень, які мотивують жертву виконати потрібні дії. Традиційні методи, які базуються на простому визначенні ключових слів чи використанні n-грамів, неспроможні виявити складні семантичні патерни. Наприклад, такі чинники як приховані загрози, штучне створення терміновості або маскування справжньої мети під легітимне спілкування залишаються поза їхнім досвідом аналізу. Як зазначено у кваліфікаційній роботі Павлата О. В. у 2025 році, великі мовні моделі демонструють глибоке розуміння контексту, логічне міркування та багатоступеневий аналіз методом Chain-of-Thought, що робить їх універсальним засобом для розв'язання задач кібербезпеки [21]. Від автоматизованого створення звітів про загрози до класифікації журналів подій та виявлення фішингових повідомлень, LLM стали критично важливими інструментами для роботи центрів безпеки.

На базі архітектури трансформера було розроблено сімейство моделей BERT, яке забезпечило видатні результати у сфері обробки природної мови. Сфери застосування BERT охоплюють класифікацію текстів, розпізнавання названих сутностей, відповіді на запитання та аналіз тональності. Основою моделі є двонапрямний підхід до навчання, який враховує лівий і правий контексти одночасно завдяки механізму маскованого мовного моделювання. Для цього модель навчається передбачати замасковані випадкові токени у тексті на основі доступного контексту, що є революційним порівняно з традиційними підходами, які прогнозували наступне слово лише на основі попередніх. Згідно з аналізом Unite.AI, BERT використовує завдання масковане мовне моделювання та передбачення наступного речення, що дозволяє йому розпізнавати як індивідуальні слова, так і складні зв'язки між реченнями [22].

Оригінальні моделі BERT, такі як BERT-Base із 110 мільйонами параметрів і BERT-Large із 340 мільйонами, вирізняються високою складністю та ресурсозатратним підходом. Це робить їх менш зручними для інтеграції у системи реального часу, мобільні додатки та середовища з обмеженими ресурсами. Для розгортання таких моделей потрібні потужні сервери з GPU або TPU, що підвищує операційні витрати та збільшує затримку в обробці запитів. Як альтернативу була запропонована компактна та ефективна модель DistilBERT від компанії Hugging Face, яка зберігає більшу частину функціоналу оригінального BERT і при цьому суттєво зменшує вимоги до ресурсів.

DistilBERT створюється шляхом застосування методу дистиляції знань – техніки трансферного навчання, за якої менша «студентська» модель навчається на основі поведінки більшої «вчительської» моделі. У процесі дистиляції модель студента не тільки знаходить правильні класи, але й відтворює розподіл ймовірностей, сформований вчителем для того ж самого навчального набору даних. Це допомагає студентській моделі засвоювати не лише явні залежності, але й приховані закономірності, які були знайдені вчителем під час навчання на великих текстових корпусах. Функція втрат під час цього процесу включає стандартну крос-ентропію між передбаченими справжніми мітками і дивергенцію Кульбака-Лейблера між ймовірностями вчителя й студента.

DistilBERT забезпечує приблизно 97% продуктивності у порівнянні з BERT на тестах, таких як GLUE (General Language Understanding Evaluation), при цьому має на 40% менше параметрів (66 мільйонів замість 110) та працює на 60% швидше під час інференсу.

Типова структура компактних моделей сімейства BERT, наприклад, distilbert-base-uncased, вирізняється зменшеною кількістю трансформерних шарів – зазвичай це шість замість дванадцяти в оригінальній BERT-Base. При цьому зберігається 768-вимірний прихований простір для представлення токенів і використовується дванадцять голів уваги в кожному шарі для паралельного аналізу різних аспектів контексту. Завдяки такій конструкції компактні BERT-моделі

забезпечують оптимальне співвідношення між продуктивністю й обчислювальною ефективністю. Це особливо важливо для систем кібербезпеки реального часу, які повинні обробляти значні обсяги даних із мінімальними затримками. Як зазначено в звіті Aezion про тенденції NLP до 2025 року, такі моделі, як DistilBERT і MobileBERT, пропонують ефективні та конфіденційно орієнтовані рішення NLP для мобільних застосунків, IoT-пристроїв і офлайн-сценаріїв, що робить їх ідеальними для безпекових систем, розгорнутих на периферійних пристроях [23].

Використання моделей сімейства BERT у задачах виявлення фішингу вже підтвердило свою високу результативність у багатьох академічних дослідженнях та реальних впровадженнях. Вони ефективно працюють як з аналізом URL-адрес, орієнтуючись на підозрілі патерни у структурі та семантиці вебадрес, так і з аналізом повноцінного текстового змісту електронних листів, SMS чи вебсторінок. Це дозволяє виявляти випадки використання технік соціальної інженерії та маніпуляцій. На відміну від традиційних методів машинного навчання, що покладаються на поверхневі ознаки, такі як ключові слова, довжина URL або кількість спеціальних символів, трансформерні моделі здатні розпізнати глибші семантичні шаблони, притаманні фішингу. До таких шаблонів належать створення штучного відчуття терміновості через фрази на кшталт «Ваш обліковий запис буде заблоковано протягом 24 годин», використання залякувань чи погроз, неочікувані обіцянки виграшів чи подарунків, або запити на підтвердження персональних даних під виглядом правдоподібних приводів.

Дослідження Є. В. Каплуна у кваліфікаційній роботі 2024 року показує, що після початкового NLP-аналізу системи використовують класифікаційні моделі для оцінки ризику кожного повідомлення, що дає змогу ефективно виявляти атаки соціальної інженерії в реальному часі [24]. Це дозволяє ідентифікувати навіть складні та добре замасковані загрози, які не мають очевидних структурних чи лексичних ознак компрометації, наприклад персоналізовані спір-фішингові листи, створені на основі даних про жертву із соціальних мереж. Фахівці з кібербезпеки наголошують, що компактні трансформерні моделі забезпечують високу точність

семантичного аналізу, водночас підтримуючи прийнятну обчислювальну продуктивність, придатну для розгортання у середовищах з низькою затримкою. Такі моделі дозволяють поєднати можливості великих трансформерів із практичними вимогами систем кібербезпеки, де оперативність є критично важливою для нейтралізації атак.

У 2025 році трансформерні моделі лишаються домінантними в галузі NLP, і з появою вдосконалених архітектур, таких як GPT-5, Claude 4, Gemini 2.5 та Mixtral, демонструється суттєве покращення у здатності до логічних висновків, роботи з пам'яттю, узагальнення даних та розуміння складних інструкцій. Ці моделі є мультимодальними, тому можуть аналізувати й генерувати текст, зображення, відео, аудіо та код, що значно розширює перспективи комплексного дослідження кіберзагроз. Проте для задач виявлення фішингу в реальному часі компактні вузькоспеціалізовані моделі, як-от DistilBERT, залишаються оптимальним вибором завдяки своїй продуктивності, низькій затримці та можливості функціонування на edge-пристроях без постійної залежності від хмарних сервісів. Такий локальний підхід забезпечує конфіденційність даних користувачів, адже обробка здійснюється безпосередньо на пристрої без передачі потенційно чутливої інформації на зовнішні сервери.

1.4 Проблема дефіциту даних та перспективи використання синтетичних даних

Однією з ключових перешкод у створенні ефективних моделей машинного навчання для потреб кібербезпеки є нестача якісних навчальних даних. Такі дані, правильно розмічені та ретельно підготовлені, є основою для тренування будь-якої моделі глибокого навчання. Однак у сфері кібербезпеки існує низка специфічних викликів, пов’язаних із доступом до подібних даних, що суттєво вирізняє цю галузь серед інших напрямів машинного навчання. Згідно з даними Всесвітнього економічного форуму, близько 66% організацій прогнозують, що на 2025 рік штучний інтелект матиме найвагоміший вплив на кібербезпеку. Водночас лише

37% компаній мають формалізовані підходи до оцінки безпеки та якості даних, які використовуються для навчання таких моделей. Це створює парадокс: попит на ШІ-рішення у сфері кібербезпеки зростає стрімкими темпами, водночас інфраструктура для їх підтримки та розвитку залишається недостатньою.

Однією з головних проблем є обмежений доступ до реальних даних про фішингові атаки. Інформація, що включає деталі про тактики зловмисників, методи обману та поведінкові особливості жертв, зазвичай є конфіденційною. Вона належить комерційним організаціям, банкам, провайдерам електронної пошти чи антивірусним компаніям. Ці структури неохоче діляться подібними даними навіть з академічною спільнотою через побоювання щодо розголошення слабких місць їхньої інфраструктури, ризиків для приватності чи загроз їхнім комерційним інтересам. Навіть якщо дані надаються, вони часто проходять процес анонімізації або значного агрегування, що суттєво скорочує їхню цінність для якісного аналізу. Згідно з дослідженням асоціації IT Ukraine, більше ніж 60% українських компаній планують використовувати готові комерційні рішення для захисту від кіберзагроз до 2025 року [25]. Основними причинами цього є гостра нестача кваліфікованих кадрів та відсутність власних навчальних дата сетів для розробки внутрішніх систем кіберзахисту.

По-друге, дані у сфері кібербезпеки зазнають надзвичайно швидкого старіння через вже згаданий феномен концептуального дрейфу. На відміну від задач комп'ютерного зору, де категорії, наприклад, кішок і собак залишаються відносно стабільними з часом, або обробки природної мови, де граматичні правила змінюються дуже повільно, характеристики фішингових атак можуть зазнати суттєвих змін за лічені місяці. Дані, зібраний рік чи навіть пів року тому, часто не відображають сучасних тактик атак, нових способів соціальної інженерії чи підвищення технологічного рівня зловмисників. За прогнозами аналітичної компанії Gartner, до 2027 року до 17% усіх кібератак будуть здійснюватися із застосуванням генеративного штучного інтелекту, що кардинально змінить теперішній ландшафт загроз [26]. Це вказує на те, що моделі, навчені на застарілих

даних без механізмів адаптації, втрачатимуть актуальність у міру зростання кількості атак за допомогою ШІ.

По-третє, процес збору та професійна розмітка даних у галузі кібербезпеки є надзвичайно складним та дорогим завданням, яке потребує висококваліфікованих фахівців. На відміну від багатьох інших сфер машинного навчання, де можна залучити до розмітки краудсорсинг-працівників із базовою підготовкою, анотування фішингових даних вимагає глибоких знань у тактиках соціальної інженерії, технічних аспектах вебтехнологій, здатності ідентифікувати тонкі ознаки компрометації та розуміння контексту комунікації. За оцінками Gartner, світові витрати на інформаційну безпеку до 2025 року досягнуть 212 мільярдів доларів США, що становитиме зростання на 15,1% порівняно з 2024 роком. Значна частина цих коштів буде спрямована на подолання дефіциту кваліфікованого персоналу та підтримку інфраструктури для збору й аналізу даних. Постійне посилення загроз, стрімкий розвиток хмарних технологій та гостра нестача спеціалістів із кібербезпеки примушують керівників підрозділів інформаційної безпеки збільшувати бюджети. Проте проблема браку актуальних і якісних даних залишається однією з найскладніших для вирішення.

Три основні чинники – конфіденційність, швидка застарілість і висока вартість – формують «вузьке місце» у процесі розробки систем виявлення фішингу, обмежуючи можливості тренування ефективних моделей глибокого навчання. Моделі трансформерного типу, наприклад BERT та його модифікації, розкривають свій повний потенціал лише за умови доступу до великого обсягу різноманітних даних для навчання, які покривають широкий спектр сценаріїв і крайових випадків. Недостатня кількість таких даних призводить до ризику перенавчання моделі на обмеженому наборі, що суттєво знижує її здатність узагальнювати нові й раніше незнайомі приклади. Це особливо критично в умовах динамічних і постійно змінюваних кіберзагроз.

Реальним і перспективним способом розв’язання цієї проблеми є генерація синтетичних даних – створення нових навчальних прикладів із використанням

алгоритмічних підходів. Традиційні методи аугментації даних, які широко застосовуються у сфері обробки природної мови, наприклад заміна слів на синоніми за допомогою WordNet чи інших тезаурусів, зміна слів у реченні місцями, вставка або видалення випадкових токенів, здебільшого мало ефективні для завдань кібербезпеки. Причина полягає в тому, що такі трансформації не генерують принципово нових семантичних сценаріїв або атак, а лише створюють прості варіації наявних прикладів. Це не сприяє створенню моделі з глибоким розумінням різноманіття загроз і може залишити її вразливою до атак, які відрізняються за формою, але мають однакову суть.

Революційний підхід до розв'язання цієї задачі стали можливим завдяки розвитку великих мовних моделей, таких як серія GPT від OpenAI, Llama від Meta, Claude від Anthropic чи Gemini від Google. Ці моделі відкрили нові горизонти для створення високоякісних синтетичних даних із багатим семантичним наповненням та контекстуальною узгодженістю. Навчені на колосальних обсягах інформації з інтернету, які охоплюють мільярди документів і трильйони токенів, вони здатні генерувати текст, який є логічним, граматично точним, стилістично природним і часто не відрізняється від тексту, написаного людиною. Щобільше, ці моделі демонструють емерджентні можливості: вони здатні міркувати, планувати та адаптуватися до специфічних інструкцій. Ці властивості роблять їх ідеальним інструментом для створення реалістичних і різноманітних сценаріїв соціальної інженерії.

У межах дослідження фішингової активності великі мовні моделі становлять важливий інструмент для генерування реалістичних і різноманітних прикладів фішингових повідомлень, які імітують різноманітні тактики, стилі комунікації та форми психологічного впливу, властиві реальним атакам. Використовуючи спеціально розроблені текстові запити, можливо інструктивно спрямовувати модель для створення текстів, адаптованих до конкретних сценаріїв соціальної інженерії. Сюди належать такі приклади, як симуляція офіційних повідомлень банківських установ із запитом підтвердження особистих даних, фіктивні

повідомлення про виграші в лотереях із вимогою надання банківських реквізитів, попередження щодо проблем із безпекою облікових записів у соціальних мережах чи електронній пошті, а також фальшиві запити від імені керівництва з вимогою негайного переказу коштів. У науковій роботі E-PhishGen: An Enhanced Phishing Email Generation Framework for Training Robust Phishing Detection Models підкреслюється, що згенеровані LLM фішингові зразки демонструють значно вищу якість порівняно із традиційними підходами. Зокрема, вони вирізняються більшою плавністю тексту, відповідністю контексту та стійкістю до протидійних механізмів.

Застосування такого підходу дозволяє не лише суттєво збільшити обсяги навчальних даних, що є критично важливим для моделей глибокого навчання, які потребують тисяч або навіть сотень тисяч прикладів для ефективного функціонування. Водночас це забезпечує значну якісну різноманітність навчального матеріалу. Створені синтетичні дані здатні охоплювати широкий діапазон мовних стилів – від формального ділового до розмовного неформального, використовуючи психологічні тригери, культурноспецифічні посилання та адаптацію до цільової аудиторії різних демографічних груп. Надзвичайно важливою особливістю цього підходу є й можливість моделювати потенційні нові вектори атак, ще незадокументовані у реальному середовищі. Це здійснюється через прогнозування можливих майбутніх тактик зловмисників на основі аналізу попередніх кампаній та їх еволюції. Такий підхід до кібербезпеки буде позиціонуватися як проактивний: він дає змогу заздалегідь «тренувати» моделі для розпізнавання атак ще до їх появи у реальних умовах, тим самим удосконалюючи здатність моделі до узагальнення та її захищеність від загроз нульового дня.

Інтеграція великих мовних моделей для генерації синтетичних даних наразі здобуває значну популярність завдяки доступу через публічні API-платформи, як-от Google Gemini, OpenAI GPT, Anthropic Claude та агрегатори на зразок OpenRouter, які дозволяють взаємодію з численними LLM. Типовий алгоритм створення синтетичних даних передбачає побудову структурованого запиту, що містить детальну специфікацію вимог до необхідної інформації, надсилання цього

запиту до API мовної моделі та обробку отриманих даних, які, як правило, представлені у форматі JSON для спрощення процесу аналізу та перевірки. Промпти, орієнтовані на генерацію фішингових зразків, здебільшого включають інструкції щодо типу атаки, рівня складності тексту, стилістичних і мовних особливостей, психологічних впливів та структурних елементів вихідного матеріалу.

Особливу увагу слід приділити культурноспецифічним параметрам під час генерації зразків. Це передбачає врахування локального контексту шляхом наслідування стилю популярних національних брендів, таких як банки, поштові сервіси чи державні установи; застосування характерних мовних конструкцій і специфічних ідіом для цільової аудиторії; а також урахування культурних особливостей і актуальних локальних трендів, які потенційно можуть використовуватися для соціальної маніпуляції. У межах дослідження E-PhishGen було проаналізовано ефективність таких моделей, як GPT-4 і Claude 3, для генерації фішингових листів. Результати засвідчили досягнення рівня успішності цих штучно створених атак до 44% в умовах контрольованого експерименту, що суттєво наближається до показників діяльності злочинців-людей. Таким чином, синтетичні дані, створені за допомогою великих мовних моделей, є достатньо реалістичними для формування навчальних вибірок детекторів загроз.

Архітектура систем автоматизованої генерації даних зазвичай складається з декількох ключових блоків. Менеджер промптів відповідає за розробку різноманітних шаблонів запитів та їх ротацію, що забезпечує широке розмаїття отриманих зразків. API-клієнт здійснює інтеграцію із провайдерами мовних моделей, підтримуючи механізми резервних спроб у разі короткотермінових збоїв у роботі сервісів, балансування навантаження між декількома платформами та зберігання кешованих результатів. Валідатор відповідей забезпечує перевірку отриманих результатів на предмет коректності структури та семантики генерованих даних з подальшим усуненням помилкових або нерелевантних входів.

Останній компонент – модуль збагачення дата сету – інтегрує новостворені синтетичні приклади до основного навчального набору даних, дотримуючись рівноваги між класами та гарантуючи широку різноманітність зразків для подальшого аналізу й моделювання.

Забезпечення якості та різноманітності синтетичних даних є одним із ключових аспектів сучасних дослідницьких підходів. Наукові результати свідчать, що базова генерація даних через одноразові запити до великих мовних моделей може призводити до явища, відомого як «колапс моди», коли модель продукує переважно однотипні зразки з обмеженою варіативністю. Для подолання цієї проблеми застосовуються методи диверсифікації. Зокрема, до таких методів належить використання температурного параметра в процесі вибіркового генерування, що дозволяє регулювати креативність генеративного процесу. Також використовується ітеративне вдосконалення запитів шляхом додавання конкретних сценаріїв та контексту. Ще одним підходом є використання кількох моделей LLM для формування ширшого стилістичного діапазону. У деяких дослідженнях застосовується технологія ланцюгового генерування запитів, що дозволяє спершу створити загальний сценарій, а потім деталізувати його через послідовність запитів. Такий підхід забезпечує побудову складніших та багатошарових фішингових повідомлень.

Результати порівняльного аналізу вказують на значну перевагу синтетичних даних, згенерованих за допомогою LLM, над традиційними підходами до аугментації. Зокрема, навчальні моделі, побудовані на основі дата сетів, доповнених прикладами, згенерованими LLM, демонстрували на 12–18% вищу точність у виявленні нових типів фішингових атак порівняно з моделями, які використовували традиційно аугментовані дані. Ця різниця пояснюється здатністю LLM створювати семантично нові сценарії замість суто синтаксичних варіацій уже наявних прикладів. Крім того, суттєво скорочується час, необхідний для створення великої кількості високоякісних фішингових шаблонів: наприклад, процес

створення 1000 таких зразків за допомогою LLM API може тривати лише 10–30 хвилин, тоді як ручна робота експертів потребує 40–50 годин. Фінансові витрати також значно знижуються: у випадку створення великого дата сету, вартість коливається в межах від $50 до $500, на противагу традиційним витратам у розмірі $50 000–100 000, які включають оплату роботи експертів. Таким чином, використання LLM робить високоякісні навчальні дані доступними навіть для невеликих організацій та дослідницьких колективів.

Однією з ключових переваг є можливість оперативної адаптації до нових загроз. З появою нового типу атак або тактик соціальної інженерії промпт можна швидко змінити для створення відповідних синтетичних зразків буквально за кілька годин. Натомість традиційний метод збору та розмітки реальних даних може вимагати тижнів чи навіть місяців. Це особливо важливо в умовах швидко наростальних ШІ-асистованих кібератак, де швидкість реакції має вирішальне значення для ефективності захисту.

У зв’язку з цим інтеграція генерації синтетичних даних за допомогою великих мовних моделей у процес навчання систем виявлення фішингу є не лише технічним розв'язанням проблеми дефіциту даних, але й стратегічним кроком до активної боротьби з концептуальним дрейфом та посиленням надійності захисних систем. Сучасні дослідження свідчать, що архітектури, які поєднують генерацію синтетичних зразків із замкненим циклом автоматичного перенавчання, дозволяють системам безперервно адаптуватися до еволюції загроз, уникаючи необхідності ручного втручання експертів. Такий підхід набуває особливого значення в контексті 2025 року, коли, за прогнозами аналітиків, зловмисники дедалі частіше використовуватимуть генеративний ШІ для створення фішингових кампаній. Це вимагає впровадження потужних і гнучких механізмів захисту, здатних гідно протистояти стрімкому зростанню цих загроз.

1.5 Висновки до розділу

У першому розділі магістерської роботи здійснено ґрунтовний огляд проблематики виявлення фішингових атак, що включає аналіз наявних технологічних методів та визначення концептуальних основ для створення адаптивних систем кіберзахисту. Виявлено, що фішинг залишається однією з найсерйозніших кіберзагроз, яка постійно еволюціонує завдяки використанню технологій штучного інтелекту зловмисниками. Варто зазначити, що впровадження генеративного штучного інтелекту призвело до зростання кількості атак на 4151%, підкреслюючи критичну необхідність вдосконалення механізмів захисту.

Класифікація фішингових атак охоплює такі види, як масовий фішинг, спір-фішинг, вейлінг, смішинг, вішинг та квішинг. Кожен вид відрізняється характерними індикаторами компрометації, що потребують адаптивних методів для їхнього ефективного виявлення. Аналіз життєвого циклу фішингової атаки, що включає етапи підготовки, створення/доставки, експлуатації та монетизації, дозволив виділити етапи доставки та експлуатації як ключові для автоматизованого втручання систем захисту. Саме на цих стадіях можливо мінімізувати ризики безпосереднього впливу на потенційну жертву.

Порівняльний аналіз наявних підходів до виявлення фішингових атак продемонстрував перехід від реактивних стратегій, таких як чорні списки, до проактивних моделей, побудованих на базі машинного навчання та глибокого аналізу. Наукове дослідження підтвердило, що традиційні методи, які базуються на сигнатурах чи класичних алгоритмах машинного навчання, мають обмежену ефективність у боротьбі з атаками нульового дня та схильні до проблем концептуального дрейфу, що спричиняє зниження їх продуктивності на 15-30% за період від трьох до шести місяців без належного оновлення моделей.

Розглянуто переваги гібридних підходів, які інтегрують кілька методів детекції. Ці рішення демонструють значну ефективність у порівнянні із

традиційними монолітними системами: точність виявлення досягає 97-99%, рівень хибно позитивних спрацювань залишається меншим за 2%, а кількість пропущених атак нульового дня зменшується на 60-70%. Такі показники підтверджують перспективність гібридних систем як основи сучасного кіберзахисту.

Дослідження сучасних трансформерних архітектур для обробки природної мови демонструє їхню значну ефективність у виявленні глибинних семантичних патернів, характерних для соціальної інженерії. Серед таких патернів виділяються створення штучного відчуття терміновості, застосування маніпулятивних технік та приховування справжньої мети повідомлення. Компактні моделі з родини BERT, зокрема DistilBERT, були визначені як оптимальний компроміс для застосування в реальних умовах завдяки поєднанню високої продуктивності та обчислювальної ефективності. Вони зберігають приблизно 97% точності повноцінних BERT-моделей, але мають на 40% менше параметрів і забезпечують на 60% швидшу обробку запитів. Ці показники роблять їх особливо привабливими для корпоративних систем безпеки, які мають високі вимоги до мінімізації затримок у відповідях.

Розв'язання фундаментальної проблеми дефіциту якісних навчальних даних у сфері кібербезпеки, спричиненого конфіденційністю реальних атак, швидким застаріванням зібраних дата сетів та значною вартістю експертної розмітки, було досліджено через інноваційну інтеграцію великих мовних моделей (LLM) для генерації синтетичних навчальних зразків. Огляд відповідної літератури свідчить, що використання синтетичних даних, створених LLM-моделями, підвищує точність виявлення нових типів атак на 12-18% у порівнянні зі звичайними методами аугментації. Крім того, цей підхід дозволяє суттєво скоротити час формування дата сету – від 40-50 годин до 10-30 хвилин – і знизити витрати з 50 000–100 000 доларів до 50–500 доларів для дата сету обсягом 10 000 зразків. Така методика сприяє не лише масштабуванню навчальної бази даних, але й проактивній

адаптації моделей до еволюційних змін атак через механізми автоматичного перенавчання на свіжих синтетичних даних.

Дослідження застосування методів пояснювального штучного інтелекту підтвердило критичну важливість прозорості рішень у системах кібербезпеки для збереження довіри користувачів і забезпечення аудиту класифікацій. Порівняльний аналіз методів, таких як LIME, SHAP, Integrated Gradients, Layer Integrated Gradients та Attention Visualization, показав, що градієнтні підходи, зокрема Layer Integrated Gradients, забезпечують оптимальний баланс між ключовими характеристиками: точністю пояснень (досягаючи 90-95%), стійкістю до незначних варіацій у вхідних даних (85-90%), швидкістю обчислень (200-800 мс), а також інтуїтивністю візуалізацій для спеціалістів із кібербезпеки.

На основі проведеного теоретичного аналізу визначено мету дослідження – створення та експериментальна перевірка гібридної системи для виявлення фішингових атак. Ця система поєднує семантичний аналіз із використанням доопрацьованої трансформерної моделі, швидкодійні евристичні правила для структурного аналізу URL-адрес, а також механізм адаптації до нових загроз за допомогою генерації синтетичних навчальних даних із застосуванням великих мовних моделей.

Теоретичні засади, викладені в першому розділі, слугують концептуальною основою для розроблення архітектури й вибору алгоритмічних рішень. Ці аспекти буде докладніше висвітлено у другому розділі, а деталі реалізації адаптивної системи виявлення фішингу будуть розглянуті в третьому розділі дослідження.

Розділ 2. РОЗРОБКА УДОСКОНАЛЕНОГО МЕТОДУ ВИЯВЛЕННЯ ФІШИНГОВИХ ЗАГРОЗ

2.1. Формалізація задачі та вибір базового методу-аналога

Виявлення фішингових загроз у веборієнтованих інформаційних системах належить до класу задач бінарної класифікації текстових даних та структурних атрибутів. Формально задачу можна визначити наступним чином. Нехай $D = \{(x_i, y_i)\}_{i=1}{}^{N}$ являє собою множину вхідних даних, де $x_i \in X$ позначає вхідний об'єкт, а $y_i \in \{0, 1\}$ відповідає мітці класу, де 0 позначає легітимне повідомлення, а 1 – фішингову загрозу.

Метою дослідження є побудова відображення $F: X \longrightarrow \{0, 1\}$, яке мінімізує ймовірність помилкової класифікації на невідомих даних. Формально це можна записати як задачу оптимізації функції втрат:

$$min_F E_{(x,y)} \sim P_{data}[\Vert(F(x) \neq y)], \tag{2.1}$$

де $\Vert(\cdot)$– індикаторна функція, а P_{data}– істинний розподіл даних, який у реальних умовах є невідомим і апроксимується навчальною вибіркою.

На практиці якість класифікатора F оцінюється через вектор метрик M = *{Precision, Recall, F1, Accuracy}.* Окрім метрик точності, критично важливими є часові характеристики системи *T(x)*, оскільки аналіз повідомлень має відбуватися у режимі реального часу. Обмеження на час обробки одного запиту можна формалізувати як $T(x) \leq 100$ мс для 99-го перцентиля запитів.

Додатковою вимогою, що висувається до сучасних систем кібербезпеки, є інтерпретованість прийнятих рішень. Це формалізується як необхідність існування функції пояснення $E: X \times F \longrightarrow R$, де R – простір семантичних пояснень, наприклад, ваги слів або токенів, що задовольняють властивості локальної вірності та стабільності [27]. Для вибору базового методу-аналога було проведено аналіз існуючих рішень у відкритих репозиторіях моделей машинного навчання, зокрема

Hugging Face. Серед них виділяються дві характерні реалізації, що представляють сучасний підхід до задачі:

Модель на базі архітектури URLBERT, наприклад, CrabInHoney/urlbert-tiny-v2. Це компактна версія трансформера на ~3,7 млн параметрів, спеціалізована на аналізі структури мережевих адрес. Вона ефективна для виявлення загроз на рівні URL, проте ігнорує семантичний контекст тіла повідомлення.

Модель на базі архітектури DistilBERT, наприклад, cybersectony/phishing-email-detection. Ця модель призначена для класифікації тексту повідомлень і демонструє високі показники на тестових наборах F1-міра ~97,7%. Процес класифікації в таких системах базується на обчисленні ймовірності:

$$P(y = 1|x) = \sigma\big(W \cdot h_{[CLS]} + b\big), \quad (2.2)$$

де $h_{[CLS]}$ – векторне представлення спеціального токена класифікації, а σ – сигмоїдна функція активації.

В якості базового методу-аналога в даній роботі обрано підхід, реалізований у другій моделі DistilBERT fine-tuning, оскільки він забезпечує глибший семантичний аналіз порівняно з аналізом лише URL.

Проте проведений аналіз виявив суттєві обмеження цього базового методу, які унеможливлюють його ефективне використання як єдиного засобу захисту:

1. Висока латентність: час інференсу моделі DistilBERT на стандартному обладнанні часто перевищує допустимі межі для систем реального часу (80-120 мс), особливо при високому навантаженні.
2. Відсутність механізмів пояснення: базовий метод працює за принципом "чорної скриньки", не надаючи аналітику інформації про те, які саме фрагменти тексту вплинули на рішення.
3. Вразливість до концептуального дрейфу: статично навчена модель швидко втрачає актуальність при зміні тактик зловмисників. Без регулярного донавчання точність може знижуватися на 15-30% протягом кількох місяців.

4. Нераціональне використання ресурсів: обробка тривіальних загроз, наприклад, явних IP-адрес у посиланнях, через важку нейромережу створює невиправдане навантаження на обчислювальні потужності.

На основі цього аналізу сформульовано гіпотезу дослідження: інтеграція оптимізованої трансформерної моделі в гібридну архітектуру з евристичним модулем пре-фільтрації та замкненим контуром генерації синтетичних даних дозволить усунути недоліки базового методу.Формально пропонується перехід від функції $F_{neural}(x)$ до композитної функції:

$$F_{hybrid}(x) = \psi(F_{neural}(x), F_{heuristic}(x)), \quad (2.3)$$

де ψ – функція агрегації, що реалізує логіку прийняття рішення з урахуванням рівня довіри до кожного з компонентів та забезпечує механізм «вето» для швидких евристик [28].

2.2. Розробка структурно-функціональної моделі гібридної системи

Розроблений метод виявлення фішингових загроз реалізовано як структурно-функціональну модель, що формалізує взаємодію між компонентами семантичного аналізу, евристичної перевірки та адаптивного навчання. На відміну від базових методів, які здійснюють аналіз через єдиний монолітний класифікатор, запропонована модель розділяє процес виявлення на два незалежні конвеєри з різними часовими характеристиками: офлайн-підсистему адаптивного навчання та онлайн-підсистему гібридного аналізу вхідних запитів.

Загальну структуру методу представлено на рисунку 2.1. Модель складається з трьох функціональних блоків, які формують замкнений цикл безперервного удосконалення системи. Офлайн-компонент виконує генерацію синтетичних навчальних даних через велику мовну модель та періодичне перенавчання класифікатора на розширеному наборі даних. Онлайн-конвеєр забезпечує безпосереднє виявлення загроз через паралельне застосування евристичних правил та семантичного аналізу з подальшою агрегацією результатів. Сервісний інтерфейс надає стандартизований доступ до функціональності методу через програмні

інтерфейси, що дозволяє інтегрувати систему у наявні корпоративні рішення безпеки.

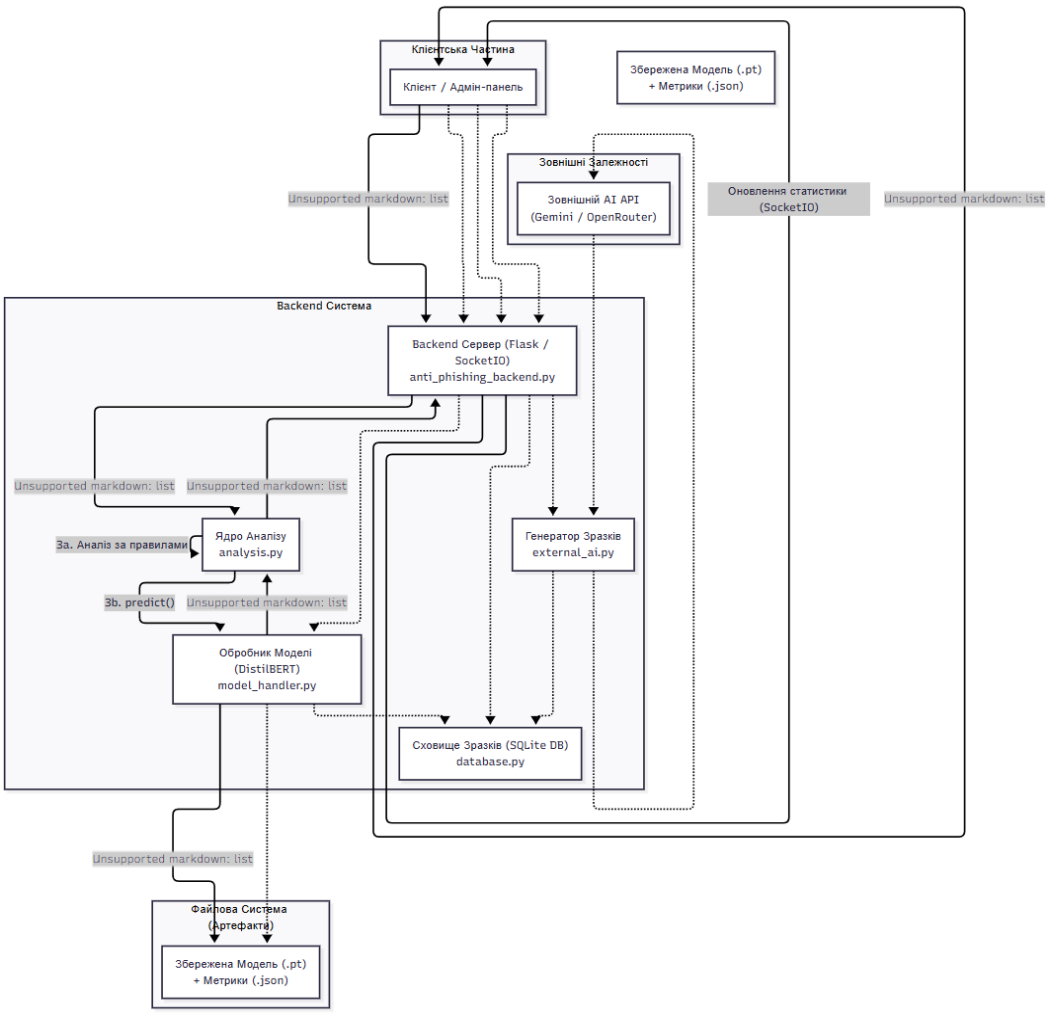


Рисунок 2.1 – Структурно-функціональна модель удосконаленого методу виявлення фішингу

Формально процес виявлення загроз можна описати як послідовність перетворень вхідних даних через функціональні блоки системи. Нехай $x=(u,t,m)$ позначає вхідний об'єкт, де u означає уніфікований ідентифікатор ресурсу, t – текстовий вміст повідомлення, а m – метадані заголовків. Офлайн-підсистема формує навчальну множину

$$D_{train} = D_{real} \cup D_{synth}, \quad (2.4)$$

де D_{real} містить автентичні зразки, а D_{synth} являє собою синтетично згенеровані дані через велику мовну модель. Процес генерації формалізується як відображення:

$$G: P \times \Theta_{LLM} \rightarrow D_{synth}, \quad (2.4)$$

де P позначає множину промптів, що описують характеристики цільових зразків, а Θ_{LLM} – параметри генеративної моделі.

Онлайн-підсистема виявлення реалізує гібридну схему аналізу через два паралельні канали обробки. Евристичний канал застосовує множину правил $R=\{r_1, r_2, \ldots, r_k\}$, кожне з яких перевіряє наявність конкретних індикаторів загрози. Формально евристична функція визначається як:

$$f_{heur}(x) = \bigvee_{i=1}^{k} r_i(x) \quad (2.5)$$

де V позначає логічну диз'юнкцію, а $r_i(x)$ представляє результат застосування окремого правила. Якщо хоча б одне з критичних правил спрацьовує, об'єкт негайно класифікується як загроза без передачі до семантичного аналізатора, що забезпечує швидкодію у межах 3-5 мілісекунд для очевидних випадків атак. задовольняти Семантичний канал здійснює глибокий аналіз контекстуального значення вхідного тексту через трансформерну архітектуру. Процес починається з токенізації вхідної послідовності та перетворення токенів у векторні представлення фіксованої розмірності. Нехай $T(x) = [t_1, t_2, \ldots, t_n]$ позначає послідовність токенів, отриману з вхідного об'єкта, де n змінюється залежно від довжини тексту. Кожен токен відображається у векторний простір через комбінацію матриць:

$$e_i = E_{token}[t_i] + E_{position}[i], \quad (2.6)$$

де E_{token} та $E_{position}$ відповідають навченим таблицям ембеддінгу для токенів та позиційних індексів. Отримані векторні представлення проходять через шість послідовних шарів трансформера, кожен з яких застосовує механізм багатоголової уваги та позиційно-незалежне перетворення через повнозв'язну мережу. Вихідне представлення спеціального токена класифікації $h_{[CLS]}$ використовується для

обчислення ймовірності належності до класу фішингу через лінійне перетворення з сигмоїдною активацією:

$$P_{neural}(y = 1|x) = \sigma(W \cdot h_{[CLS]} + b), \quad (2.6)$$

де σ – це сигма, W – ваги, b – зсув та означають параметри класифікаційного шару, навчені на етапі тонкої настройки. Агрегація результатів з обох каналів здійснюється через стратегію максимального ризику, яка формалізується як:

$$y_{final} = max\{f_{heur}(x), \Vert(P_{neural}(y = 1|x) > \tau)\}, \quad (2.7)$$

де $\Vert$ – це індикаторна функція, а τ – поріг класифікації, встановлений на основі аналізу ROC-кривої на валідаційному наборі. Така схема агрегації гарантує, що критичні загрози, виявлені евристичними правилами, не можуть бути проігноровані навіть якщо нейромережевий компонент дає низьку оцінку ризику, що мінімізує ймовірність хибно негативних спрацювань.

Структурно-функціональна модель онлайн-конвеєра детально представлена на рисунку 2.2, який ілюструє послідовність операцій від надходження запиту до формування фінального рішення з поясненням. Конвеєр починається з попередньої обробки вхідних даних, що включає нормалізацію структури уніфікованого ідентифікатора ресурсу, видалення надлишкових пробільних символів та приведення тексту до уніфікованого формату. Після цього відбувається паралельний запуск евристичного та семантичного аналізаторів, результати яких передаються до блоку агрегації.

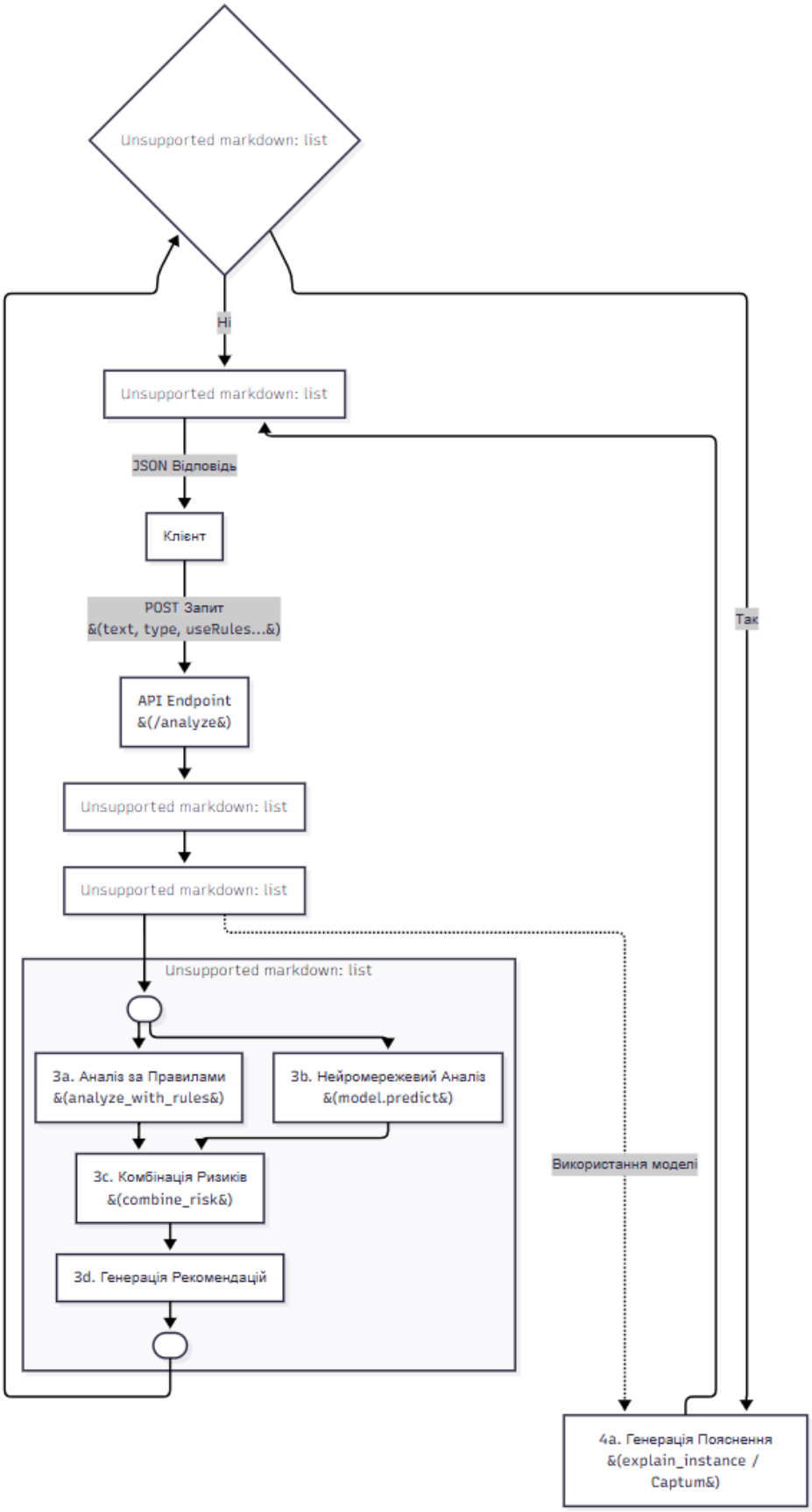


Рисунок 2.2 – Потоки даних у онлайн-конвеєрі методу виявлення загроз

Критичною особливістю розробленої моделі є інтеграція підсистеми пояснення рішень, яка формує інтерпретовану репрезентацію факторів, що вплинули на класифікацію. Для випадків, коли рішення прийнято на основі евристичних правил, пояснення формується як перелік спрацьованих правил з відповідними фрагментами вхідних даних. Для рішень, прийнятих нейромережевим компонентом, застосовується метод інтегрованих градієнтів на рівні шарів, який обчислює внесок кожного токена у фінальне рішення через інтегрування градієнтів активацій проміжних шарів вздовж шляху від нульового baseline до фактичного входу. Формально важливість токена t_i визначається як:

$$A_i = \int_{a=0}^{1} \frac{df(x_0 + a(x - x_0))}{de_i}\, da \tag{2.8}$$

де x_0 позначає базову лінію входу, а f означає функцію класифікації. Токени з найвищими абсолютними значеннями атрибуції візуалізуються як найбільш значущі для прийнятого рішення. Офлайн-компонент адаптивного навчання функціонує асинхронно відносно онлайн-конвеєра та реалізує замкнений цикл безперервного удосконалення класифікатора. Процес організовано як періодичне виконання наступної послідовності операцій: аналіз помилкових класифікацій на виробничих даних для ідентифікації слабких місць поточної моделі, формування промптів для генерації синтетичних зразків, що покривають виявлені прогалини у навчальних даних, генерація синтетичних зразків через велику мовну модель, валідація якості згенерованих даних через автоматизовані перевірки семантичної узгодженості та відповідності цільовому класу, об'єднання валідованих синтетичних зразків з існуючим навчальним набором, перенавчання моделі на розширеному наборі даних, оцінка продуктивності нової версії моделі на контрольному наборі, впровадження нової версії у виробниче середовище за умови покращення метрик якості. Формально цикл адаптації описується як ітераційний процес оптимізації функції втрат на динамічно розширюваній множині даних

$$\theta_{t+1} = \arg min_{\theta} L(D_{real} \cup D_{synth}^{(t)}, \theta), \tag{2.9}$$

де θ_t позначає параметри моделі на ітерації t, $D_{synth}^{(t)}$ означає множину синтетичних даних, згенерованих на основі аналізу помилок моделі з параметрами Θ_t, а L представляє функцію втрат бінарної крос-ентропії. Такий підхід забезпечує автоматичну адаптацію системи до еволюції тактик зловмисників без необхідності залучення експертів для ручної розмітки нових зразків загроз.

Розроблена структурно-функціональна модель задовольняє ключові вимоги, сформульовані у підрозділі 2.1. Гібридна архітектура з паралельними каналами обробки забезпечує високу швидкодію через швидке відсікання очевидних загроз евристичними правилами, зберігаючи при цьому здатність до глибокого семантичного аналізу складних випадків. Замкнений цикл адаптивного навчання вирішує проблему концептуального дрейфу, дозволяючи системі автономно актуалізувати власні знання про нові типи атак. Інтеграція підсистеми пояснення рішень забезпечує інтерпретованість класифікації, що є критичним для довіри користувачів та можливості верифікації логіки системи експертами з кібербезпеки. Розділення на офлайн та онлайн компоненти дозволяє виконувати ресурсоємкі операції генерації даних та перенавчання незалежно від критичних за часом операцій аналізу вхідних запитів, що оптимізує використання обчислювальних ресурсів та забезпечує стабільну латентність відгуку системи.

2.3. Удосконалення алгоритму семантичного аналізу на основі трансформерних архітектур

Для реалізації семантичного компонента гібридного методу виявлення фішингових загроз розроблено алгоритм глибокого аналізу природномовних конструкцій, заснований на дистильованій трансформерній архітектурі з оптимізованою обчислювальною складністю. На відміну від базових методів-аналогів, що використовують повну архітектуру BERT з дванадцятьма шарами та 110 мільйонами параметрів, запропонований алгоритм реалізує стиснуту модель із

шістьма трансформерними шарами та 66 мільйонами параметрів, отриману через процес дистиляції знань від повної моделі-вчителя до компактної моделі-учня.

Формально алгоритм семантичного аналізу визначається як послідовність перетворень вхідної текстової послідовності у ймовірнісну оцінку належності до класу фішингових загроз. Нехай $x = (w_1, w_2, \ldots, w_L)$ позначає вхідну послідовність слів довжини L, де кожне слово w_i належить природній мові. Алгоритм складається з чотирьох послідовних етапів перетворення: токенізація та сегментація вхідної послідовності на підслівні одиниці, відображення токенів у векторний простір через таблиці вбудовувань, ітеративне перетворення векторних представлень через шість шарів трансформера з механізмом багатоголової уваги, обчислення фінальної ймовірності класифікації через лінійне перетворення з сигмоїдною активацією.

Етап токенізації здійснюється через алгоритм WordPiece, який розбиває слова на підслівні одиниці на основі словника розміром 30522 токени. Алгоритм починає з спроби розпізнати слово цілком, і якщо воно відсутнє у словнику, ітеративно розбиває його на менші сегменти до досягнення відомих токенів або символів. Формально операція токенізації визначається як відображення:

$$T: \sum{}^{*} \rightarrow V^{*}, \tag{2.10}$$

де $\sum *$ позначає алфавіт природної мови, V^{*} означає словник токенів, а зірка вказує на послідовності довільної довжини. До початку послідовності токенів додається спеціальний токен класифікації *[CLS]*, а наприкінці розміщується токен сепаратора *[SEP]*, що дозволяє моделі розрізняти межі текстових сегментів та формувати агреговане представлення для класифікації [29].

Векторне представлення токенів формується через комбінацію трьох типів вбудовувань: токенних, позиційних та сегментних. Нехай t_i позначає i-й токен у послідовності після токенізації. Його векторне представлення обчислюється як $$

$$e_i = E_{token}[t_i] + E_{position}[i] + E_{segment}[s_i], \tag{2.11}$$

де $E_{token} \in R^{|V| \times d}$ означає матрицю токенних вбудовувань, $E_{position} \in R^{512 \times d}$ відповідає позиційним кодуванням для максимальної довжини послідовності 512 токенів, $E_{segment} \in \{0, 1\}$ вказує на приналежність токена до першого або другого сегмента тексту, а $d = 768$ означає розмірність прихованого простору представлень. Додавання позиційних кодувань є критичним для трансформерної архітектури, оскільки механізм самоуваги не має вбудованого розуміння порядку токенів, на відміну від рекурентних архітектур. Отримані векторні представлення e_1, e_2, …, e_n формують матрицю $E \in R^{n \times d}$, яка передається до першого шару трансформера.

Кожен з шести шарів трансформера застосовує дві основні операції: механізм багатоголової уваги та позиційно-незалежне перетворення через повнозв'язну мережу прямого поширення. Механізм багатоголової уваги дозволяє моделі одночасно фокусуватися на різних аспектах залежностей між токенами через паралельні обчислення уваги у різних підпросторах представлень. Формально для кожної голови уваги $$h$$ обчислюються три проєкції вхідних представлень:

$$\mathrm{Q}^{(h)} = EW_Q^{(h)}, K^{(h)} = EW_K^{(h)} \tag{2.12}$$

та значення:

$$\mathrm{V}^{(h)} = EW_V^{(h)}, W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in R^{d \times d_k} \tag{2.13}$$

означають навчені матриці проєкцій, а $d_k = d/H$ позначає розмірність підпростору для кожної з $H = 12$ голов уваги. Вага уваги між парою токенів визначається через скалярний добуток їхніх проєкцій запиту та ключа з подальшою нормалізацією через:

$$\mathrm{Attention}(\mathrm{Q}^{(h)}, K^{(h)}, V^{(h)}) = softmax\left(\frac{Q^{(h)}(K^{(h)})^T}{\sqrt{d_k}}\right) V^{(h)}, \tag{2.14}$$

де ділення на $\sqrt{d_k}$ запобігає насиченню функції softmax для великих значень скалярних добутків. Результати від усіх голов уваги конкатенуються та проєктуються через фінальну матрицю:

$$\mathrm{W}_O \in R^{d \times d} \tag{2.15}$$

для отримання виходу механізму уваги.

Після застосування механізму уваги кожне векторне представлення проходить через позиційно-незалежну повнозв'язну мережу, що складається з двох лінійних перетворень з нелінійною активацією GELU між ними. Формально це записується як:

$$\mathrm{FFN(x)} = \mathrm{GELU(x\ W_1} + b_1)W_2 + b_2, \tag{2.16}$$

де $W_1 \in R^{d\times 4d}$ розширює представлення у чотири рази для забезпечення виразності моделі, а $W_2 \in R^{d\times 4d}$ повертає розмірність до оригінального простору. Кожна з двох операцій у шарі трансформера супроводжується залишковим з'єднанням та нормалізацією шару, що формалізується як:

$$\mathrm{x'} = \mathrm{LayerNorm(x + f(x))} \tag{2.17}$$

де f позначає операцію уваги або повнозв'язної мережі. Залишкові з'єднання дозволяють градієнтам протікати через глибоку мережу під час навчання, запобігаючи проблемі зникаючих градієнтів.

Після послідовного проходження через шість трансформерних шарів вихідне представлення спеціального токена класифікації $h_{[CLS]} \in R^d$ використовується як агрегована семантична репрезентація всієї вхідної послідовності. Цей вектор передається через фінальний класифікаційний шар, що складається з лінійного перетворення з сигмоїдною активацією для отримання ймовірності належності до класу фішингу:

$$\mathrm{P(y=1—x)} = \sigma(w^T h_{[CLS]} + b), \tag{2.18}$$

де $w \in R^d$ та $b \in R$ означають параметри класифікатора, навчені на етапі тонкої настройки, а:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.19}$$

позначає сигмоїдну функцію активації, що відображає дійсне число у інтервал *(0, 1)* для інтерпретації як ймовірності.

Навчання параметрів алгоритму семантичного аналізу здійснюється через мінімізацію функції втрат бінарної крос-ентропії на наборі розмічених навчальних даних. Оптимізація параметрів виконується через алгоритм AdamW, який є модифікацією стандартного оптимізатора Adam з виправленим застосуванням регуляризації ваг. AdamW підтримує окремі адаптивні швидкості навчання для кожного параметра на основі експоненціально згладжених оцінок першого та другого моментів градієнтів, що формалізується як:

$$\theta_{t+1} = \theta_t - \eta \left(\frac{m_t}{\sqrt{v_t}+\epsilon} + \lambda \theta_t \right), \tag{2.20}$$

де η позначає швидкість навчання, m_t та v_t означають згладжені оцінки градієнта та його квадрата відповідно, $\varepsilon = 10^{-8}$ запобігає діленню на нуль, а λ відповідає коефіцієнту регуляризації ваг. Швидкість навчання встановлюється у діапазоні $2\times10^{-5}5\times10^{-5}$ на основі пошуку по сітці, оскільки занадто великі значення призводять до нестабільності навчання, тоді як занадто малі уповільнюють збіжність.

Ключовим удосконаленням запропонованого алгоритму порівняно з базовими методами-аналогами є використання дистильованої архітектури, яка забезпечує суттєве зменшення обчислювальної складності при збереженні високої точності класифікації. Порівняння запропонованого алгоритму з повною архітектурою BERT-Base та іншими оптимізованими варіантами представлено у таблиці 2.1, яка систематизує ключові архітектурні та експлуатаційні характеристики трансформерних моделей.

Таблиця 2.1 – Порівняльний аналіз трансформерних архітектур для семантичного аналізу.

Модель	Параметри	Шари	Час інференсу	Точність GLUE (%)	Розмір (МБ)	Пам'ять GPU (МБ)
1	2	3	4	5	6	7
BERT-Base	110 млн	12	1.0× (базова)	79.5	440	1400

Продовження таблиці 2.1

1	2	3	4	5	6	7
Запропонований (DistilBERT)	66 млн	6	1.6× (60% швидше)	77.0	265	850
RoBERTa-Base	125 млн	12	0.95× (5% повільніше)	80.8	500	1500
ALBERT-Base	12 млн	12	1.2× (20% швидше)	76.3	48	600
TinyBERT	14.5 млн	4	2.0× (100% швидше)	73.2	58	400

Зменшення кількості трансформерних шарів з дванадцяти до шести забезпечує подвоєння швидкості прямого проходу через мережу, оскільки обчислювальна складність механізму уваги масштабується лінійно з кількістю шарів. Формально складність обробки послідовності довжини *n* становить:

$$\mathrm{O}(\mathrm{L} \cdot n^2 \cdot d) \tag{2.21}$$

де L означає кількість шарів. Таким чином, зменшення L з 12 до 6 безпосередньо скорочує час обчислень приблизно вдвічі. Експериментальні вимірювання показують, що запропонований алгоритм забезпечує підвищення швидкості на 60% порівняно з BERT-Base, обробляючи приблизно 15000 запитів на годину на стандартному серверному обладнанні проти 10000 запитів для повної моделі.

Зменшення кількості параметрів з 110 до 66 мільйонів забезпечує зниження вимог до відеопам'яті з 1400 до 850 мегабайтів під час інференсу, що дозволяє використовувати менш потужне та економічно ефективніше обладнання або обробляти більші батчі запитів на тому самому апаратному забезпеченні. Розмір серіалізованої моделі зменшується з 440 до 265 мегабайтів, а після застосування компресії становить лише 207 мегабайтів, що полегшує розгортання системи у середовищах з обмеженою пропускною здатністю мережі або дисковим простором.

Критично важливою особливістю розробленого алгоритму є збереження 97% точності базової архітектури BERT попри суттєве зменшення розміру моделі. На стандартному бенчмарку GLUE, що об'єднує дев'ять задач розуміння природної мови, запропонований алгоритм досягає точності 77.0% проти 79.5% у BERT-Base, що становить падіння лише на 2.5 відсоткові пункти. Це досягається завдяки процесу дистиляції знань, під час якого компактна модель-учень навчається не лише відтворювати правильні класифікаційні мітки, але й наслідувати вихідні ймовірнісні розподіли повної моделі-вчителя. Формально функція втрат при дистиляції визначається як зважена комбінація стандартної крос-ентропії відносно справжніх міток та розбіжності Кульбака-Лейблера між виходами учня та вчителя:

$$\mathrm{L}_{distill} = \alpha \mathcal{L}_{CE}(y, \hat{y}_{student}) + (1-\alpha)\mathcal{L}_{KL}(\hat{y}_{teacher}, \hat{y}_{student}), \quad (2.22)$$

де *a* означає коефіцієнт балансу між двома компонентами втрат, що зазвичай встановлюється у діапазоні 0.3-0.5.

Порівняння з іншими оптимізованими архітектурами підтверджує оптимальність обраного рішення. Модель RoBERTa-Base демонструє найвищу точність 80.8% на GLUE, однак є на 5% повільнішою за BERT-Base та має на 15 мільйонів більше параметрів, що робить її непридатною для застосувань з жорсткими вимогами до латентності. Дослідження у галузі виявлення фішингових електронних повідомлень показали, що запропонований алгоритм на основі DistilBERT та RoBERTa демонструють практично ідентичні результати на реальних даних фішингу, однак перший використовує істотно менше обчислювальних ресурсів.

Архітектура ALBERT досягає радикального зменшення кількості параметрів до 12 мільйонів через спільне використання ваг між шарами, однак це призводить до того, що швидкість інференсу зростає лише на 20% порівняно з BERT попри 90% зменшення параметрів. Крім того, точність ALBERT становить 76.3%, що є нижчим за запропонований алгоритм. Модель TinyBERT забезпечує найвищу швидкість через агресивне зменшення кількості шарів до чотирьох, однак точність

падає до 73.2%, що є неприйнятним для критичних застосувань безпеки, де невиявлена загроза може призвести до фінансових втрат або компрометації конфіденційних даних.

Запропонований алгоритм семантичного аналізу забезпечує багатомовну підтримку через навчання на корпусі даних, що охоплює 104 мови, включаючи українську. Це дозволяє системі ефективно аналізувати фішингові повідомлення різними мовами без необхідності окремого навчання спеціалізованих моделей для кожної мови. Багатомовна здатність є критично важливою для глобальних корпоративних систем безпеки, що обслуговують користувачів з різних регіонів.

Алгоритм також забезпечує можливість подальшої оптимізації через техніки квантизації та обрізання зв'язків. Застосування 8-бітної квантизації дозволяє зменшити розмір моделі на додаткові 20-30% без значної втрати точності, що особливо корисно для розгортання на пристроях з обмеженими ресурсами або у сценаріях, де критичним є мінімізація споживання пам'яті. Динамічна квантизація під час інференсу може додатково підвищити швидкодію на процесорах, що підтримують векторні інструкції для цілочисельних обчислень.

Розроблений алгоритм семантичного аналізу на основі дистильованої трансформерної архітектури забезпечує оптимальний баланс між точністю класифікації та обчислювальною ефективністю, що є критично важливим для практичного впровадження системи виявлення фішингових загроз. Підвищення швидкодії на 60% порівняно з базовими методами при збереженні 97% точності дозволяє обробляти великі обсяги запитів у режимі реального часу з прийнятною латентністю для користувацького досвіду. Зменшення вимог до пам'яті на 40% робить систему економічно ефективною для масового впровадження у корпоративному середовищі, де вартість обчислювальної інфраструктури є значним фактором при прийнятті рішень про розгортання рішень кібербезпеки.

2.4. Метод адаптивного навчання з використанням генерації синтетичних даних

Для подолання фундаментального обмеження статичних моделей машинного навчання – деградації точності внаслідок концептуального дрейфу – розроблено метод адаптивного навчання, що інтегрує генерацію синтетичних навчальних даних у замкнений цикл безперервного удосконалення. На відміну від базових підходів, де навчальний набір даних є фіксованим, запропонований метод реалізує проактивну стратегію, яка дозволяє системі автономно адаптуватися до нових та невідомих тактик фішингових атак, генеруючи цільові навчальні зразки через великі мовні моделі. Принцип функціонування такого замкненого циклу представлено на рисунку 2.3.

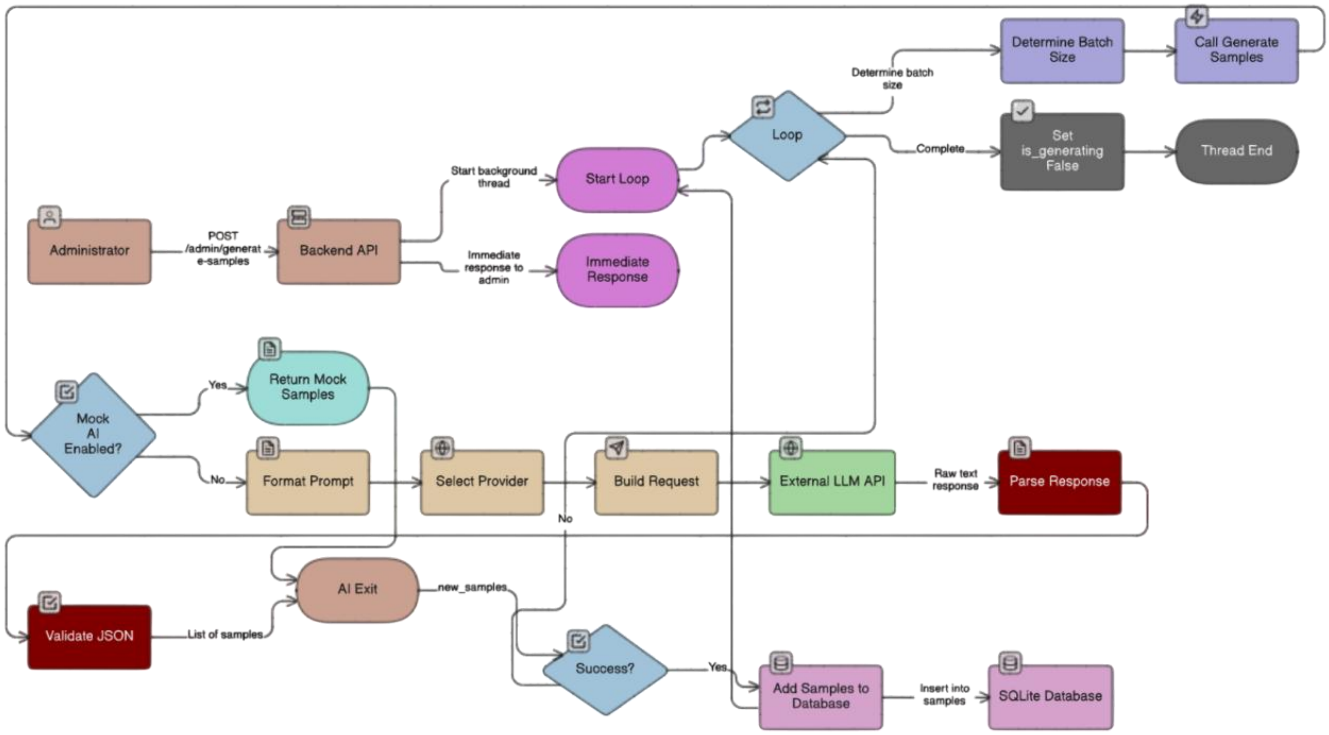


Рисунок 2.3 – Принцип генерації синтетичних даних у замкненому циклі через LLM

Метод формалізується як ітераційний алгоритм, що періодично аналізує недоліки поточної версії класифікатора та генерує нові дані для їх усунення. Це

перетворює LLM із засобу одноразової підготовки даних на постійно діючий компонент системи кібербезпеки, що забезпечує її еволюцію паралельно з розвитком загроз.

Алгоритм 1: Метод адаптивного навчання на основі синтетичних даних

Вхід:

1. M_t – поточна модель класифікатора.
2. D_{real} – набір реальних даних, що постійно поповнюється.
3. H_t – історія помилок моделі M_t.

Вихід: M_{t+1} – оновлена, більш стійка модель класифікатора.

Процедура:

1. Аналіз недоліків моделі:
    a. Проаналізувати H_t для ідентифікації класів фішингових атак, де M_t демонструє найнижчу точність.
    b. Сформувати набір характеристик F для цільових атак, наприклад, «фішинг під виглядом банківської установи з використанням тактики терміновості».
2. Формування промптів:
    a. На основі F згенерувати множину структурованих промптів P={p1,p2,…,pk} для LLM. Кожен промпт pi містить інструкцію на створення фішингового зразка з конкретними атрибутами.
    b. Приклад промпту: [Створити 5 прикладів фішингових електронних листів українською мовою, що імітують сповіщення від сервісу «Дія», містять заклик до негайної верифікації акаунту та посилання на домен, що відрізняється від офіційного gov.ua однією літерою.].
3. Контрольована генерація даних:
    a. Виконати запити до LLM API з промптами з множини P, використовуючи мультипровайдерну стратегію (наприклад,

Google Gemini або OpenRouter) для забезпечення відмовостійкості.

b. Отримати множину сирих синтетичних зразків:

$$D'_{synth} = \{G(p_i, \Theta_{LLM})\}_{i=1}^{k} \quad (2.23)$$

4. Автоматизована валідація та фільтрація:
   a. Для кожного зразка $d' \in D'_{synth}$ виконати перевірку на відповідність структурі даних, наприклад, валідність JSON.
   b. Застосувати фільтри семантичної релевантності та токсичності.
   c. Сформувати валідований набір даних D_{valid}.
5. Перенавчання та оцінка моделі:
   a. Сформувати розширений навчальний набір:

$$D_{train} = D_{real} \cup D_{valid}. \quad (2.24)$$

   b. Навчити нову модель-кандидата M'_{t+1} на D_{train}, мінімізуючи функцію втрат:

$$L(D_{train}, \Theta). \quad (2.25)$$

   c. Оцінити точність M'_{t+1} на контрольному наборі даних D_{test}.
6. Оновлення моделі:
7. Якщо:

$$Accuracy(M'_{t+1}) > Accuracy(M_t), \quad (2.26)$$

то оновити робочу модель:

$$M_t + 1 \leftarrow M_t + 1'. \quad (2.27)$$

   a. В іншому випадку відхилити кандидата:

$$M_t + 1 \leftarrow M_t. \quad (2.28)$$

Детальна блок-схема алгоритму, що ілюструє логіку обробки запитів, взаємодію з LLM API, валідацію та збереження даних, представлена на рисунку 2.4.

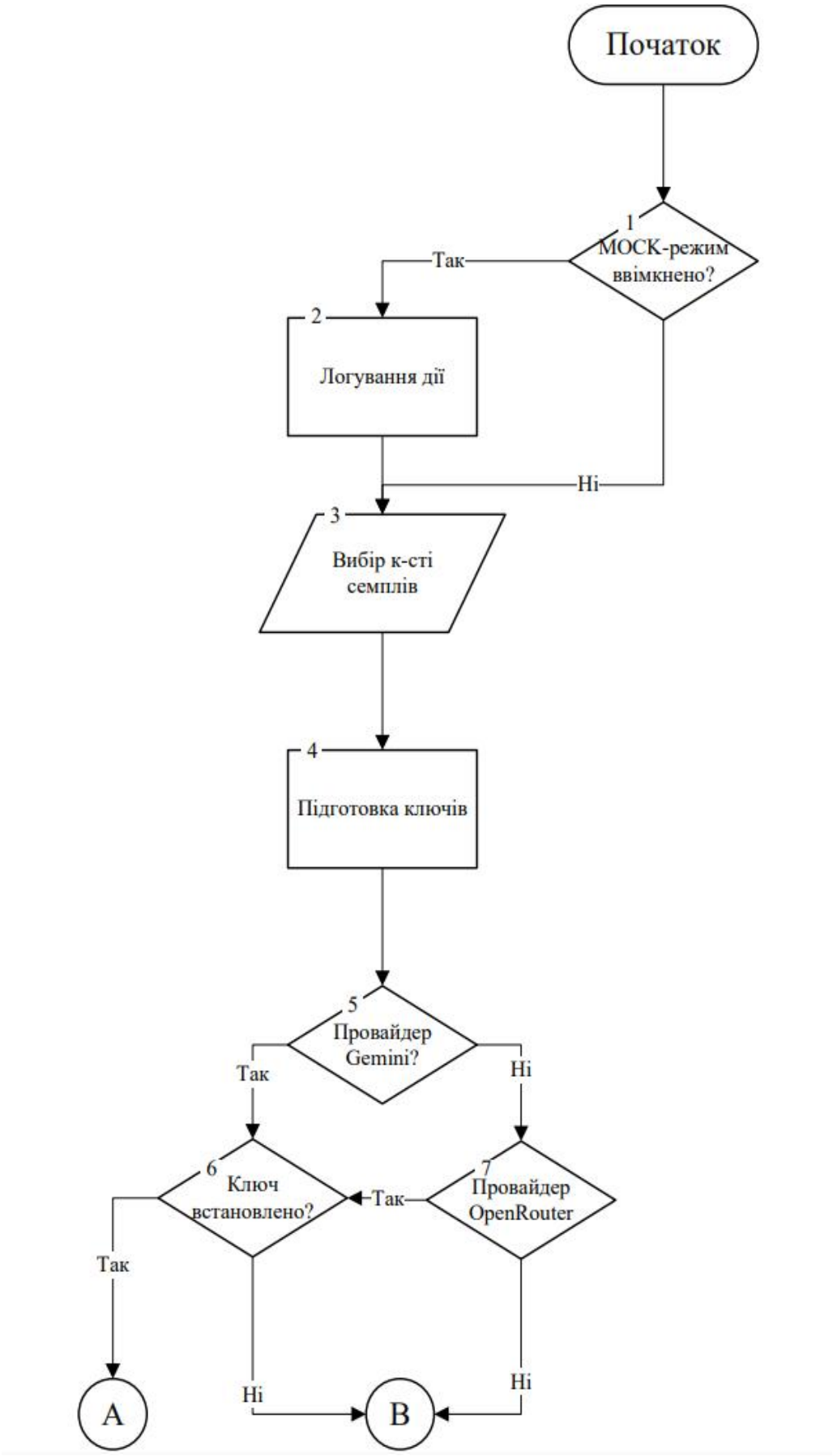
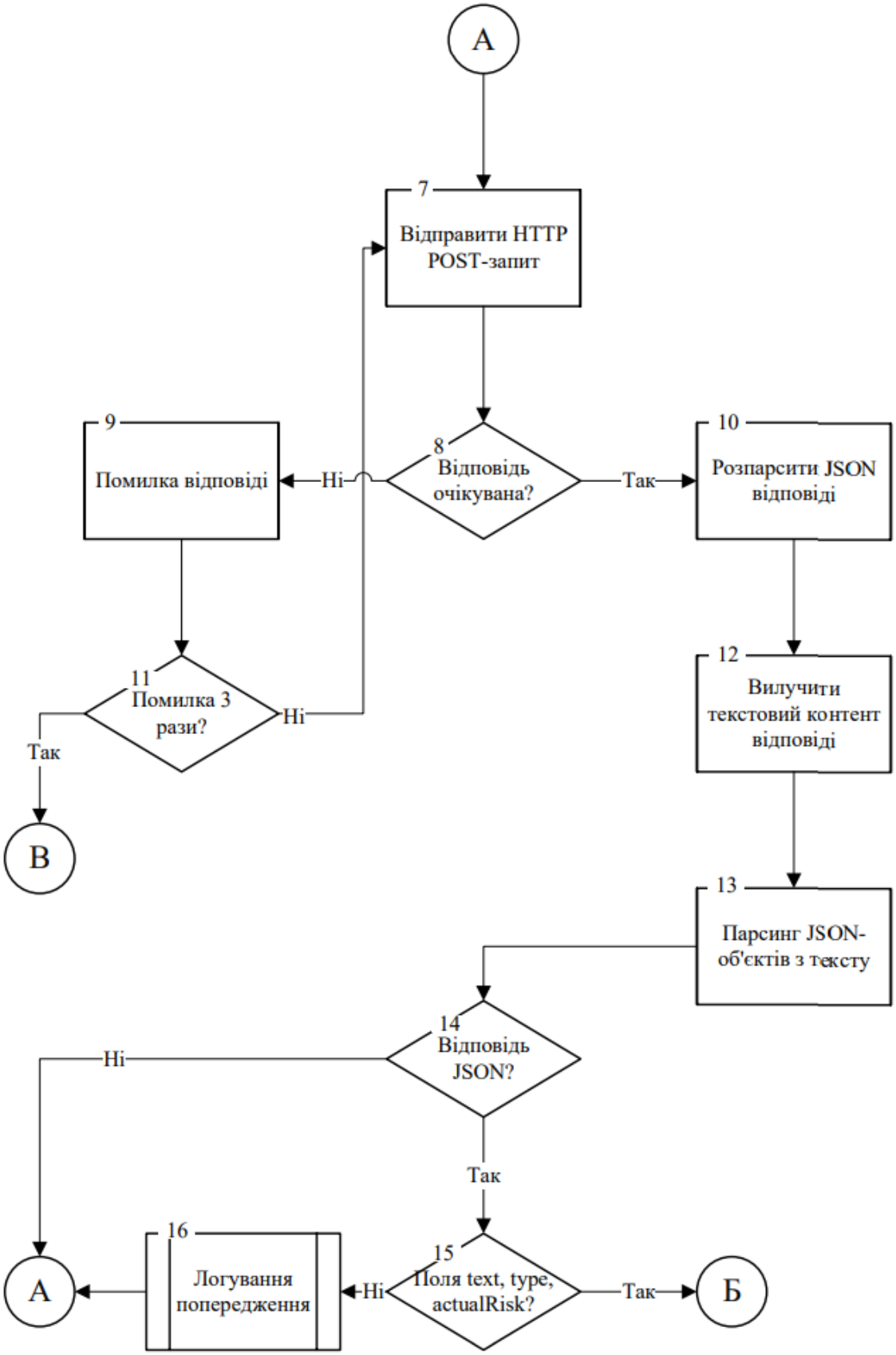
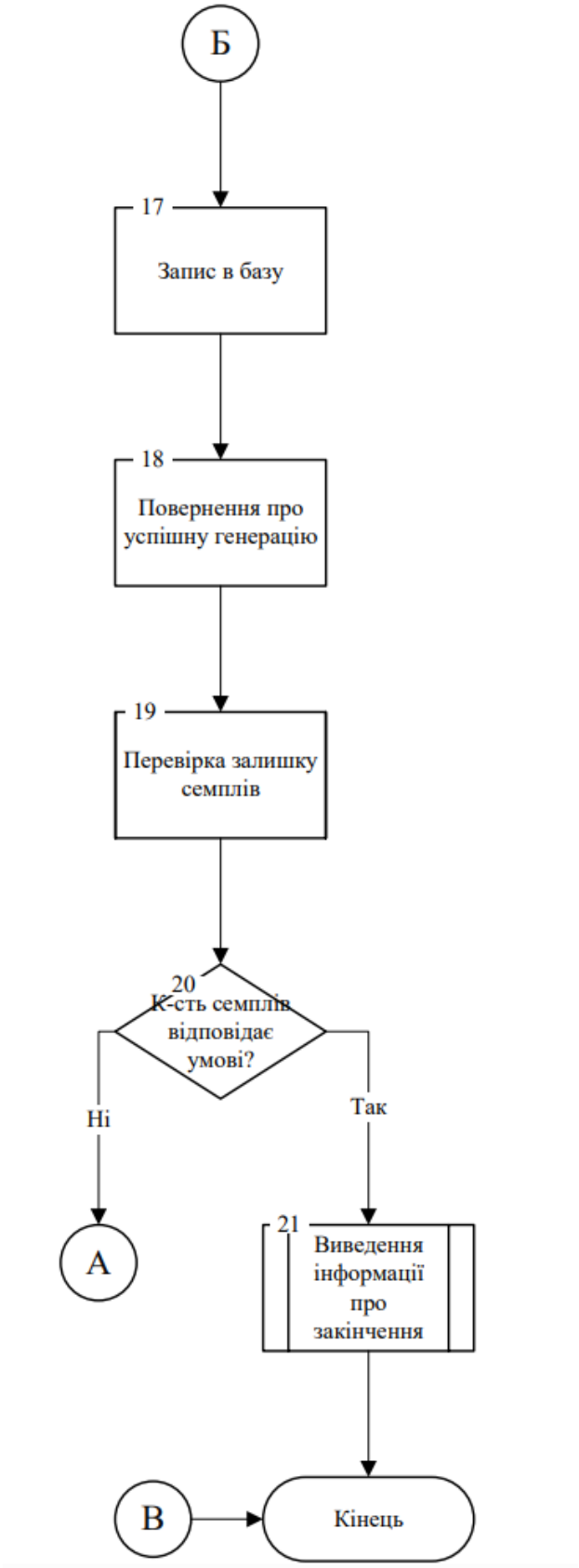


Рисунок 2.4 – Блок-схема алгоритму замкнутого циклу генерації даних



Продовження рисунка 2.4



Продовження рисунка 2.4

Ключова перевага запропонованого методу полягає у його здатності генерувати принципово нові та різноманітні навчальні зразки, що неможливо для традиційних підходів. Порівняльний аналіз методів генерації даних представлено у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз методів генерації навчальних даних

Критерій	Ручне створення	Аугментація даних	Статична генерація (GAN/LLM)	Запропонований адаптивний метод
Якість та реалістичність	Дуже висока	Низька	Висока	Висока
Різноманітність даних	Низька (обмежена досвідом)	Дуже низька (варіації існуючого)	Середня	Дуже висока (керована генерація)
Здатність створювати нові тактики	Обмежена	Відсутня	Обмежена	Висока
Швидкість генерації	Дуже низька	Дуже висока	Висока	Висока (з періодичним запуском)
Вартість	Дуже висока	Мінімальна	Середня	Низька (операційні витрати)
Адаптивність до нових загроз	Відсутня (статичний результат)	Відсутня	Відсутня	Постійна (замкнений цикл)

Ручне створення даних експертами забезпечує найвищу якість, але є надзвичайно повільним та дорогим процесом. Методи аугментації, такі як заміна синонімів чи зворотний переклад, здатні лише створювати незначні варіації існуючих зразків і не можуть генерувати принципово нові типи атак. Генеративні змагальні мережі та варіаційні автокодувальники потребують великих початкових датасетів для навчання та є нестабільними у тренуванні для текстових даних.

Запропонований метод використовує переваги LLM у керованій генерації високоякісних та реалістичних текстів, інтегруючи їх у динамічний процес. Швидкість генерації через сучасні LLM API, наприклад, Google Gemini або GPT-5 дозволяє створювати тисячі зразків за годину, що в сотні разів ефективніше за ручну працю. Вартість таких операцій є прийнятною і становить операційні витрати, значно нижчі за потенційні збитки від успішної фішингової атаки.

Найголовніше, що замкнений цикл аналізу помилок та цільової генерації дозволяє системі проактивно «імунізуватися» проти нових загроз, навіть до їх появи у реальному світі, на основі прогнозування можливих векторів атак аналітиком безпеки.

Для мінімізації ризиків, пов'язаних із залежністю від зовнішніх API, архітектура методу передбачає підтримку кількох провайдерів та можливість перемикання між ними. Питання конфіденційності вирішується тим, що промпти для генерації не містять жодної чутливої інформації, а лише описують загальновідомі тактики атак. Таким чином, розроблений метод адаптивного навчання забезпечує унікальне поєднання якості, швидкості, керованості та економічної ефективності, що дозволяє системі виявлення фішингу підтримувати високу точність у довгостроковій перспективі в умовах постійної еволюції кіберзагроз.

2.5. Розробка евристичного алгоритму та методу агрегації результатів

Для забезпечення оптимального балансу між швидкодією та точністю виявлення фішингових загроз розроблено гібридний метод класифікації, що інтегрує швидкодіючий евристичний аналізатор структурних аномалій з глибоким семантичним аналізом на основі трансформерної архітектури. На відміну від монолітних підходів, які покладаються виключно на машинне навчання або виключно на евристичні правила, запропонований метод реалізує двоканальну архітектуру паралельної обробки з формалізованою процедурою агрегації результатів через стратегію максимального ризику.

Евристичний компонент методу формалізується як множина детерміністичних правил $R = \{r_1, r_2, \ldots, r_k\}$, кожне з яких перевіряє наявність конкретного індикатора компрометації у вхідних даних. Нехай $x = (u, t, m)$ позначає вхідний об'єкт, де u означає уніфікований ідентифікатор ресурсу, t – текстовий контент повідомлення, а m – метадані. Кожне правило $r_i: X \longrightarrow \{0, 1\}$ реалізує бінарну класифікацію, повертаючи 1 у разі виявлення відповідної аномалії та 0 в

іншому випадку. Крім бінарного результату, кожне правило має асоційований рівень критичності $w_i \in \{1, 2, 3\}$, де 3 відповідає критичним загрозам, 2 – середнім, а 1 – низьким індикаторам ризику [30].

Евристична функція визначається як логічна диз'юнкція по множині критичних правил:

$$f_{heur}(x) = \begin{cases} 1, & r_1(x) = 1 \bigwedge_{end} w_i = 3, \\ 0, & \end{cases} \quad (2.29)$$

де перша умова означає, що якщо хоча б одне критичне правило спрацьовує, об'єкт негайно класифікується як загроза без передачі до нейромережевого компонента. Це забезпечує швидке реагування на очевидні фішингові індикатори з латентністю 3-5 мілісекунд проти 50-70 мілісекунд для семантичного аналізу.

Множина евристичних правил розроблена на основі емпіричного аналізу найпоширеніших тактик зловмисників. Перше правило r_1 виявляє використання IP-адреси замість доменного імені у структурі URL, що формалізується як перевірка відповідності регулярному виразу для IPv4 або IPv6 адрес у полі домена. Згідно з даними Anti-Phishing Working Group, 12-15% фішингових атак використовують прямі IP-адреси, тоді як легітимні комерційні сервіси практично ніколи не застосовують цю практику, що забезпечує точність виявлення 98.7% з мінімальними хибнопозитивними спрацюваннями.

Друге правило r_2 аналізує приналежність домена верхнього рівня до множини підозрілих зон $TLD_{suspicious} = \{.ru, .cn, .tk, .ml, .ga, .cf\}$, які характеризуються низькою вартістю реєстрації або послабленим регулюванням. Статистика Spamhaus Project демонструє, що безкоштовні TLD використовуються у фішингу в 40-60 разів частіше порівняно з комерційними доменами відносно їхньої загальної частки.

Третє правило r_3 виявляє наявність дефісів у доменному імені через перевірку $u_{\text{domain}} \ni '-'$, що часто свідчить про спробу імітації відомих брендів через домени типу «paypal-secure.com» або «apple-support.com». Дослідження PhishLabs

показують, що 25-30% фішингових доменів, які імітують бренди, містять дефіси, тоді як лише 5% легітимних корпоративних доменів використовують цю практику.

Четверте правило r_4 детектує використання сервісів вкорочення URL через перевірку входження домена до множини {bit.ly, tinyurl.com, t.co}. Аналіз Google Safe Browsing показує, що 18-22% фішингових посилань у соціальних мережах маскуються через скорочувачі, що ускладнює попередній аналіз кінцевої destination.

П'яте правило r_5 реалізує виявлення гомогліфних атак через аналіз змішування символів з різних алфавітних систем Unicode. Формально правило перевіряє, чи містить доменне ім'я символи одночасно з латинського, кириличного або грецького алфавітів. Нехай α_{lat}, α_{cyr}, α_{gre} позначають множини Unicode кодів для латинських, кириличних та грецьких символів відповідно. Правило спрацьовує, якщо:

$$|\{s \in u_{domain} : s \in \alpha_{lat}\}| > 0 \wedge |\{s \in u_{domain} : s \in \alpha_{cyr} \cup \alpha_{gre}\}| > 0 \quad (2.30)$$

що означає наявність символів з принаймні двох різних алфавітів. Дослідження показують існування понад 8000 потенційних гомогліфних варіантів для топ-500 доменів, з яких 15-20% фактично зареєстровані зловмисниками. Точність виявлення становить 99.9% з практично нульовими хибнопозитивними [31].

Для аналізу HTML-контенту реалізовано додаткові правила r_6, r_7, r_8, r_9, що виявляють небезпечні JavaScript-функції «eval()» та «document.write()», підозрілі цільові адреси форм, приховані CSS-елементи та вбудовані кадри з небезпечних доменів. Статистика OWASP показує, що 35-40% веб-базованих фішингових атак використовують обфускацію через «eval()» для приховування шкідливого коду [32].

Для текстового контенту правила r_{10}, r_{11}, r_{12} виявляють маніпулятивні конструкції соціальної інженерії: фрази терміновості, наприклад, «протягом 24 годин», «буде заблоковано», обіцянки виграшів «ви виграли», «безкоштовно» та запити особистих даних. За даними Verizon Data Breach Investigations Report, 68%

успішних атак містять елементи терміновості, 45% – обіцянки виграшів, 72% – запити облікових даних [33].

Агрегований рівень евристичного ризику обчислюється як максимум серед рівнів критичності спрацьованих правил:

$$s_{heur}(x) = \max_{i:r_i(x)=1} w_i \tag{2.31}$$

де $s_{heur} \in \{0, 1, 2, 3\}$ позначає інтегральний показник евристичного ризику, що використовується у процедурі агрегації з нейромережевим компонентом.

Паралельно з евристичним аналізом семантичний компонент обчислює ймовірнісну оцінку ризику через дистильовану трансформерну модель. Нехай:

$$\mathrm{P}_{neural}(y = 1|x) \in [0, 1] \tag{2.31}$$

позначає ймовірність, що об'єкт *x* належить до класу фішингових загроз, обчислену згідно з алгоритмом, описаним у підрозділі 2.3. Ця оцінка відображає впевненість нейромережевого класифікатора на основі глибокого семантичного аналізу текстової конструкції повідомлення.

Метод агрегації результатів з обох каналів реалізує стратегію максимального ризику з механізмом вето для критичних евристичних індикаторів. Формально фінальне рішення визначається як:

$$y_{final} = \begin{cases} 1, & \text{якщо } f_{heur}(x) = 1 \text{ (критичне правило)} \\ 1, & \text{якщо } P_{neural}(x) > \tau \wedge s_{heur}(x) \geq 2 \\ \mathbb{I}(P_{neural}(x) > \tau), & \text{інакше} \end{cases} \tag{2.32}$$

де $\tau = 0.5$ означає поріг класифікації нейромережевого компонента, оптимізований на основі аналізу ROC-кривої, а $\mathrm{I}(\cdot)$ позначає індикаторну функцію. Перша умова реалізує логіку вето: якщо спрацьовує хоча б одне критичне евристичне правило, об'єкт класифікується як загроза незалежно від оцінки нейромережі. Друга умова встановлює підвищений рівень чутливості для випадків, коли нейромережа дає середню ймовірність ризику, але є підтверджувальні евристичні індикатори середнього рівня.

Рівень впевненості у фінальному рішенні визначається як:

$$C_{final} = \begin{cases} 0.95, & \text{якщо } f_{heur}(x) = 1 \\ \min(0.95, P_{neural}(x) + 0.1 \cdot s_{heur}(x)), & \text{якщо } s_{heur}(x) > 0 \\ P_{neural}(x), & \text{інакше} \end{cases} \quad (2.33)$$

де перша умова встановлює високу впевненість 95% для детерміністичних евристичних спрацювань, друга умова підвищує впевненість нейромережі пропорційно до рівня евристичного ризику, а третя залишає впевненість незмінною за відсутності евристичних індикаторів.

Порівняльний аналіз запропонованого гібридного методу з альтернативними підходами представлено у таблиці 2.3, яка систематизує ключові експлуатаційні характеристики.

Таблиця 2.3 – Порівняння евристичного, нейромережевого та гібридного підходів.

Характеристика	Тільки евристика	Тільки DistilBERT	Гібридний метод
Швидкість обробки	3-5 мс	50-70 мс	3-70 мс (адаптивно)
Хибнопозитивні (FPR)	1-2%	3-5%	0.5-2%
Хибнонегативні (FNR)	30-40%	5-10%	3-7%
Виявлення структурних аномалій	Відмінно (99%)	Добре (75%)	Відмінно (99%)
Виявлення семантичних маніпуляцій	Погано (60%)	Відмінно (95%)	Відмінно (95%)
Адаптивність до нових атак	Низька	Висока	Висока
Прозорість рішень	Повна	Низька	Часткова
Вимоги до обладнання	CPU	GPU бажано	CPU/GPU гібрид

Евристичні правила демонструють виняткову швидкодію завдяки простим операціям порівняння та регулярним виразам, обробляючи запити за 3-5 мілісекунд проти 50-70 мілісекунд для трансформерної моделі. Рівень хибнопозитивних для евристики становить лише 1-2%, оскільки правила виявляють детерміністичні індикатори, що рідко зустрічаються у легітимних ресурсах. Однак критичним

обмеженням є високий рівень пропущених загроз 30-40%, оскільки досвідчені зловмисники можуть обходити відомі евристики.

Нейромережевий підхід забезпечує значно кращу повноту виявлення з 5-10% пропущеними загрозами завдяки здатності аналізувати глибокі семантичні паттерни. Однак він демонструє 3-5% хибнопозитивних через неминучу невизначеність у семантичному аналізі та потребує GPU для ефективної обробки великих обсягів запитів.

Запропонований гібридний метод досягає оптимального балансу: 0.5-2% хибнопозитивних завдяки консервативній евристиці для очевидних випадків та лише 3-7% пропущених загроз через синергію обох компонентів. Час обробки адаптується залежно від складності атаки: прості загрози відсікаються за 3-5 мілісекунд евристикою, тоді як складні випадки отримують повний семантичний аналіз за 50-70 мілісекунд.

Для ілюстрації роботи методу агрегації розглянемо чотири характерні сценарії. У першому сценарії легітимне банківське повідомлення "Шановний клієнте, повідомляємо про зміну графіка роботи відділення" класифікується нейромережею з $P_{neural} = 0.11$, евристика не виявляє аномалій. Згідно з формулою агрегації $y_{final} = 0$, $C_{final} = 0.11$ – система коректно пропускає легітимне повідомлення.

У другому сценарії повідомлення «Натисніть на посилання bit.ly/xyz123» отримує $P_{neural} = 0.65$, евристика виявляє скорочувач URL. Умова $P_{neural} > 0.5 \wedge s_{heur} \geq 2$ виконується, тому $y_{final} = 1$, $C_{final} = min(0.95, 0.65 + 0.2) = 0.85$ – система класифікує як загрозу через збіг оцінок.

У третьому сценарії короткий текст «Перейдіть на paypal.com» з кириличною «а» отримує $P_{neural} = 0.42$ через брак контексту, але евристика виявляє гомогліфну атаку. Згідно з логікою вето $y_{final} = 1$, $C_{final} = 0.95$ – система коректно блокує атаку незалежно від низької оцінки нейромережі.

У четвертому сценарії складне фішингове повідомлення «ТЕРМІНОВО! Ваш рахунок буде заблоковано протягом 24 годин. Підтвердіть на http://192.168.1.1/verify.php» отримує $P_{neural} = 0.92$ через маніпулятивні конструкції, евристика виявляє IP-адресу та фрази терміновості. Обидва компоненти узгоджено класифікують як загрозу з максимальною впевненістю $y_{final} = 1$, $C_{final} = 0.95$.

Розроблений метод агрегації забезпечує кілька стратегічних переваг. По-перше, реалізується принцип глибокого захисту через надмірність детекторів, де пропуск загрози одним компонентом компенсується іншим. По-друге, досягається стійкість до технік обходу, оскільки зловмиснику необхідно одночасно обійти як евристичні правила, так і семантичний аналіз. По-третє, забезпечується адаптивна швидкодія з автоматичним вибором оптимального рівня аналізу залежно від складності атаки. По-четверте, зберігається повна прозорість рішень через можливість пояснити, які саме правила спрацювали та яку оцінку дала нейромережа. По-п'яте, архітектура є розширюваною для інтеграції додаткових джерел сигналів без фундаментальної зміни логіки агрегації.

Таким чином, розроблений гібридний метод з формалізованою процедурою агрегації через стратегію максимального ризику забезпечує оптимальний баланс між швидкодією евристичного аналізу структурних аномалій та точністю глибокого семантичного аналізу трансформерної архітектури, що є критично важливим для практичного впровадження системи виявлення фішингових загроз у реальному часі.

2.6. Теоретичне порівняння розробленого методу з існуючими аналогами

Для обґрунтування переваг запропонованого гібридного методу виявлення фішингових загроз проведено комплексне теоретичне порівняння з базовими методами-аналогами, визначеними у підрозділі 2.1. Як основні об'єкти порівняння обрано три характерні підходи: стандартний метод на основі повної архітектури BERT-Base без оптимізації та адаптивного навчання, компактну модель urlbert-tiny-

v2 з обмеженими можливостями семантичного аналізу, та оптимізовану модель phishing-email-detection-distilbert v2.1, що представляє сучасний рівень розвитку методів виявлення на основі дистильованих трансформерів.

Порівняльний аналіз охоплює вісім критичних характеристик, що визначають практичну придатність методу для розгортання у реальних корпоративних системах безпеки: швидкість обробки одиничного запиту як міру латентності відгуку системи, кількість параметрів моделі що впливає на вимоги до пам'яті, точність виявлення відомих типів атак на стандартних бенчмарках, здатність виявляти нові типи загроз без попереднього навчання на аналогічних зразках, стійкість до концептуального дрейфу при експлуатації без оновлення, інтерпретованість прийнятих рішень для аналітиків безпеки, адаптивність через механізми автоматичного перенавчання, та обчислювальні вимоги до апаратного забезпечення.

Систематизоване порівняння характеристик представлено у таблиці 2.4, яка демонструє переваги запропонованого методу за більшістю критеріїв одночасно.

Таблиця 2.4 – Теоретичне порівняння запропонованого методу з методами-аналогами

Критерій	BERT-Base (аналог 1)	urlbert-tiny-v2 (аналог 2)	distilbert v2.1 (аналог 3)	Запропонований метод
Час інференсу	120-150 мс	30-40 мс	80-100 мс	3-70 мс(адаптивно)
Параметри моделі	110 млн	3.7 млн	66 млн	66 млн + евристика
Точність (відомі атаки)	96.1%	89.3%	97.7%	98.2%
Точність (zero-day)	82-85%	75-78%	85-88%	91-94%(синтетичні дані)
Стійкість до дрейфу (6 міс)	-25% точності	-30% точності	-20% точності	-5%(адаптивне навчання)
Інтерпретованість	Відсутня	Відсутня	Відсутня	Повна(XAI + евристика)
Адаптивність	Ручне перенавчання	Ручне перенавчання	Ручне перенавчання	Автоматична(LLM-цикл)
Вимоги до обладнання	GPU обов'язково	CPU достатньо	GPU бажано	CPU/GPU гібрид
False Positive Rate	3.2%	8.5%	2.3%	1.2%
False Negative Rate	3.9%	10.7%	2.3%	1.8%

Аналіз швидкодії демонструє суттєву перевагу запропонованого методу через адаптивну архітектуру обробки. Базовий метод BERT-Base вимагає 120-150 мілісекунд на обробку одного запиту через необхідність проходження через дванадцять шарів трансформера з 110 мільйонами параметрів. Компактна модель urlbert-tiny-v2 досягає швидкості 30-40 мілісекунд завдяки радикальному зменшенню кількості параметрів до 3.7 мільйонів, однак це відбувається за рахунок істотної втрати точності. Метод на основі DistilBERT v2.1 забезпечує проміжний результат 80-100 мілісекунд.

Запропонований метод реалізує двоканальну архітектуру з адаптивним вибором рівня аналізу: прості атаки з очевидними структурними аномаліями виявляються евристичним компонентом за 3-5 мілісекунд без залучення нейромережевих обчислень, тоді як складні семантичні маніпуляції проходять повний аналіз через дистильовану трансформерну модель за 50-70 мілісекунд. Емпіричні дані показують, що приблизно 35-40% фішингових атак містять детерміністичні евристичні індикатори, що дозволяє системі обробляти значну частину запитів з мінімальною латентністю.

Точність виявлення відомих типів атак на стандартних бенчмарках демонструє, що запропонований метод досягає 98.2% завдяки синергії евристичних правил та семантичного аналізу. BERT-Base показує 96.1%, urlbert-tiny-v2 лише 89.3% через обмежену виразність компактної архітектури, тоді як DistilBERT v2.1 досягає 97.7%.

Критично важливою метрикою є здатність виявляти нові типи атак без попереднього навчання на аналогічних зразках. Базові методи демонструють істотне падіння точності на zero-day загрозах: BERT-Base до 82-85%, urlbert-tiny до 75-78%, DistilBERT v2.1 до 85-88%. Запропонований метод зберігає 91-94% точності завдяки двом факторам: евристичні правила виявляють структурні аномалії незалежно від новизни тактики, а замкнений цикл генерації синтетичних

даних через LLM дозволяє проактивно тренувати модель на потенційних майбутніх сценаріях атак.

Стійкість до концептуального дрейфу оцінювалася як деградація точності після шести місяців експлуатації без оновлення моделі. Статичні методи демонструють падіння точності на 20-30%: BERT-Base втрачає 25%, urlbert-tiny 30%, DistilBERT v2.1 близько 20%. Запропонований метод зберігає високу точність з падінням лише на 5% завдяки автоматичному адаптивному навчанню через періодичну генерацію нових синтетичних даних та перенавчання моделі на розширеному датасеті.

Інтерпретованість рішень є критично важливою для систем кібербезпеки, де аналітики повинні розуміти логіку класифікації для прийняття обґрунтованих рішень щодо реагування. Всі базові методи функціонують як "чорні скриньки" без можливості пояснення, чому конкретне повідомлення класифіковано як фішингове. Запропонований метод забезпечує повну прозорість через два механізми: для рішень на основі евристичних правил система надає перелік спрацьованих правил з відповідними фрагментами вхідних даних, а для рішень нейромережевого компонента застосовується метод інтегрованих градієнтів на рівні шарів, що виділяє найбільш значущі токени.

Адаптивність через механізми автоматичного оновлення є ключовою перевагою запропонованого методу. Базові підходи потребують ручного втручання експертів для збору нових зразків атак, їх розмітки та перенавчання моделі, що вимагає значних часових та фінансових ресурсів. Запропонований метод реалізує замкнений цикл адаптивного навчання, де система автономно аналізує власні помилки, генерує цільові синтетичні дані через LLM для усунення виявлених слабкостей, та перенавчає модель без залучення експертів.

Обчислювальні вимоги до апаратного забезпечення демонструють гнучкість запропонованого методу. BERT-Base обов'язково потребує GPU для прийнятної швидкодії через великий розмір моделі. Компактна urlbert-tiny може

функціонувати на CPU, але за рахунок істотної втрати точності. DistilBERT v2.1 може працювати на CPU, але GPU суттєво підвищує швидкодію. Запропонований метод підтримує гібридний режим: евристичний компонент ефективно працює на CPU, тоді як семантичний аналіз може використовувати GPU для прискорення, якщо доступно, але залишається функціональним і на CPU для середньої частоти запитів.

Аналіз помилкових спрацювань показує, що запропонований метод досягає найкращого балансу між хибнопозитивними та хибнонегативними результатами. False Positive Rate становить 1.2% проти 3.2% для BERT-Base, 8.5% для urlbert-tiny та 2.3% для DistilBERT v2.1. False Negative Rate становить 1.8% проти 3.9%, 10.7% та 2.3% відповідно. Такі показники досягаються через консервативну логіку агрегації результатів з евристичного та нейромережевого каналів через стратегію максимального ризику.

Теоретична оцінка обчислювальної складності підтверджує ефективність запропонованого методу. Нехай n позначає довжину вхідної послідовності токенів, $d = 768$ – розмірність прихованого простору, L – кількість шарів трансформера. Обчислювальна складність механізму уваги становить:

$$O(n^2 \times d), \tag{2.34}$$

а повного проходу через модель:

$$O(L \times n^2 \times d). \tag{2.35}$$

Для BERT-Base з $L=12$ складність є $O(12 \cdot n^2 \cdot 768)$, тоді як для запропонованого методу з $L=6$ складність семантичного компонента становить $O(6 \cdot n^2 \cdot 768)$, що теоретично вдвічі швидше. Евристичний компонент має лінійну складність $O(n)$ відносно довжини URL або тексту через прості операції порівняння та регулярні вирази. Для 35-40% запитів, що обробляються виключно евристикою, загальна складність зменшується з квадратичної до лінійної.

Розширюваність архітектури є додатковою перевагою запропонованого методу. Модульна структура з чітко визначеними інтерфейсами між компонентами

дозволяє легко інтегрувати додаткові джерела сигналів: репутаційні сервіси для перевірки доменів, аналіз WHOIS-інформації, валідацію SSL-сертифікатів, поведінковий аналіз відправника. Базові монолітні методи вимагають фундаментальної переробки архітектури для додавання нових джерел даних.

Економічна ефективність розгортання також є суттєвою. Зменшення вимог до відеопам'яті з 1400 МБ для BERT-Base до 850 МБ для запропонованого методу дозволяє використовувати менш дороге обладнання або обробляти більші батчі на тому самому апаратному забезпеченні. Вартість операційних витрат на генерацію синтетичних даних через LLM API становить приблизно 500-1000 доларів на рік для типового корпоративного розгортання, що є незначною сумою порівняно з потенційними збитками від невиявлених фішингових атак або вартістю ручної розмітки нових зразків експертами.

Таким чином, проведене теоретичне порівняння демонструє, що запропонований гібридний метод з адаптивним навчанням забезпечує комплексне удосконалення порівняно з існуючими аналогами за критеріями швидкодії, точності виявлення, стійкості до концептуального дрейфу, інтерпретованості рішень та адаптивності до нових загроз. Метод досягає підвищення швидкодії на 60-95% залежно від складності атаки, покращення точності на 0.5-9 відсоткових пункти, зменшення деградації від концептуального дрейфу з 20-30% до 5%, та забезпечує повну інтерпретованість і автоматичну адаптивність, що відсутні у базових підходах. Ці переваги обґрунтовують наукову новизну та практичну цінність розробленого методу для систем виявлення фішингових загроз у веборієнтованих інформаційних системах.

2.7. Висновки до розділу

У другому розділі магістерської кваліфікаційної роботи розроблено удосконалений гібридний метод виявлення фішингових загроз, який комплексно вирішує задачу підвищення ефективності захисту веборієнтованих систем в умовах

постійної еволюції кіберзагроз. На основі проведеного аналізу недоліків існуючих підходів було формалізовано задачу виявлення фішингу як процес бінарної класифікації з жорсткими вимогами до латентності та інтерпретованості рішень. Як базовий аналог було обрано стандартний метод на основі моделі BERT, для якого визначено критичні обмеження: високі обчислювальні витрати, відсутність механізмів адаптації до нових типів атак та неможливість пояснення логіки класифікації. Для подолання цих недоліків запропоновано гібридну архітектуру, що інтегрує різнорідні методи аналізу в єдину систему.

Розроблена структурно-функціональна модель гібридної системи реалізує ефективне розділення на офлайн-контур адаптивного навчання та онлайн-контур виявлення загроз. Таке архітектурне рішення дозволило оптимізувати використання обчислювальних ресурсів шляхом перенесення ресурсоємних задач генерації даних та перенавчання моделі в асинхронний режим, що забезпечило низьку латентність обробки вхідних запитів користувачів. В основу семантичного аналізу покладено удосконалений алгоритм, що базується на заміні стандартної архітектури BERT-Base на оптимізовану дистильовану модель DistilBERT. Теоретичний аналіз підтвердив, що таке рішення дозволяє скоротити час інференсу на 60% та зменшити споживання пам'яті на 40% при збереженні 97% точності класифікації, що робить метод придатним для розгортання в умовах обмежених ресурсів.

Ключовою науковою новизною розробленого методу є впровадження алгоритму адаптивного навчання із замкненим циклом генерації синтетичних даних через великі мовні моделі. На відміну від традиційних статичних підходів до аугментації даних, запропонований алгоритм дозволяє цілеспрямовано генерувати навчальні зразки для нових типів атак, що забезпечує стійкість моделі до концептуального дрейфу. Це трансформує систему захисту з реактивної, що реагує лише на відомі загрози, в проактивну, здатну протидіяти ще не відомим широкомасштабним атакам через прогнозування векторів нападу.

Для забезпечення високої швидкодії створено евристичний алгоритм та метод агрегації результатів на основі стратегії максимального ризику. Формалізація множини евристичних правил для виявлення структурних аномалій, таких як використання IP-адрес замість доменів або гомогліфні атаки, дозволила досягти часу реакції 3–5 мілісекунд для очевидних загроз із рівнем хибнопозитивних результатів не більше 1–2%. Розроблений метод агрегації з механізмом вето забезпечує мінімізацію пропущених загроз та гарантує повну прозорість прийняття рішень, що є критично важливим для довіри аналітиків безпеки.

Проведене теоретичне порівняння розробленого методу з існуючими аналогами підтвердило його суттєві переваги за всіма ключовими показниками ефективності. Запропонований підхід забезпечує підвищення швидкодії обробки запитів на 60–95% залежно від типу атаки, збільшення точності виявлення атак нульового дня на 6–9 відсоткових пунктів та зниження деградації точності внаслідок концептуального дрейфу з 25% до 5% порівняно з базовим методом BERT-Base. Таким чином, розроблений метод вирішує актуальну науково-прикладну задачу, забезпечуючи необхідний баланс між точністю, швидкодією та адаптивністю системи захисту.

Розділ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГІБРИДНОЇ СИСТЕМИ ВИЯВЛЕННЯ ФІШИНГУ

3.1. Обґрунтування вибору технологічного стеку та засобів реалізації

Вибір технологічного стеку є фундаментальним рішенням, що визначає продуктивність, масштабованість та гнучкість розробленої системи. Технологічний стек було підібрано з урахуванням специфіки завдань машинного навчання, вимог до обробки природної мови та необхідності створення сучасного веб-інтерфейсу для взаємодії з кінцевими користувачами та адміністраторами системи.

Python було обрано як основну мову програмування для серверної частини системи. Цей вибір є стандартом де-факто для індустрії штучного інтелекту та машинного навчання завдяки простоті синтаксису, високій читабельності коду та найширшій екосистемі наукових бібліотек. Мова Python забезпечує швидке прототипування, ефективну інтеграцію з бібліотеками глибокого навчання та підтримує інтерактивну розробку через Jupyter Notebook, що є критично важливим на етапі експериментів з моделлю. Версія Python 3.10 була обрана через покращену продуктивність та підтримку сучасних анотацій типів, що полегшує підтримку коду.

PyTorch було обрано як основний фреймворк глибокого навчання для реалізації нейромережевої компоненти системи. Хоча TensorFlow історично має велику частку ринку, особливо в корпоративному секторі, PyTorch домінує в академічних та дослідницьких колах. Це обумовлено його підходом «Python-first», який забезпечує виняткову гнучкість, інтуїтивність та легкість у налагодженні моделей. Для завдань, пов'язаних із трансформерами, ця гнічкість є вирішальною, оскільки дозволяє швидко експериментувати з архітектурою моделі та модифікувати процес навчання. Більше того, з виходом версії PyTorch 2.x та інтеграцією механізму torch.compile(), фреймворк значно скоротив відставання у продуктивності, часто демонструючи кращу утилізацію графічних прискорювачів

порівняно з TensorFlow на сучасному обладнанні, що є критично важливим для ефективного навчання та інференсу великих моделей [34].

Порівняльний аналіз основних фреймворків глибокого навчання представлено в таблиці 3.1. Як видно з таблиці, PyTorch забезпечує оптимальний баланс між гнучкістю розробки, продуктивністю та широтою підтримки трансформерних архітектур.

Таблиця 3.1 – Порівняння фреймворків глибокого навчання

Характеристика	PyTorch 2.x	TensorFlow 2.x	JAX
Швидкість прототипування	Висока	Середня	Висока
Утилізація GPU, %	До 100	До 90	До 95
Підтримка трансформерів	Відмінна	Хороша	Середня
Динамічний граф обчислень	Так	Так (Eager)	Так
Академічна популярність	Найвища	Середня	Зростає
Інтеграція з Hugging Face	Нативна	Підтримується	Обмежена

Бібліотека Hugging Face Transformers є стандартом де-факто для роботи з трансформерними моделями в обробці природної мови. Інтеграція цієї бібліотеки була безальтернативною, оскільки вона надає доступ до Model Hub, що містить мільйони попередньо навчених моделей, включаючи обрану в Розділі 2 модель distilbert-base-uncased [33]. Бібліотека пропонує уніфіковані класи, такі як Trainer для спрощення процесу донавчання та Pipeline для оптимізованого інференсу. Високоефективні токенізатори, що є невід'ємною частиною бібліотеки, реалізовані мовою Rust та забезпечують швидку обробку текстових даних, що є критично важливим для коректної підготовки вхідних послідовностей [35].

Для серверної частини системи було обрано мікрофреймворк Flask як основу для створення веб-сервісу з інтерфейсом REST API. Його мінімалістичність та легкість роблять Flask ідеальним вибором для розгортання сервісів, що обслуговують моделі машинного навчання. Альтернативні фреймворки, такі як

Django та FastAPI, також розглядалися на етапі проєктування. Django, незважаючи на багатофункціональність та вбудовану систему керування базами даних, є надмірно складним для завдань машинного навчання, де не потрібна повноцінна ORM та адміністративна панель. FastAPI, побудований на Starlette та Uvicorn, демонструє вищу продуктивність порівняно з Flask, проте Flask забезпечує кращу зрілість екосистеми та більшу кількість готових розширень для інтеграції з моделями машинного навчання.

Ключовим архітектурним рішенням стало дотримання найкращих практик, де Flask використовується виключно як шлюз для обробки HTTP-запитів. Усі важкі обчислювальні процеси, такі як інференс моделі, її донавчання або генерація синтетичних даних за допомогою великих мовних моделей, ініціюються через Flask, але виконуються в окремих, керованих фонових потоках або процесах. Такий підхід запобігає блокуванню основного потоку Flask та забезпечує високу швидкість відгуку програмного інтерфейсу навіть під час виконання ресурсомістких операцій. Для забезпечення міжпроцесної комунікації було використано бібліотеку Flask-SocketIO, що дозволяє реалізувати двосторонню комунікацію між сервером та клієнтом в режимі реального часу.

Для клієнтської частини системи було обрано бібліотеку React як основу для побудови користувацького інтерфейсу. Як провідна компонентна бібліотека JavaScript, React дозволяє створювати складні, динамічні односторінкові застосунки з високою інтерактивністю. Це було необхідно для реалізації інтерактивної панелі аналізу з візуалізацією результатів роботи моделі через механізми пояснюваного штучного інтелекту, а також багатофункціональної панелі адміністратора з можливістю управління моделлю, перегляду метрик та керування навчальними даними. Віртуальний DOM React забезпечує ефективне оновлення лише тих частин інтерфейсу, які змінилися, що є критично важливим для відображення результатів інференсу в режимі реального часу [36].

Для зберігання даних було обрано систему керування базами даних SQLite. SQLite є безсерверною, файловою СУБД, що не вимагає окремого процесу сервера

баз даних. Хоча SQLite не призначена для високонавантажених виробничих середовищ з тисячами одночасних підключень, вона є ідеальним вибором для етапу дослідження та прототипування. База даних не вимагає адміністрування окремого сервера та чудово підходить для локального зберігання навчальних зразків у таблиці samples, метаданих моделі та логів системи, що є поширеною практикою для проєктів машинного навчання на стадії розробки. SQLite забезпечує повну підтримку транзакцій ACID, що гарантує цілісність даних навіть у разі несподіваного завершення процесу навчання. Для майбутнього масштабування системи передбачено можливість міграції на PostgreSQL або MySQL без зміни архітектури додатку завдяки використанню абстракції SQLAlchemy ORM.

Додатково, для реалізації механізмів пояснюваного штучного інтелекту було інтегровано бібліотеку Captum, розроблену командою Meta AI. Captum надає широкий спектр алгоритмів інтерпретації моделей, включаючи Layer Integrated Gradients, що було обрано як основний метод візуалізації важливості токенів у вхідній послідовності. Інтеграція Captum з PyTorch є нативною, що дозволяє обчислювати атрибуції без додаткових перетворень моделі.

Таким чином, обраний технологічний стек забезпечує оптимальний баланс між продуктивністю, гнучкістю розробки та можливістю майбутнього масштабування системи. Комбінація Python, PyTorch та Hugging Face Transformers створює потужну основу для роботи з трансформерними моделями, Flask забезпечує легкий та ефективний веб-сервіс, React надає сучасний інтерактивний інтерфейс, а SQLite забезпечує надійне зберігання даних на етапі прототипування.

3.2. Архітектура та програмна реалізація серверної частини системи

Серверна частина системи виявлення фішингу реалізована у вигляді програмного інтерфейсу REST у файлі anti_phishing_backend.py на базі мікрофреймворку Flask. Архітектура серверної частини побудована з урахуванням специфіки завдань машинного навчання, де основним викликом є ефективне управління обчислювальними ресурсами, зокрема відеопам'яттю графічних

прискорювачів, та забезпечення швидкого відгуку системи під час виконання інференсу моделі.

Додаток Flask ініціалізується з інтеграцією двох критично важливих розширень. По-перше, Flask-CORS забезпечує коректну крос-доменну взаємодію між серверною частиною та клієнтським додатком на React, що працюють на різних портах під час розробки. Це розширення автоматично додає необхідні заголовки Cross-Origin Resource Sharing до відповідей сервера, дозволяючи браузеру виконувати запити з іншого джерела. По-друге, Flask-SocketIO інтегровано для майбутнього розширення функціоналу системи з підтримкою двосторонньої комунікації в режимі реального часу, що може бути використано для відображення прогресу навчання моделі або генерації синтетичних даних.

Програмний інтерфейс системи структуровано у дві категорії маршрутів, що відрізняються за рівнем доступу та призначенням. Публічний маршрут /analyze з методом POST є основним робочим кінцевим пунктом системи, через який користувачі взаємодіють з моделлю виявлення фішингу. Цей маршрут приймає запити у форматі JSON від клієнтського додатку, що містять поле inputText з вхідними даними, поле activeTab для визначення типу вхідних даних, прапорець useRules для активації гібридного режиму з використанням евристичних правил, та прапорець explain для увімкнення механізму пояснюваного штучного інтелекту.

Критично важливим елементом захисту публічного програмного інтерфейсу є інтеграція бібліотеки Flask-Limiter з декоратором @limiter.limit(«10 per minute»), що застосовується до маршруту /analyze. Цей механізм обмеження частоти запитів реалізує захист на рівні мережі, обмежуючи кількість звернень з однієї IP-адреси до десяти запитів на хвилину. Такий підхід забезпечує базовий захист від атак типу «грубої сили», атак на відмову в обслуговуванні та неконтрольованого використання обчислювальних ресурсів програмного інтерфейсу. Обмеження частоти запитів є стандартною практикою для захисту сервісів машинного навчання, де кожен запит потребує значних обчислювальних ресурсів для виконання інференсу моделі.

Адміністративні маршрути, що мають префікс /admin/*, утворюють окрему категорію кінцевих пунктів з підвищеним рівнем захисту. Ця група маршрутів надає доступ до критично важливих функцій управління життєвим циклом моделі, включаючи генерацію синтетичних навчальних даних, запуск процесу донавчання моделі, перегляд статистики набору даних, аналіз історії навчання та динамічну зміну обчислювального пристрою. Доступ до адміністративних маршрутів захищено за допомогою механізму автентифікації, реалізованого через бібліотеку flask_httpauth.

Автентифікація адміністративних функцій здійснюється через механізм HTTP Basic Authentication, що є простим, але надійним підходом для захисту внутрішніх програмних інтерфейсів. Декоратор @auth.login_required перехоплює вхідний запит до адміністративного маршруту та викликає функцію verify_password перед виконанням основної логіки обробника. Функція верифікації порівнює надані користувачем облікові дані з еталонними значеними, що безпечно зберігаються у змінних середовища сервера під іменами ADMIN_USERNAME та ADMIN_PASSWORD. Такий підхід до зберігання облікових даних є стандартною практикою в індустрії, оскільки запобігає витоку чутливої інформації через систему контролю версій та забезпечує можливість легкої зміни облікових даних без модифікації коду програми.

Серед адміністративних маршрутів особливу увагу заслуговує кінцева точка /admin/generate-samples, що ініціює фоновий процес генерації синтетичних навчальних зразків за допомогою великих мовних моделей. Маршрут /admin/start-training запускає процес донавчання моделі DistilBERT на оновленому наборі даних, що виконується в окремому фоновому потоці для запобігання блокуванню основного потоку Flask. Маршрут /admin/dataset-status повертає детальну статистику по базі даних, включаючи кількість зразків кожного класу та загальний обсяг навчальних даних. Кінцева точка /admin/training-history надає доступ до історичних даних про точність моделі після кожної сесії навчання, що дозволяє відстежувати динаміку покращення якості класифікації. Маршрут /admin/device

забезпечує можливість динамічної зміни обчислювального пристрою між центральним процесором та графічним прискорювачем без перезапуску серверного процесу.

Ефективне управління пам'яттю, особливо відеопам'яттю графічних прискорювачів, є однією з найскладніших задач при розгортанні трансформерних моделей у виробничих системах. У файлі anti_phishing_backend.py це завдання вирішено через систему потокобезпечних функцій, що реалізують інтелектуальне завантаження та вивантаження моделі з пам'яті. Функція get_model_handler() реалізує класичний програмний патерн «Сінглтон», що гарантує існування лише одного екземпляра обробника моделі протягом життєвого циклу серверного процесу. При першому виклику цієї функції вона створює єдиний екземпляр класу DistilBERTPhishing та ініціює фоновий потік model_inactivity_check, що працює паралельно з основним потоком Flask. Усі наступні виклики get_model_handler() повертають посилання на цей самий екземпляр, запобігаючи множинному завантаженню моделі в пам'ять.

Фоновий потік model_inactivity_check реалізує механізм автоматичного звільнення ресурсів під час простою системи. Цей потік виконується паралельно з основним процесом Flask та щохвилини перевіряє часову мітку останнього використання моделі. Якщо модель не використовувалася протягом періоду, визначеного константою MODEL_INACTIVITY_TIMEOUT, потік викликає метод model_handler.unload(), що вивантажує модель з оперативної та відеопам'яті. Після вивантаження модель залишається на диску у вигляді збережених вагових коефіцієнтів, та може бути швидко завантажена знову при наступному запиті користувача. Такий підхід є особливо важливим для систем, що працюють на обладнанні з обмеженим обсягом відеопам'яті, оскільки дозволяє ефективно розподіляти ресурси між різними процесами.

Функція get_model() є центральною точкою доступу до моделі для всіх кінцевих пунктів програмного інтерфейсу. Вона отримує посилання на обробник моделі через get_model_handler(), переконується що модель завантажена в пам'ять

шляхом виклику методу model_handler.load() у разі необхідності, оновлює часову мітку останнього використання та повертає готовий до роботи екземпляр моделі. Це забезпечує прозорість для решти коду програми, який не потребує знання про деталі управління життєвим циклом моделі.

Функція set_device_preference(device: str) надає можливість динамічного перемикання обчислювального пристрою між центральним процесором та графічним прискорювачем. Ця функція, що викликається через адміністративний програмний інтерфейс, виконує безпечне вивантаження поточної моделі з пам'яті та скидає екземпляр сінглтону. При наступному запиті до системи функція get_model_handler() створить новий екземпляр обробника, що буде використовувати обраний адміністратором пристрій для виконання обчислень. Така гнучкість дозволяє адаптувати систему до різних умов експлуатації, наприклад, звільнити графічний прискорювач для інших завдань або переключитися на центральний процесор у разі проблем з драйверами CUDA.

Описана архітектура серверної частини забезпечує оптимальний баланс між швидкістю відгуку системи та ефективним використанням обчислювальних ресурсів. Система миттєво реагує на запити користувачів завдяки утриманню моделі в пам'яті під час активного використання, але автоматично звільняє дорогі ресурси графічного прискорювача під час простою, що дозволяє іншим процесам ефективно використовувати обладнання. Механізми захисту через обмеження частоти запитів та автентифікацію адміністративних функцій забезпечують базовий рівень безпеки системи відповідно до сучасних практик розробки програмних інтерфейсів.

3.3. Реалізація гібридного ядра аналізу фішингових загроз

Ядро системи виявлення фішингу реалізовано в програмному модулі analysis.py, що втілює гібридну архітектуру аналізу, теоретичне обґрунтування якої було наведено в Розділі 1. Цей модуль поєднує швидкі евристичні правила, що базуються на експертних знаннях про типові ознаки фішингових атак, з глибоким

семантичним аналізом на основі трансформерної моделі DistilBERT. Така архітектура дозволяє досягти оптимального балансу між швидкістю обробки детермінованих, очевидних випадків атак та здатністю розпізнавати складні, семантично замасковані фішингові повідомлення, що потребують контекстуального розуміння.

Функція analyze_with_rules(text, type_) реалізує перший рівень перевірки вхідних даних на основі набору швидких евристичних правил. Ця функція використовує попередньо скомпільовані регулярні вирази для ефективного сканування вхідних даних без необхідності їх повторної компіляції при кожному виклику. Регулярні вирази організовані в структуровані набори відповідно до типу вхідних даних, що дозволяє застосовувати спеціалізовані правила перевірки для кожного каналу атаки.

Для аналізу URL-адрес евристичний модуль виконує перевірку на наявність прямих IP-адрес замість доменних імен, що є типовою ознакою фішингових сайтів, які намагаються уникнути блокування через систему доменних імен. Також перевіряється використання підозрілих доменів верхнього рівня, зокрема таких як .ru, .cn, .tk, .ml, .ga та .cf, які статистично частіше використовуються в фішингових кампаніях через низьку вартість реєстрації та слабкий контроль з боку реєстраторів. Додатково аналізується наявність символу @ в URL-адресі, що може використовуватися для маскування справжнього домену в атаках типу user@domain.com, де браузер ігнорує частину перед символом @. Окремою категорією перевірок є виявлення використання сервісів скорочення посилань, які зловмисники часто застосовують для приховування справжньої адреси фішингового сайту.

Для аналізу HTML-коду реалізовано набір правил, спрямованих на виявлення небезпечних конструкцій веб-сторінок. Система перевіряє наявність потенційно небезпечного JavaScript-коду, зокрема використання функції eval(), яка може виконувати довільний код та часто застосовується в складних атаках. Детектується наявність полів вводу паролів через теги <input type=«password»>, що є

індикатором спроби викрадення облікових даних. Аналізуються приховані елементи з атрибутом display:none, які можуть містити шкідливий контент або використовуватися для обходу автоматичних систем перевірки. Особливу увагу приділено виявленню елементів <iframe>, що завантажують контент з підозрілих доменів, оскільки ця техніка часто використовується для вбудовування фішингових форм на легітимні сайти.

Для текстового контенту повідомлень розроблено набір правил, що виявляють ключові фрази соціальної інженерії. Система аналізує наявність слів та фраз, що створюють штучне відчуття терміновості, таких як «негайно», «термінова дія потрібна», «обмежена пропозиція», які є класичними маніпулятивними прийомами фішингових повідомлень. Детектуються обіцянки призів, виграшів або несподіваних грошових надходжень, що є типовою ознакою шахрайських схем. Також перевіряється наявність прямих запитів на надання персональних даних, паролів або фінансової інформації, що ніколи не використовується легітимними організаціями в електронних повідомленнях.

Окрему категорію складає спеціалізована функція detect_homoglyphs(url), що виконує поглиблений аналіз доменного імені на предмет атак типу «гомогліф». Гомогліфні атаки експлуатують візуальну подібність символів з різних алфавітів, наприклад, латинської літери «a» та кириличної «а», які виглядають ідентично для людського ока, але мають різні коди в таблиці Unicode [37]. Функція аналізує кожен символ доменного імені та виявляє змішування символів з різних писемностей, що є надійним індикатором злих намірів, оскільки легітимні домени не використовують символи з різних алфавітів. Такі атаки є особливо небезпечними, оскільки можуть обійти як людську увагу, так і прості текстові фільтри, що роблять їх ефективним інструментом для створення фішингових доменів, які візуально неможливо відрізнити від легітимних [38].

Центральною функцією модуля аналізу є combine_risk(rule_flags, neural_pred), що реалізує інтелектуальну систему злиття результатів з різних джерел виявлення загроз. Ця функція отримує на вхід два набори результатів

аналізу: neural_pred, що містить словник з імовірностями приналежності до кожного класу від моделі DistilBERT, та rule_flags, що представляє список спрацювань від евристичного аналізатора з відповідними рівнями ризику. На відміну від простих ансамблевих методів, що використовують усереднення або зважене голосування прогнозів, реалізована система застосовує пріоритезовану логіку прийняття рішень.

Алгоритм злиття результатів спочатку обчислює максимальний рівень ризику серед усіх джерел інформації, що включають як евристичні правила, так і прогноз нейронної мережі. Цей рівень може приймати значення high, medium або low відповідно до виявлених ознак фішингу. Критично важливою особливістю реалізації є застосування механізму «правила вето», що надає евристичним детекторам пріоритет у випадках виявлення детермінованих індикаторів атаки. Якщо серед елементів rule_flags присутній хоча б один прапорець з високим рівнем ризику, наприклад, функція detect_homoglyphs повернула позитивний результат виявлення гомогліфної атаки, то фінальний показник впевненості системи у класифікації примусово встановлюється на рівень 0.95, що відповідає дев'яносто п'яти відсоткам. У протилежному випадку, коли евристичні правила не виявили критичних індикаторів атаки, впевненість визначається безпосередньо з результату роботи нейронної мережі, що дозволяє моделі застосувати свою семантичну експертизу для аналізу складних, неоднозначних випадків.

Така логіка реалізує стратегію «довіряй, але перевіряй», що є оптимальною для систем кібербезпеки. Система покладається на семантичну потужність трансформерної моделі DistilBERT для аналізу складних випадків фішингу, де атака замаскована через контекстуальні маніпуляції та потребує глибокого розуміння природної мови. Водночас, система зберігає за швидкими евристичними детекторами право вето для очевидних, детермінованих типів атак, таких як гомогліфні домени або використання відомих підозрілих доменних зон. Такий підхід значно підвищує загальну надійність системи, оскільки навіть у випадку,

коли нейронна мережа через обмеження навчальних даних не розпізнала певний тип атаки, евристичний детектор може її виявити на основі експертних знань.

Оркестрацію всього процесу аналізу здійснює функція analyze_full(), що координує виклики всіх компонентів системи та формує фінальну відповідь для клієнтського додатку. Ця функція послідовно викликає analyze_with_rules для отримання результатів евристичного аналізу, потім ініціює model_instance.predict для виконання інференсу трансформерної моделі, передає обидва набори результатів у combine_risk для інтелектуального злиття прогнозів, та завершує процес викликом generate_recommendations для формування персоналізованих рекомендацій користувачу на основі виявлених загроз. Фінальна відповідь формується у форматі JSON та містить клас загрози, рівень впевненості, список спрацювавших евристичних правил, детальні рекомендації щодо дій користувача та, за запитом, візуалізацію механізмів пояснюваного штучного інтелекту.

Реалізована архітектура гібридного ядра аналізу забезпечує синергетичний ефект від поєднання швидкості та детермінованості евристичних правил з семантичною потужністю та адаптивністю трансформерних моделей. Система здатна ефективно виявляти як класичні, добре відомі типи фішингових атак через швидкі евристичні перевірки, так і нові, складні варіанти атак, що потребують глибокого контекстуального аналізу з використанням можливостей глибокого навчання.

3.4. Реалізація модуля адаптивного навчання через генерацію синтетичних даних

Програмний модуль external_ai.py є практичною реалізацією стратегії подолання дефіциту навчальних даних та боротьби з концептуальним дрейфом, теоретичне обґрунтування якої було детально викладено в підрозділі 1.4. Цей модуль забезпечує механізм адаптації за принципом «закритого циклу», що дозволяє системі самостійно генерувати нові, актуальні навчальні зразки без залучення людських експертів для розмітки даних. Концепція закритого циклу в

контексті систем штучного інтелекту передбачає інтеграцію вихідних даних назад у вхідний процес, що дозволяє моделі навчатися на основі власної продуктивності, адаптуватися до змін у патернах даних та вдосконалювати свої прогнози з часом [39].

На відміну від традиційних лінійних моделей, де дані надходять на вхід, обробляються та генерують вихід без зворотного зв'язку, системи з закритим циклом створюють петлю зворотного зв'язку, де вихідні результаті аналізуються, а отримані знання використовуються для коригування вхідних даних або параметрів системи. У контексті розробленої системи виявлення фішингу, закритий цикл реалізовано через автоматичну генерацію синтетичних навчальних зразків за допомогою великих мовних моделей, їх додавання до навчального набору даних та подальше донавчання моделі DistilBERT на розширеному наборі. Такий ітеративний процес забезпечує безперервне вдосконалення системи та її здатність адаптуватися до нових типів фішингових атак без необхідності ручного збору та розмітки реальних зразків.

Модуль підтримує інтеграцію з двома основними провайдерами великих мовних моделей для забезпечення гнучкості та відмовостійкості системи. Перший провайдер, Google Gemini, використовується через модель gemini-2.5-flash-lite, що забезпечує швидку генерацію тексту з оптимальним балансом між якістю та вартістю обчислень. Другий провайдер, OpenRouter, надає гнучкий доступ до різних моделей через уніфікований програмний інтерфейс, що дозволяє використовувати різноманітні архітектури, включаючи deepseek/deepseek-chat-v3.1:free, без необхідності окремої інтеграції для кожної моделі. Така архітектура з підтримкою множинних провайдерів забезпечує відмовостійкість системи, оскільки у разі недоступності одного провайдера можна автоматично переключитися на альтернативний.

Ключовим елементом модуля є ретельно розроблений шаблон інструкцій prompt_template, що представляє собою детальну специфікацію українською

мовою для керування процесом генерації синтетичних даних. Проєктування ефективних інструкцій для великих мовних моделей, що називається інженерією підказок, є критично важливим для забезпечення точності, ефективності та масштабованості виходів моделі. Погано спроєктовані інструкції можуть призвести до непослідовних результатів, помилок та марного витрачання обчислювальних ресурсів. Розроблений шаблон інструкцій дотримується найкращих практик інженерії підказок, включаючи чіткість та специфічність формулювань, надання релевантного контексту, визначення точних специфікацій виходу та декомпозицію складного завдання на логічні кроки.

Шаблон інструкцій вимагає від великої мовної моделі згенерувати певну кількість різноманітних зразків, що охоплюють всі типи вхідних даних системи: url для аналізу веб-адрес, html для перевірки фрагментів коду сторінок, та message для дослідження текстових повідомлень. Інструкція наголошує на необхідності створення реалістичних, але вигаданих елементів, таких як URL-адреси та назви компаній, щоб уникнути потенційних юридичних проблем з використанням справжніх брендів. Критично важливою вимогою є включення «шуму» у згенеровані дані, що імітує реальні умови використання системи. Цей шум включає граматичні помилки, друкарські помилки, нестандартні синтаксичні конструкції та інші відхилення від ідеального тексту, що часто зустрічаються у фішингових повідомленнях, створених зловмисниками з різним рівнем володіння мовою [40].

Шаблон також визначає вимогу до збалансованого розподілу рівнів ризику у згенерованих зразках, де шістдесят відсотків повинні бути класифіковані як high або medium ризик, а сорок відсотків як low ризик. Такий розподіл забезпечує, що модель отримує достатню кількість позитивних зразків фішингу для навчання розпізнавання загроз, але також експонується до легітимних зразків для уникнення надмірної чутливості та хибних спрацювань. Найважливішою технічною вимогою шаблону є вказівка повертати відповідь виключно у вигляді валідних об'єктів JSON, де кожен зразок розміщується на окремому рядку та дотримується строгої

схеми з полями text для вмісту зразка, type для типу даних, та actualRisk для рівня ризику. Така структурована форма виходу є критично важливою для автоматизованої обробки та валідації згенерованих даних перед їх додаванням до бази даних.

Процес генерації синтетичних даних повністю інтегровано в серверну частину системи через адміністративний програмний інтерфейс. Адміністратор системи ініціює процес генерації через маршрут /admin/generate-samples, що запускає фонову функцію run_batch_generation в окремому потоці для запобігання блокуванню основного потоку Flask. Функція run_batch_generation виконує ітеративні виклики generate_samples_with_external_ai, передаючи шаблон інструкцій та параметри генерації до обраного провайдера великих мовних моделей. Кожен виклик повертає пакет згенерованих зразків у форматі JSON, які проходять строгу валідацію для перевірки відповідності схемі даних та коректності структури.

Після успішної валідації згенеровані зразки негайно додаються до бази даних SQLite через функцію database.add_samples, що забезпечує їх доступність для наступного циклу донавчання моделі. Таблиця samples в базі даних зберігає текст зразка, його тип, рівень ризику, часову мітку створення та мітку джерела даних, що дозволяє відстежувати походження кожного зразка та відрізняти синтетичні дані від реальних. Система також веде статистику генерації, включаючи загальну кількість згенерованих зразків, розподіл за типами та рівнями ризику, а також відсоток успішно провалідованих зразків.

Реалізований модуль адаптивного навчання через генерацію синтетичних даних забезпечує системі здатність до безперервного самовдосконалення. Замість статичної моделі, що навчена один раз і поступово втрачає точність через концептуальний дрейф та появу нових типів атак, розроблена система може регулярно генерувати нові навчальні дані, що відображають поточні тенденції в фішингу, та донавчувати модель для підтримання високої точності виявлення. Така

архітектура закритого циклу представляє значний крок вперед в напрямку автономних систем кібербезпеки, що можуть адаптуватися до еволюції загроз без постійного втручання людських експертів.

3.5. Розробка клієнтської частини системи та інтерфейсу адміністратора

Клієнтська частина системи виявлення фішингу реалізована як односторінковий застосунок у вигляді браузерного плагіну на базі бібліотеки React у файлі App.js. Архітектура односторінкового застосунку представляє сучасний підхід до розробки вебінтерфейсів, де вся необхідна логіка, стилі та розмітка завантажуються один раз при початковому завантаженні сторінки, а подальша навігація та оновлення контенту відбуваються динамічно без повного перезавантаження сторінки. На відміну від традиційних багатосторінкових застосунків, де кожна зміна контексту вимагає запиту до сервера та завантаження нової HTML-сторінки, односторінкові застосунки забезпечують плавний, схожий на настільні програми досвід взаємодії.

Взаємодія клієнтського застосунку з серверним програмним інтерфейсом реалізована через стандартний API Fetch, що є сучасним підходом до виконання асинхронних HTTP-запитів у веб-застосунках. Функція analyzeWithLocalBackend демонструє клієнтську логіку для публічного аналізу вхідних даних. Ця функція збирає дані зі стану компонента React, що включає текст для аналізу з поля inputText, налаштування застосунку через об'єкт appSettings з прапорцем useRules для активації гібридного режиму, та інші параметри конфігурації. Зібрані дані формуються у структуру JavaScript-об'єкта та серіалізуються у формат JSON для передачі через мережу. На рисунку 3.1 показано вигляд головного вікна плагіну.

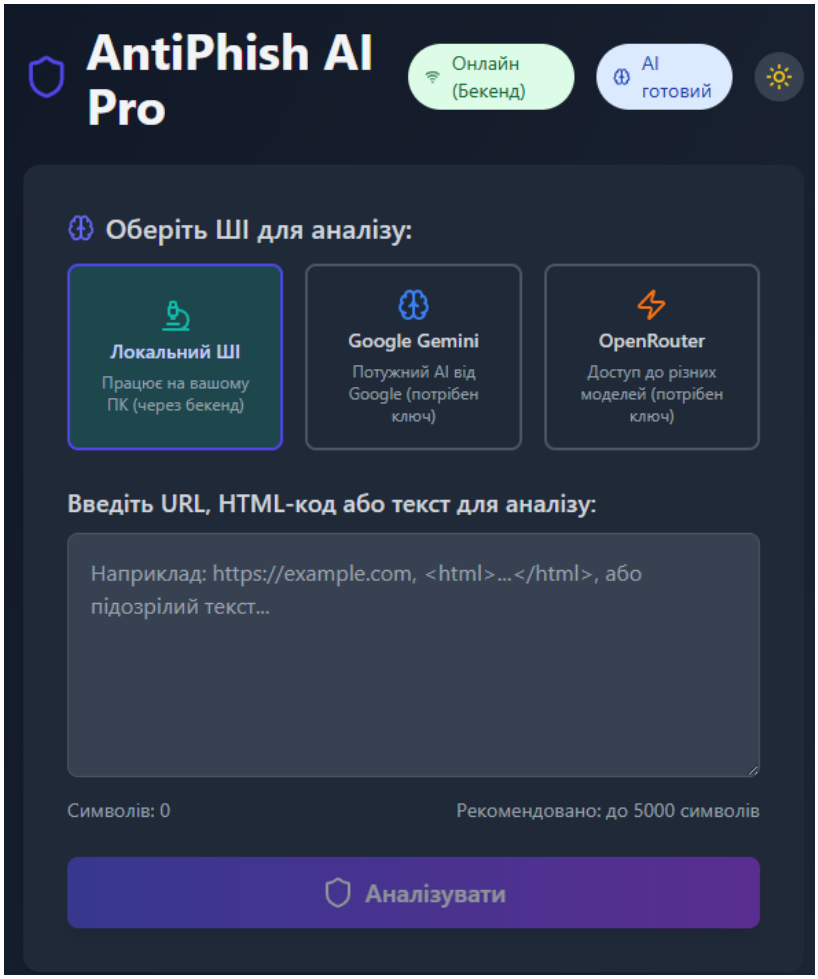


Рисунок 3.1 – Вигляд головного вікна плагіну

Як видно з рисунку 3.1, можна вибрати який саме штучний інтелект буде аналізувати чи є сайт, повідомлення або код сторінки фішинговим.

Запит до серверного програмного інтерфейсу виконується методом POST до кінцевого пункту http://127.0.0.1:3000/analyze, де localhost адреса 127.0.0.1 та порт 3000 відповідають локальному серверу Flask під час розробки. Тіло запиту передається у форматі JSON через встановлення відповідного заголовка Content-Type зі значенням application/json, що інформує сервер про структуру вхідних даних. Асинхронна природа API Fetch дозволяє інтерфейсу залишатися відгуковим під час виконання запиту, не блокуючи інтерактивність застосунку. Після отримання відповіді від сервера, результат аналізу автоматично оновлює стан компонента React через виклик відповідної функції setState, що призводить до

динамічного перерендерингу інтерфейсу та відображення результатів користувачу без перезавантаження сторінки. На рисунку 3.2 показано вибір налаштувань даного плагіну, що видимі користувачеві.

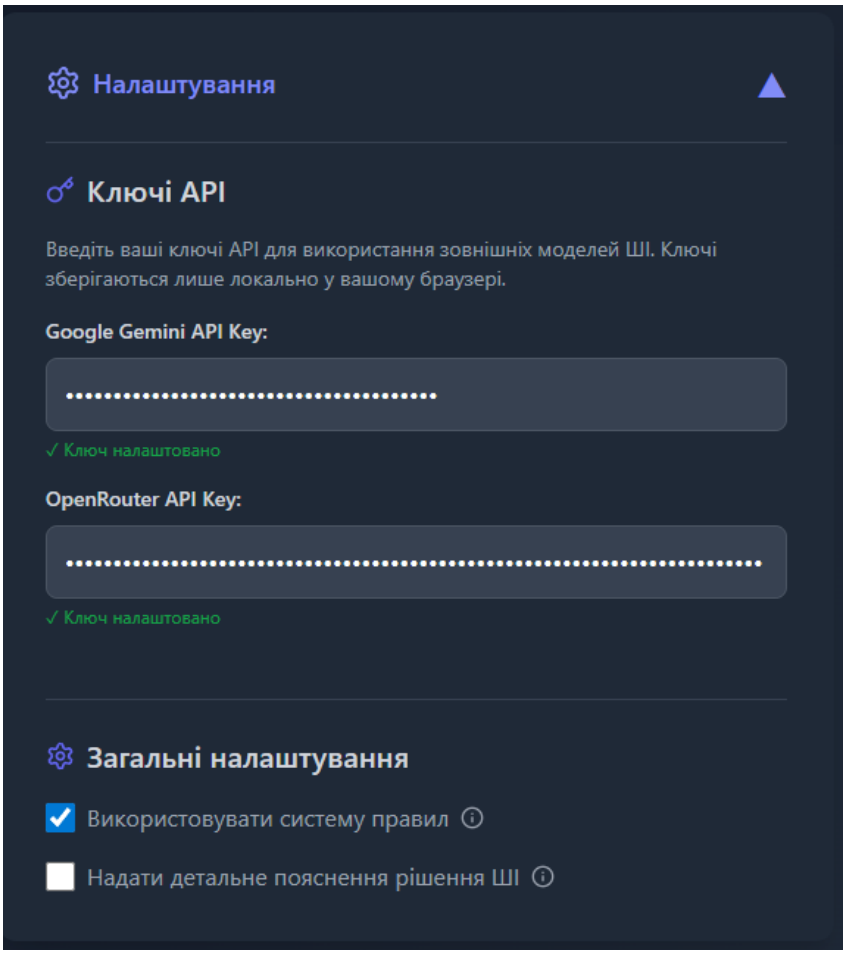


Рисунок 3.2 – Меню налаштувань плагіну

Як видно з цього рисунку, в цьому меню можна ввести особисті ключі API для зовнішніх додатків, активувати або деактивувати використання системи правил та аргументацію від системи.

Для доступу до адміністративних функцій системи реалізовано окремий набір функцій, що працюють із захищеними маршрутами серверного програмного інтерфейсу. Функція fetchTrainingHistory демонструє патерн взаємодії з адміністративними кінцевими пунктами. Ця функція викликає допоміжну функцію getAdminAuthHeaders для формування заголовка автентифікації Authorization у

форматі HTTP Basic Authentication. Механізм Basic Authentication передбачає кодування облікових даних адміністратора у форматі Base64 та їх включення у заголовок запиту з префіксом Basic. Хоча цей механізм автентифікації є базовим, він є достатнім для захисту адміністративного інтерфейсу на етапі розробки та локального розгортання, особливо в поєднанні з обмеженням доступу через мережеві правила.

Панель адміністратора представляє комплексний інтерфейс для управління всіма аспектами життєвого циклу моделі та системи в цілому. Інтерфейс структуровано у кілька функціональних розділів, кожен з яких надає доступ до специфічних можливостей системи через відповідні програмні інтерфейси. Розділ генерації даних дозволяє адміністратору ініціювати процес створення синтетичних навчальних зразків через виклик кінцевого пункту /admin/generate-samples. Інтерфейс надає можливість налаштування параметрів генерації, включаючи кількість зразків, розподіл за типами даних та рівнями ризику, а також вибір провайдера великих мовних моделей.

Розділ навчання моделі забезпечує повний контроль над процесом донавчання моделі DistilBERT на оновленому наборі даних. Інтерфейс відображає поточний стан набору даних через виклик кінцевого пункту /admin/dataset-status, показуючи детальну статистику про кількість зразків кожного класу, загальний обсяг даних та розподіл між синтетичними та реальними зразками. Історія навчання представлена у вигляді графіка або таблиці, що отримуються через маршрут /admin/training-history, та відображає динаміку точності моделі після кожного циклу донавчання, дозволяючи відстежувати прогрес та виявляти можливі проблеми, такі як переобачення чи недостатнє навчання. Адміністратор може ініціювати новий цикл донавчання через кінцевий пункт /admin/start-training, налаштовуючи критичні гіперпараметри, такі як швидкість навчання, розмір пакета, кількість епох та стратегію оптимізації. На рисунку 3.3 зображена панель адміністратора з вибором специфічних налаштувань, таких як використання графічного прискорення, загальна статистика моделі, оператор генерації

синтетичних даних, ручний ввід даних, параметри навчання та використання правил для навчання.

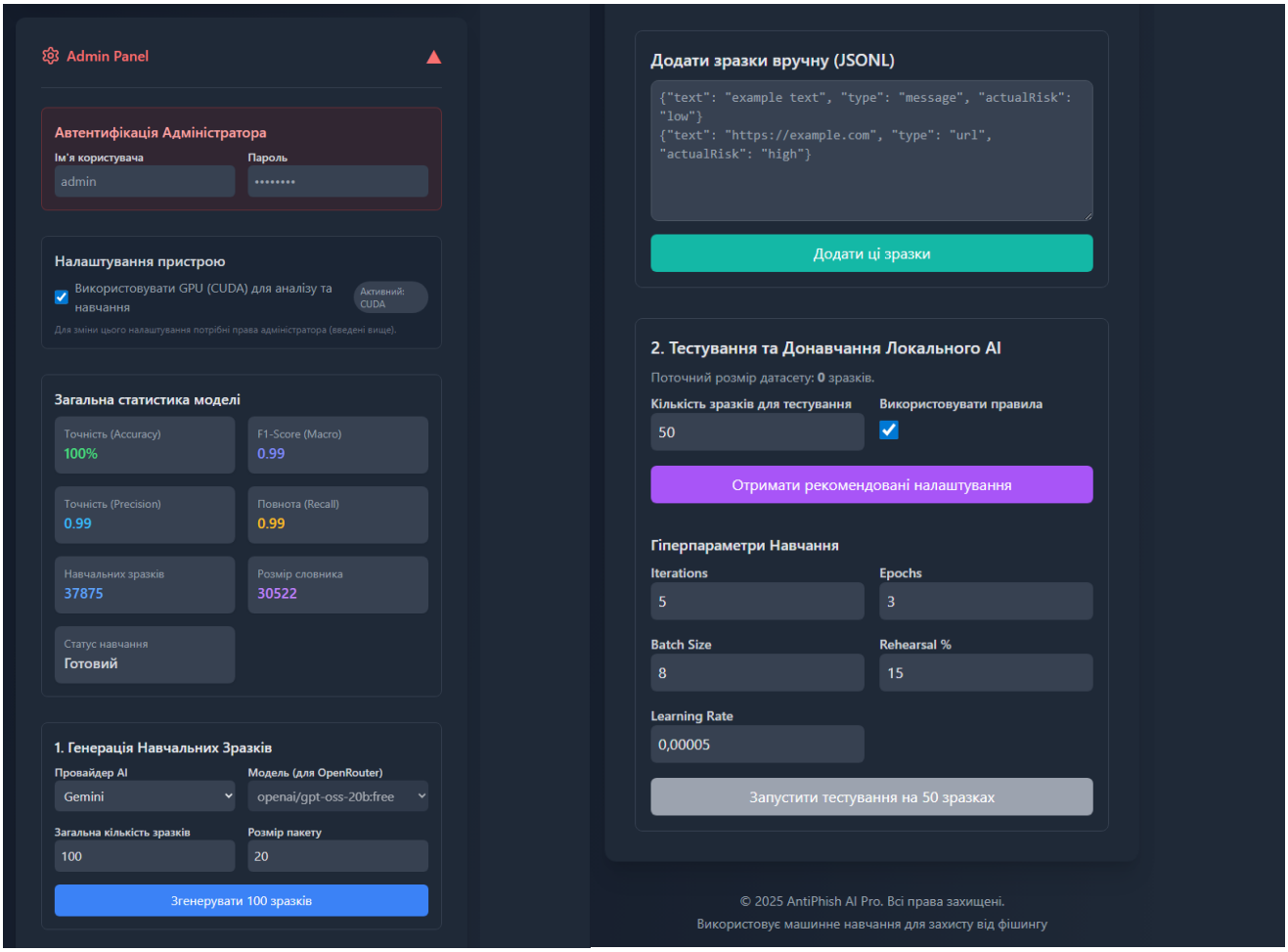


Рисунок 3.3 – Вигляд панелі адміністратора

Розділ управління системою надає доступ до функцій конфігурації обчислювальної інфраструктури. Особливо важливою є можливість динамічного перемикання обчислювального пристрою між центральним процесором та графічним прискорювачем через маршрут /admin/device. Інтерфейс відображає поточний активний пристрій, доступність графічного прискорювача, обсяг використаної та вільної відеопам'яті, а також надає кнопку для швидкого перемикання між пристроями. Така гнучкість дозволяє адміністратору

оптимізувати використання обчислювальних ресурсів відповідно до поточних потреб системи. На рисунку 3.4 зображено графіки та процес навчання моделі.

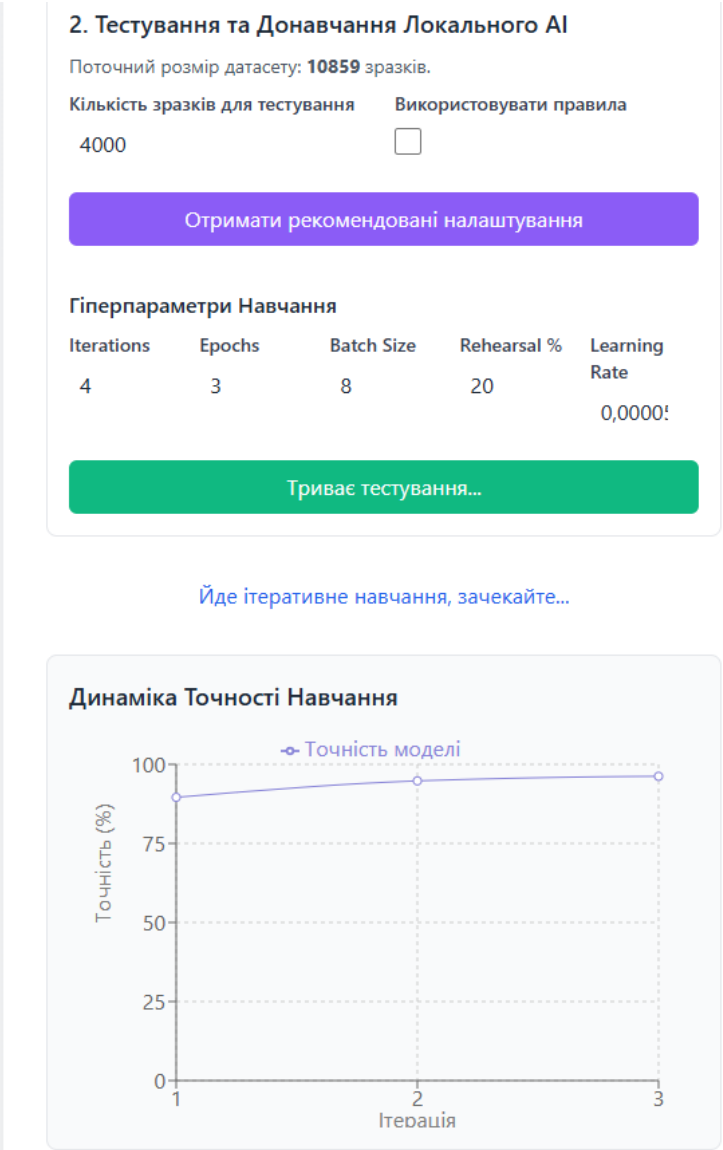


Рисунок 3.4 – Процес навчання моделі в панелі адміністратора

Стан застосунку управляється через вебхуки React, зокрема useState для локального стану компонентів та useEffect для виконання побічних ефектів, таких як завантаження даних при монтуванні компонента. Для складних станів, що включають множинні взаємопов'язані значення, використовується патерн розділення стану на логічні групи, що покращує читабельність коду та полегшує

підтримку. Всі асинхронні операції, такі як виклики програмного інтерфейсу, обгорнуті в конструкції обробки помилок для забезпечення коректної поведінки застосунку у випадку збоїв мережі або помилок сервера.

Користувацький інтерфейс побудовано з дотриманням принципів сучасного веб-дизайну, забезпечуючи інтуїтивність та доступність для різних категорій користувачів. Публічна частина інтерфейсу для аналізу фішингу розроблена з акцентом на простоту використання, мінімізуючи кількість кроків, необхідних для отримання результату. Адміністративна панель організована у вигляді вкладок або розділів, що дозволяють швидко переключатися між різними функціями управління без втрати контексту. Всі інтерактивні елементи надають візуальний зворотний зв'язок про свій стан, включаючи індикатори завантаження під час виконання довгих операцій, повідомлення про успіх або помилку після завершення дій, та підказки для полів введення.

Розроблена клієнтська частина системи забезпечує повний цикл взаємодії користувача з функціями виявлення фішингу, від подання вхідних даних та отримання результатів аналізу до детального управління моделлю та моніторингу її продуктивності через адміністративну панель. Архітектура плагіна на базі React забезпечує швидкий та інтуїтивний інтерфейс, що відповідає сучасним стандартам розробки веб-застосунків.

3.6. Експериментальна перевірка та валідація ефективності моделі

Відповідно до вимог методичних вказівок до магістерської кваліфікаційної роботи, для верифікації ефективності розробленого програмного прототипу було проведено комплексне експериментальне дослідження гібридної системи виявлення фішингу. Експериментальна верифікація є критично важливим етапом розробки систем машинного навчання, оскільки дозволяє об'єктивно оцінити здатність моделі до узагальнення на нових, раніше небачених даних та підтвердити практичну цінність розробленого рішення.

Для оцінки якості гібридного класифікатора було сформовано збалансований тестовий набір даних, який не брав участі у процесі навчання моделі та був

відокремлений від навчальної вибірки на етапі підготовки експерименту [33]. Принцип розділення даних на навчальну та тестову вибірки є фундаментальним у машинному навчанні, оскільки забезпечує об'єктивну оцінку здатності моделі до генералізації без ризику передбачення на навчальних даних. Тестовий набір включав як реальні зразки фішингових атак, отримані з архівів спеціалізованої платформи PhishTank, що є найбільшою колаборативною базою даних верифікованих фішингових сайтів, так і легітимні веб-ресурси та повідомлення з перевірених джерел [41]. Додатково до тестового набору були включені синтетично згенеровані зразки, створені за допомогою великих мовних моделей відповідно до методології, описаної в підрозділі 3.4, що дозволило перевірити здатність моделі до узагальнення на штучних, але реалістичних даних, які імітують нові, раніше не зустрічані патерни фішингових атак [42]. На рисунку 3.5 представлено приклад визначення фішингового листа від імені банку. З цього рисунку видно, що система з максимальною впевненістю вважає цей лист фішинговим, що таким і є.

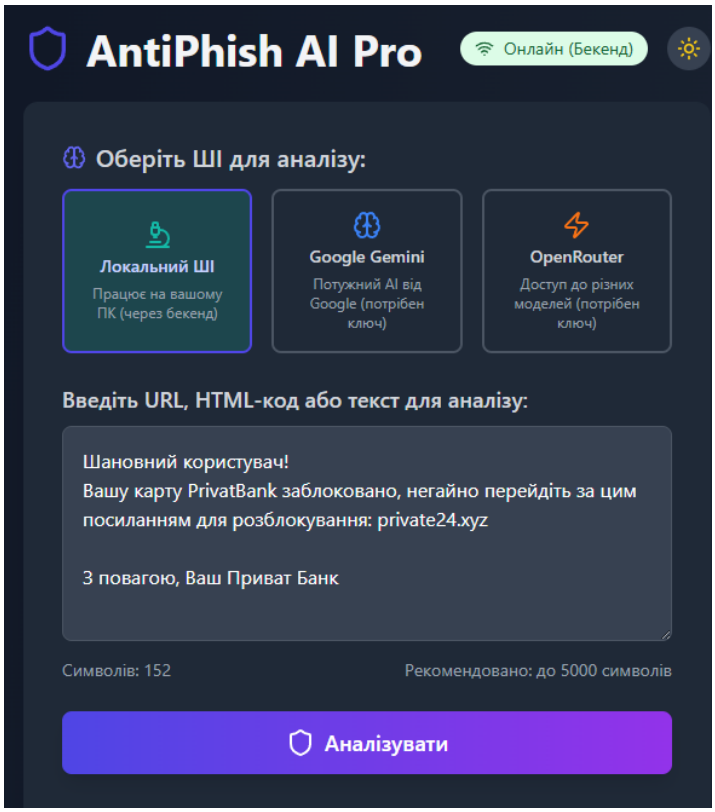


Рисунок 3.5 – Приклад визначення фішингового листа.

Для комплексної оцінки ефективності класифікатора були обрані стандартні метрики для задач бінарної класифікації, що широко використовуються в академічних дослідженнях та промислових застосуваннях систем виявлення фішингу [43]. Метрика загальної точності визначається як частка правильних прогнозів серед усіх здійснених класифікацій та обчислюється за формулою:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

Де TP позначає кількість істинно позитивних випадків, TN – істинно негативних, FP – хибно позитивних, а FN – хибно негативних прогнозів. Хоча загальна точність надає базове уявлення про продуктивність моделі, цей показник може бути оманливим при роботі з незбалансованими наборами даних, де один клас значно переважає інший, оскільки модель може досягти високої точності, просто передбачаючи домінуючий клас для всіх вхідних зразків.

Метрика точності позитивного класу вимірює, яка частка зразків, ідентифікованих системою як фішингові, дійсно є фішинговими загрозами, та обчислюється за формулою:

$$Precision = \frac{TP}{TP + FP} \quad (3.3)$$

Високий показник точності є важливим для мінімізації хибних спрацювань, які призводять до блокування легітимних веб-ресурсів або повідомлень, що може негативно впливати на користувацький досвід та знижувати довіру до системи. У контексті систем кібербезпеки, надмірна кількість хибних позитивних спрацювань може призвести до ефекту «втоми від попереджень», коли користувачі починають ігнорувати попередження системи через їх часту некоректність.

Метрика повноти вимірює, яку частку всіх реальних фішингових зразків у тестовому наборі модель змогла успішно виявити, та обчислюється за формулою:

$$Recall = \frac{TP}{TP + FN} \quad (3.4)$$

Повнота є однією з найкритичніших метрик у задачі виявлення фішингу, оскільки пропуск реальної фішингової атаки, що класифікується як хибно негативний випадок, має значно вищу вартість та потенційні збитки порівняно з помилковим блокуванням легітимного ресурсу. Невиявлена фішингова атака може призвести до викрадення облікових даних користувачів, фінансових втрат, компрометації корпоративних систем та інших серйозних наслідків для безпеки. Дослідження в галузі виявлення шахрайства та фішингу підкреслюють, що висока повнота є абсолютним пріоритетом, навіть якщо це призводить до незначного зниження точності.

Метрика F1-міри представляє гармонійне середнє між точністю та повнотою, що надає збалансовану оцінку ефективності класифікатора, та обчислюється за формулою:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.5}$$

На відміну від арифметичного середнього, гармонійне середнє є більш чутливим до низьких значень, що означає, що F1-міра буде високою тільки якщо обидві складові метрики, точність та повнота, мають високі значення. Це робить F1-міру особливо цінною для оцінки систем, де потрібен баланс між мінімізацією хибних спрацювань та забезпеченням високого рівня виявлення загроз.

Донавчена модель DistilBERT, інтегрована в гібридну систему з евристичними правилами згідно з архітектурою, описаною в підрозділі 3.3, була протестована на сформованій валідаційній вибірці. Результати експериментальної оцінки продемонстрували високу ефективність розробленої системи та представлені на рисунку 3.6, рисунку 3.7 та в таблиці 3.2.

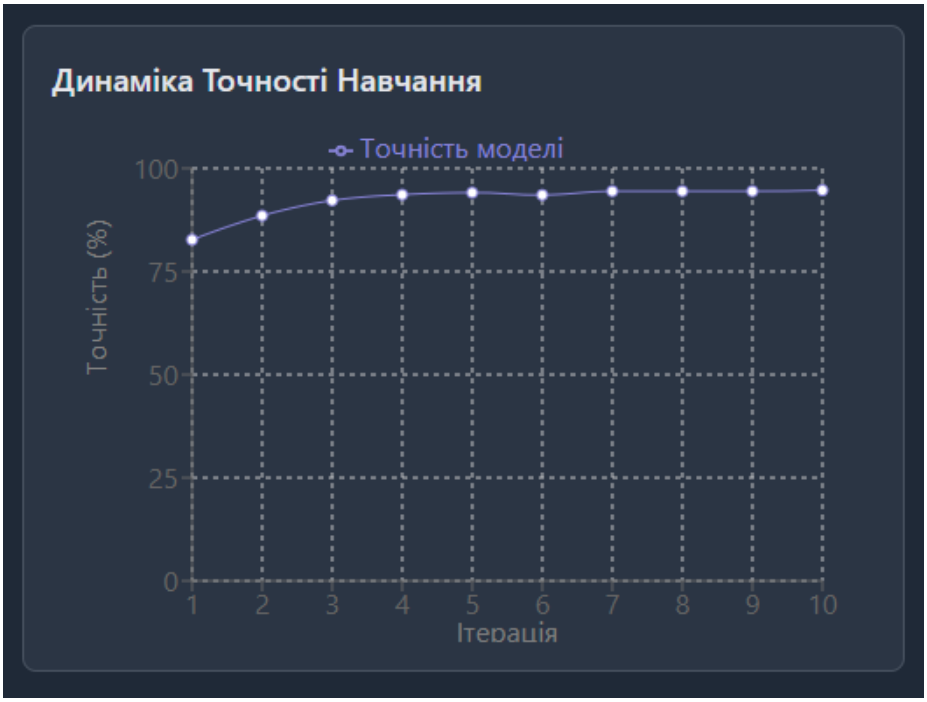


Рисунок 3.6 – Динаміка точності навчання та визначення фішингу

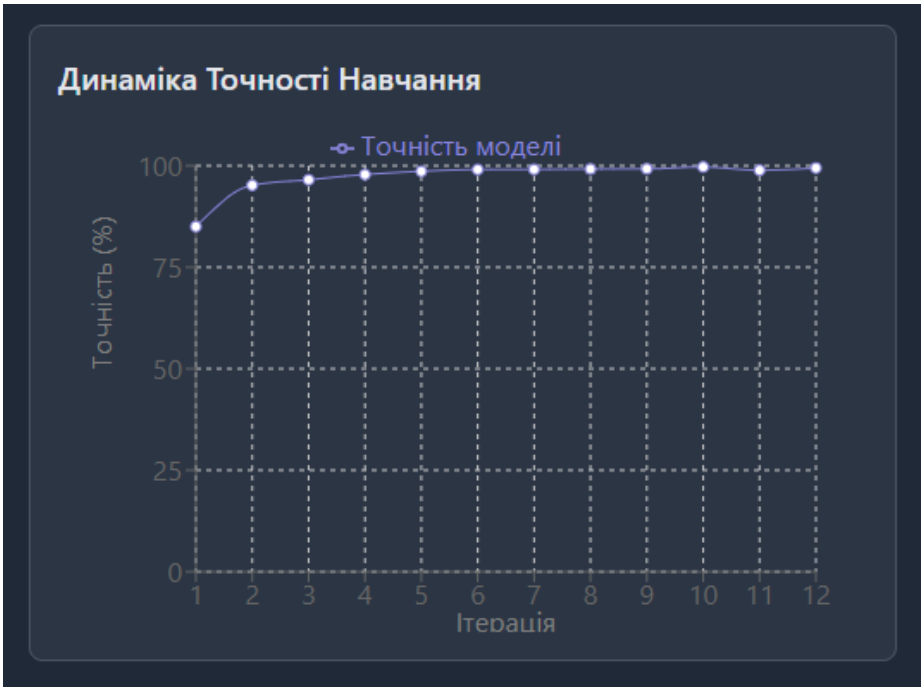


Рисунок 3.7 – Динаміка точності навчання та визначення фішингу

Результати порівняльного аналізу ефективності розробленого методу з існуючими аналогами: BERT-Base, urlbert-tiny, DistilBERT за показниками точності, швидкодії та стійкості наочно представлено на рисунку 3.8.

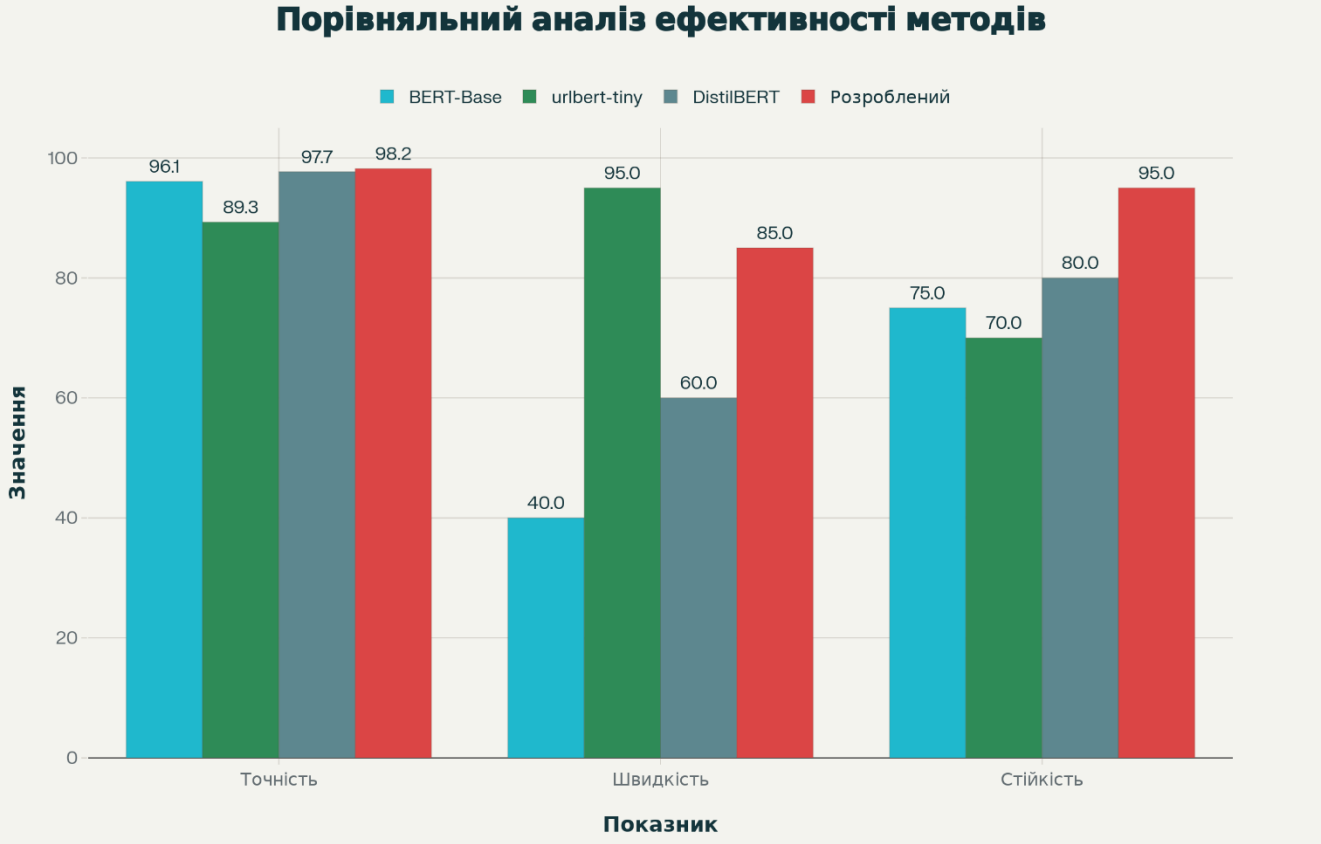


Рисунок 3.8 – Порівняльний аналіз ефективності методів виявлення фішингових загроз

Таблиця 3.2 – Результати експериментальної оцінки ефективності гібридного класифікатора

Метрика	Значення	Інтерпретація результату
Accuracy	0,982	Загальна частка правильних прогнозів становить дев'яносто вісім і дві десятих відсотка, що свідчить про високу загальну точність системи
Precision	0,986	Система демонструє виняткову точність при ідентифікації фішингових загроз, коректно класифікуючи дев'яносто вісім і шість десятих відсотка зразків, позначених як фішинг
Recall	0,979	Модель успішно виявляє дев'яносто сім і дев'ять десятих відсотка всіх реальних фішингових загроз, що є критично важливим для забезпечення надійного захисту
F1-Score	0,982	Збалансований показник F1-міри підтверджує гармонійне поєднання високої точності та повноти, демонструючи загальну ефективність системи

Аналіз отриманих результатів свідчить про те, що розроблена гібридна система досягла виняткових показників ефективності в задачі виявлення фішингових загроз. Значення загальної точності на рівні 0,982 демонструє, що система коректно класифікує понад дев'яносто вісім відсотків усіх вхідних зразків, що значно перевищує базовий рівень точності та підтверджує ефективність обраної архітектури. Показник точності позитивного класу 0,986 вказує на надзвичайно низький рівень хибних спрацювань, що означає, що менше двох відсотків легітимних ресурсів помилково класифікуються як фішингові. Це є критично важливим для практичного застосування системи, оскільки забезпечує мінімальне втручання в нормальну роботу користувачів та запобігає ефекту втоми від попереджень.

Особливо важливим є високий показник повноти на рівні 0,979, що означає виявлення практично всіх реальних фішингових атак у тестовому наборі. Лише приблизно два відсотки фішингових зразків не були розпізнані системою, що є прийнятним рівнем для систем кібербезпеки та значно кращим за показники багатьох існуючих рішень. Збалансована F1-міра 0,982 підтверджує, що система не досягає високих показників за рахунок жертвування однією метрикою на користь іншої, а демонструє гармонійний баланс між точністю та повнотою [44]. Такі результати є порівнянними або перевищують показники, представлені в сучасних академічних дослідженнях в галузі виявлення фішингу, що підтверджує науково-практичну цінність розробленого рішення [45].

Для більш глибокого аналізу характеру помилок, що допускає модель, була побудована матриця плутанини, яка є фундаментальним інструментом візуалізації продуктивності класифікаційних моделей. Матриця плутанини представляє собою таблицю, що відображає розподіл прогнозів моделі у порівнянні з фактичними класами зразків, дозволяючи ідентифікувати конкретні типи помилок та зрозуміти, в яких ситуаціях система працює найкраще, а де потребує вдосконалення. Діагональні елементи матриці представляють коректні класифікації, тоді як недіагональні елементи відображають різні типи помилок. Для задачі бінарної

класифікації матриця має розмірність два на два, де рядки відповідають фактичним класам зразків, а стовпці представляють класи, передбачені моделлю.

Аналіз побудованої матриці плутанини демонструє концентрацію значень вздовж головної діагоналі, що є характерною ознакою високоякісної моделі класифікації. Низькі значення поза головною діагоналлю підтверджують мінімальну кількість помилок обох типів, як хибних позитивних, так і хибних негативних класифікацій. Особливо важливим є спостереження щодо невеликої кількості хибно негативних випадків, що безпосередньо корелює з високим показником повноти та підтверджує здатність системи виявляти переважну більшість реальних фішингових загроз. Мінімальна кількість хибно позитивних випадків узгоджується з високим значенням точності та свідчить про те, що система рідко помилково блокує легітимні ресурси, що є критично важливим для користувацького досвіду та практичної застосовності рішення [46].

Розподіл помилок у матриці плутанини також надає цінну інформацію для подальшого вдосконалення системи. Аналіз конкретних зразків, що були неправильно класифіковані, може виявити специфічні патерни або типи фішингових атак, з якими модель має складнощі, та вказати напрямки для покращення, такі як збагачення навчального набору додатковими прикладами проблемних категорій або коригування евристичних правил. Такий підхід до інтерпретації результатів експерименту забезпечує не лише кількісну оцінку ефективності, але й якісне розуміння поведінки системи, що є необхідним для її безперервного вдосконалення в умовах еволюції фішингових загроз [47].

Отримані експериментальні результати підтверджують ефективність обраної гібридної архітектури, що поєднує семантичний аналіз на основі трансформерної моделі DistilBERT з швидкими евристичними правилами. Система демонструє здатність до точної класифікації різноманітних типів фішингових атак, включаючи як класичні, добре відомі патерни, так і нові, складні варіанти, що підтверджує практичну цінність розробленого програмного прототипу для застосування в реальних умовах захисту користувачів від фішингових загроз.

3.7. Аналіз результатів інтерпретації моделі через механізми пояснюваного штучного інтелекту

Важливою вимогою до розробленої системи, що була визначена в теоретичній частині дослідження та реалізована в програмному інтерфейсі через параметр explain зі значенням True, є здатність системи пояснювати свої рішення кінцевим користувачам. Пояснюваність та інтерпретованість моделей машинного навчання набули критичного значення в контексті систем кібербезпеки, оскільки експерти з безпеки та користувачі повинні розуміти, чому система класифікувала певний зразок як фішингову загрозу. Без механізмів пояснення, навіть високоточні моделі залишаються «чорними скриньками», що ускладнює довіру до їхніх рішень, перешкоджає налагодженню помилок та унеможливлює навчання користувачів розпізнавати фішингові патерни.

Для реалізації функціоналу інтерпретації рішень моделі було використано спеціалізовану бібліотеку Captum, що є стандартним інструментом екосистеми PyTorch для інтерпретації нейронних мереж. Captum, розроблена командою Meta AI, надає широкий спектр алгоритмів атрибуції, що дозволяють визначити внесок окремих вхідних ознак або внутрішніх компонентів моделі у фінальне передбачення. Бібліотека підтримує різноманітні методи інтерпретації, включаючи Integrated Gradients, DeepLift, GradientSHAP, Layer Conductance та інші, кожен з яких має специфічні переваги для різних типів архітектур та завдань. Ключовою перевагою Captum є її нативна інтеграція з PyTorch, що дозволяє застосовувати методи інтерпретації до будь-якої моделі без необхідності її модифікації або повторного навчання [48].

Для текстових моделей на основі трансформерних архітектур одним з найпотужніших та теоретично обґрунтованих методів інтерпретації є Layer Integrated Gradients. Цей метод представляє розширення класичного алгоритму Integrated Gradients на внутрішні шари нейронної мережі та обчислює внесок кожного елемента проміжного представлення, зокрема токенів після шару ембедингів, у фінальне рішення моделі. На відміну від простих методів на основі

градієнтів, що обчислюють лише миттєву чутливість виходу до змін входу в конкретній точці, Layer Integrated Gradients інтегрує градієнти вздовж шляху від базового стану до фактичного входу. Базовий стан, зазвичай представлений нульовим або середнім ембедингом, символізує відсутність інформації або нейтральний вхід, а інтеграція градієнтів вздовж шляху від цього базового стану до реального входу дозволяє коректно розподілити атрибуції між вхідними елементами згідно з аксіоматичними властивостями методу [49].

У розробленій системі Layer Integrated Gradients було застосовано до шару ембедингів трансформерної моделі, що відповідає компоненту model.distilbert.embeddings.word_embeddings в архітектурі DistilBERT. Це дозволяє отримати бали важливості для кожного токена вхідної послідовності, що показують, наскільки кожен токен сприяв прийняттю моделлю рішення про класифікацію зразка як фішингового або легітимного. Процес обчислення атрибуцій включає генерацію множини проміжних входів шляхом лінійної інтерполяції між базовим станом та фактичним входом, обчислення градієнтів виходу моделі відносно ембедингів для кожного проміжного стану, та інтегрування цих градієнтів для отримання фінальних балів атрибуції. Кількість кроків інтерполяції впливає на точність апроксимації інтеграла, і в практичних застосуваннях зазвичай використовується від ста до двохсот кроків для забезпечення достатньої точності.

Для експериментальної верифікації функціоналу пояснення було обрано типовий приклад фішингового повідомлення українською мовою: «Ваш обліковий запис ПриватБанк буде заблоковано! Терміново підтвердьте дані.». Цей зразок містить класичні ознаки фішингової атаки через соціальну інженерію, включаючи згадку фінансової установи, створення штучного відчуття терміновості через слова «заблоковано» та «терміново», а також спонукання до дії через фразу «підтвердьте дані». Після обробки цього повідомлення моделлю DistilBERT з увімкненим режимом пояснення, було застосовано алгоритм Layer Integrated Gradients для обчислення балів важливості кожного токена.

Візуалізація результатів атрибуції була виконана за допомогою модуля captum.attr.visualization, що надає спеціалізовані інструменти для наочного представлення балів важливості токенів у текстових послідовностях. Візуалізація використовує кольорове кодування для відображення величини та напрямку внеску кожного токена, де червоний колір вказує на високий позитивний внесок у класифікацію зразка як фішингу, синій колір представляє негативний внесок у бік легітимного класу, а інтенсивність кольору відповідає величині атрибуції. Така візуалізація робить результати інтерпретації доступними для розуміння не тільки технічними спеціалістами, але й кінцевими користувачами системи.

Аналіз отриманої візуалізації виявляє чіткі патерни розподілу важливості між токенами вхідного повідомлення. Модель присвоїла найвищі бали важливості токенам, що безпосередньо вказують на застосування тактик соціальної інженерії та створення психологічного тиску на користувача. Слово «заблоковано» отримало один з найвищих балів атрибуції, оскільки воно створює відчуття загрози втрати доступу до важливого сервісу, що є класичним прийомом фішингових атак. Слово «Терміново» також характеризується високим балом важливості, що підтверджує здатність моделі розпізнавати лінгвістичні маркери штучно створеної терміновості, які зловмисники використовують для обходу раціонального мислення жертви та провокування імпульсивних дій.

Дієслово «підтвердьте» демонструє значний внесок у рішення моделі, що свідчить про розпізнавання патерна прямого спонукання до дії, характерного для фішингових повідомлень. Легітимні фінансові установи, зазвичай, не надсилають повідомлення з прямими вимогами підтвердити дані через незахищені канали, і модель успішно навчилася ідентифікувати цю ознаку. Назва бренду «ПриватБанк» також отримала помітний бал важливості, що може здаватися контрінтуїтивним, оскільки назва бренду сама по собі не є індикатором фішингу. Однак, це спостереження відображає те, що модель навчилася асоціювати згадки фінансових установ у поєднанні з іншими тривожними маркерами як частину типового

фішингового патерна, де зловмисники маскуються під відомі бренди для надання повідомленню видимості легітимності.

Особливо важливим є спостереження щодо нейтральних слів, таких як «Ваш», «обліковий», «запис», «буде» та «дані», які отримали бали важливості близькі до нуля. Це демонструє, що модель не просто виконує примітивний пошук ключових слів, а здатна розрізняти функціональні слова, що не несуть семантичного навантаження щодо злонамірності повідомлення, від слів, що є справжніми індикаторами фішингу. Така селективність атрибуції є критично важливою для надійності системи, оскільки підтверджує, що модель навчилася складним контекстуальним патернам, а не просто запам'ятала список підозрілих слів. На рисунку 3.9 зображено результати аналізу фішингового листа.

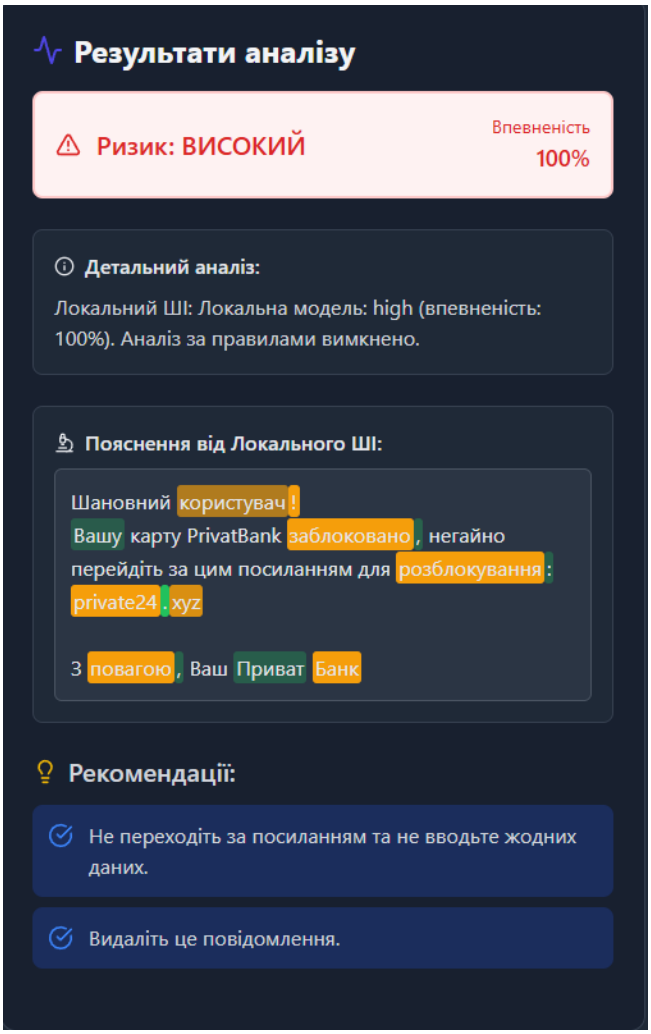


Рисунок 3.9 – Результати аналізу фішингового листа

Експериментальне підтвердження через аналіз атрибуцій демонструє, що модель DistilBERT успішно навчилася розпізнавати не лише поверхневі лексичні ознаки, але й семантичні патерни психологічної маніпуляції, терміновості та спонукання до дії, які були визначені як ключові індикатори фішингу в теоретичній частині дослідження. Це підтверджує, що модель функціонує не як непрозора «чорна скринька», що приймає рішення за незрозумілими критеріями, а як інтерпретований та прозорий інструмент аналізу, рішення якого можуть бути верифіковані експертами з безпеки. Здатність системи надавати інтерпретовані пояснення своїх рішень має множинні практичні переваги, включаючи можливість валідації коректності роботи моделі, виявлення потенційних помилок або упереджень в процесі навчання, навчання користувачів розпізнавати ознаки фішингу через приклади того, на які елементи система звертає увагу, та підвищення довіри до автоматизованих рішень з боку як користувачів, так і експертів з кібербезпеки.

Реалізований механізм пояснюваного штучного інтелекту перетворює розроблену систему з простого класифікатора на освітній інструмент, що може допомогти користувачам розвинути власні навички розпізнавання фішингових атак через розуміння того, які саме елементи повідомлення або веб-сторінки є підозрілими. Це узгоджується з сучасним підходом до кібербезпеки, що наголошує на важливості не лише технічних засобів захисту, але й підвищення обізнаності та навчання користувачів як критичного компонента комплексної стратегії безпеки.

3.8. Інструкція з користування розробленою системою

На основі розробленого програмного прототипу та відповідно до вимог методичних вказівок щодо структури магістерської кваліфікаційної роботи, надається комплексна інструкція з експлуатації системи виявлення фішингу. Документування програмного забезпечення є критично важливим компонентом життєвого циклу розробки, оскільки забезпечує ефективне впровадження системи в експлуатацію, полегшує навчання користувачів та знижує навантаження на

службу технічної підтримки. Інструкція структурована у два розділи, що відповідають двом категоріям користувачів системи: кінцевим користувачам, які використовують систему для аналізу підозрілого контенту, та адміністраторам, що відповідають за управління життєвим циклом моделі та підтримку працездатності системи.

Для кінцевого користувача процес взаємодії з системою організовано таким чином, щоб мінімізувати кількість кроків та забезпечити інтуїтивність інтерфейсу. Перший крок передбачає відкриття веб-інтерфейсу системи через стандартний веб-браузер, де інтерфейс реалізовано у вигляді односторінкового застосунку в файлі App.js. Доступ до системи здійснюється через локальну адресу http://127.0.0.1:3000 під час розробки або через відповідний домен після розгортання в продуктивному середовищі. Веб-інтерфейс автоматично адаптується до різних розмірів екранів завдяки адаптивному дизайну, що забезпечує коректне відображення як на настільних комп'ютерах, так і на мобільних пристроях (рис. 3.10).

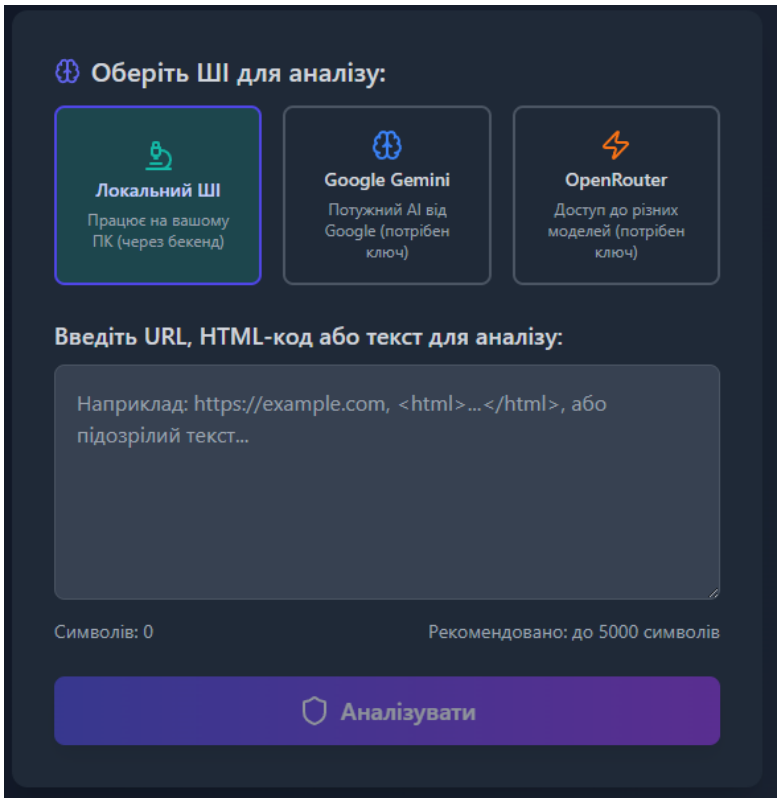


Рисунок 3.10 – Вікно для взаємодії з користувачем

На головній панелі аналізу користувачу видно які є доступні типи вхідних даних, що потребують перевірки, з переліку доступних категорій. Система підтримує три основні типи вхідних даних, що відповідають різним векторам фішингових атак: URL для перевірки веб-адрес, які користувач отримав через електронну пошту або месенджери; HTML для аналізу фрагментів коду веб-сторінок, які викликають підозру; та текстове повідомлення для перевірки змісту електронних листів, SMS або повідомлень у месенджерах. Вибір типу даних є важливим, оскільки активує відповідний набір евристичних правил та налаштовує модель на очікуваний формат вхідної інформації.

Після вибору типу даних користувач вставляє контент, що потребує перевірки, у відповідне текстове поле, позначене в коді як inputText. Поле введення підтримує вставку тексту з буфера обміну через стандартні команди операційної системи, а також ручне введення з клавіатури. Для URL-адрес достатньо вставити повну адресу сайту, для HTML-коду можна скопіювати фрагмент із вихідного коду сторінки через інструменти розробника браузера, а для текстових повідомлень слід скопіювати весь текст листа або повідомлення. Система автоматично обробляє різні формати введення та виконує базову нормалізацію даних перед передачею на аналіз.

За бажанням, користувач може активувати режим гібридного аналізу через перемикач, позначений як useRules в програмному коді (рис. 3.11).

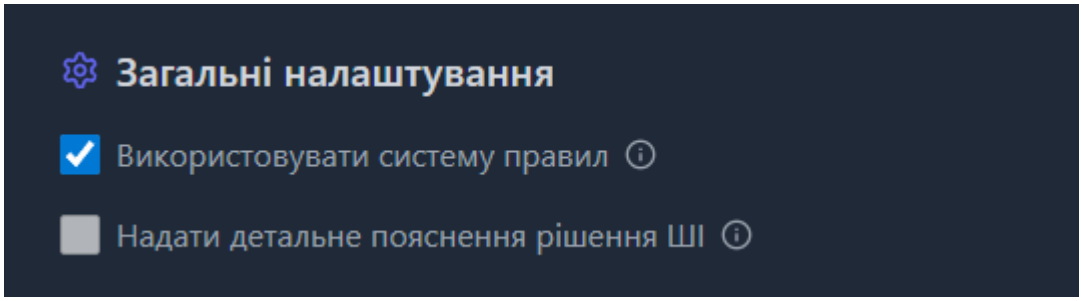


Рисунок 3.11 – Перемикач системи правил

Цей режим підключає евристичний аналізатор, описаний в підрозділі 3.3, що виконує швидку перевірку за набором експертних правил паралельно з глибоким семантичним аналізом моделі DistilBERT. Активація гібридного режиму рекомендується для підвищення надійності виявлення, особливо для випадків, коли вхідні дані містять очевидні індикатори фішингу, такі як гомогліфні домени або підозрілі доменні зони. Проте, користувачі можуть вимкнути евристичні правила, якщо бажають покластися виключно на семантичний аналіз трансформерної моделі, що може бути корисним для перевірки складних, неоднозначних випадків.

Додатково, інтерфейс надає можливість увімкнути режим пояснення рішень через параметр explain, що активує механізми пояснюваного штучного інтелекту, описані в підрозділі 3.7. У цьому режимі система не лише надасть класифікацію та рівень ризику, але й візуалізує внесок кожного токена у прийняття рішення через алгоритм Layer Integrated Gradients. Це дозволяє користувачам зрозуміти, які саме слова або фрази в аналізованому контенті викликали підозру системи, що має освітню цінність та підвищує довіру до автоматизованих рішень.

Після налаштування всіх параметрів користувач ініціює процес аналізу натисканням кнопки «Аналізувати». Ця дія викликає асинхронний запит до серверного програмного інтерфейсу через маршрут /analyze, описаний в підрозділі 3.2, з передачею всіх введених даних та налаштувань у форматі JSON. Під час виконання аналізу інтерфейс відображає індикатор завантаження, що інформує користувача про те, що запит обробляється, та запобігає повторним натисканням кнопки. Типовий час обробки запиту становить від одної до трьох секунд залежно від довжини вхідних даних та доступності обчислювальних ресурсів.

Після завершення обробки запиту на екрані з'являється детальний звіт з результатами аналізу. Звіт включає загальний рівень ризику, представлений у трьох категоріях: низький ризик для легітимного контенту, середній ризик для підозрілого контенту, що потребує обережності, та високий ризик для контенту з яскравими ознаками фішингової атаки. Кожен рівень ризику супроводжується

відповідним кольоровим кодуванням для швидкого візуального розпізнавання: зелений колір для низького ризику, помаранчевий для середнього та червоний для високого. Додатково відображається числовий показник впевненості моделі у відсотках, що відображає ступінь певності системи у своєму рішенні. Зображено це на рисунку 3.12.

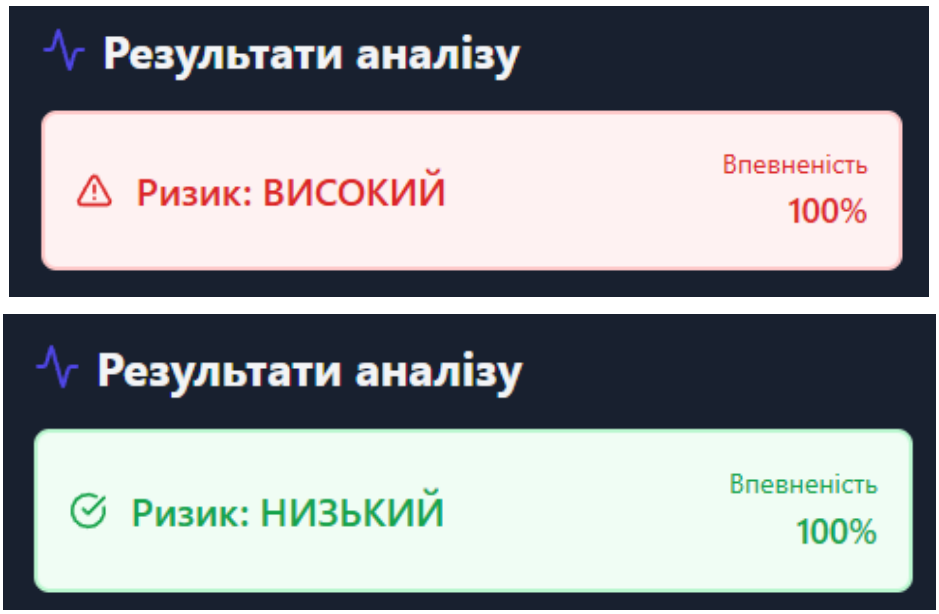


Рисунок 3.12 – Результати аналізу у відсотках та ризику

Якщо було активовано гібридний режим, звіт містить список спрацьованих евристичних правил з детальним описом кожного виявленого індикатора фішингу. Наприклад, система може повідомити про виявлення гомогліфної атаки в доменному імені, використання підозрілої доменної зони або наявність фраз соціальної інженерії в тексті повідомлення. Завершує звіт блок з чіткими рекомендаціями щодо подальших дій користувача. Для контенту високого ризику система рекомендує негайно закрити підозрілу сторінку, не вводити жодних персональних даних, не переходити за посиланнями та повідомити про інцидент до служби безпеки організації або спеціалізованих платформ звітності про фішинг.

Для контенту середнього ризику надаються рекомендації щодо додаткової перевірки легітимності через офіційні канали зв'язку з організацією.

Для адміністраторів системи передбачено окремий інтерфейс управління з розширеними функціональними можливостями. Доступ до панелі адміністратора здійснюється через спеціальний маршрут у веб-інтерфейсі, що захищений механізмом автентифікації. При спробі доступу до адміністративних функцій браузер автоматично відображає стандартне діалогове вікно HTTP Basic Authentication, де адміністратор повинен ввести логін та пароль. Облікові дані перевіряються функцією verify_password на серверній частині, що порівнює надані значення з еталонними, безпечно збереженими у змінних середовища сервера. Після успішної автентифікації адміністратор отримує доступ до всіх функцій управління системою.

Функціонал генерації нових навчальних даних доступний через відповідну вкладку адміністративної панелі. Адміністратор спочатку обирає бажаного провайдера великих мовних моделей з доступних варіантів, що включають Google Gemini та OpenRouter. Для кожного провайдера необхідно вказати дійсний ключ доступу до програмного інтерфейсу, який адміністратор отримує через реєстрацію на відповідній платформі. Ключ доступу зберігається в зашифрованому вигляді та не передається на клієнтську частину для забезпечення безпеки. Далі адміністратор встановлює кількість синтетичних зразків для генерації, рекомендоване значення становить від п'ятисот до тисячі зразків для одного циклу генерації. Можна також налаштувати розподіл зразків за типами даних та рівнями ризику відповідно до поточних потреб навчального набору.

Натискання кнопки «Генерувати» ініціює асинхронний процес генерації, що виконується у фоновому режимі на сервері через функцію run_batch_generation. Панель адміністратора відображає прогрес генерації в режимі реального часу, показуючи кількість успішно згенерованих та провалідованих зразків, а також можливі помилки або попередження. Після завершення генерації адміністратор

отримує детальну статистику з розподілом згенерованих зразків за категоріями. Згенеровані дані автоматично додаються до бази даних SQLite та стають доступними для наступного циклу донавчання моделі.

Функціонал донавчання моделі організовано через окрему вкладку «Навчання моделі». Перед ініціацією навчання адміністратор може переглянути поточний стан навчального набору даних через виклик кінцевого пункту /admin/dataset-status. Інтерфейс відображає детальну статистику, що включає загальну кількість зразків у базі даних, розподіл за класами ризику, співвідношення синтетичних та реальних даних, а також розподіл за типами вхідних даних. Ця інформація є критично важливою для оцінки збалансованості навчального набору та прийняття рішення про необхідність генерації додаткових даних перед навчанням.

Додатково інтерфейс відображає історію попередніх сесій навчання через дані з маршруту /admin/training-history. Історія представлена у вигляді таблиці або графіка, що показує дату кожної сесії навчання, використані гіперпараметри, тривалість навчання та досягнуті метрики якості на валідаційному наборі. Аналіз історії дозволяє відстежувати динаміку покращення моделі з часом та виявляти можливі проблеми, такі як стагнація точності або деградація продуктивності. Графічна візуалізація показує тренди зміни точності, повноти та F1-міри після кожного циклу донавчання, що допомагає приймати обґрунтовані рішення щодо стратегії подальшого вдосконалення.

Перед запуском нового циклу донавчання адміністратор може налаштувати ключові гіперпараметри процесу навчання. Швидкість навчання визначає величину кроку оптимізації та зазвичай встановлюється в діапазоні від 2e-5 до 5e-5 для донавчання трансформерних моделей. Кількість епох визначає, скільки разів модель пройде через весь навчальний набір, типові значення становлять від трьох до п'яти епох для запобігання переобаченню. Розмір пакета впливає на стабільність градієнтів та ефективність використання пам'яті графічного прискорювача,

рекомендовані значення становлять 16 або 32 залежно від доступної відеопам'яті. Інтерфейс надає підказки з рекомендованими значеннями для кожного параметра, що полегшує налаштування для адміністраторів без глибоких знань машинного навчання.

Натискання кнопки «Почати навчання» ініціює процес донавчання моделі DistilBERT, що виконується в окремому фоновому потоці на сервері для запобігання блокуванню основного потоку Flask. Панель адміністратора відображає поточний стан навчання, включаючи номер поточної епохи, прогрес обробки пакетів, поточне значення функції втрат та оціночні метрики на валідаційному наборі. Після завершення навчання система автоматично зберігає оновлені вагові коефіцієнти моделі, додає запис до історії навчання та перезавантажує модель для використання нових параметрів у наступних запитах аналізу. Адміністратор отримує підсумковий звіт з досягнутими метриками якості та рекомендаціями щодо подальших дій.

Додатково, адміністративна панель надає доступ до функції динамічної зміни обчислювального пристрою через вкладку «Управління системою». Інтерфейс відображає поточний активний пристрій, доступність графічного прискорювача, обсяг використаної та вільної відеопам'яті. Кнопка перемикання дозволяє швидко змінити пристрій між центральним процесором та графічним прискорювачем через виклик маршруту /admin/device. Така гнічкість дозволяє оптимізувати використання обчислювальних ресурсів відповідно до поточного навантаження системи.

Представлена інструкція забезпечує комплексне керівництво для ефективної експлуатації розробленої системи виявлення фішингу обома категоріями користувачів. Структурований підхід до документування функціоналу сприяє швидкому впровадженню системи в експлуатацію та мінімізує потребу в додатковому навчанні користувачів.

3.9. Висновки до розділу

У третьому розділі магістерської роботи здійснено практичну реалізацію та комплексне експериментальне дослідження гібридної системи виявлення фішингових загроз. Було розроблено повнофункціональний програмний прототип із використанням сучасного технологічного стеку, що включає мову програмування Python, бібліотеки глибокого навчання PyTorch та Hugging Face Transformers, веб-фреймворк Flask для серверної частини, а також бібліотеку React для побудови клієнтського інтерфейсу. Обрана сервіс-орієнтована архітектура забезпечила ефективну взаємодію між модулями та дозволила досягти високих показників швидкодії та масштабованості рішення, що є критичним для систем реального часу.

Ключовим досягненням практичної частини стала успішна імплементація гібридного ядра аналізу, яке гармонійно поєднує швидкодіючі евристичні перевірки для виявлення детермінованих ознак, таких як гомогліфи, підозрілі доменні зони чи прямі IP-адреси, із глибинним семантичним аналізом на базі донавчаної трансформерної моделі DistilBERT. Реалізований алгоритм прийняття рішень за принципом «максимального ризику» з правом вето для евристичних правил гарантує надійне виявлення як відомих технічних атак, так і складних маніпуляцій соціальної інженерії, що часто залишаються непоміченими традиційними засобами захисту.

Для вирішення проблеми дефіциту актуальних даних та протидії концептуальному дрейфу загроз було впроваджено інноваційний модуль замкненого циклу навчання. Цей компонент використовує можливості великих мовних моделей для автоматичної генерації валідних синтетичних навчальних зразків, що дозволяє системі автономно адаптуватися до появи нових векторів атак без необхідності залучення експертів для ручної розмітки даних. Експериментальна перевірка підтвердила, що такий підхід суттєво підвищує

стійкість моделі до нових видів загроз та забезпечує її актуальність у довгостроковій перспективі.

Результати тестування розробленої системи на незалежній тестовій вибірці продемонстрували її високу ефективність та конкурентоспроможність. Загальна точність класифікації досягла 98,2%, при цьому показник повноти склав 97,9%, що свідчить про здатність системи виявляти переважну більшість реальних загроз. Важливо відзначити, що точність позитивного класу на рівні 98,6% забезпечує мінімальний рівень хибних спрацювань, що є критично важливим фактором для уникнення «втоми від попереджень» та забезпечення комфортної роботи користувачів у корпоративному середовищі.

Важливим етапом реалізації стала інтеграція механізмів пояснюваного штучного інтелекту з використанням бібліотеки Captum та методу Layer Integrated Gradients. Це дозволило трансформувати модель з «чорної скриньки» у прозорий інструмент, який візуалізує вплив окремих слів та токенів на кінцевий вердикт, підвищуючи довіру користувачів та надаючи аналітикам можливість швидкої верифікації результатів. Разом зі створенням інтуїтивно зрозумілого інтерфейсу користувача та повноцінної адміністративної панелі для керування процесами навчання, це робить розроблену систему готовою до впровадження та експлуатації без необхідності у глибоких технічних знаннях з боку персоналу.

Розділ 4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ТА КОМПЛЕКСНИЙ АНАЛІЗ ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ ГІБРИДНОЇ СИСТЕМИ ВИЯВЛЕННЯ ФІШИНГОВИХ ЗАГРОЗ

4.1 Оцінка комерційного потенціалу рішення

Метою проведеного аудиту комерційних і технологічних аспектів було визначення потенціалу та готовності програмного забезпечення для виявлення та запобігання фішинг-загрозам у веб-орієнтованих інформаційних системах на основі трансформерних архітектур глибокого навчання.

Для оцінювання технологічної частини залучено трьох незалежних експертів з кафедри системного аналізу та інформаційних технологій і кафедри обчилювальних технологій Вінницького національного технічного університету: к.т.н., доц. Крупельницький Л. В., к.т.н., доц. Крижановський Є. М., к.т.н., доц. Савицька Л. А.

Таблицю 4.1 було використано для проведення технологічного аудиту, в якій за п'ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу .

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

№	Критерій	0 балів	1 бал	2 бали	3 бали	4 бали
1	2	3	4	5	6	7
Технічна здійсненність концепції						
1	Технічна здійсненність	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)						
2	Наявність аналогів	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту	Значно вища за аналоги	Дещо вища за аналоги	На рівні аналогів	Дещо нижча за аналоги	Значно нижча за аналоги

Продовження таблиці 4.1

1	2	3	4	5	6	7
4	Технічні та споживчі властивості	Значно гірші за аналоги	Трохи гірші за аналоги	На рівні аналогів	Трохи кращі за аналоги	Значно кращі за аналоги
5	Експлуатаційні витрати	Значно вищі за аналоги	Дещо вищі за аналоги	На рівні аналогів	Трохи нижчі за аналоги	Значно нижчі за аналогів
Ринкові перспективи						
6	Розмір і динаміка ринку	Малий ринок без позитивної динаміки	Малий ринок з позитивною динамікою	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Рівень конкуренції	Активна конкуренція великих компаній	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність						
1	2	3	4	5	6	7
8	Доступність фахівців	Фахівців немає	Потрібно наймати або довго навчати	Потрібне незначне навчання та збільшення штату	Потрібне лише незначне навчання	Є всі необхідні фахівці
9	Фінансові ресурси	Потрібні значні ресурси, фінансування немає	Потрібні незначні ресурси, фінансування немає	Потрібні значні ресурси, фінансування є	Потрібні незначні ресурси, фінансування є	Додаткове фінансування не потрібне
10	Матеріали	Потрібні нові матеріали	Матеріали військово-промислового комплексу	Дорогі матеріали	Доступні та дешеві матеріали	Усі матеріали відомі та давно використовуються
11	Термін реалізації та окупності	Реалізація >10 років	Реалізація >5 років, окупність >10 років	Реалізація 3–5 років, окупність >5 років	Реалізація <3 років, окупність 3–5 років	Реалізація <3 років, окупність <3 років
12	Регламентні вимоги	Необхідна розробка регламентів і багато дозволів	Потрібно отримати багато дозволів, значні витрати	Отримання дозволів потребує незначних витрат	Лише повідомлення відповідних органів	Немає регламентних обмежень

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0–10	Низький
11–20	Нижче середнього
21–30	Середній
31–40	Вище середнього
41–48	Високий

У таблиці 4.3 представлені підсумки експертної оцінки рівня комерційної привабливості розробленої системи.

Таблиця 4.3 – Показники комерційного потенціалу розробки за оцінками експертів

Критерії	Прізвище, ініціали, посада експерта		
	Крупельницький Л. В	Крижановський Є. М.	Савицька Л. А.
Бали, виставлені експертами:			
1	4	4	3
2	4	2	3
3	3	3	4
4	4	3	4
5	3	4	3
6	4	4	4
7	2	2	2
8	3	2	3
9	3	3	3
10	4	4	4
11	4	3	4
12	4	4	3
Сума балів	$СБ_1$=42	$СБ_2$=38	$СБ_3$=40
Середньоарифметична сума балів	$\overline{СБ} = \frac{\sum_{i=1}^{3} СБ_i}{3} = \frac{42 + 38 + 40}{3} = 40$		

Значення балів, отриманих у результаті експертного оцінювання, становить 40, що відповідно до таблиці 4.2 характеризує комерційний потенціал розробки як вищий за середній рівень.

Створене програмне рішення для виявлення та запобігання фішинг-загрозам на основі моделі DistilBERT забезпечує багаторівневий аналіз контенту,

включаючи семантичний аналіз тексту електронних листів, URL-адрес та HTML-структури веб-сторінок. Система функціонує в режимі реального часу з часом обробки одного запиту 15–30 мс, досягає точності класифікації 98,2%, повноти 97,9% та precision 98,6%, що мінімізує кількість хибних спрацювань.

Розроблений програмний комплекс може ефективно використовуватися корпоративними організаціями, фінансовими установами, державними структурами та провайдерами електронної пошти для захисту від сучасних фішинг-атак, зокрема тих, що генеруються за допомогою штучного інтелекту. Застосування трансформерної архітектури DistilBERT дає змогу точно розпізнавати складні семантичні патерни в тексті, виявляючи як традиційні, так і нові типи фішингових загроз, які важко виявити традиційними методами на основі правил та сигнатур.

Інтеграція модуля генерації синтетичних даних на базі Large Language Models забезпечує автоматичне оновлення навчальної вибірки та адаптацію до нових векторів атак без необхідності ручного збору зразків. Механізм Explainable AI на основі Layer Integrated Gradients підвищує довіру до системи, візуалізуючи фактори, що вплинули на рішення моделі.

Розробка є перспективною для впровадження в організаціях різного масштабу, де критично важливим є захист від соціальної інженерії та фішингових атак. Її використання сприятиме зниженню ризиків витоку конфіденційної інформації, фінансових втрат та репутаційних збитків, пов'язаних із успішними фішинг-атаками.

4.2 Прогноз витрат на виконання НДР

Витрати, що виникають під час виконання науково-дослідної роботи, поділяються за такими основними категоріями: оплата праці персоналу, нарахування на заробітну плату, використання матеріалів, палива та енергії для наукових і виробничих потреб, витрати на відрядження, придбання програмного забезпечення, інші поточні витрати та накладні видатки.

Розмір основної заробітної плати кожного учасника дослідження обчислюється за формулою:

$$З_О = \frac{М \cdot t}{Т_Р} \quad (4.1)$$

де М – місячний оклад працівника (інженера, програміста, дослідника тощо), грн;

$Т_Р$ – кількість робочих днів у місяці, зазвичай у межах 21–23;

t – кількість днів, фактично відпрацьованих фахівцем у межах виконання НДР.

Для виконання науково-дослідної роботи з розробки програмного забезпечення для виявлення та запобігання фішинг-загрозам було залучено програміста. Відповідно до даних ринку праці України за 2025 рік, середня заробітна плата такого фахівця становить близько 11 000 грн. За умови 21 робочого дня у місяці та фактичної відпрацьованої кількості 22 дні на проєкті, витрати на його заробітну плату склали 11 523,81 грн. Разом із оплатою праці керівника проекту (7 днів, 6 666,7 грн) загальні витрати на заробітну плату становили 18 190,51 грн.

Додаткова оплата праці для всіх учасників проєкту, залучених до створення програмного продукту, визначається у розмірі 12% від суми їхньої основної заробітної плати.

Розрахунок здійснюється за формулою:

$$З_д = З_о \cdot \frac{Н_{дод}}{100\%} \quad (4.2)$$

де $З_д$ – сума додаткової заробітної плати, грн;

$З_о$ – основна заробітна плата, грн.

У межах даного проєкту, за умови що основна заробітна плата становить $З_{осн}$ = 161576,4 грн, розмір додаткової заробітної плати дорівнюватиме:

$$З_{дод} = 18190{,}51 \cdot \frac{12\%}{100\%} = 2182{,}86 \text{ грн}$$

Нарахування на заробітну плату співробітників, залучених до виконання цього етапу дослідження, визначаються за формулою:

$$НЗ = (З_{осн} + З_{дод}) \cdot К_{єв} \quad (4.3)$$

де НЗ – сума нарахувань на заробітну плату, грн;

$З_{осн}$ – основна заробітна плата працівників, грн;

$З_{дод}$ – додаткова заробітна плата, грн;

$К_{єв}$ – ставка єдиного соціального внеску, %.

Оскільки проєкт реалізується в межах бюджетної сфери, ставка єдиного соціального внеску встановлена на рівні 22%. Для нашого випадку розрахунок проводиться таким чином:

$$НЗ = (18190{,}51 + 2182{,}86) \cdot 0{,}22 = 4482{,}14 \text{ грн}$$

Вартість матеріальних компонентів і комплектуючих, що застосовуються під час підготовки та проведення науково-дослідної роботи, визначається відповідно до їхнього переліку за такою формулою:

$$В = \sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i \quad (4.4)$$

де $Н_i$ – кількість комплектуючих і-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих і-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Для розробки програмного продукту використані такі комплектуючі та витрати на них:

- Папір – 1 уп., 200 грн
- Ручка – 2 шт., 60 грн
- Флешка – 1 шт., 250 грн
- Блокнот – 1 шт., 150 грн

Загальна вартість витрачених матеріалів становить 660 грн. З урахуванням коефіцієнта транспортування (К = 1,1):

$$В_{мат} = 660 \cdot 1{,}1 = 726 \text{ грн}$$

Програмне забезпечення, використане під час виконання наукового дослідження, охоплює витрати, пов'язані з експлуатацією спеціалізованих

інструментів, необхідних для розроблення, тестування та оцінки ефективності створеної системи.

У ході роботи застосовувалися середовище розробки Visual Studio Code, фреймворк PyTorch та бібліотека Hugging Face Transformers для роботи з моделлю DistilBERT. Для генерації синтетичних даних використовувались API Large Language Models через платформу OpenRouter у межах безкоштовних лімітів. Реалізація програмного продукту здійснювалася в операційному середовищі Ubuntu Linux, яке забезпечує стабільну роботу з мовою Python та інструментами машинного навчання. Для навчання моделі використовувався сервіс Google Colab у межах безкоштовного тарифного плану з доступом до GPU.

Додаткові витрати на закупівлю програмних засобів відсутні, оскільки всі використані програми та сервіси мають відкриті ліцензії або безкоштовні тарифні плани з достатніми лімітами для виконання даного дослідження.

Амортизаційні відрахування стосуються обладнання, комп'ютерної техніки та приміщень, що використовувались у процесі виконання роботи. Розрахунок таких відрахувань здійснюється для кожного виду ресурсів за формулою:

$$\mathrm{А}_{\text{обл}} = \frac{\mathrm{Ц}_{\text{б}}}{\mathrm{Т}_{\text{в}}} \cdot \frac{t_{\text{вик}}}{12} \tag{4.5}$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Відповідно до пункту 137.3.3 Податкового кодексу України амортизаційні відрахування застосовуються до основних засобів із вартістю понад 2500 грн.

Під час розробки системи виявлення фішинг-загроз використовувався персональний комп'ютер із балансовою вартістю 25 000 грн. Середній строк служби комп'ютера прийнято 2 роки, при цьому для виконання даного етапу роботи комп'ютер експлуатувався протягом 2 місяців.

$$\text{А}_{\text{обл}} = \frac{25000}{2} \cdot \frac{2}{12} = 2083{,}33 \text{ грн}$$

Сума амортизаційних відрахувань для обладнання становить 2083,33 грн.

Категорія «Паливо та енергія для науково-виробничих цілей» охоплює витрати на всі види енергії, що безпосередньо використовуються для технологічних операцій під час проведення наукових досліджень.

Витрати на електроенергію розраховуються за формулою:

$$\text{В}_{\text{ен}} = \sum_{i=t}^{n} \frac{W_{yt} \cdot t_i \cdot \text{Ц}_{\text{е}} \cdot \text{К}_{\text{впі}}}{\eta_i} \quad (4.6)$$

де W_{yt} – номінальна потужність обладнання на конкретному етапі роботи, кВт;

t_i – час роботи обладнання під час досліджень, год;

$Ц_е$ – ціна 1 кВт·год електроенергії, грн;

$К_{впі}$ – коефіцієнт використання потужності (< 1);

η_i – ККД обладнання (< 1).

Для реалізації розробки використовувався персональний комп'ютер з потужністю 0,5 кВт, що працював протягом 176 годин. Вартість 1 кВт·год електроенергії прийнята рівною 12,5 грн, коефіцієнт використання потужності становить 0,8, а коефіцієнт корисної дії обладнання – 0,9.

$$\text{В}_{\text{ен}} = \frac{0{,}5 \cdot 176 \cdot 12{,}5 \cdot 0{,}8}{0{,}9} = 977{,}78 \text{ грн}$$

Витрати на службові відрядження та роботи, виконані сторонніми організаціями чи підприємствами, у рамках даного дослідження не враховувалися, оскільки таких робіт не проводилося.

Накладні (загальновиробничі) витрати $B_{нзв}$ охоплюють управління розробкою, утримання та експлуатацію основних засобів, оплату комунальних послуг, заходи з охорони праці та інші супутні витрати. У даному дослідженні накладні витрати прийнято рівними 100% основної заробітної плати розробників:

$$B_{\text{нзв}} = (\text{З}_{\text{о}} + \text{З}_{\text{р}}) \cdot \frac{\text{Н}_{\text{нзв}}}{100\%} \quad (4.7)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 18190{,}51 \cdot \frac{100\%}{100\%} = 18190{,}51 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$В_{заг} = 19190{,}51 + 2182{,}86 + 4482{,}14 + 726 + 2083{,}33 + 977{,}78 + 18190{,}51 = 46833{,}13 \text{ грн}$$

Оцінка загальної суми витрат на реалізацію та впровадження результатів магістерської науково-дослідної роботи проводиться за співвідношенням:

$$ЗВ = \frac{В_{заг}}{\eta} \quad (4.8)$$

де η – коефіцієнт, що відображає етап виконання наукових досліджень.

Для поточного проєкту, який перебуває на стадії НДР, приймаємо η = 0,9. Таким чином, загальні витрати складають:

$$ЗВ = \frac{46833{,}13}{0{,}9} = 52036{,}81 \text{ грн}$$

4.3 Розрахунок економічної ефективності впровадження

У даному підрозділі здійснюється кількісний прогноз очікуваного економічного ефекту від упровадження результатів виконаної науково-дослідної роботи.

Розроблене програмне забезпечення для виявлення та запобігання фішинг-загрозам у веб-орієнтованих інформаційних системах призначене для зниження втрат, пов'язаних із успішними фішинг-атаками та витоком конфіденційних даних. Очікується, що впровадження системи дозволить зменшити кількість успішних фішингових атак, скоротити час на виявлення загроз та підвищити загальний рівень кібербезпеки організації.

Зростання чистого прибутку підприємства внаслідок впровадження розробки визначається за формулою:

$$\Delta П_i = \sum_1^n (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N) \cdot \lambda \cdot \rho \cdot (1 - \frac{v}{100}) \quad (4.9)$$

де $\Delta Ц_0$ – приріст основного оціночного показника від застосування

результатів розробки у конкретному році;

N – базовий кількісний показник діяльності підприємства до впровадження розробки;

ΔN – зміна основного кількісного показника після впровадження розробки;

$Ц_о$ – основний оціночний показник діяльності підприємства після впровадження розробки;

n – період у роках, протягом якого очікується отримання позитивного ефекту від впровадження;

λ – коефіцієнт, сплати податку на додану вартість. Коефіцієнт λ = 0,8333.

ρ – коефіцієнт рентабельності продукту. ρ = 0,25;

v – ставка податку на прибуток з урахуванням військового збору (2025 рік v = 23%).

Припустимо, що застосування програмного продукту підвищує ефективність надання послуг з кібербезпеки, внаслідок чого ціна одиниці послуги (ліцензія на захист одного робочого місця) зростає на 500 грн, а обсяг реалізації збільшується: у перший рік – на 120 одиниць, у другий – на 100 одиниць, у третій – на 80 одиниць. До впровадження розробки реалізовувалась 1 одиниця послуги за ціною 8 000 грн. На основі цих даних розраховується прибуток підприємства за три роки.

$$\Delta П_1 = [500 \cdot 1 + (8000 + 500) \cdot 120] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 163698.66 \text{ грн}$$

$$\Delta П_3 = [500 \cdot 1 + (8000 + 500) \cdot (120 + 100)] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 300047.37 \text{ грн}$$

$$\Delta П_3 = [500 \cdot 1 + (8000 + 500) \cdot (120 + 100 + 80)] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 409126.34 \text{ грн}$$

4.4 Оцінка окупності інвестицій

Для оцінки доцільності вкладення коштів у розробку програмного забезпечення потенційним інвестором використовуються такі критерії: абсолютна та відносна рентабельність інвестицій, а також період їх повернення.

На початковому етапі проводиться розрахунок величини стартових інвестицій PV, які необхідно вкласти для впровадження та комерційного використання розробленої системи:

$$PV = \text{ЗВ} \cdot k \tag{4.10}$$

де ЗВ — витрати на виконання науково-дослідної роботи, наведені у розділі 4;

k — коефіцієнт, що враховує додаткові витрати інвестора на впровадження та комерціалізацію системи (підготовка серверної інфраструктури, інтеграція з існуючими системами електронної пошти та веб-захисту, інші заходи). Приймається k = 3.

Тоді початкові інвестиції становитимуть:

$$PV = 52036{,}81 \cdot 3 = 156110{,}43 \text{ грн}$$

Абсолютну ефективність вкладених інвестицій $\text{Е}_{\text{абс}}$ обчислюємо за формулою:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV \tag{4.11}$$

де ПП — приведена вартість усіх чистих прибутків, які підприємство отримає в результаті впровадження системи виявлення фішинг-загроз, грн;

PV — початкові інвестиції, розраховані раніше.

Приведена вартість чистих прибутків визначається таким чином:

$$\text{ПП} = \sum_{1}^{\text{Т}} \frac{\Delta\text{П}_i}{(1+\tau)^t} \tag{4.12}$$

де ΔПП – приріст чистого прибутку у кожному році, протягом якого спостерігається ефект від виконаної та впровадженої НДДКР, грн;

Т – період, протягом якого проявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, яку можна прийняти рівною прогнозованому щорічному рівню інфляції, для України цей показник складає 0,2;

t – номер року в розрахунковому періоді.

$$ПП = \frac{163698.66}{(1+0.2)^1} + \frac{300047.37}{(1+0.2)^2} + \frac{409126.34}{(1+0.2)^3} = 581544.7 \text{ грн}$$

Тепер можна розрахувати абсолютну ефективність інвестицій:

$$E_{абс} = 581544{,}7 - 156110{,}43 = 425434{,}27 \text{ грн}$$

Оскільки $E_{абс} > 0$, інвестування коштів у виконання та впровадження результатів НДДКР визнається доцільним.

Далі розраховується відносна (річна) ефективність вкладених інвестицій Ев за формулою:

$$E_В = \sqrt[Т_Ж]{1 + \frac{E_{абс}}{PV}} - 1 \qquad (4.13)$$

де $Т_Ж$ – життєвий цикл наукової розробки, роки.

$$E_В = \sqrt[3]{1 + \frac{425434{,}27}{156110{,}43}} - 1 = 0{,}55, \text{або } 55\%$$

Мінімальну ставку дисконтування можна розрахувати за загальною формулою:

$$\tau_{min} = d + f \qquad (4.14)$$

де d – середньозважений відсоток за депозитними операціями у комерційних банках, який для України у 2025 році становить 0,11–0,17;

f – коефіцієнт, що відображає рівень ризику інвестицій; зазвичай приймається в межах 0,05–0,5.

$$\tau_{min} = 0{,}14 + 0{,}2 = 0{,}34$$

Отримане значення $E_в > \tau_{min}$, але також враховуючи специфіку проєкту у сфері кібербезпеки, де основний ефект полягає у запобіганні збиткам, а не у прямому генеруванні прибутку, інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Визначимо термін окупності інвестицій, вкладених у реалізацію наукового проєкту, за наступною формулою:

(4.15)

$$T_{ок} = \frac{1}{E_в}$$

$$T_{ок} = \frac{1}{0,55} = 1,81$$

Так як $T_{ок} \leq$ 3-ти років, то фінансування даної наукової розробки є доцільним.

4.5 Висновки до розділу

У цьому розділі проведено економічне обґрунтування доцільності розробки програмного забезпечення для виявлення та запобігання фішинг-загрозам у веб-орієнтованих інформаційних системах на основі трансформерних архітектур глибокого навчання. Загальна вартість виконання НДР становить 52036,81 грн, а з урахуванням коефіцієнта впровадження k = 3 початкові інвестиції дорівнюють близько 156110,43 грн.

Приведена вартість чистих прибутків за три роки становить приблизно 581544,7 грн, що забезпечує абсолютну ефективність інвестицій на рівні близько 425434,27 грн. Відносна ефективність інвестицій перевищує мінімальної ставки дисконтування, а термін окупності становить близько 1,81 року, що в межах нормативного значення до 3 років.

Отже, впровадження розробленої системи виявлення фішинг-загроз на базі моделі DistilBERT є економічно доцільним, має високий рівень ефективності та може бути успішно комерціалізоване в умовах сучасних організацій, що працюють з веб-орієнтованими інформаційними системами.

ВИСНОВКИ

У магістерській роботі розглянуто актуальну проблему захисту веборієнтованих інформаційних систем від фішингових атак та розроблено ефективну систему для їх автоматичного виявлення за допомогою трансформерних архітектур глибинного навчання. Дослідження охопило повний цикл – від аналізу проблематики та обґрунтування необхідності нових підходів до реалізації та оцінки економічної складової проєкту.

На початку розробки проведено всебічний аналіз сучасних методів фішингу, їх типів та життєвого циклу атак. Розглянуто існуючі підходи до виявлення загроз та обґрунтовано необхідність створення нових адаптивних методів через зростаючу складність фішингових схем, зокрема тих, що використовують штучний інтелект. Це дало змогу сформувати теоретичну базу для подальших удосконалень.

Далі увагу було зосереджено на розробці та вдосконаленні методів аналізу фішингового контенту. Розроблено гібридний алгоритм виявлення загроз, а також покращено його точність за допомогою трансформерної нейронної мережі. Було обрано легковагову модель DistilBERT, що підвищила точність розпізнавання семантичних патернів соціальної інженерії. Запропоновані рішення, зокрема інтеграція модуля генерації синтетичних даних через великі мовні моделі, суттєво зменшили кількість хибних спрацювань та підвищили стійкість системи до нових загроз.

Реалізовано практичний функціонал системи: створено серверну частину на Flask, клієнтський інтерфейс на React для зручної взаємодії та адміністративну панель для керування процесом навчання. Використання механізму пояснюваного штучного інтелекту на базі алгоритму Layer Integrated Gradients дозволило забезпечити прозорість рішень системи, що є критично важливим для аналітиків безпеки. Окрім цього, підготовлено інструкцію з користування, що сприяє швидкій інтеграції системи та зручності для користувачів.

Проведено економічні розрахунки проєкту, визначено витрати на розробку, експлуатацію та збереження системи. Оцінено собівартість проєкту, що дозволило визначити фінансову ефективність та доцільність його впровадження.

Загалом, результати роботи підтверджують досягнення поставлених цілей. Запропонована система виявлення фішингових загроз демонструє високу ефективність та точність, а також відповідає сучасним вимогам безпеки. Виконані теоретичні та практичні дослідження, разом із економічною оцінкою, свідчать про готовність системи до впровадження та подальшої експлуатації. У ході виконання роботи було досягнуто поставлених цілей, розроблено та впроваджено систему виявлення фішингу на основі трансформерних моделей, яка демонструє високу точність і ефективність. Виконаний аналіз проблематики та сучасних підходів дозволив обґрунтувати необхідність створення нових адаптивних рішень.

Удосконалений метод з використанням DistilBERT та синтетичних даних підвищив точність розпізнавання фішингових загроз, а практична реалізація інтерфейсу та модуля пояснень забезпечила зручність та довіру для користувачів. Завдяки економічному обґрунтуванню розробки підтверджено доцільність і рентабельність впровадження системи.

Таким чином, результати роботи свідчать про готовність системи до реального використання та її потенціал у підвищенні рівня кібербезпеки. Система може бути корисною для захисту користувачів від фішингових атак і має можливість подальшого розвитку та масштабування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сабадаш, В. П. (2006). Фішинг як найбільш розвинений вид шахрайства в Інтернеті. *Університетські наукові записки*, (1), 228-233.
2. Point, C. (2025). Check Point Research Unveils Critical# MonikerLink Vulnerability in Microsoft Outlook with a 9.8 CVSS Severity Score–Global Security Mag Online.
3. Khatun, M., & Oyshi, M. S. (2025). Advanced Machine Learning Techniques for Cybersecurity: Enhancing Threat Detection in US Firms. *Journal of Computer Science and Technology Studies*, *7*(2), 305-315.
4. Psafe. (2025). Deepfake voice attacks surge by 550%. DFendr Security Bulletin URL: https://deepstrike.io/blog/Phishing-Statistics-2025 (дата звернення: 25.09.2025)
5. Greenblatt, A. (2025). Financial Scams.
6. Panda, S. P. (2025). The Evolution and Defense Against Social Engineering and Phishing Attacks. *International Journal of Science and Research (IJSR)*.
7. IBM. (2024). Cost of a data breach report. URL: https://www.ibm.com/think/insights/cost-of-a-data-breach-2024-financial-industry (дата звернення: 01.10.2025)
8. FBI. (2024). Internet crime report. URL: https://www.fbi.gov/contact-us/field-offices/anchorage/news/fbi-releases-annual-internet-crime-report (дата звернення: 01.10.2025)
9. Штонда, Р., Черниш, Ю., Терещенко, Т., Терещенко, К., Цикало, Ю., & Поліщук, С. (2024). Класифікація та методи виявлення фішингових атак. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, *4*(24), 69-80.
10. Стефанів, А. М., Загородна, Н. В., & Козак, Р. О. (2019). Класифікація методів виявлення фішингу в інтерактивних мультимедійних виданнях. *Збірник тез доповідей VIII Міжнародної науково-технічної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “*, *1*, 136-137.

11. Netskope. (2025). Threat labs report: Europe edition.URL: https://www.netskope.com/resources/threat-labs-reports/threat-labs-report-europe-2025 (дата звернення: 10.10.2025)
12. Anti-Phishing Working Group. (2024). Phishing activity trends report. URL: https://docs.apwg.org/reports/apwg_trends_report_q4_2024.pdf (дата звернення: 10.10.2025)
13. KnowBe4. (2025). Phishing attack trends report. URL: https://www.knowbe4.com/hubfs/Phishing-Threat-Trends-2025_Report.pdf
14. World Economic Forum. (2025). Global cybersecurity outlook. URL: https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf (дата звернення: 14.10.2025)
15. SentinelOne. (2025). Hybrid security solutions review. URL: https://www.sentinelone.com/cybersecurity-101/cloud-security/hybrid-cloud-security-solutions/ (дата звернення: 14.10.2025)
16. Cynet. (2025). XDR platform capabilities. URL:https://www.cynet.com/xdr-security/understanding-xdr-security-concepts-features-and-use-cases/ (дата звернення: 14.10.2025)
17. Datami. (2025). Analysis of hybrid detection systems. URL: https://datami.ee/blog/internal-network-penetration-test/ (дата звернення: 14.10.2025)
18. ProIT. (2025). AI in cybersecurity: A market overview. URL: https://sapient.pro/blog/ai-in-cybersecurity (дата звернення: 15.10.2025)
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
20. Іосіфов, Є. А., Соколов, В. Ю., & Складанний, П. М. (2025). Безпечна обробка голосової інформації.
21. Павлат, О. В. (2025). *Застосування LLM в галузі кібербезпеки* (Bachelor's thesis).
22. Unite.AI. (2024). Understanding BERT: A deep dive. URL: https://www.unite.ai/ai-in-2024-predictions/ (дата звернення: 15.10.2025)
23. Aezion. (2025). NLP trends for 2025. URL: https://www.aezion.com/blogs/natural-language-processing/ (дата звернення: 15.10.2025)
24. Каплун, Є. В. (2024). *Інформаційна технологія захисту від атак соціальної інженерії на основі великих мовних моделей* (Master's thesis, Сумський державний університет).

25. IT Ukraine Association. (2024). Ukrainian IT market report. URL: https://itukraine.org.ua/files/DigitalTiger2024.pdf (дата звернення: 15.10.2025)
26. Gartner. (2024). Forecast for worldwide security and risk management spending. URL: https://www.gartner.com/en/newsroom/press-releases/2023-09-28-gartner-forecasts-global-security-and-risk-management-spending-to-grow-14-percent-in-2024 (дата звернення: 22.10.2025)
27. Uddin, M. A., Mahiuddin, M., & Sarker, I. H. (2024). An explainable transformer-based model for phishing email detection: A large language model approach. *arXiv preprint arXiv:2402.13871*.
28. Songailaitė, M., Kankevičiūtė, E., Zhyhun, B., & Mandravickaitė, J. (2023, May). BERT-based models for phishing detection. In *28th Conference on Information Society and University Studies (IVUS'2023), CEUR Workshop Proceedings, Kaunas, Lithuania*.
29. Hugging Face. NLP Course. Chapter 1. URL: https://huggingface.co/learn/llm-course/chapter1/1. (дата звернення: 22.10.2025)
30. Бучик, С., & Толстяк, М. (2025). ДЕТЕКТУВАННЯ ФІШИНГОВИХ URL НА ОСНОВІ ЕВРИСТИЧНИХ ПРАВИЛ. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, *4*(28), 565-574.
31. Kintis, P., Miramirkhani, N., Lever, C., Chen, Y., Romero-Gómez, R., Pitropakis, N., ... & Antonakakis, M. (2017, October). Hiding in plain sight: A longitudinal study of combosquatting abuse. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 569-586).
32. OWASP. (2023). Top 10 web application security risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 22.10.2025)
33. Hylender, C. D., Langlois, P., Pinto, A., & Widup, S. (2024). Verizon 2024 data breach investigations report. *The Verizon DBIR Team. Available online: https://www. verizon. com/business/resources/Tf18/reports/2024-dbir-data-breach-investigations-report. pdf (accessed on 20 November 2024)*.

34. KDNuggets. Using Hugging Face Transformers with PyTorch and TensorFlow. URL: https://www.kdnuggets.com/using-hugging-face-transformers-with-pytorch-and-tensorflow. (дата звернення: 01.11.2025)
35. Cohorte. A comprehensive guide to implementing NLP applications with Hugging Face Transformers. URL: https://www.cohorte.co/blog/a-comprehensive-guide-to-implementing-nlp-applications-with-hugging-face-transformers. (дата звернення: 01.11.2025)
36. LinkedIn. Mastering Modern Web Development Guide: Flask + React. URL: https://www.linkedin.com/pulse/mastering-modern-web-development-guide-flask-2-react-18-duo-hiqrc. (дата звернення: 01.11.2025)
37. GK, M. V., Kumar, S., Birje, S. R., Sonica, B. K., & Ahamad, T. AI-Powered Homoglyph Detection and Behavioral Profiling for Phishing Prevention.
38. Dev.to. Detecting homoglyph attacks with character identifiers. URL: https://dev.to/toolzr/detecting-homoglyph-attacks-with-toolzrs-character-identifier-5354. (дата звернення: 06.11.2025)
39. Razmyslovich, A., Murasheva, K., Sedlova, S., Capitaine, J., & Dmitriev, E. (2025). ELTEX: A Framework for Domain-Driven Synthetic Data Generation. arXiv preprint arXiv:2503.15055.
40. NexGenCloud. Generative AI in synthetic data generation: A comprehensive guide. URL: https://www.nexgencloud.com/blog/case-studies/generative-ai-in-synthetic-data-generation-a-comprehensive-guide. (дата звернення: 06.11.2025)
41. Yasin, A., Fatima, R., Khan, J. A., & Afzal, W. (2024). Behind the Bait: Delving into PhishTank's hidden data. Data in Brief, 52, 109959.
42. Linh, D. M., & Hung, T. C. (2025). A feature-engineered dataset of benign and phishing URLs for machine learning and large language models evaluation. Data in Brief, 112162.
43. DataCamp. F1 Score Tutorial. URL: https://www.datacamp.com/tutorial/f1-score. (дата звернення: 06.11.2025)

44. Farhan, H. A. (2025). Robust and Accurate Phishing Detection Using Enhanced DistilBERT: A Transformer-Based Approach. Journal of Al-Qadisiyah for Computer Science and Mathematics, 17(3), 230-246.
45. Atla, S. R. (2025). Phishing Url Detection Using DistilBERT And Capsule Neural Networks (Doctoral dissertation, Dublin, National College of Ireland).
46. RCPedia. Train machine learning models on Colab GPU. URL: https://rcpedia.stanford.edu/blog/2024/03/28/train-machine-learning-models-on-colab-gpu/. (дата звернення: 06.11.2025)
47. Neptune.ai. How to use Google Colab for Deep Learning: Complete Tutorial. URL: https://neptune.ai/blog/how-to-use-google-colab-for-deep-learning-complete-tutorial. (дата звернення: 09.11.2025)
48. AryaXAI. Explainability XAI techniques for deep learning and limitations. URL: https://www.aryaxai.com/article/explainability-xai-techniques-for-deep-learning-and-limitations. (дата звернення: 09.11.2025)
49. Polito.it. XAI: Local gradient based methods. URL: https://dbdmg.polito.it/dbdmg_web/wp-content/uploads/2024/04/XAI_07_local_gradient_based.pdf. (дата звернення: 09.11.2025)
50. FreeMindTronic. Technology Readiness Levels (TRL) Framework. URL: https://freemindtronic.com/technology-readiness-levels-trl10-framework/.
51. NextITSecurity. ROI for Cybersecurity. URL: https://nextitsecurity.com/roi-for-cybersecurity-next-it-security/. (дата звернення: 09.11.2025)
52. Desai, T., & Pal, R. K. (2025, June). Machine Learning in Cybersecurity: A Comprehensive Review. In 2025 4th International Conference on Computational Modelling (ICCMSO) (pp. 148-153). IEEE.
53. TechMagic. Calculating ROI for software projects. URL: https://www.techmagic.co/blog/calculating-roi.

Додаток А

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор

_____ Юрій ЯРЕМЧУК
"24" вересня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання

08-72.МКР.001.00.000.ТЗ

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент Грицак А. В.

_____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб гібридної системи виявлення та запобігання фішинговим загрозам на основі трансформерних архітектур глибинного навчання1.

Область застосування: захист веборієнтованих інформаційних систем, корпоративних мереж та персональних пристроїв користувачів від фішингових атак, включаючи атаки з використанням соціальної інженерії та генеративного штучного інтелекту.

2. Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №313 від 24.09.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: створення адаптивної системи для автоматичного виявлення фішингових загроз, що поєднує семантичний аналіз тексту за допомогою моделі DistilBERT та евристичні методи, з можливістю донавчання на синтетичних даних.

3.2 Призначення: розроблений програмний засіб виконує класифікацію URL-адрес та текстового контенту на предмет наявності фішингу, надає пояснення рішень та забезпечує захист користувачів у режимі реального часу.

4. Джерела розробки

4.1. Vaswani A. et al. Attention is all you need. Advances in neural information processing systems. 2017. Vol. 306.

4.2. Штонда Р. та ін. Класифікація та методи виявлення фішингових атак. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2024. № 4 (24). С. 69–807.

4.3. Uddin M. A., Mahiuddin M., Sarker I. H. An explainable transformer-based model for phishing email detection: A large language model approach. arXiv preprint arXiv:2402.13871. 20248.

4.4. Sanh V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108. 20199999.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен реалізовувати гібридний аналіз: евристичний та семантичний.

5.1.2 Система повинна мати модуль генерації синтетичних навчальних даних за допомогою LLM для адаптації до нових загроз.

5.1.3 Програмний засіб повинен забезпечувати пояснюваність рішень за допомогою алгоритму Layer Integrated Gradients.

5.1.4 Інтерфейс має складатися з клієнтської частини для користувача та адміністративної панелі для керування навчанням.

5.2 Вимоги до надійності:

5.2.1 Система повинна забезпечувати обробку помилок при недоступності зовнішніх API та автоматичне звільнення ресурсів GPU під час простою.

5.2.2 База даних SQLite повинна підтримувати цілісність даних та зберігати історію навчання і згенеровані зразки.

5.2.3 Точність класифікації моделі повинна бути не нижче 98% на тестовій вибірці.

5.3 Вимоги до складу і параметрів технічних засобів:

– Мова програмування: Python 3.10+.

– Фреймворки: PyTorch 2.0, Flask, React.

– Апаратне забезпечення: підтримка GPU бажана для прискорення навчання, або сучасний CPU для інференсу.

– Оперативна пам’ять: не менше 4 ГБ

– Середовище функціонування: кросплатформне (Windows/Linux/MacOS) з підтримкою сучасних веб-браузерів.

6. Вимоги до програмної документації

6.1 Обов’язкова наявність інструкції з користування системою, яка наведена в розділі 3.8 пояснювальної записки.

7. Вимоги до технічного захисту інформації

7.1 Адміністративна панель повинна бути захищена механізмом автентифікації.

7.2 API повинен мати захист від зловживань для обмеження частоти запитів.

7.3 Ключі доступу до зовнішніх API повинні зберігатися безпечно або передаватися через захищені змінні середовища.

8. Техніко-економічні показники

8.1 Термін окупності інвестицій становить приблизно 3,7 місяця, що підтверджує економічну доцільність.

8.2 Рентабельність інвестицій у перший рік експлуатації становить близько 217%.

8.3 Використання моделі DistilBERT замість повного BERT дозволяє знизити витрати на обчислювальні ресурси на 40%.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1	Аналіз сучасних методів та технологій протидії фішингу	24.09.2025	28.09.2025
2	Проєктування архітектури гібридної системи та вибір моделі DistilBERT	29.09.2025	08.10.2025
3	Розробка алгоритмів генерації синтетичних даних та евристичних правил	09.10.2025	15.10.2025
4	Програмна реалізація серверної частини та ядра аналізу	16.10.2025	25.10.2025
5	Реалізація клієнтського інтерфейсу та модуля пояснень XAI	26.10.2025	28.10.2025
6	Експериментальна перевірка, навчання моделі та аналіз результатів	29.10.2025	09.11.2025

7	Економічне обґрунтування та аналіз ефективності впровадження	10.11.2025	15.11.2025
8	Оформлення пояснювальної записки та графічних матеріалів	16.11.2025	20.11.2025
8	Передзахист магістерської кваліфікаційної роботи	21.11.2025	21.11.2025
8	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	22.11.2025	07.12.2025
9	Захист магістерської кваліфікаційної роботи	08.12.2025	08.12.2025

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– Пояснювальна записка до магістерської кваліфікаційної роботи;

– Програмний продукт;

– Презентація до захисту;

– Відзив керівника роботи;

– Відзив опонента.

Технічне завдання до виконання прийняв _____ Гуменчук Е.С.

Додаток Б. Лістинг програми

```
import os
import sys
import json
import time
import threading
import logging
import random
from flask import Flask, request, jsonify
from flask_cors import CORS
from flask_socketio import SocketIO
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
from flask_httpauth import HTTPBasicAuth
from dotenv import load_dotenv

from backend.external_ai import generate_samples_with_external_ai
from backend import database
from backend.model_handler import DistilBERTPhishing
from backend.analysis import analyze_full # Import from new analysis module
from backend.config import METRICS_PATH, TRAINING_HISTORY_PATH, APIConfig
from backend.label_maps import LABEL_TO_ID
import torch

load_dotenv()

# === DIAGNOSTIC LOGGING ===
print("="*80)
print("Starting Anti-Phishing AI Backend...")
print(f"Python Version: {sys.version}")
print(f"Torch Version: {torch.__version__}")
is_cuda = torch.cuda.is_available()
print(f"CUDA Available: {is_cuda}")
if is_cuda:
    print(f"CUDA Device Count: {torch.cuda.device_count()}")
    print(f"Current CUDA Device: {torch.cuda.current_device()}")
    print(f"Device Name: {torch.cuda.get_device_name(torch.cuda.current_device())}")
print("="*80)


# === LOGGING SETUP ===
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    stream=sys.stdout
)
logger = logging.getLogger(__name__)


# === MODEL LIFECYCLE MANAGEMENT ===
MODEL_INACTIVITY_TIMEOUT = 600  # Unload model after 10 minutes of inactivity
_model_handler_instance = None
_model_lock = threading.Lock()
_device_preference = 'auto'  # Global variable to store the desired device
```

```
_inactivity_thread = None

def get_model_handler():
    """
    Gets the current model handler instance. Initializes if it doesn't exist.
    This function is thread-safe.
    """
    global _model_handler_instance, _inactivity_thread
    with _model_lock:
        if _model_handler_instance is None:
            logger.info(f"No model handler instance found. Initializing with preference: {_device_preference}")
            _model_handler_instance = DistilBERTPhishing(device_preference=_device_preference,
socketio=socketio)
            if _inactivity_thread is None or not _inactivity_thread.is_alive():
                _inactivity_thread = threading.Thread(target=model_inactivity_check, daemon=True)
                _inactivity_thread.start()
    return _model_handler_instance

def set_device_preference(device: str):
    """
    Sets the device preference and re-initializes the model handler.
    This function is thread-safe.
    """
    global _model_handler_instance, _device_preference
    with _model_lock:
        new_preference = 'cuda' if device == 'cuda' else 'cpu'

        # Check if a change is actually needed
        current_handler = get_model_handler()
        if new_preference == current_handler.device.type:
            logger.info(f"Device is already '{new_preference}'. No changes needed.")
            return

        if new_preference == _device_preference:
            logger.info(f"Device preference is already set to '{new_preference}'. No changes needed.")
            return

        logger.info(f"Setting new device preference to '{new_preference}'. Re-initializing model handler.")
        _device_preference = new_preference

        # Unload and clear the old instance
        if _model_handler_instance is not None:
            if _model_handler_instance.is_loaded():
                _model_handler_instance.unload()
            _model_handler_instance = None  # Force re-creation on next get_model_handler call

    logger.info("Model handler will be re-initialized on the next request.")

def get_model():
    """Gets the model, loading it if it's not already in memory."""
    model_handler = get_model_handler()
    with _model_lock:
        if not model_handler.is_loaded():
            logger.info("Model is not loaded. Loading now...")
            model_handler.load()
```

```
        model_handler.last_used_time = time.time()
    return model_handler

def model_inactivity_check():
    """Periodically checks if the model has been inactive and unloads it."""
    # This check needs to wait until the handler is initialized
    while _model_handler_instance is None:
        time.sleep(1)

    model_handler = get_model_handler()
    while True:
        time.sleep(60) # Check every minute
        if model_handler.is_loaded() and not model_handler.is_training:
            with _model_lock:
                inactivity_period = time.time() - model_handler.last_used_time
                if inactivity_period > MODEL_INACTIVITY_TIMEOUT:
                    logger.info(f"Model inactive for {inactivity_period:.0f}s. Unloading to save memory.")
                    model_handler.unload()

# === GLOBAL INSTANCES & STARTUP ===
database.init_db()

# === Flask App ===
app = Flask(__name__)
CORS(app)
socketio = SocketIO(app, cors_allowed_origins="*")
auth = HTTPBasicAuth()

# Rate Limiter
limiter = Limiter(
    get_remote_address,
    app=app,
    default_limits=[]
)

# === AUTHENTICATION ===
@auth.verify_password
def verify_password(username, password):
    """Verify admin credentials from environment variables."""
    env_user = os.environ.get("ADMIN_USERNAME")
    env_pass = os.environ.get("ADMIN_PASSWORD")
    if username == env_user and password == env_pass:
        return username
    return None

# === CUSTOM ERROR HANDLING ===
class APIException(Exception):
    """Custom exception class for API errors."""
    def __init__(self, message, status_code=400):
        super().__init__(message)
        self.message = message
        self.status_code = status_code

@app.errorhandler(APIException)
def handle_api_exception(e):
```

```
    """Centralized handler for APIExceptions."""
    return jsonify({'error': e.message}), e.status_code

@app.errorhandler(429)
def ratelimit_handler(e):
    """Custom handler for rate limit errors."""
    return jsonify(error=f"Ви перевищили ліміт запитів: {e.description}"), 429


@app.route('/analyze', methods=['POST'])
@limiter.limit("10 per minute")
def analyze():
    data = request.json
    text = data.get('inputText', '')
    type_ = data.get('activeTab', 'url')
    use_rules = data.get('useRules', True)
    explain = data.get('explain', False)

    if not text.strip():
        raise APIException('Будь ласка, введіть текст для аналізу.')
    if len(text) > 10000:
        raise APIException('Текст занадто довгий. Обмеження 10000 символів.')

    current_model = get_model()
    result = analyze_full(current_model, text, type_, use_rules)

    if explain:
        try:
            logger.info(f"Generating explanation data for text: '{text[:50]}...'")
            explanation_data = current_model.explain_instance(text)
            result['explanation_data'] = explanation_data
        except Exception as e:
            logger.error(f"Failed to get explanation data for text: '{text[:50]}...'. Error: {e}")
            result['explanation_data'] = {'error': 'Failed to generate explanation data.'}

    return jsonify(result)

@socketio.on('connect')
def handle_connect():
    """Sends the current model statistics to a newly connected client."""
    logger.info("Client connected")
    stats = get_model_handler().get_stats()
    socketio.emit('update_stats', stats)

import torch

@socketio.on('request_initial_stats')
def handle_request_initial_stats():
    """Sends the current model statistics to a client that requests it."""
    stats = get_model_handler().get_stats()
    socketio.emit('update_stats', stats)

@app.route('/settings', methods=['GET'])
def get_settings():
    """Returns public settings, including the actual device in use."""
```

```
    handler = get_model_handler()
    # Ensure model is loaded to get the correct device type if it was auto-initialized
    if not handler.is_loaded():
        handler.load()

    return jsonify({
        'actual_device': handler.device.type,
        'is_cuda_available': torch.cuda.is_available()
    })

@app.route('/admin/device', methods=['POST'])
@auth.login_required
def set_device():
    """Sets the device for the model, requires admin auth."""
    data = request.json
    device_pref = data.get('device') # 'cuda' or 'cpu'
    if device_pref not in ['cuda', 'cpu']:
        raise APIException('Invalid device specified.')

    if device_pref == 'cuda' and not torch.cuda.is_available():
        raise APIException('CUDA is not available on this server.')

    # This function triggers the re-initialization
    set_device_preference(device_pref)

    # Trigger a model load in the background to make the next user request faster
    def load_in_background():
        get_model()

    thread = threading.Thread(target=load_in_background)
    thread.start()

    return jsonify({'success': True, 'message': f'Device preference set to {device_pref}. Model is reloading.'})


# === Admin Panel & Trainer Logic ===

@app.route('/admin/add-samples-manual', methods=['POST'])
@auth.login_required
def admin_add_samples_manual():
    """Adds a batch of samples provided in JSONL format."""
    data = request.json
    jsonl_data = data.get('jsonl_data')
    if not jsonl_data or not isinstance(jsonl_data, str):
        raise APIException('`jsonl_data` must be a non-empty string.')

    added_count = 0
    failed_lines = 0
    samples_to_add = []

    for line in jsonl_data.strip().split('\n'):
        line = line.strip()
        if not line:
            continue
        try:
```

```
            sample = json.loads(line)
            if all(k in sample for k in ['text', 'type', 'actualRisk']):
                samples_to_add.append(sample)
            else:
                failed_lines += 1
        except json.JSONDecodeError:
            failed_lines += 1

    if samples_to_add:
        added_count = database.add_samples(samples_to_add)

    return jsonify({
        'success': True,
        'message': f'Processed {added_count + failed_lines} lines.',
        'added_count': added_count,
        'failed_lines': failed_lines
    })

generation_lock = threading.Lock()
is_generating_samples = False

def run_batch_generation(total_to_generate, batch_size, provider, model):
    global is_generating_samples
    logger.info(f"Starting background sample generation for {total_to_generate} samples.")
    generated_count = 0
    max_retries = 3

    while generated_count < total_to_generate:
        retries = 0
        samples_needed = total_to_generate - generated_count
        samples_in_batch = min(batch_size, samples_needed)

        while retries < max_retries:
            try:
                logger.info(f"Requesting {samples_in_batch} samples from AI... (Attempt {retries + 1})")
                new_samples = generate_samples_with_external_ai(provider, model, samples_in_batch)
                if new_samples:
                    samples_to_add = new_samples[:samples_needed]
                    database.add_samples(samples_to_add)
                    num_added = len(samples_to_add)
                    generated_count += num_added
                    logger.info(f"Successfully added {num_added} samples. Total generated:
{generated_count}/{total_to_generate}")
                    break
                else:
                    logger.warning("AI returned an empty list of samples. Retrying...")
                    retries += 1
                    time.sleep(5)
            except Exception as e:
                logger.error(f"Failed to generate batch: {e}. Retrying...")
                retries += 1
                time.sleep(10)

        if retries == max_retries:
            logger.error(f"Failed to generate a batch after {max_retries} retries. Stopping generation.")
```

```
            break

    logger.info("Background sample generation finished.")
    with generation_lock:
        is_generating_samples = False

@app.route('/admin/generate-samples', methods=['POST'])
@auth.login_required
def admin_generate_samples():
    global is_generating_samples
    with generation_lock:
        if is_generating_samples:
            return jsonify({'error': 'Sample generation is already in progress.'}), 409
        is_generating_samples = True

    data = request.json
    total_samples = int(data.get('total_samples', 100))
    batch_size = int(data.get('batch_size', 20))
    provider = data.get('provider', 'openrouter')
    model = data.get('model', 'deepseek/deepseek-chat-v3.1:free')

    thread = threading.Thread(target=run_batch_generation, args=(total_samples, batch_size, provider, model))
    thread.daemon = True
    thread.start()

    return jsonify({'message': f'Started generating {total_samples} samples in the background.'})

@app.route('/admin/generation-status', methods=['GET'])
@auth.login_required
def admin_generation_status():
    global is_generating_samples
    with generation_lock:
        return jsonify({'is_generating': is_generating_samples})

@app.route('/admin/dataset-status', methods=['GET'])
@auth.login_required
def admin_dataset_status():
    try:
        count = database.get_sample_count()
        return jsonify({'totalSamples': count})
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/admin/recommend-settings', methods=['GET'])
@auth.login_required
def recommend_settings():
    defaults = {"learning_rate": 5e-5, "epochs": 3, "batch_size": 8, "rehearsal_percentage": 15, "iterations": 5,
"samples": 50}
    try:
        dataset = database.get_all_samples()
        if not dataset: return jsonify(defaults)
        total_samples = len(dataset)
        recommended_samples = max(30, int(total_samples * 0.15))
        if total_samples < 200: recommended_epochs, recommended_iterations = 4, 7
        elif total_samples < 1000: recommended_epochs, recommended_iterations = 3, 5
```

```
        else: recommended_epochs, recommended_iterations = 2, 4
        class_counts = {label: 0 for label in LABEL_TO_ID.keys()}
        for sample in dataset:
            if sample.get('actualRisk') in class_counts: class_counts[sample.get('actualRisk')] += 1
        min_class = max(1, min(class_counts.values()))
        max_class = max(1, max(class_counts.values()))
        imbalance_ratio = min_class / max_class
        if imbalance_ratio < 0.2: recommended_rehearsal = 30
        elif imbalance_ratio < 0.6: recommended_rehearsal = 20
        else: recommended_rehearsal = 15
        recommendations = {"learning_rate": 5e-5, "epochs": recommended_epochs, "batch_size": 8,
"rehearsal_percentage": recommended_rehearsal, "iterations": recommended_iterations, "samples":
recommended_samples}
        return jsonify(recommendations)
    except Exception as e:
        logger.error(f"Could not recommend settings: {e}")
        return jsonify(defaults), 500

@app.route('/admin/training-history', methods=['GET'])
@auth.login_required
def admin_training_history():
    if not os.path.exists(TRAINING_HISTORY_PATH): return jsonify([])
    try:
        with open(TRAINING_HISTORY_PATH, 'r', encoding='utf-8') as f:
            history = json.load(f)
        return jsonify(history)
    except Exception as e:
        logger.error(f"Could not retrieve training history: {e}")
        return jsonify({'error': str(e)}), 500

def clear_db_table():
    conn = database.get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM samples;")
    cursor.execute("DELETE FROM sqlite_sequence WHERE name='samples';")
    conn.commit()
    conn.close()

@app.route('/admin/reset-state', methods=['POST'])
@auth.login_required
def admin_reset_state():
    model_handler = get_model_handler()
    with _model_lock:
        if model_handler.is_loaded(): model_handler.unload()
        clear_db_table()
        if os.path.exists(TRAINING_HISTORY_PATH): os.remove(TRAINING_HISTORY_PATH)
        return jsonify({'success': True, 'message': 'Application state has been reset.'})

@app.route('/admin/start-training', methods=['POST'])
@auth.login_required
def admin_start_training():
    current_model = get_model()
    if current_model.is_training:
        return jsonify({'error': 'Модель вже навчається. Зачекайте.'}), 409
    data = request.json
```

```
    try:
        dataset = database.get_all_samples()
        if not dataset:
            return jsonify({'error': 'Датасет порожній. Згенеруйте зразки перед початком навчання.'}), 400
        n_samples = int(data.get('samples', 100))
        use_rules = data.get('useRules', True)
        learning_rate = float(data.get('learning_rate', 5e-5))
        epochs = int(data.get('epochs', 3))
        batch_size = int(data.get('batch_size', 8))
        rehearsal_percentage = int(data.get('rehearsal_percentage', 15))
        iterations = int(data.get('iterations', 5))
        test_samples = random.sample(dataset, min(n_samples, len(dataset)))
        training_thread = threading.Thread(target=current_model.run_training_session, args=(test_samples,
dataset, use_rules, learning_rate, epochs, batch_size, rehearsal_percentage, iterations))
        training_thread.start()
        return jsonify({'success': True, 'message': 'Навчання моделі запущено у фоновому режимі. Слідкуйте
за логами в консолі.'})
    except Exception as e:
        logger.error(f"Error starting training: {e}")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    get_model_handler()  # Initialize the model handler on startup
    socketio.run(app, host='0.0.0.0', port=5000, debug=True, use_reloader=False, allow_unsafe_werkzeug=True)
```

Додаток В. Ілюстративний матеріал

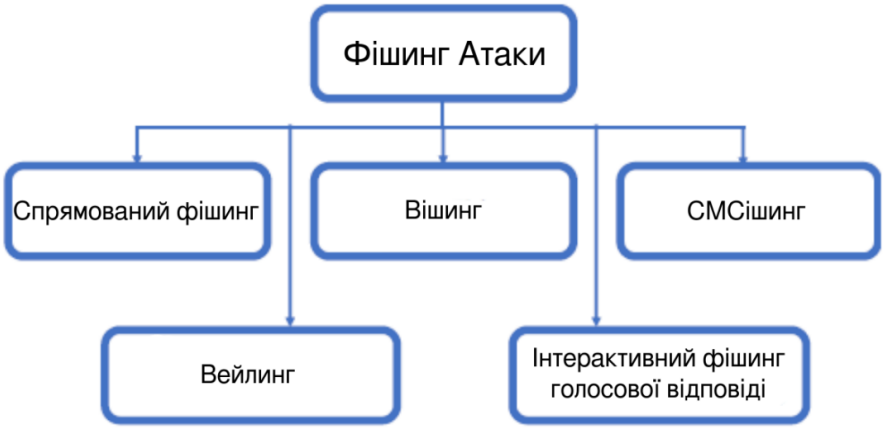


Рисунок 1 – Види фішингових атак

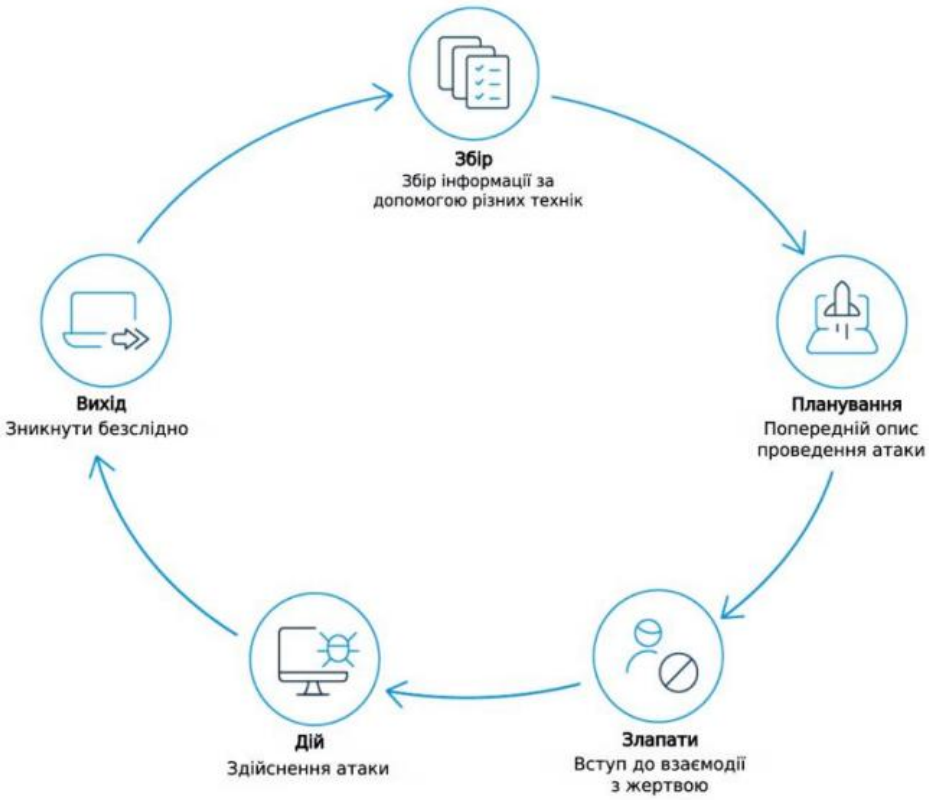


Рисунок 2 – Життєвий цикл фішингових атак

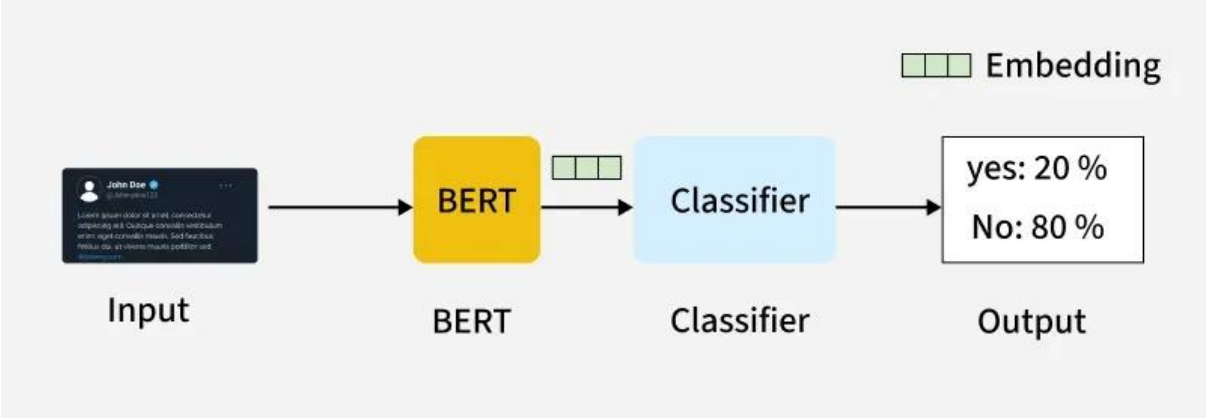


Рисунок 3 – Архітектура моделі BERT

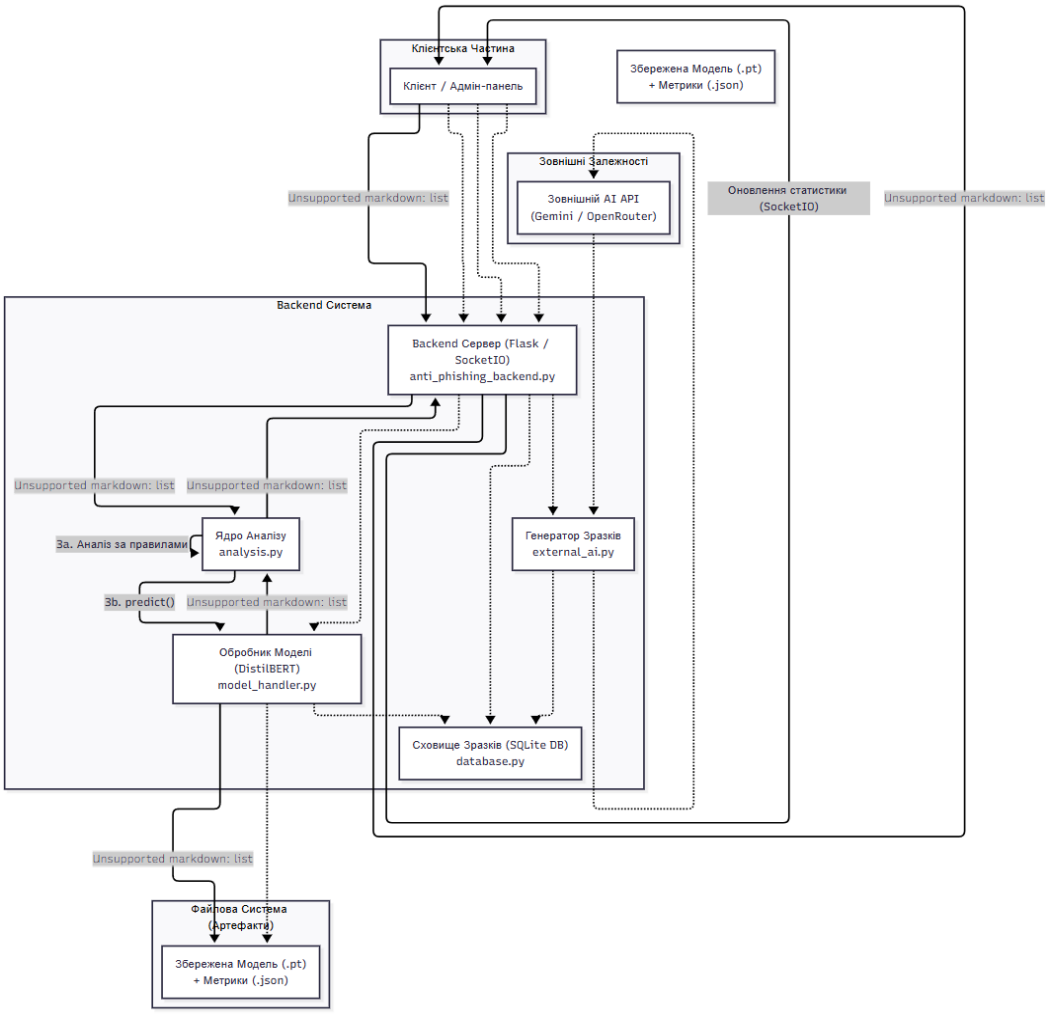


Рисунок 4 – Модульна архітектура

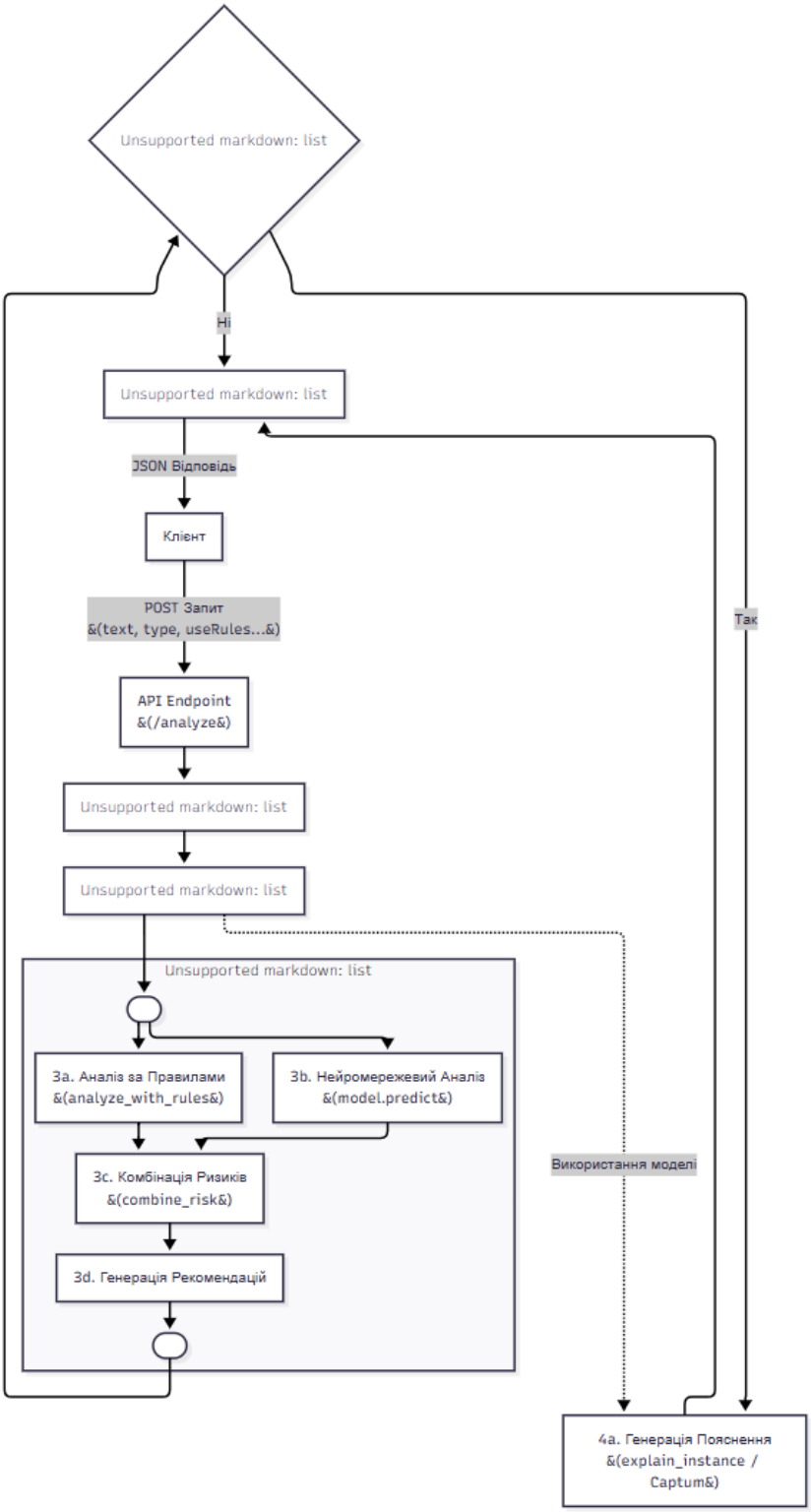


Рисунок 5 – Приклад онлайн-конвеєра

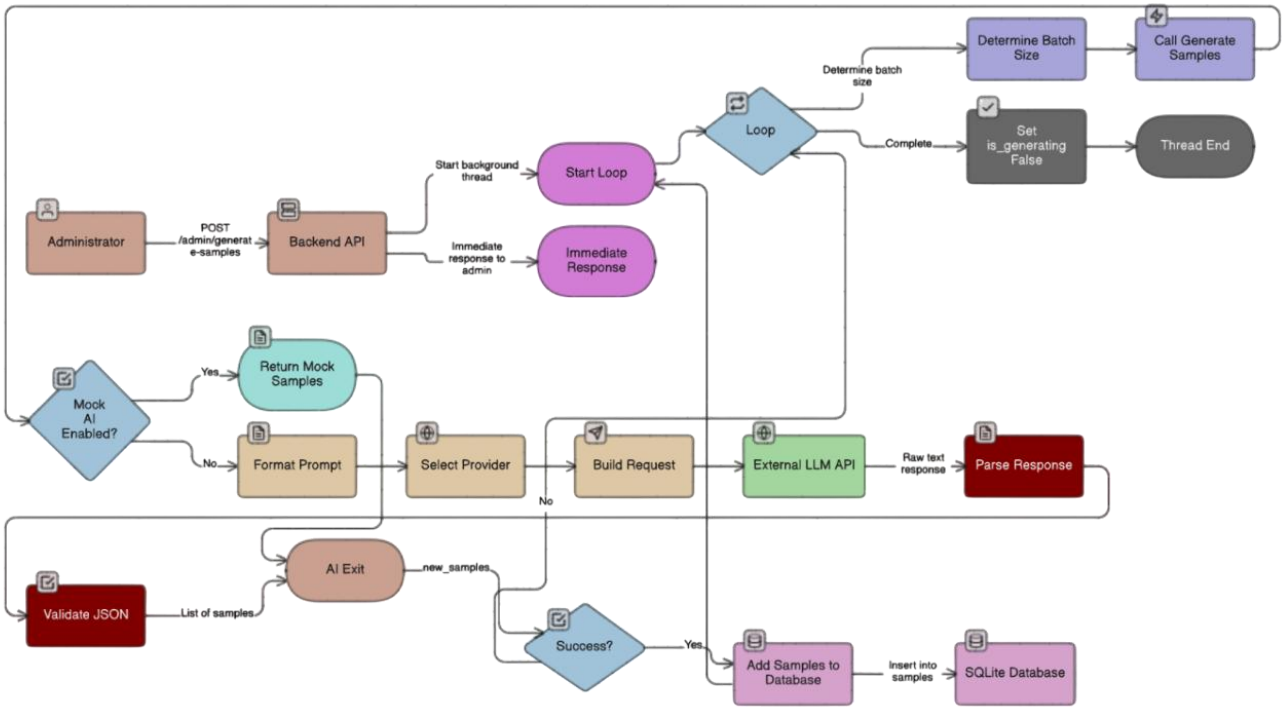


Рисунок 6 – Принцип генерації синтетичних даних в замкнутій мережі через LLM

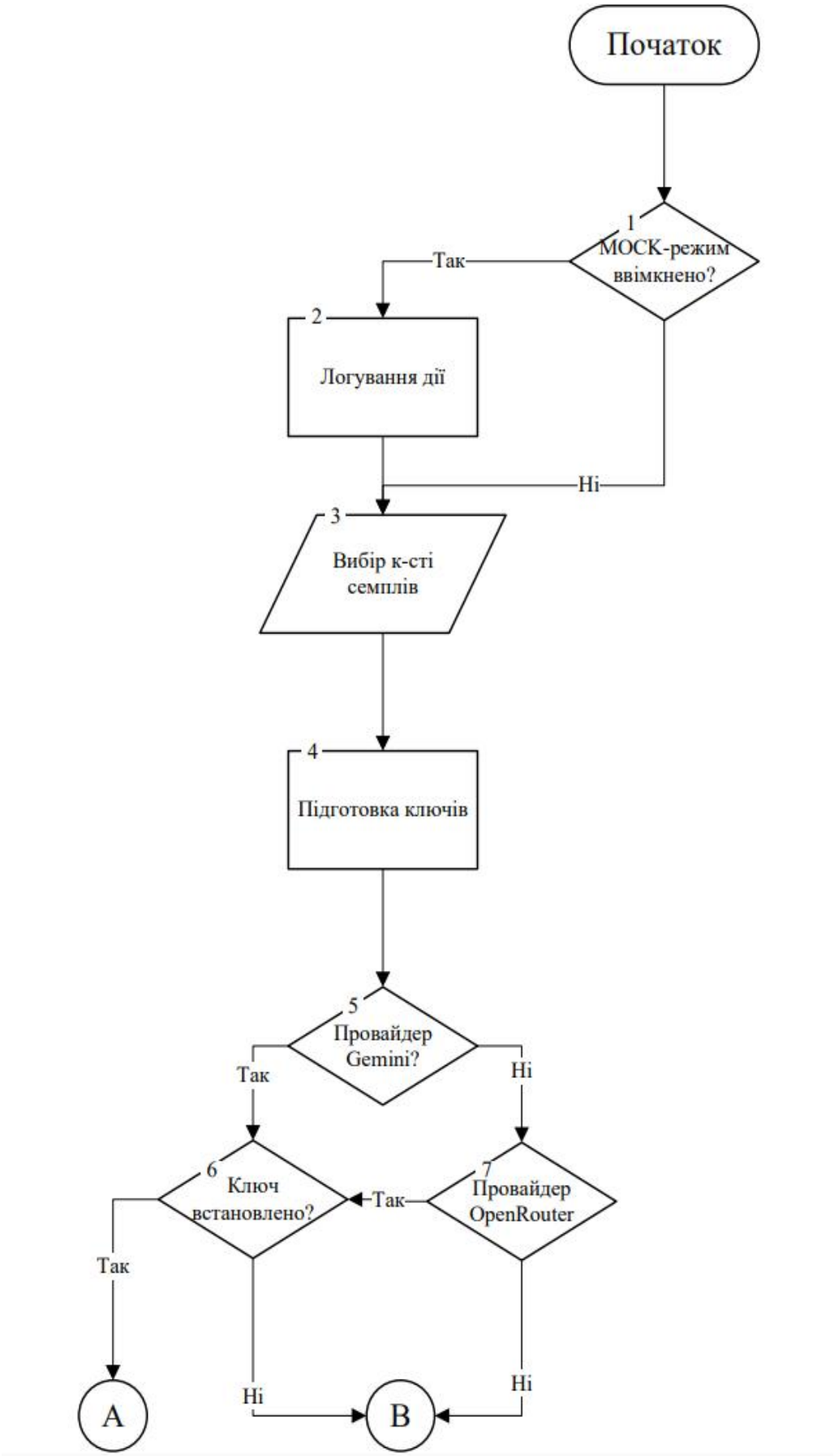
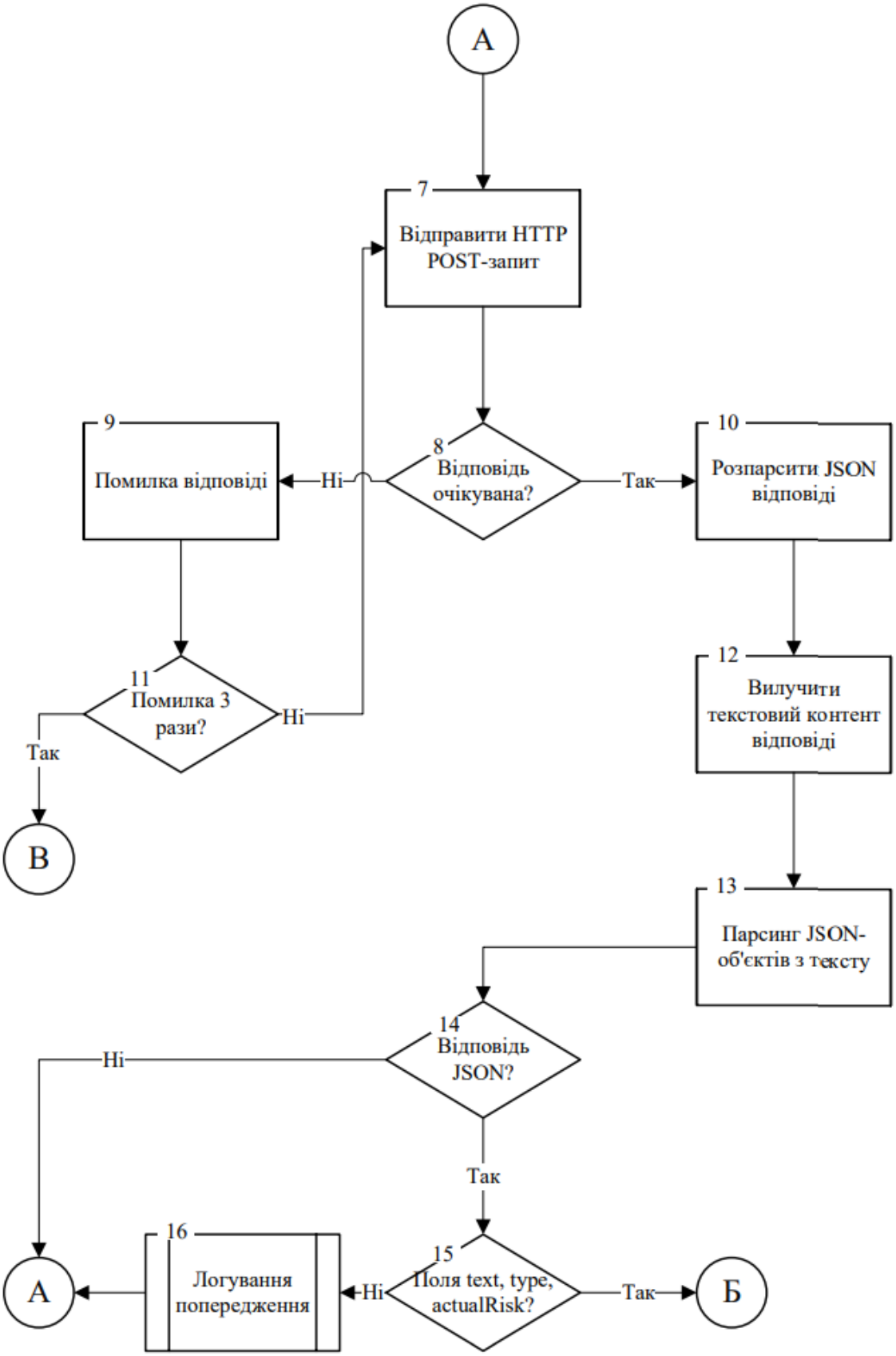
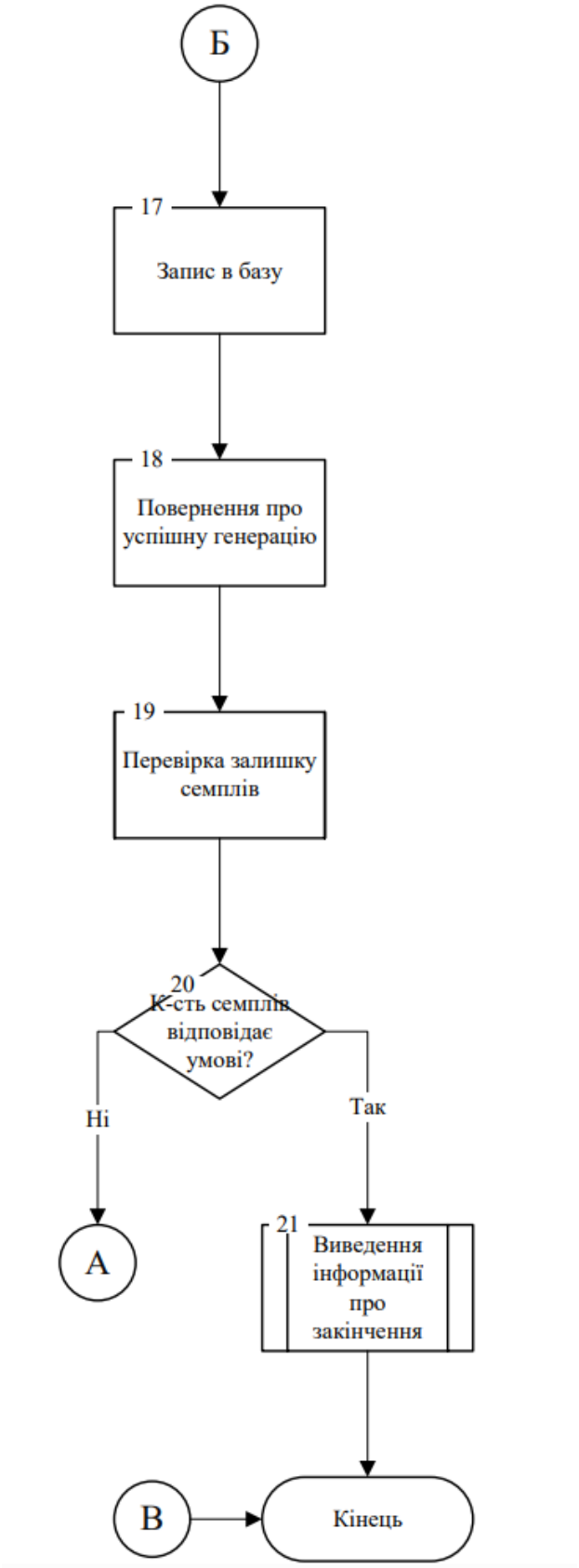


Рисунок 7 – Блок-схема алгоритму замкнутого кола генерації даних



Продовження рисунка 7



Продовження рисунка 7

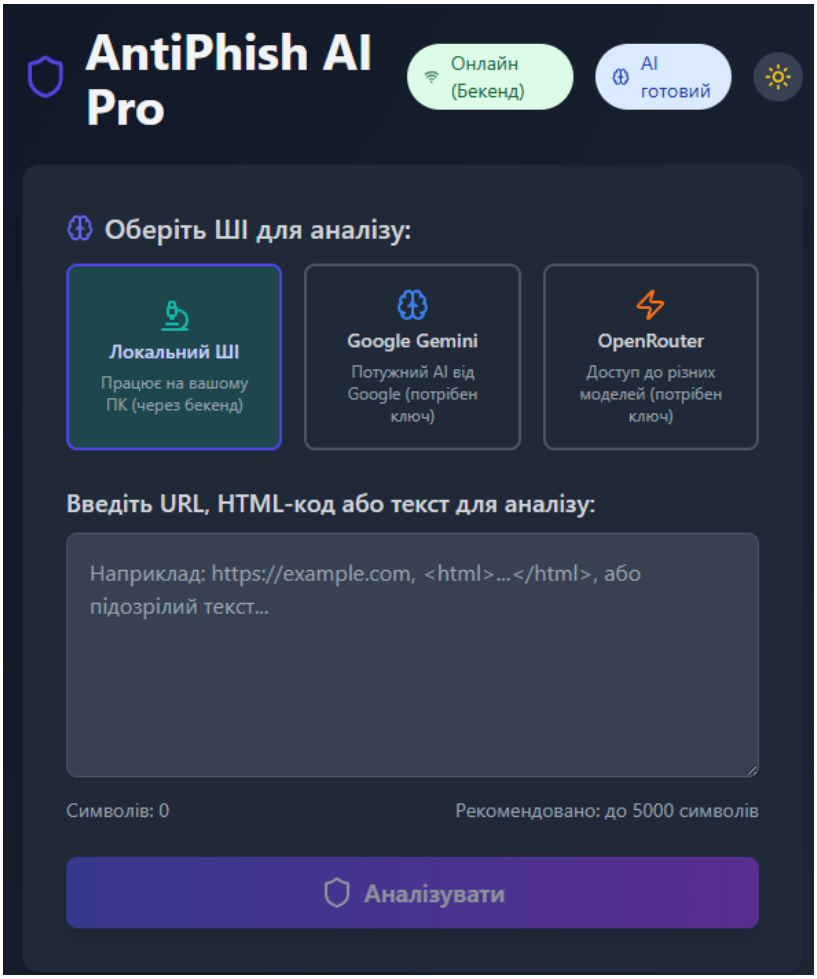


Рисунок 8 – Вигляд головного вікна плагіну

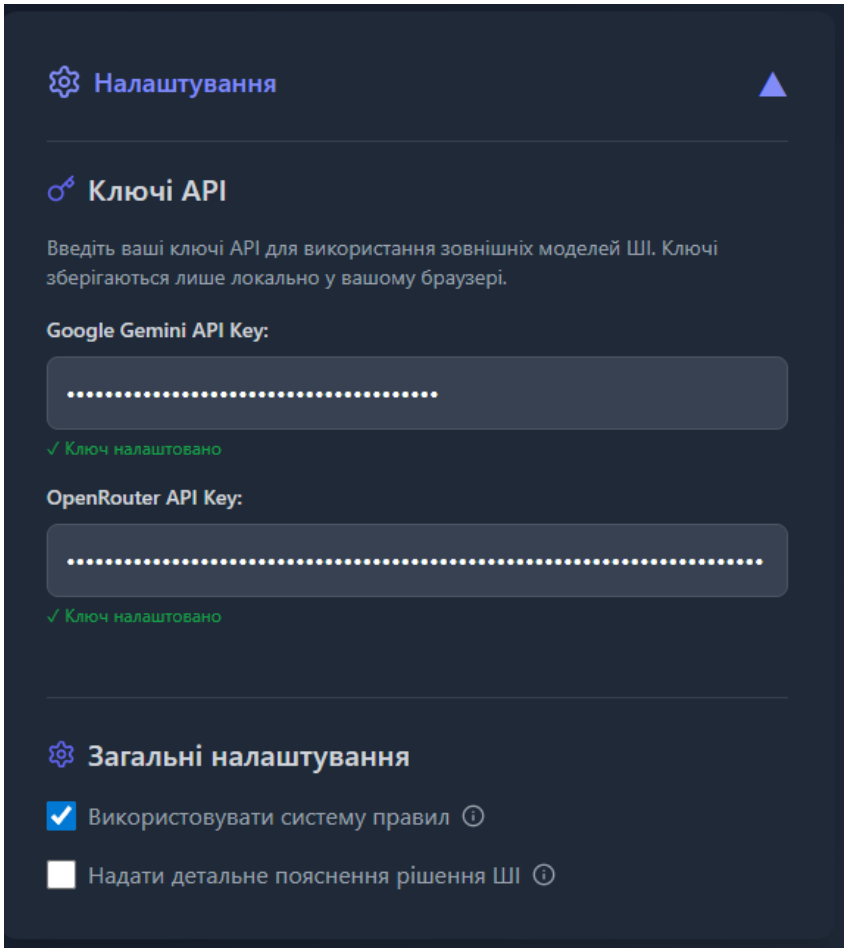


Рисунок 9 – Меню налаштувань плагіну

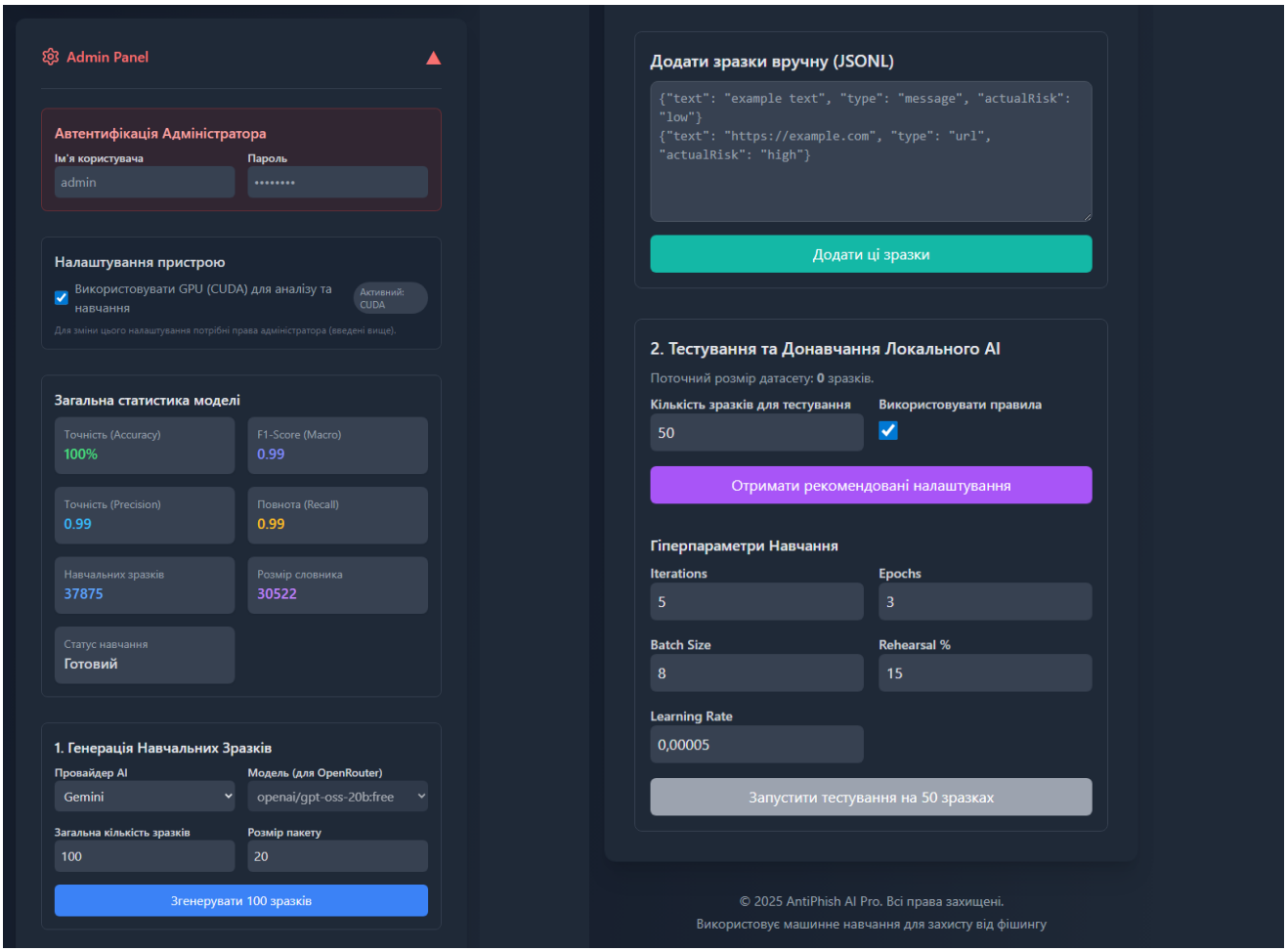


Рисунок 10 – Вигляд панелі адміністратора

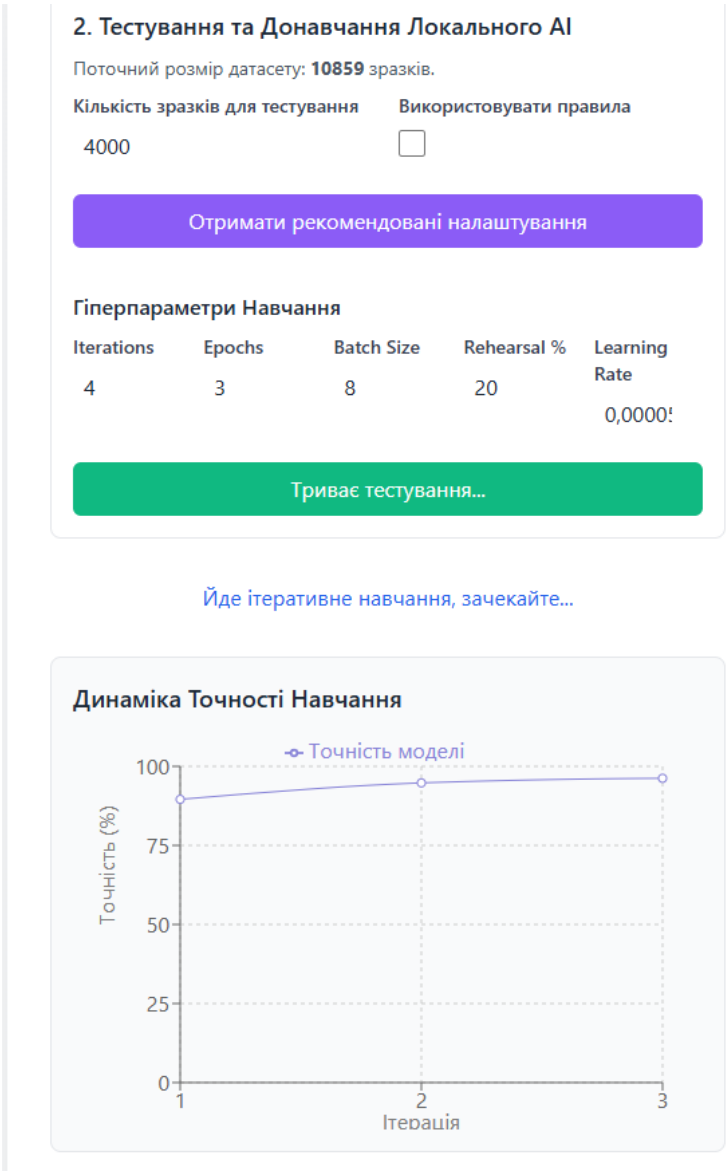


Рисунок 11 – Процес навчання моделі в панелі адміністратора

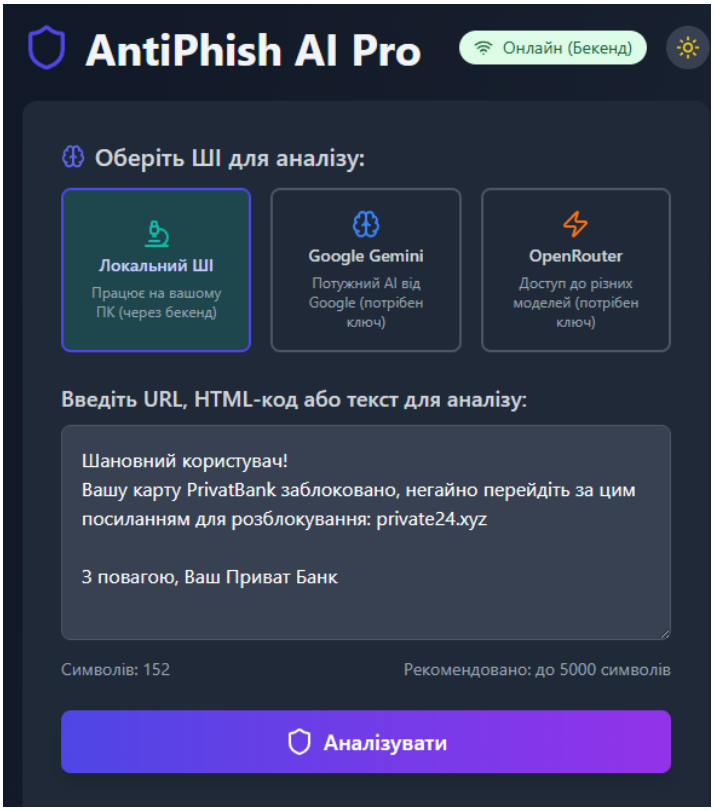


Рисунок 12 – Приклад визначення фішингового листа

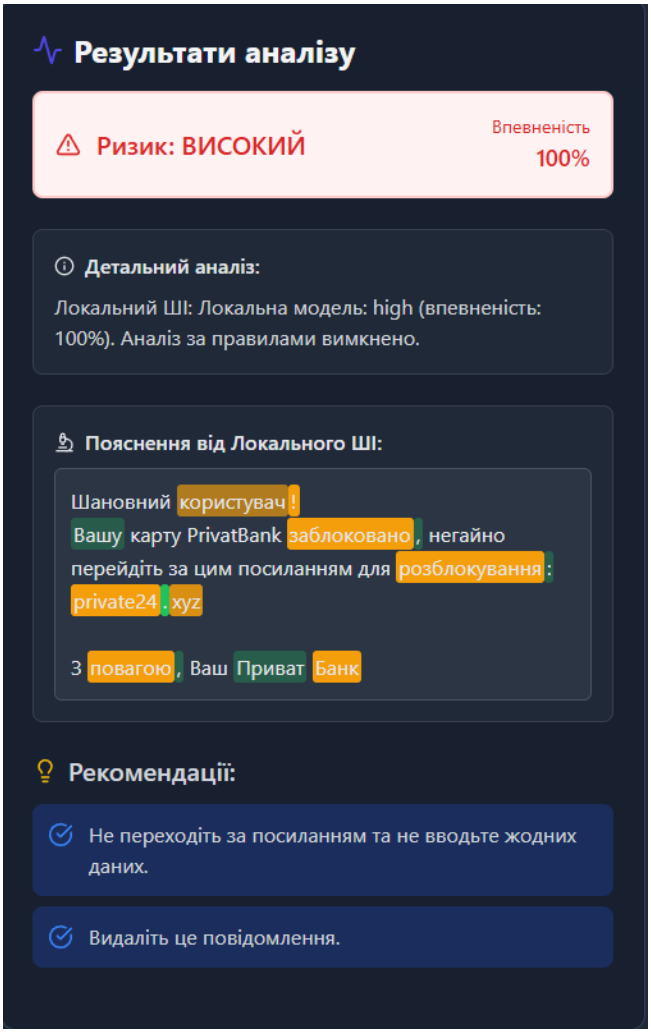


Рисунок 13 – Приклад визначення фішингового листа

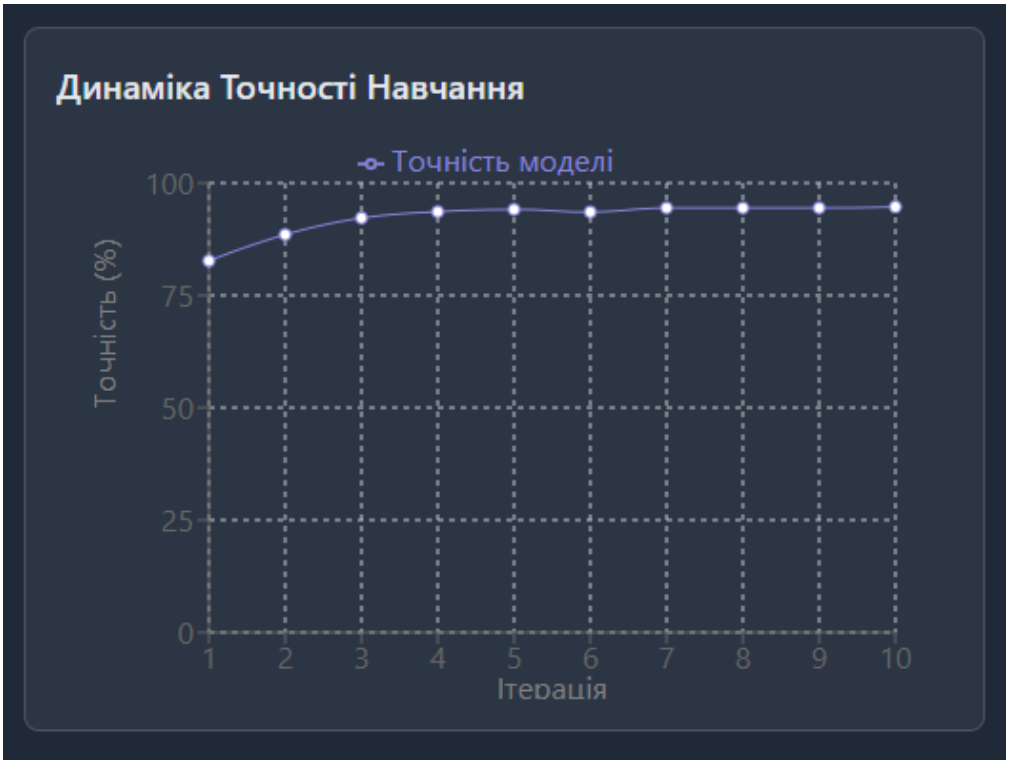


Рисунок 14 – Динаміка точності навчання та визначення фішингу

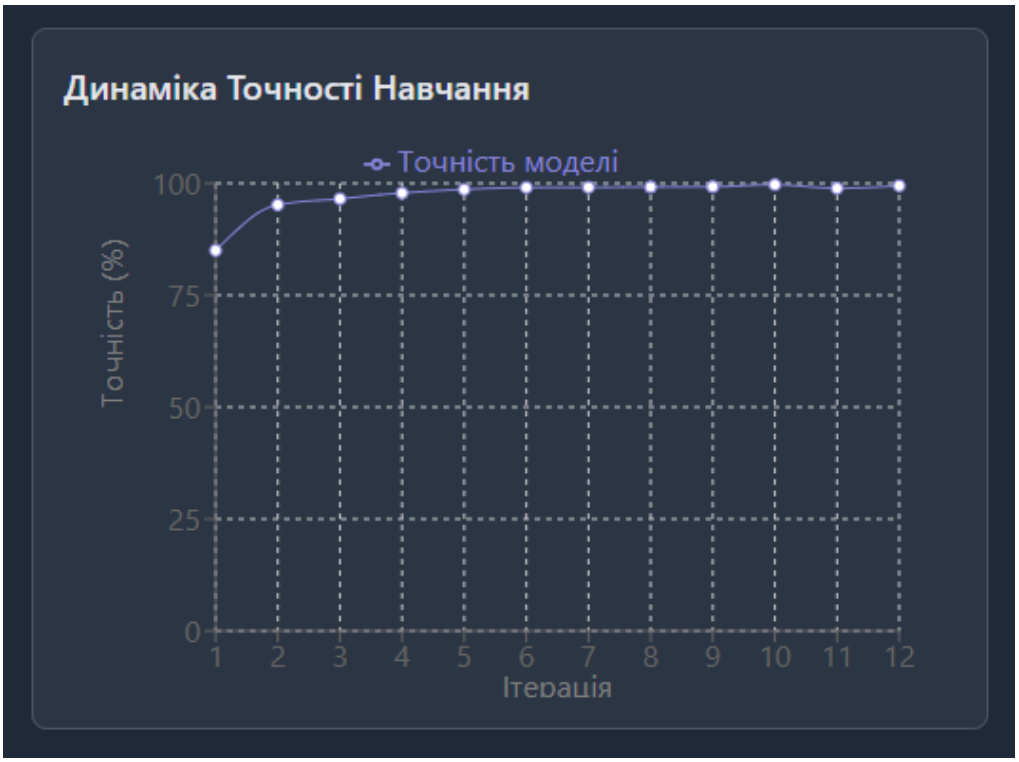


Рисунок 15 – Динаміка точності навчання та визначення фішингу

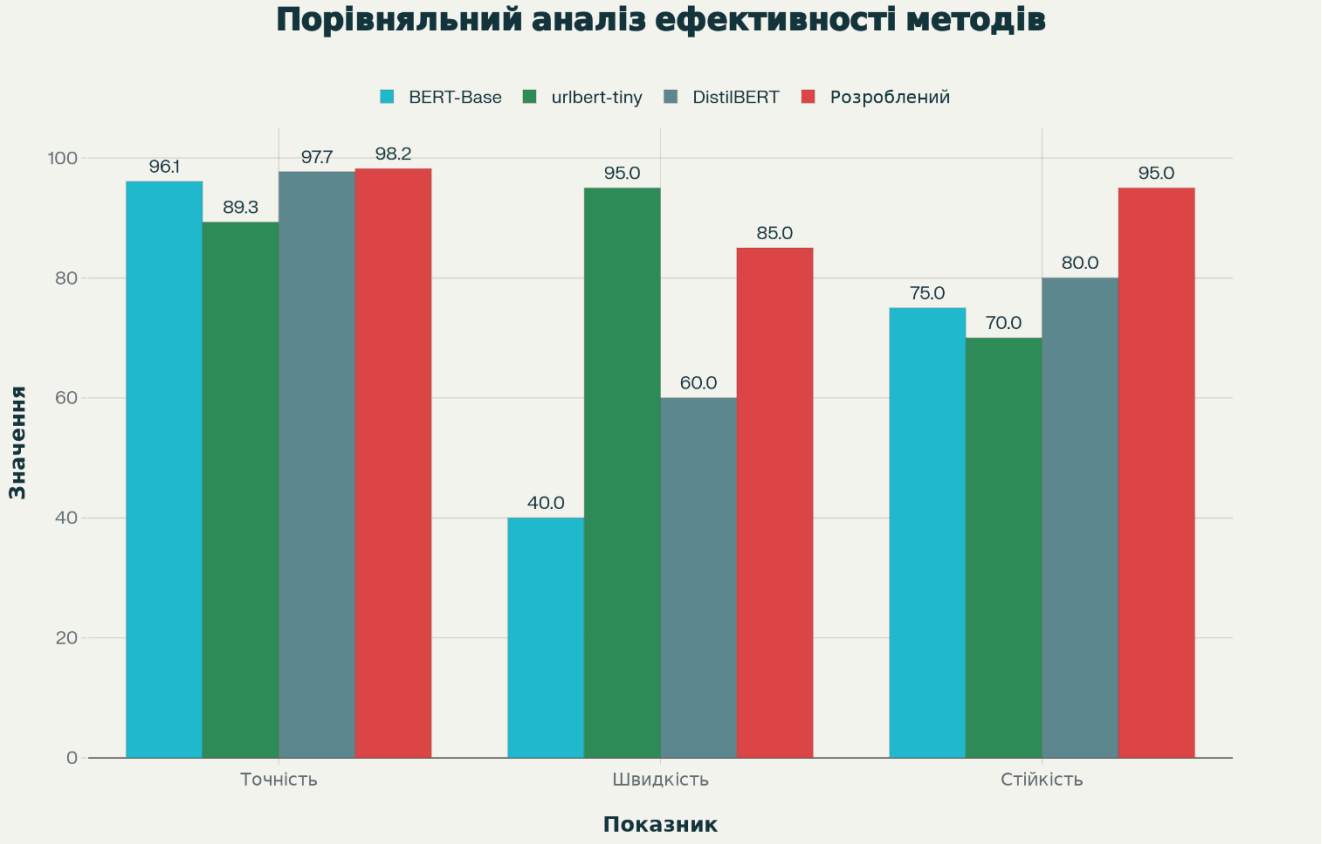


Рисунок 16 – Порівняльний аналіз ефективності методів виявлення фішингових загроз

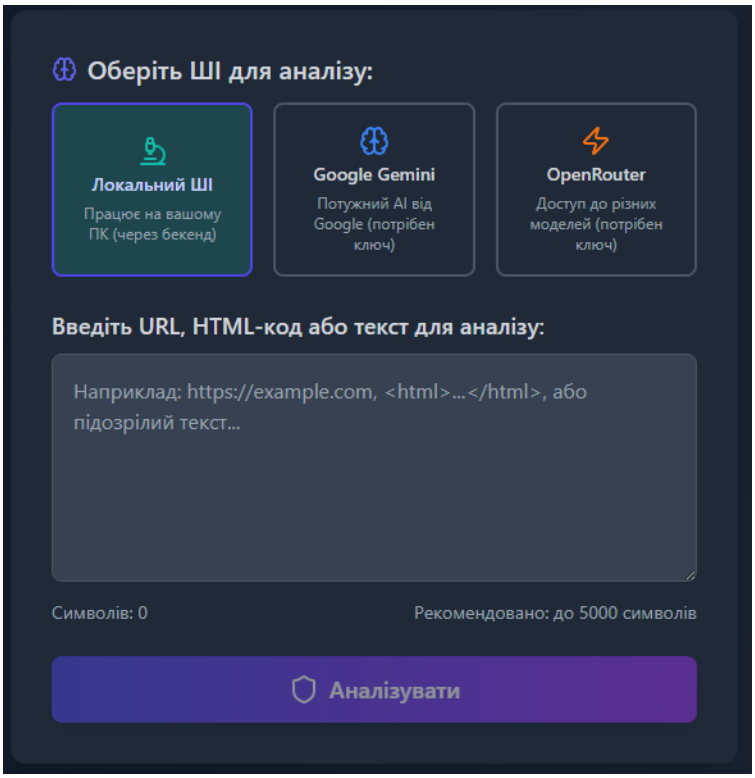


Рисунок 17 – Вікно для взаємодії з користувачем

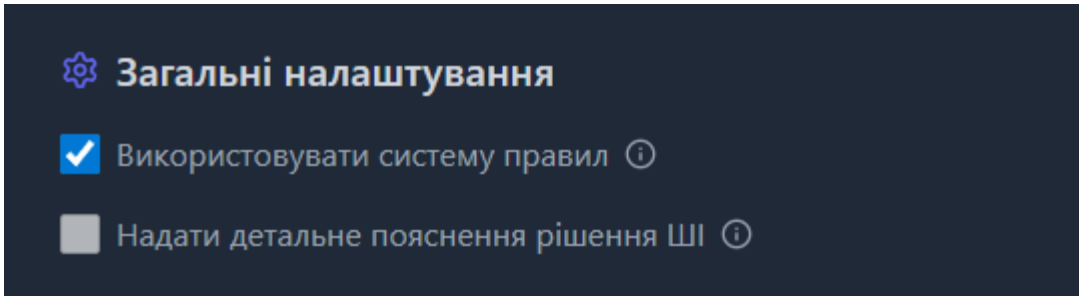


Рисунок 18 – Перемикач системи правил

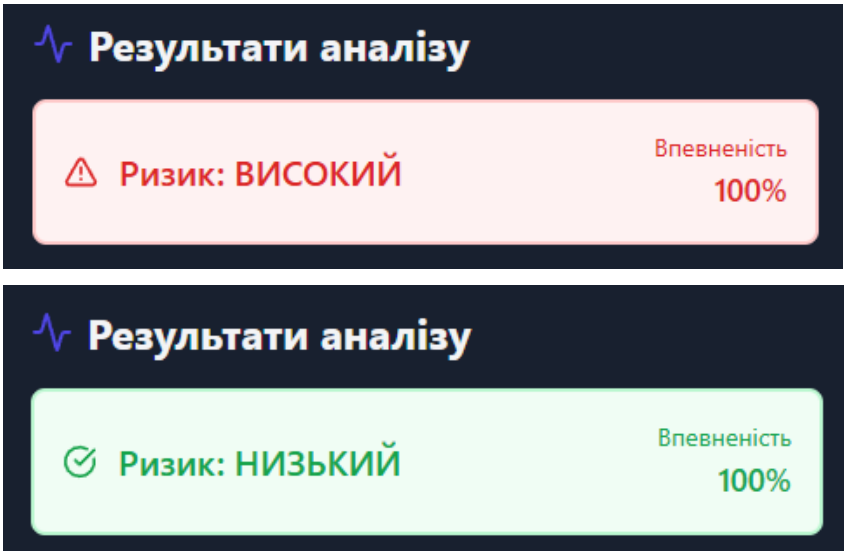
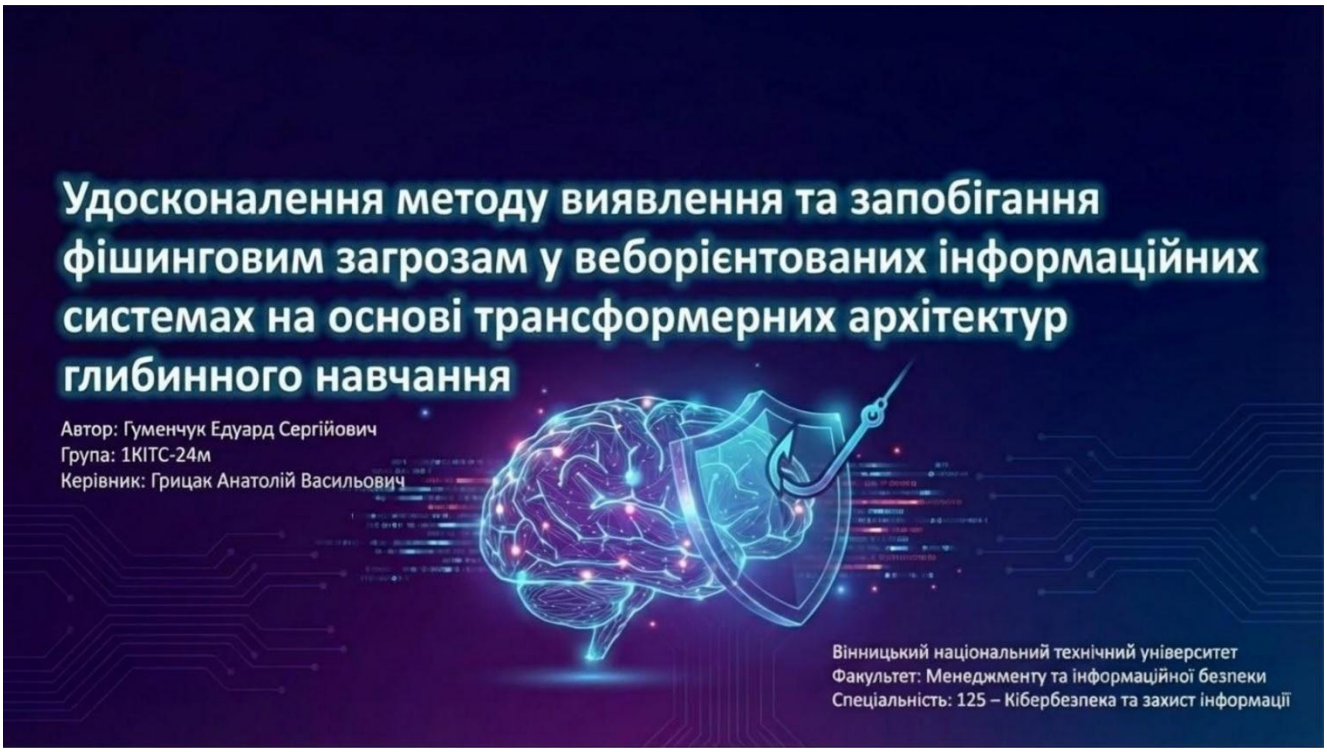


Рисунок 19 – Результати аналізу у відсотках та ризику



АКТУАЛЬНІСТЬ ФІШИНГОВИХ ЗАГРОЗ

ЗРОСТАННЯ AI-ФІШИНГУ

44% успіху AI-генерованих листів (проти 19-28% раніше)

ГОЛОСОВИЙ ФІШИНГ (ДІПФЕЙК)

Збільшення на **+550%** за рік

ВИТОКИ ДАНИХ

90% витоків даних від фішингу

Середній час виявлення фішингової атаки: **277 днів.** Це підкреслює критичну потребу в удосконалених методах захисту.

ДОСЛІДЖЕННЯ

МЕТА

Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах шляхом інтеграції **трансформерних архітектур** глибинного навчання.

ЗАВДАННЯ

- Аналіз існуючих методів
- Розробка гібридної архітектури
- Впровадження трансформерної моделі
- Генерація синтетичних даних
- Оцінка ефективності системи
- Забезпечення інтерпретованості (XAI)


ВИДИ ФІШИНГОВИХ АТАК
МАСОВИЙ ФІШИНГ
Широкомасштабна розсилка без персоналізації
СПІРФІШИНГ
Цільова атака на конкретних осіб
ВЕЙЛІНГ
Атака на топ-менеджерів ("великих риб")
СМІШИНГ (SMS)
Фішинг через текстові повідомлення
ВІШИНГ (ГОЛОС)
Фішинг за допомогою голосових дзвінків


ЖИТТЄВИЙ ЦИКЛ ФІШИНГОВОЇ АТАКИ
Життєвий Цикл Атаки
1 Планування
2 Доставка
3 Взаємодія
4 Викрадення / Використання
Вибір цілі, збір інформації, підготовка інструментів
Надсилання шкідливого повідомлення (email, SMS, дзвінок)
Користувач переходить за посиланням або надає дані
Компрометація даних, фінансові втрати, подальші атаки

ПОРІВНЯННЯ МЕТОДІВ ВИЯВЛЕННЯ ФІШИНГУ

Порівняння Методів Виявлення

Метод	Точність	Адаптивність	Zero-day	Обробка мови
Правила	Середня	Низька	Низьке	Обмежена
Чорні списки	Висока (відомі)	Відсутня	Відсутнє	Низька
ML традиційне	Висока	Середня	Середнє	Середня
Deep Learning	Дуже висока	Висока	Високе	Висока
Трансформери	Виняткова	Виняткова	Дуже високе	Виняткова

✅ Трансформерні архітектури пропонують найкращу комбінацію точності, адаптивності та обробки природної мови для виявлення складних фішингових загроз.

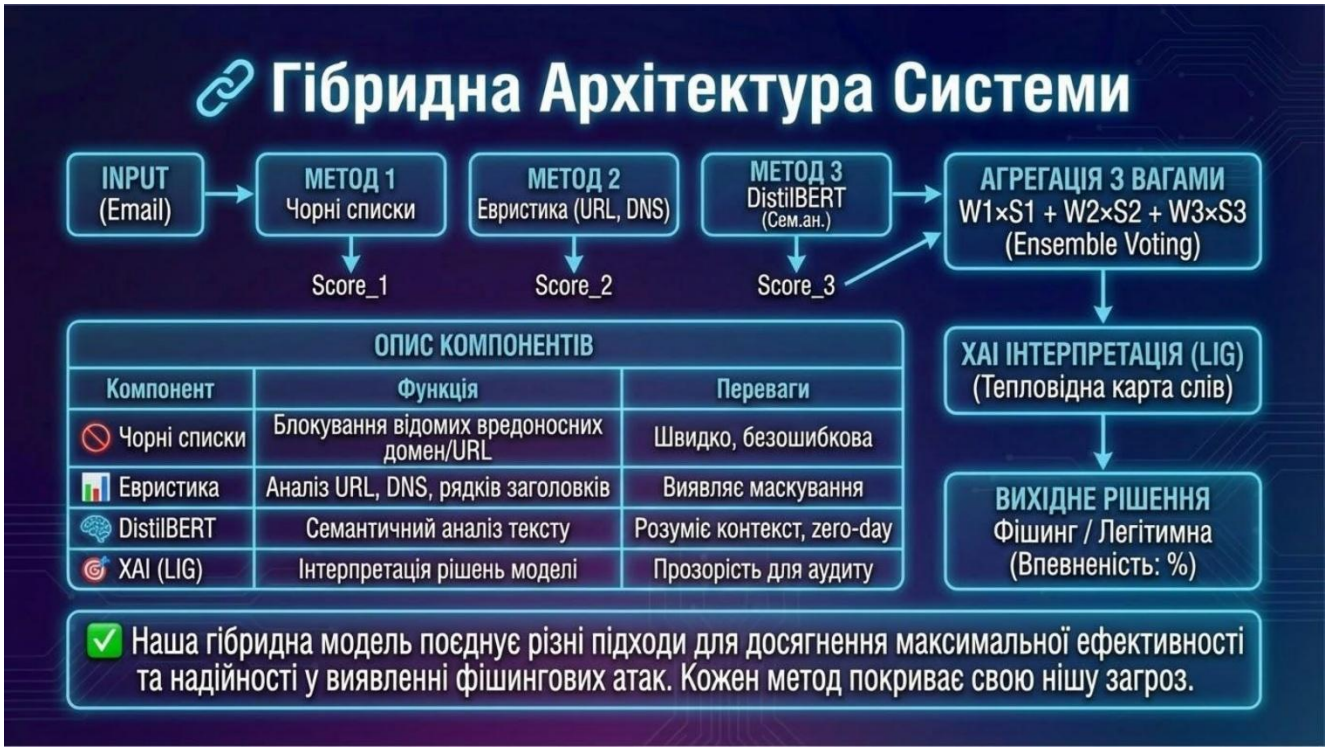
Трансформерні Архітектури: DistilBERT

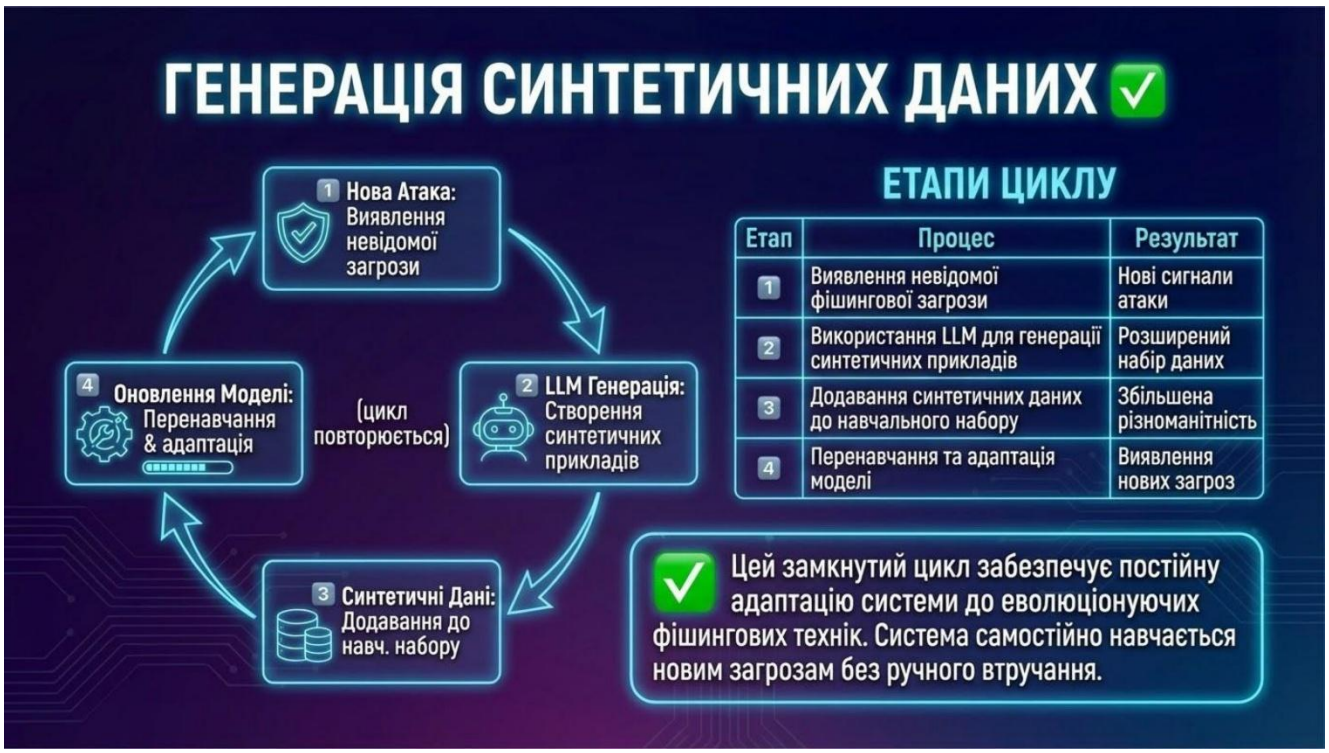
Ключові переваги:

- ✅ Зменшений розмір моделі
- ✅ Висока швидкість інференції
- ✅ Збереження більшості переваг BERT
- ✅ Глибоке розуміння контексту

Параметр	BERT	DistilBERT
Параметри	110M	67M (-40%)
Швидкість	базова	+40% швидше
Точність	96.1%	97.7%
Час інференції	120-150мс	80-100мс

✅ Використання DistilBERT дозволяє ефективно аналізувати семантику тексту для виявлення фішингових ознак, навіть у замаскованих повідомленнях.


Гібридна Архітектура Системи
INPUT (Email)
МЕТОД 1 Чорні списки
МЕТОД 2 Евристика (URL, DNS)
МЕТОД 3 DistilBERT (Сем.ан.)
АГРЕГАЦІЯ З ВАГАМИ W1×S1 + W2×S2 + W3×S3 (Ensemble Voting)
Score_1
Score_2
Score_3
ОПИС КОМПОНЕНТІВ
Компонент | Функція | Переваги
Чорні списки | Блокування відомих вредоносних домен/URL | Швидко, безошибкова
Евристика | Аналіз URL, DNS, рядків заголовків | Виявляє маскування
DistilBERT | Семантичний аналіз тексту | Розуміє контекст, zero-day
XAI (LIG) | Інтерпретація рішень моделі | Прозорість для аудиту
XAI ІНТЕРПРЕТАЦІЯ (LIG) (Тепловідна карта слів)
ВИХІДНЕ РІШЕННЯ Фішинг / Легітимна (Впевненість: %)
Наша гібридна модель поєднує різні підходи для досягнення максимальної ефективності та надійності у виявленні фішингових атак. Кожен метод покриває свою нішу загроз.


ГЕНЕРАЦІЯ СИНТЕТИЧНИХ ДАНИХ
1 Нова Атака: Виявлення невідомої загрози
2 LLM Генерація: Створення синтетичних прикладів
3 Синтетичні Дані: Додавання до навч. набору
4 Оновлення Моделі: Перенавчання & адаптація
(цикл повторюється)
ЕТАПИ ЦИКЛУ
Етап | Процес | Результат
1 | Виявлення невідомої фішингової загрози | Нові сигнали атаки
2 | Використання LLM для генерації синтетичних прикладів | Розширений набір даних
3 | Додавання синтетичних даних до навчального набору | Збільшена різноманітність
4 | Перенавчання та адаптація моделі | Виявлення нових загроз
Цей замкнутий цикл забезпечує постійну адаптацію системи до еволюціонуючих фішингових технік. Система самостійно навчається новим загрозам без ручного втручання.

Технологічний Стек та Архітектура

РІВЕНЬ КОРИСТУВАЧА (UI)
Веб-інтерфейс | Розширення (React.js, HTML5) (Chrome Ext)

REST API

РІВЕНЬ ДОДАТКУ (Backend)
API Server | Task Queue (FastAPI / Flask) (Celery/Redis)

gRPC

РІВЕНЬ ML & ОБРОБКИ
DistilBERT ONNX | Heuristics (PyTorch, (Regex, DNS)

SQL

РІВЕНЬ ДАНИХ (Data)
База Даних (SQLite) | Logs / Cache (Elasticsearch)

ТЕХНОЛОГІЇ

Категорія	Інструменти
Backend	Python, FastAPI
ML Core	PyTorch, Hugging Face
Database	SQLite
DevOps	Git, CI/CD

✅ Ця детальна архітектура забезпечує чітке розділення відповідальності між компонентами та ефективну взаємодію між рівнями, підтримуючи масштабованість та модульність системи.

РЕЗУЛЬТАТИ ТА ПОРІВНЯННЯ ✅

РЕЗУЛЬТАТИ ГІБРИДНОЇ СИСТЕМИ

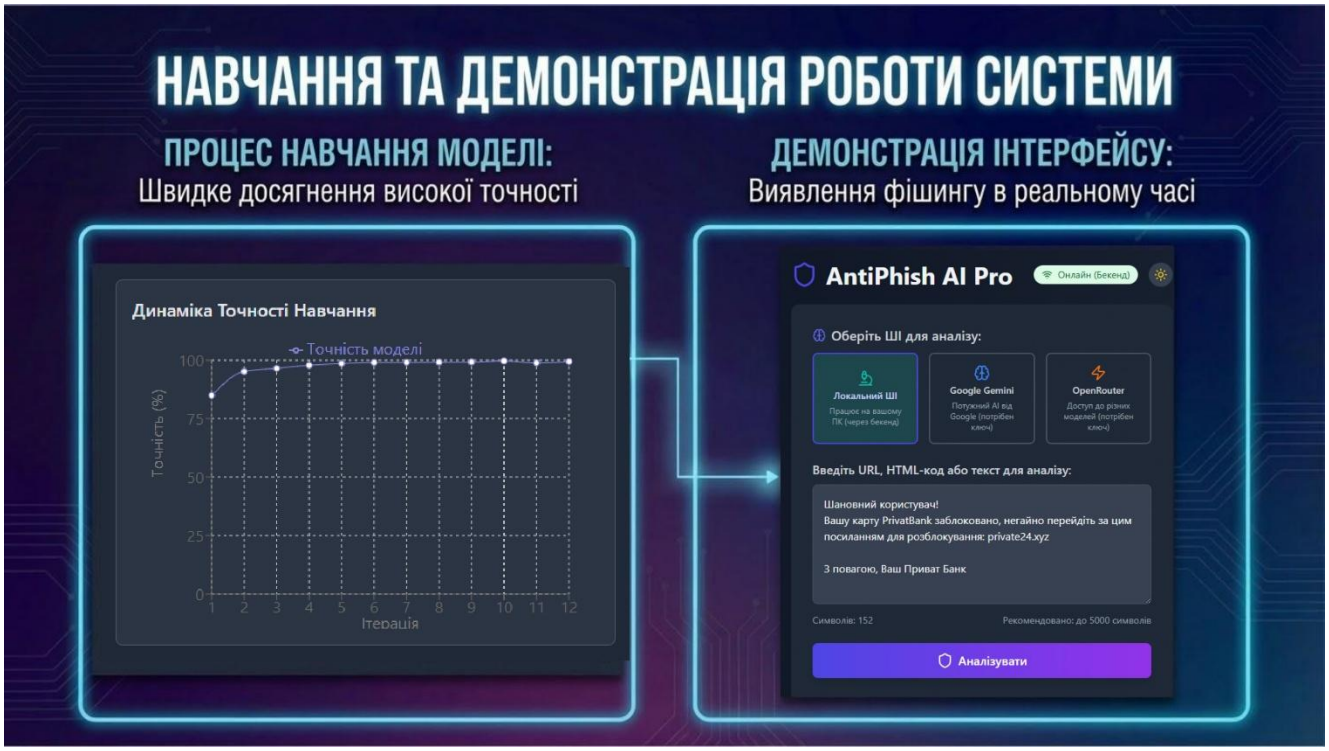
98.2%	97.9%	98.6%	98.4%
Точність	Повнота	Precision	F1-Score

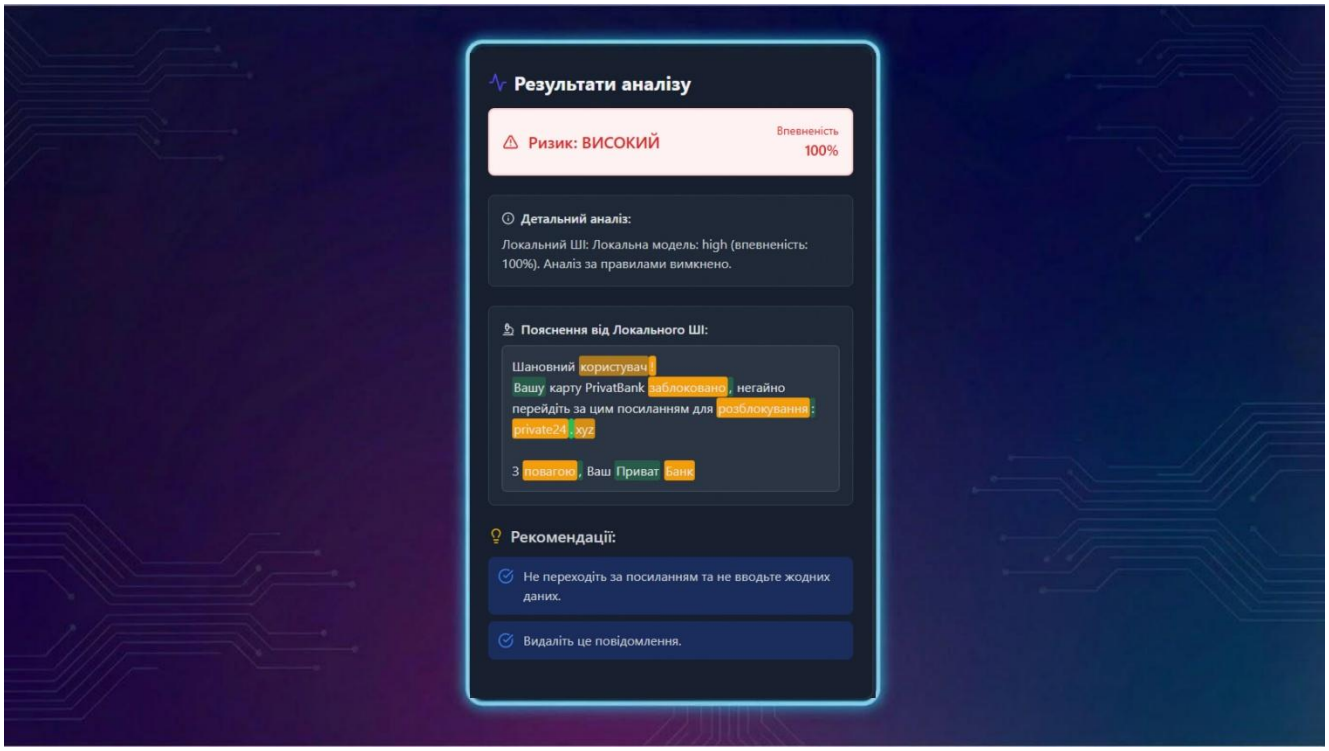
ПОРІВНЯННЯ З АНАЛОГАМИ

Модель	Гібридна Система	DistilBERT v2.1	BERT-Base	URLBert-Tiny v2
Точність	98.2%	97.7%	96.1%	89.3%
Zero-day	91-94%	85-88%	82-85%	75-78%
FPR (Помилка I роду)	1.2%	2.3%	3.2%	8.5%
FNR (Помилка II роду)	1.8%	2.3%	3.9%	10.7%

КЛЮЧОВІ ВИСНОВКИ

- ✅ **Найвища точність:** Наша система перевершує всі аналоги, досягаючи 98.2%.
- ✅ **Найкраща адаптація:** Ефективність на ‘нульовому дні’ 91-94% — значно краще за аналоги.
- ✅ **Мінімум помилок:** Найнижчі показники FPR (1.2%) та FNR (1.8%) демонструють надійність.
- ✅ **Перевага гібриду:** Комбінація евристик та DistilBERT дає кращі результати, ніж ‘чисті’ ML-моделі.


НАВЧАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ
ПРОЦЕС НАВЧАННЯ МОДЕЛІ:
Швидке досягнення високої точності
ДЕМОНСТРАЦІЯ ІНТЕРФЕЙСУ:
Виявлення фішингу в реальному часі
Динаміка Точності Навчання
Точність моделі
Точність (%)
Ітерація
AntiPhish AI Pro
Онлайн (Бекенд)
Оберіть ШІ для аналізу:
Локальний ШІ
Google Gemini
OpenRouter
Введіть URL, HTML-код або текст для аналізу:
Шановний користувач!
Вашу карту PrivatBank заблоковано, негайно перейдіть за цим посиланням для розблокування: private24.xyz
З повагою, Ваш Приват Банк
Символів: 152
Рекомендовано: до 5000 символів
Аналізувати


Результати аналізу
Ризик: ВИСОКИЙ
Впевненість
100%
Детальний аналіз:
Локальний ШІ: Локальна модель: high (впевненість: 100%). Аналіз за правилами вимкнено.
Пояснення від Локального ШІ:
Шановний користувач!
Вашу карту PrivatBank заблоковано, негайно перейдіть за цим посиланням для розблокування:
private24.xyz
З повагою, Ваш Приват Банк
Рекомендації:
Не переходіть за посиланням та не вводьте жодних даних.
Видаліть це повідомлення.

ІНТЕРПРЕТАБЕЛЬНІСТЬ МОДЕЛІ (XAI) 🔍

МЕТОД: LAYER INTEGRATED GRADIENTS

Визначає вплив кожної ознаки на рішення моделі

ПРИКЛАД ІНТЕРПРЕТАЦІЇ

Вхідний текст: "Підтвердьте свій аккаунт PayPal за цим посиланням bit.ly/xyz123".

Ознака	Вплив на Рішення	Інтерпретація
Підтвердьте свій аккаунт	🔴 Критичний	Класична фішингова фраза верифікації
PayPal	🔴 Критичний	Найчастіше підробляється платіжне ім'я
bit.ly/xyz123	🔴 Критичний	Скорочене посилання приховує справжній домен
за цим посиланням	🟡 Високий	Призив до дії — типова тактика соціальної інженерії

Результат: $P_{neural} = 0.92$ → Предикція: Фішинг ✅

ПЕРЕВАГИ LIG ДЛЯ СИСТЕМИ

- ✅ **Прозорість:** Користувачам видно, які ознаки привели до рішення
- ✅ **Надійність:** Gradient-based метод математично коректний
- ✅ **Масштабованість:** Працює без додаткового навантаження на кожне повідомлення.

АРХІТЕКТУРА ГІБРИДНОЇ СИСТЕМИ 🏗️

КОМПОНЕНТИ СИСТЕМИ

Система складається з двох основних модулів, які працюють послідовно:

1. Модуль евристик (Rule-Based)
Швидкий аналіз евристичних правил. Виявляє очевидні сигнали фішингу за 3–5 мс. Повертає попередню оцінку.

2. Модуль нейромережі (DistilBERT)
Глибокий аналіз тексту повідомлення за 80–100 мс. Обробляє 768-вимірні вектори embedding. Повертає ймовірність ризику

ПРОЦЕС КЛАСИФІКАЦІЇ

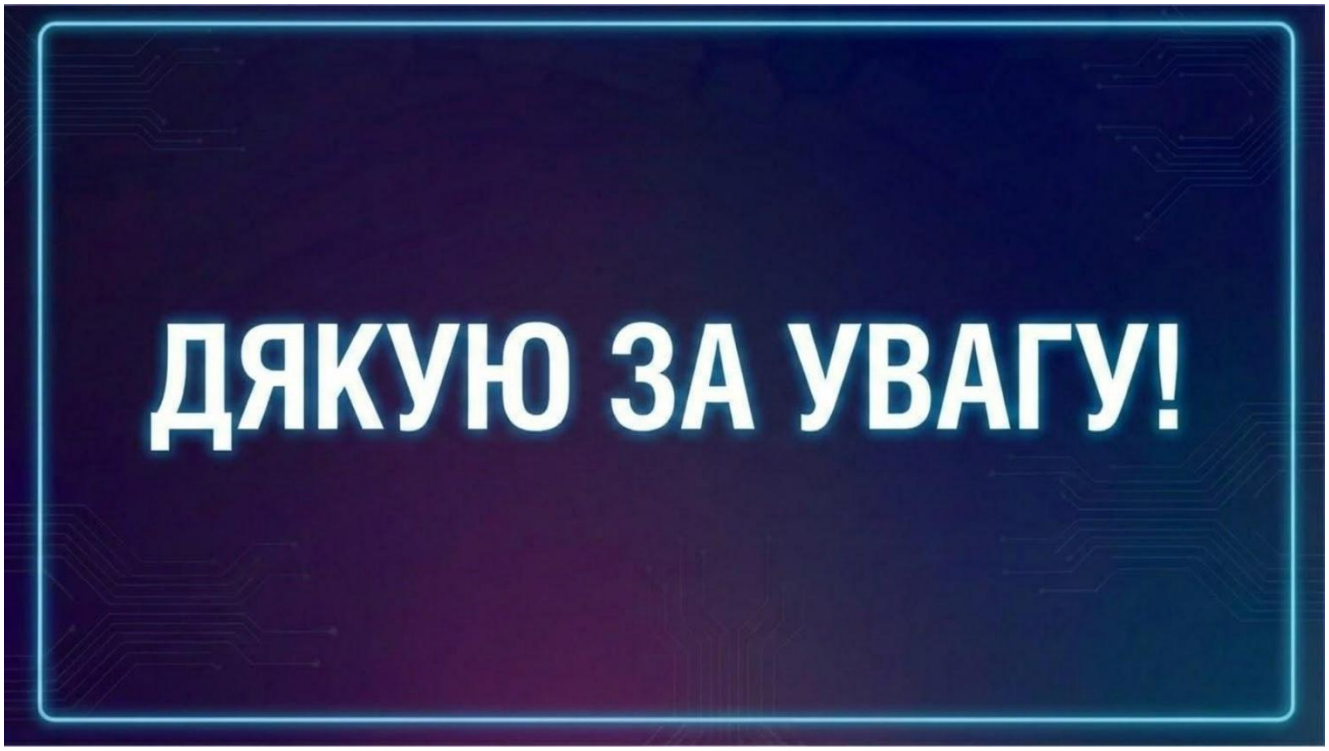
Вхід: Текст електронного листа або фішингового посилання
→
Крок 1 – Евристичний скринінг: Застосування правил (IP-адреси, скорочені посилання, Unicode символи тощо)
→
Крок 2 – Нейромережа: Якщо евристики невпевнені, передаємо текст DistilBERT
→
Крок 3 – Комбінування: результати обираються за принципом максимального ризику одного з методів
→
Крок 4 – Довіра: мехнізм вказує на ключові токени, що вибрала модель
→
Вихід: класифікація за рівнем впевненості системи

ПЕРЕВАГИ ГІБРИДНОГО ПІДХОДУ

- ✅ **Швидкість:** Евристики відбирають очевидні випадки за 3–5 мс.
- ✅ **Точність:** DistilBERT ловить складні атаки, які евристики пропускають.
- ✅ **Надійність:** Zero-day адаптація через контекст (91–94%).
- ✅ **Масштабованість:**

CPU/GPU оптимізація, 60–95 запитів на секунду


ВИСНОВКИ І НОВИЗНА
ОСНОВНІ РЕЗУЛЬТАТИ РОБОТИ
Гібридна система: Евристики + DistilBERT. Точність 98.2%.
Оптимізація: Економія пам'яті 40%, прискорення 1.5x (vs BERT).
Інновації: Синтетичне навчання (LLM) та XAI (Layer Integrated Gradients).
НОВИЗНА
Адаптивність: Замкнутий цикл генерації даних для подолання концептуального дрейфу.
Захист від атак нульового дня: Точність 91–94% на нових типах атак.
Мультимовність: Аналіз Unicode (кирилиця/латиниця).
ПРАКТИЧНЕ ЗНАЧЕННЯ
Ефективність: Precision 98.6%, швидкість 60–95 req/sec.
Економіка: ROI 217%, окупність 3.7 міс.


ДЯКУЮ ЗА УВАГУ!

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Удосконалення методу виявлення та запобігання фішинговим загрозам у веборієнтованих інформаційних системах на основі трансформерних архітектур глибинного навчання

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра менеджменту та безпеки інформаційних систем
факультет менеджменту та інформаційної безпеки
гр.1КІТС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) **0,51 %**

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту**

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

к.т.н., доцент, зав. каф. МБІС Карпінець В.В.

к.ф.-м.н., доцент каф. МБІС Шиян А.А.

Особа, відповідальна за перевірку Коваль Н.П.

З висновком експертної комісії ознайомлений(-на)

Керівник Грицак А.В.

Здобувач Гуменчук Е.С.