МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ІДЕНТИФІКАЦІЇ ФІШИНГУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач _____ Анастасія ЦИМБАЛ

«___»_____ 2025 р.

Керівник д-р техн. наук, професор _____ Ірина КАЛІНІНА

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Цимбал Анастасії Юріївни
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтелектуальна система ідентифікації фішингу».
Керівник роботи: Калініна Ірина Олександрівна, професор кафелри ІІС, д-р. техн. наук.
Затверджена наказом ЧНУ ім. Петра Могили від «7» липня 2025 р. № 184.
2. Строк представлення кваліфікаційної роботи «16» грудня 2025 р.
3. Очікуваний результат роботи та початкові дані:_набори даних фішингових та легітимних повідомлень та вебпосилань з відкритих джерел (Phishing Dataset, Phishing Email Dataset, Phishing URL Dataset); користувацькі (кастомні) приклади повідомлень і посилань, що формуються через веб-інтерфейс системи; попередньо оброблені текстові вектори для навчання моделей; методи машинного навчання (Logistic Regression, Naive Bayes, Support Vector Machine, Random Forest, Gradient Boosting) для класифікації контенту; метрики оцінки якості моделей (precision,

recall, f1-score, accuracy). Очікуваний результат: інтелектуальна система для автоматичного виявлення фішингових повідомлень і посилань.

4. Перелік питань, що підлягають розробці:_аналіз сучасного стану проблеми фішингових атак, їхніх видів та особливостей реалізації; огляд та порівняння існуючих методів і підходів до виявлення фішингових атак; вибір та обґрунтування методів машинного навчання, застосованих для класифікації фішингових повідомлень та URL-адрес; формування наборів ознак для побудови моделей класифікації, зокрема текстових характеристик повідомлень та ознак URL-адрес; розробка інтелектуальної системи ідентифікації фішингових повідомлень та посилань; тестування та оцінка ефективності розробленої системи.

5. Перелік графічних матеріалів: презентація, рисунки, таблиці.

Керівник роботи _____ _____Ірина КАЛІНІНА_____
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ ____Анастасія ЦИМБАЛ_____
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «07» липня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система ідентифікації фішингу

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	27.06.2025	30.06.2025	
2	Аналіз предметної області та постановка задачі	1.07.2025	20.07.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз публікацій та аналогічних систем, щодо ідентифікації фішингу	21.07.2025	30.07.2025	
4	Огляд алгоритмів машинного навчання для вирішення поставленої задачі	01.09.2025	25.10.2025	
5	Реалізація обраних технологій з аналізом отриманих результатів	26.10.2025	21.11.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	25.11.2025	26.11.2025	
7	Корегування роботи за результатами попереднього захисту	27.11.2025	05.12.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	09.12.2025	09.12.2025	
9	Доробка та остаточне оформлення КР	10.12.2025	12.12.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.12.2025	16.12.2025	

Керівник роботи _____ Ірина КАЛІНІНА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Анастасія ЦИМБАЛ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«07» липня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 601м ЧНУ ім. Петра Могили

Цимбал Анастасії Юріївни

на тему: **“ІНТЕЛЕКТУАЛЬНА СИСТЕМА ІДЕНТИФІКАЦІЇ ФІШИНГУ”**

Фішингові атаки є одними з найпоширеніших видів кіберзлочинів, спрямованих на отримання конфіденційної інформації користувачів шляхом соціальної інженерії та підробки вебресурсів. Щороку кількість таких атак зростає, завдаючи значної шкоди як окремим користувачам, так і великим організаціям. У зв’язку з цим, розробка інтелектуальних систем для автоматичного виявлення фішингових повідомлень та посилань є важливим напрямом підвищення рівня кібербезпеки та запобігання кіберзагрозам.

Об'єктом дослідження є процеси автоматичного виявлення та класифікації фішингових повідомлень у цифрових комунікаційних системах, а також підробних URL-адрес, що використовуються у фішингових атаках.

Предметом дослідження є методи і моделі машинного навчання для ідентифікації фішингових повідомлень і посилань на основі аналізу текстових і URL-ознакових характеристик.

Метою дослідження є підвищення рівня кібербезпеки, своєчасного попередження користувачів про потенційні загрози та зниження ризику несанкціонованого доступу до конфіденційної інформації. Для досягнення мети використано методи машинного навчання, обробки текстових даних та аналізу URL-адрес.

Дана робота складається з чотирьох розділів, у яких розглянуто аналіз проблеми фішингових атак і сучасних підходів до їх виявлення, зроблено проєктування архітектури інтелектуальної системи та вибір методів машинного навчання, програмну реалізацію системи PhisIdent, а також наведено результати навчання, тестування й оцінку ефективності розробленої системи з урахуванням користувацького зворотного зв’язку.

Кваліфікаційна робота містить 98 сторінок, 33 рисунка, 14 таблиць, 28 використаних джерела та 7 додатків.

Ключові слова: *виявлення фішингових атак, фішингові повідомлення, фішингові посилання, кібербезпека, машинне навчання, інтелектуальні системи, класифікація текстів, аналіз даних.*

ABSTRACT

to the qualification work by the student of the group 601m of Petro Mohyla Black Sea National University

Tsymbal Anastasiia

“INTELLIGENT PHISHING IDENTIFICATION SYSTEM”

Phishing attacks are among the most widespread types of cybercrime, aimed at obtaining users’ confidential information through social engineering techniques and the spoofing of web resources. Each year, the number of such attacks increases, causing significant damage to both individual users and large organizations. In this context, the development of intelligent systems for the automatic detection of phishing messages and links is an important direction for enhancing cybersecurity and preventing cyber threats.

The object of the study is the processes of automatic detection and classification of phishing messages in digital communication systems, as well as fraudulent URL addresses used in phishing attacks.

The subject of the study is machine learning methods and models for identifying phishing messages and links based on the analysis of textual features and URL-based characteristics.

The aim of the research is to increase the level of cybersecurity, provide timely warnings to users about potential threats, and reduce the risk of unauthorized access to confidential information. To achieve this aim, machine learning methods, text data processing techniques, and URL analysis are employed.

This work consists of four chapters, which cover the analysis of the phishing attack problem and modern approaches to its detection, the design of the intelligent system architecture and the selection of machine learning methods, the software implementation of the PhisIdent system, as well as the results of training, testing, and evaluating the effectiveness of the developed system with consideration of user feedback.

The qualification thesis comprises 98 pages, 33 figures, 14 tables, 28 references, and 7 appendices.

Keywords: *phishing attack detection, phishing messages, phishing links, cybersecurity, machine learning, intelligent systems, text classification, data analysis.*

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІС	– інтелектуальна система
ПЗ	– програмне забезпечення
AI	– Artificial Intelligence
AJAX	– Asynchronous JavaScript and eXtensible Markup Language
API	– Application Programming Interface
BDI	– Brand Domain Identification
BERT	– Bidirectional Encoder Representations from Transformers
CSS	– Cascading Style Sheet
DevOps	– Development & Operations
DKIM	– DomainKeys Identified Mail
DMARC	– Domain-based Message Authentication, Reporting, and Conformance
DNS	– Domain Name System
GB	– Gradient Boosting
GIF	– Graphics Interchange Format
HTML	– HyperText Markup Language
HTTP	– HyperText Transfer Protocol
IDE	– Integrated Development Environment
IP	– Internet Protocol
JPCERT	– Japan Computer Emergency Response Team Coordination Center
JS	– JavaScript
JSON	– JavaScript Object Notation
LightGBM	– Light Gradient Boosting Framework
LR	– Logistic Regression
ML	– Machine Learning
NB	– Naive Bayes
NLP	– Natural Language Processing

PAPO – Phishing Attack Process Ontology

QR – Quick Response code

RBF – Radial Basis Function

RF – Random Forest

SMS – Short Message Service

SPF – Sender Policy Framework

SSL – Secure Sockets Layer

SVM – Support Vector Machine

TF-IDF – Term Frequency – Inverse Document Frequency

UFO – Unified Foundational Ontology

UML – Unified Modeling Language

URL – Uniform Resource Locator

UX – User Experience

VoIP – Voice over Internet Protocol

VS Code – Visual Studio Code

XGBoost – eXtreme Gradient Boosting

XSS – Cross Site Scripting

ВСТУП

З розвитком інформаційних технологій та збільшенням обсягу цифрової інформації питання кібербезпеки набувають дедалі більшої актуальності. Однією з найпоширеніших загроз у сучасному цифровому середовищі є фішингові атаки, що спрямовані на отримання конфіденційної інформації користувачів шляхом соціальної інженерії та підробки вебресурсів. Щороку кількість таких атак зростає, завдаючи значної шкоди як окремим користувачам, так і великим організаціям.

Існуючі рішення у сфері виявлення фішингу мають певні обмеження. Багато традиційних методів базуються на статичних сигнатурах або чорних списках URL-адрес, що робить їх неефективними проти нових, невідомих та адаптивних фішингових загроз. Методи аналізу, які використовуються більшістю антивірусних програм, часто виявляються недостатніми для своєчасного виявлення фішингових повідомлень або посилань, що створює додаткові ризики для безпеки користувачів.

Провідні компанії та дослідники у сфері кібербезпеки, такі як Symantec, Kaspersky, McAfee та інші, активно працюють над удосконаленням технологій виявлення фішингових атак, впроваджуючи методи машинного навчання та обробки природної мови. Проте, незважаючи на значний прогрес, проблема ефективного виявлення фішингових повідомлень та підробних посилань залишається актуальною.

Актуальність теми полягає у тому, що фішингові атаки є дуже поширеним видом кіберзлочинів, спрямованих на отримання конфіденційної інформації користувачів шляхом соціальної інженерії та підробки вебресурсів. Щороку кількість таких атак зростає, завдаючи значної шкоди як окремим користувачам, так і великим організаціям. У зв’язку з цим, розробка інтелектуальних систем для автоматичного виявлення фішингових повідомлень та посилань є важливим напрямом підвищення рівня кібербезпеки та запобігання кіберзагрозам.

Дана робота пов’язана з численними науковими дослідженнями у галузі кібербезпеки, зокрема зі створенням систем штучного інтелекту для аналізу

текстових даних, виявлення аномалій та фішингових ознак у повідомленнях і URL-адресах. Результати цієї роботи можуть бути використані для підвищення ефективності існуючих систем захисту інформації та розробки нових рішень для боротьби з фішингом.

Мета дослідження полягає у підвищенні рівня кібербезпеки, своєчасного попередження користувачів про потенційні загрози та зниження ризику несанкціонованого доступу до конфіденційної інформації. Для досягнення поставленої мети використано методи машинного навчання, обробки текстових даних та аналізу URL-адрес.

Об’єктом дослідження є процеси автоматичного виявлення та класифікації фішингових повідомлень у цифрових комунікаційних системах, а також підробних URL-адрес, що використовуються у фішингових атаках.

Предметом дослідження є методи і моделі машинного навчання для ідентифікації фішингових повідомлень і посилань на основі аналізу текстових і URL-ознакових характеристик.

У процесі розробки інтелектуальної системи використовувались сучасні інструментальні засоби та платформи. Мова програмування Python застосована для реалізації методів ML та обробки текстових даних, JS разом із HTML та CSS – для створення зручного користувацького інтерфейсу, а середовище VS Code забезпечило зручну розробку та налагодження проєкту.

Отже, дана робота спрямована на розробку інтелектуальної системи автоматичного виявлення фішингових повідомлень і посилань, що забезпечує підвищення рівня кібербезпеки користувачів шляхом своєчасного попередження про потенційно небезпечні ресурси. Реалізована система поєднує методи ML, обробку природної мови та аналіз структурних ознак URL, що дає змогу ефективно ідентифікувати нові, раніше невідомі фішингові загрози. Запропонований підхід сприяє вдосконаленню механізмів виявлення шахрайських повідомлень, зниженню ризику витоку конфіденційних даних та зміцненню загального рівня захисту інформаційних систем.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Види фішингових атак та сучасний стан проблеми

З розвитком цифрових технологій, поширенням інтернет-банкінгу, електронної комерції та онлайн-сервісів питання кібербезпеки стає дедалі актуальнішим. Однією з найбільш небезпечних та поширених кіберзагроз є фішингові атаки, що спрямовані на отримання конфіденційної інформації користувачів (логінів, паролів, платіжних даних, персональних відомостей) шляхом маскування під легітимні ресурси або використання методів соціальної інженерії. За даними звітів провідних антивірусних компаній (Kaspersky, Symantec, McAfee тощо), кількість фішингових атак щороку зростає, а їхні методи ускладнюються [1]. Існують різні види фішингових атак (див. рис. 1.1).

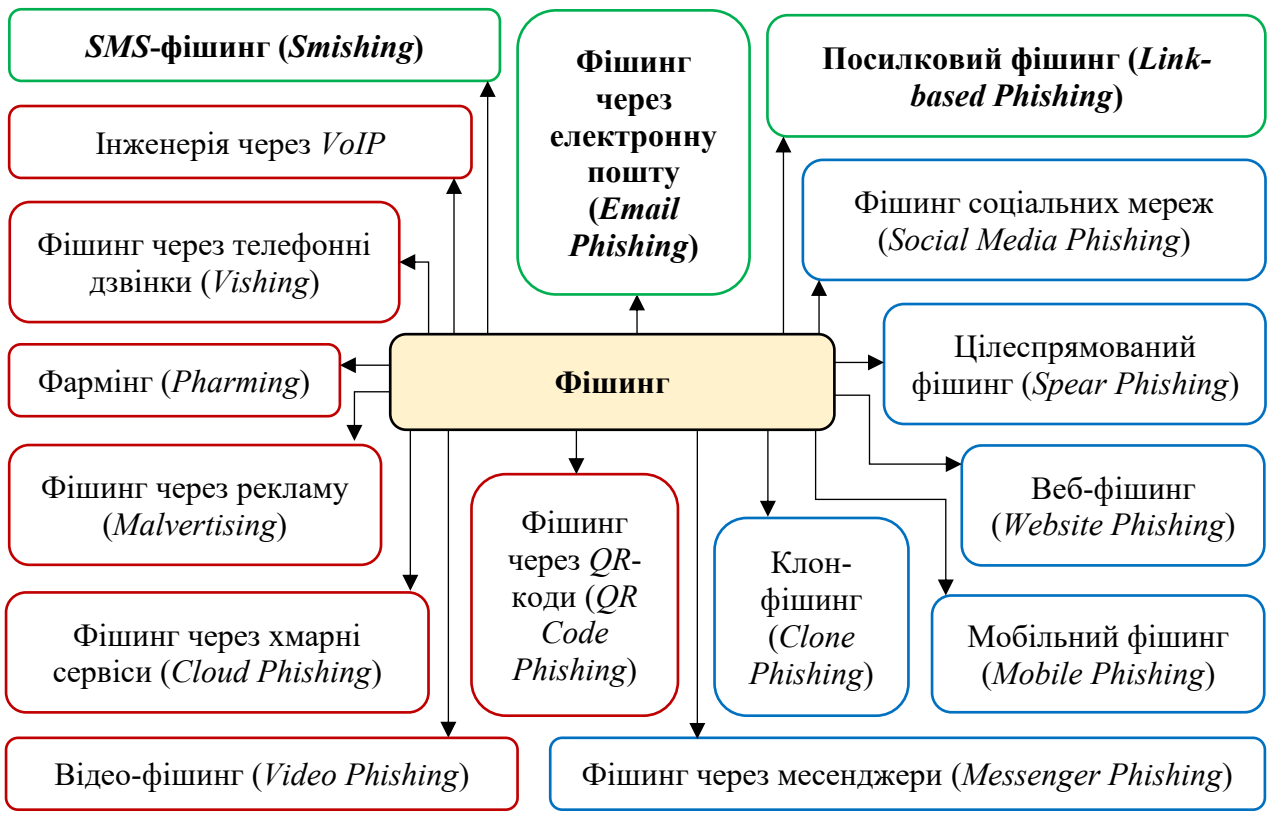

Рисунок 1.1 – Види фішингових атак

Схема ілюструє різновиди фішингових атак (зеленим кольором позначено ті, що безпосередньо використані у даній роботі, синім – частково використані або згадані, а червоним – ті, що не розглядалися у дослідженні). Пояснення згаданих видів фішингу:

– фішинг через електронну пошту (найпоширеніший вид фішингу, коли користувачам надсилають електронні листи зі шкідливими посиланнями або вкладеннями, замаскованими під легітимні повідомлення від банків, державних установ чи сервісів);

– смішинг (атака через SMS, що містять шкідливі посилання або інструкції для викрадення конфіденційних даних користувача);

– посилковий фішинг (використання підроблених або маскованих URL-адрес для перенаправлення користувача на фішинговий сайт з метою викрадення його даних);

– фішинг соціальних мереж (створення підробних акаунтів або повідомлень у соціальних мережах для збору особистої інформації користувачів або поширення шкідливих посилань);

– клон-фішинг (копіювання легітимного електронного листа, який користувач раніше отримував, із заміною посилання або вкладення на шкідливе);

– цілеспрямований фішинг (атака, спрямована на конкретну особу або організацію, що містить персоналізовану інформацію для підвищення довіри жертви);

– мобільний фішинг (фішингові атаки, орієнтовані на мобільні пристрої, зокрема через SMS, push-сповіщення та мобільні додатки);

– веб-фішинг (створення фальшивих вебсайтів, візуально ідентичних легітимним, з метою збору облікових даних користувачів);

– фішинг через месенджери (розсилка шкідливих посилань або повідомлень через месенджери (Viber, WhatsApp, Telegram) для викрадення даних користувачів);

– інженерія через VoIP (атаки через VoIP-телефонію для обману користувачів із метою отримання їх конфіденційної інформації);

– фішинг через телефонні дзвінки (соціальна інженерія через дзвінки, під час яких зловмисник видає себе за співробітника банку чи служби безпеки);

– фішинг через QR-коди (створення шкідливих QR-кодів, що перенаправляють користувачів на фішингові ресурси);

– фішинг через хмарні сервіси (атаки, спрямовані на користувачів хмарних сервісів (Google Drive, OneDrive) з метою викрадення облікових даних);

– фармінг (перенаправлення користувача на фальшивий сайт шляхом підміни DNS-записів без зміни URL у браузері);

– фішинг через рекламу (використання шкідливої реклами для перенаправлення користувачів на фішингові сайти або завантаження шкідливого ПЗ);

– відео-фішинг (фішингові атаки у форматі відео або відеоповідомлень, що містять шкідливі посилання або заклики до переходу за ними).

Основна проблема фішингових атак полягає у тому, що вони базуються не тільки на технічних вразливостях, але й на людському факторі, що значно ускладнює їхнє виявлення звичайними антивірусними засобами.

1.2 Способи виявлення фішингових атак та заходи протидії

Фішингові атаки становлять серйозну загрозу для кібербезпеки користувачів і організацій. У зв’язку з цим, для ефективного захисту користувачів, облікових записів та інформаційних ресурсів, необхідно використовувати комплексні підходи, що включають як способи виявлення фішингових атак, так і заходи протидії їм. Способи виявлення фішингових атак:

– чорні списки (Blacklists) – бази даних із забороненими URL-адресами або IP-адресами, що були помічені як фішингові;

– фільтрація за сигнатурами (Signature-based Detection) – виявлення шкідливих повідомлень або сайтів на основі відомих шаблонів, ключових слів чи ознак;

– евристичний аналіз (Heuristic Analysis) – використання правил для перевірки URL-адрес (наявність IP-адрес замість домену, довжина URL, кількість піддоменів) або аналізу структури повідомлень;

– алгоритми NLP – аналіз тексту повідомлень для виявлення фішингових шаблонів, тональності та стилю повідомлень;

– методи ML – використання методів класифікації (RF, LR, SVM, GB, NB) для аналізу текстових характеристик повідомлень або ознак URL-адрес.

Заходи протидії фішинговим атакам включають наступне (див. рис. 1.2):

– проведення регулярних тренінгів щодо ознак фішингових атак, перевірки URL-адрес, безпечної поведінки в Інтернеті;

– додаткові фактори входу (SMS-код, біометрія) зменшують ризик компрометації облікових записів у разі викрадення паролів;

– використання сучасних антивірусних програм із функціями виявлення фішингових сайтів;

– використання SPF, DKIM та DMARC для захисту доменів від підробки вхідних та вихідних листів;

– регулярне оновлення браузерів, антивірусів та операційних систем для усунення відомих вразливостей атак.

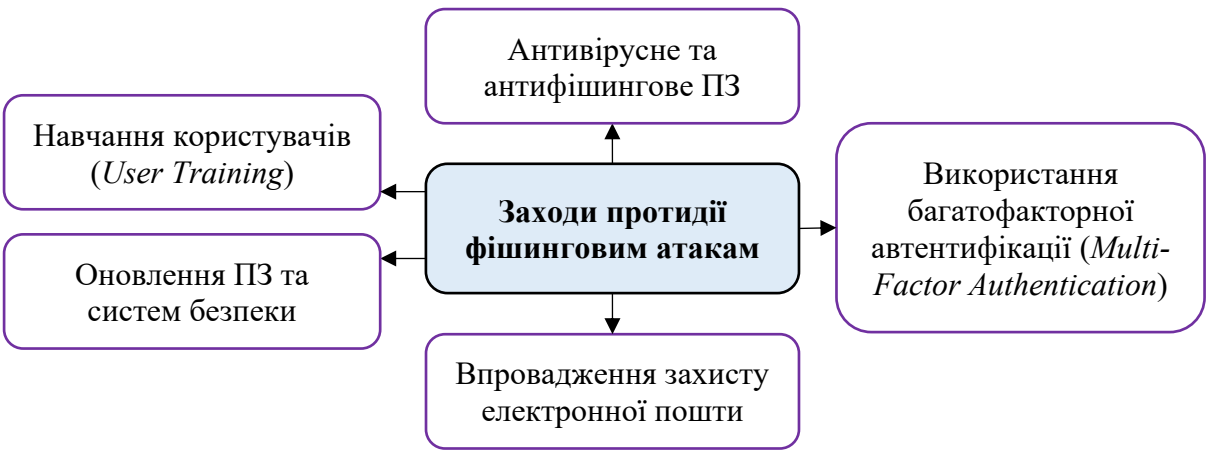

Рисунок 1.2 – Заходи протидії фішингу

Таким чином, для забезпечення максимальної ефективності захисту від фішингових атак доцільно поєднувати різні способи виявлення шкідливих повідомлень та посилань із комплексними заходами протидії, що включають як технічні рішення, так і організаційні та освітні методи. Такий підхід дозволяє своєчасно виявляти загрози та мінімізувати ризики компрометації конфіденційної інформації користувачів та організацій.

1.3 Огляд останніх публікацій та існуючих систем ідентифікації фішингу

Аналіз актуальних публікацій дозволяє визначити ефективні методи, інструменти та технології, що можуть бути використані або вдосконалені під час створення інтелектуальної системи виявлення фішингу. Наприклад, у роботі P. Rehida *et al.* [2] автори дослідили проблематику уникнення виявлення зловмисного ПЗ шляхом застосування антиемуляційних та поліморфних технік. Автори запропонували метод виявлення шкідливих програм на основі аналізу змін їх поведінки під час виконання у модифікованих ізольованих середовищах, зокрема використовуючи емуляцію та пісочниці. Розроблена архітектура системи дозволяє формувати множину модифікованих середовищ для детектування зловмисних проявів, що значно підвищує ефективність виявлення інфікованих програм, навіть якщо вони використовують поліморфні техніки для уникнення стандартних методів захисту.

Дослідники D. Mehed *et al.* [3] проаналізували сучасні загрози інформаційній безпеці та спрогнозували основні напрямки кібератак у майбутньому. Вони відзначили, що незважаючи на розвиток технологій захисту, методи соціальної інженерії, зокрема фішинг, продовжують залишатися одним з найефективніших інструментів кіберзлочинців. У статті наведено детальний огляд найпоширеніших технік фішингових атак, а також сформульовано комплекс необхідних заходів для зниження їх успішності, що включає організаційні, технічні та освітні методи протидії.

Крім того, H. Saleem *et al.* [4] дослідили виявлення веб-фішингових атак шляхом поєднання системи правил з методами ML. Було використано набір даних із 14 ключових ознак фішингової поведінки для навчання моделей SVM та RF, серед

яких найвищу точність показав RF. Отримані правила автори інтегрували у браузерний інструмент Phish Net у вигляді розширення Google Chrome, що здійснює перевірку вебсторінок у реальному часі та сповіщає користувачів про підозрілі сайти. Таким чином, дослідники продемонстрували практичну ефективність ML-підходів у підвищенні захисту від фішингових загроз.

Результати роботи M. Hassnain *et al.* [5] зосередилися на виявленні нових (zero-day) фішингових атак за допомогою алгоритмів AI. Дослідники підкреслили, що традиційні методи, засновані на чорних списках або аналізі вебконтенту, мають низьку ефективність проти нових атак. Вони застосували підхід на основі візуальної схожості, аналізуючи скріншоти фішингових та легітимних сайтів, а також доповнили існуючий датасет PhishTank зображеннями офіційних сайтів відомих брендів. Запропонована модель продемонструвала до 84% точності при виявленні фішингових сторінок, що перевищує результати існуючих методів на основі візуальної схожості.

В свою чергу, дослідження Í. Oliveira *et al.* [6] запропонувало новий підхід до формалізації процесу фішингових атак шляхом розробки PAPO на основі UFO та моделювальної мови OntoUML. Запропонована онтологія дозволяє чітко структурувати знання про фішингові атаки, що підвищує якість і зрозумілість моделей для дослідників і розробників. PAPO сприяє покращенню систем взаємодії, моделювання даних, розробки знання-орієнтованих систем і оцінки ефективності механізмів кіберзахисту відповідно до міжнародних стандартів управління ризиками.

Автор J. P. Rajput [7] запропонував підхід до виявлення фішингових вебсайтів на основі методів ML, аналізуючи ознаки URL та контенту. Було протестовано LR, Multinomial NB та XGBoost, серед яких LR досягла найвищої точності – 96.63%. Для підготовки даних було застосовано токенізацію, стемінг та векторизацію, а розгортання моделі здійснено через FastAPI, що забезпечило зручний інтерфейс користувача та можливість реального виявлення фішингових загроз.

Висновки роботи R. Mishra *et al.* [8] наголосили на ефективності ознак, що використовуються у BDI для виявлення фішингових сайтів. Було проаналізовано 5

ключових ознак, серед яких Common Name Information, Logo Domain та Form Action Domain, на вибірці з ~4.5 тис. легітимних і фішингових сайтів. Результати показали, що жодна з ознак не є надлишковою, а найкращим класифікатором виявився RF з точністю 99.8% та середнім часом відповіді 0.08 секунди. Дослідження підкреслює важливість використання доменних ознак для створення ефективних та масштабованих систем детектування фішингу в реальному часі.

Зі свого боку, C. V. Swetha *et al.* [9] запропонували нову мультимодальну архітектуру для виявлення фішингових електронних листів, що поєднує текстові та метадані ознаки за допомогою трансформерів і механізму крос-уваги. Модель використовує попередньо натренований трансформер для обробки тексту та спеціальні шари для аналізу метаданих (час, тема, відправник), об'єднуючи їх через кросувагу для підвищення точності. Запропонована система показала високу ефективність, досягнувши 99% точності на тестовому наборі English-Arabic Phishing Email Corpus. Таким чином, автори довели, що мультимодальний підхід із ранньою інтеграцією ознак здатен виявляти приховані шаблони фішингу, недоступні для традиційних ML- або rule-based методів, роблячи вагомий внесок у розвиток технологій кіберзахисту.

Отже, аналіз сучасних досліджень у цій галузі підтвердив ефективність застосування різноманітних методів та технологій для виявлення фішингових атак та зловмисного ПЗ (див. табл. 1.1). Дослідження демонструють ефективність застосування RF, LR, трансформерів та візуальної схожості, а також підкреслюють необхідність поєднання різних підходів, таких як аналіз поведінки, контенту, метаданих і доменних ознак, для підвищення точності систем виявлення фішингу. Отримані результати та запропоновані архітектури можуть слугувати основою для розробки інтелектуальних систем захисту, що здатні протидіяти як відомим, так і новим атакам, забезпечуючи комплексний підхід до кібербезпеки.

Таблиця 1.1 – Методи та моделі виявлення фішингових атак і шкідливого ПЗ

Метод / модель	Опис методу / моделі та застосування
Емуляція та пісочниці	[2] Аналіз змін поведінки зловмисного ПЗ у модифікованих ізольованих середовищах для виявлення антиемуляційних та поліморфних технік
Огляд фішингових технік та заходів протидії	[3] Аналіз сучасних загроз та формулювання комплексних заходів протидії фішингу (організаційні, технічні, освітні)
RF (ML) + система правил	[4] Виявлення веб-фішингу за допомогою RF; інтеграція правил у Chrome-розширення Phish Net для реального часу
AI з візуальною схожістю	[5] Використання аналізу скріншотів сайтів (фішингових та легітимних) для виявлення zero-day атак
Онтологічне моделювання (PAPO)	[6] Формалізація процесів фішингових атак на основі UFO та OntoUML для підвищення якості моделей та систем кіберзахисту
LR (ML)	[7] Аналіз ознак URL та контенту; модель LR досягла точності 96.63%, розгортання через FastAPI
RF (BDI)	[8] Виявлення фішингових сайтів за доменними ознаками (Common Name Info, Logo Domain, Form Action Domain) з точністю 99.8%
Multimodal Transformer	[9] Мультимодальна архітектура з крос-увагою для виявлення фішингових повідомлень за текстовими та метаданими ознаками; точність 99%

Таким чином, проведений огляд показав, що найбільш поширеними та ефективними підходами до виявлення фішингових атак є методи ML, зокрема RF, LR, SVM та NB, які використовуються для аналізу текстових і URL-ознакових характеристик. Саме ці методи складають основу розробленої в даній роботі інтелектуальної системи. Водночас, результати розглянутих досліджень демонструють, що поєднання ML-підходів з методами глибокого навчання, онтологічного моделювання та аналізу візуальної схожості також є перспективним напрямком, який може підвищити ефективність виявлення як відомих, так і нових фішингових атак та шкідливого ПЗ у сучасних умовах розвитку кіберзагроз.

1.4 Характеристика фішингових повідомлень і URL-адрес

Фішингові повідомлення та URL-адреси є основними інструментами кіберзлочинців для здійснення атак соціальної інженерії. Їхня характеристика включає як

текстові, так і технічні ознаки, які дозволяють ідентифікувати потенційно небезпечні повідомлення або посилання. Фішингові повідомлення зазвичай мають такі особливості:

– наявність термінових закликів до дії (наприклад, «Ваш акаунт буде заблоковано», «Негайно підтвердіть дані»);

– граматичні та стилістичні помилки, які свідчать про автоматичний або машинний переклад;

– імітація офіційної комунікації (від банків, платіжних сервісів, державних установ) із використанням логотипів, підписів та брендових кольорів;

– використання емоційного тиску (страх втрати доступу до акаунта, штрафів, блокування картки) для спонукання користувача негайно виконати вказівки.

URL-адреси, що використовуються у фішингових атаках, часто мають такі характеристики:

– доменне ім’я, подібне до легітимного, але з незначними відмінностями (наприклад, bankofarnerica.com замість bankofamerica.com);

– використання субдоменів або довгих URL для приховування справжнього доменного імені;

– наявність IP-адрес замість доменного імені, що є нетиповим для офіційних сайтів;

– додавання зайвих слів чи символів для введення користувача в оману (наприклад, secure-login-bank.com).

Розуміння зазначених характеристик є критично важливим для створення ефективних систем виявлення фішингу, адже більшість методів ML використовують ці ознаки як ключові для класифікації повідомлень та URL-адрес на фішингові або безпечні. Зокрема, аналіз текстових ознак повідомлення у поєднанні з технічним аналізом посилань дозволяє суттєво підвищити точність виявлення фішингових атак.

1.5 Постановка задачі дослідження

На підставі мети дослідження необхідно було вирішити низку завдань, спрямованих на розробку інтелектуальної системи для автоматичного виявлення фішингових повідомлень та URL-адрес, що забезпечить підвищення рівня кібербезпеки користувачів та своєчасне інформування про потенційні загрози. До основних завдань дослідження належать:

– аналіз сучасного стану проблеми фішингових атак, їхніх видів та особливостей реалізації з метою визначення ключових характеристик, що можуть бути використані для ідентифікації;

– огляд та порівняння існуючих методів і підходів до виявлення фішингових атак, зокрема методів ML та обробки текстових даних, для вибору оптимальних алгоритмів класифікації;

– вибір та обґрунтування методів ML, що застосовані для класифікації фішингових повідомлень та URL-адрес, серед яких GB, RF, LR, SVM та NB;

– формування наборів ознак для побудови моделей класифікації, зокрема текстових характеристик повідомлень (наявність термінових закликів, ключових слів тощо) та ознак URL-адрес (довжина, наявність IP, підозрілі символи, схожість на легітимні домени);

– розробка архітектури інтелектуальної системи, що включає модулі класифікації повідомлень та URL-адрес, а також користувацький інтерфейс для взаємодії з системою;

– реалізація програмної системи з використанням Python для ML-моделей та JS, HTML і CSS для побудови користувацького інтерфейсу;

– проведення тестування розробленої системи на реальних та симульованих даних для оцінки ефективності та точності роботи обраних алгоритмів класифікації;

– аналіз результатів тестування з метою визначення переваг і недоліків застосованих методів, а також формулювання рекомендацій щодо подальшого удосконалення системи.

При формулюванні зазначених завдань враховувалися сучасні вимоги до систем кібербезпеки, зокрема необхідність забезпечення високої точності класифікації фішингових загроз, швидкості обробки запитів користувачів та зручності інтеграції системи в існуючі захисні платформи.

Таким чином, постановка задачі дослідження передбачає комплексну розробку алгоритмів та програмних засобів, які дозволять здійснювати автоматичне виявлення фішингових повідомлень і посилань на основі аналізу текстових та URL-ознакових характеристик, забезпечуючи ефективний захист користувачів від кіберзагроз у цифрових комунікаційних системах.

Висновки до розділу 1

У даному розділі розглянуто підхід до аналізу проблеми фішингових атак та постановки задачі розробки інтелектуальної системи для їх виявлення. Його сутність полягає у комплексному вивченні видів фішингових атак, характеристик фішингових повідомлень та URL-адрес, а також сучасних методів протидії цим кіберзагрозам. Проведено огляд наукових публікацій та існуючих систем, що продемонструвало ефективність використання методів ML, зокрема RF, LR, SVM, GB та NB, у поєднанні з аналізом текстових та технічних ознак для виявлення фішингових атак.

Особливу увагу приділено визначенню ключових характеристик фішингових повідомлень, серед яких наявність термінових закликів, емоційного тиску, граматичних помилок та підозрілих URL-адрес. Окрім того, проаналізовано сучасні підходи до виявлення фішингових загроз, такі як чорні списки, евристичний аналіз, NLP та ML-методи, що дозволяє сформувати надійну основу для розробки ефективної системи захисту.

Постановка задачі дослідження відображає системний підхід, що охоплює аналіз проблеми, вибір методів ML, формування наборів ознак, розробку архітектури програмної системи, її реалізацію та тестування. Цей підхід дозволяє створити інтелектуальну систему, здатну автоматично виявляти фішингові повідомлення та посилання, забезпечуючи високий рівень точності, швидкість роботи та практичну застосовність у сфері кібербезпеки.

Отже, розділ не лише визначає актуальність проблеми фішингових атак, але й окреслює стратегію розробки інтелектуальної системи, що базується на сучасних методах ML, комплексному аналізі характеристик атак та потреб користувачів, забезпечуючи ефективність, надійність та безпеку цифрових комунікаційних систем.

2 СТРУКТУРА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ІДЕНТИФІКАЦІЇ ФІШИНГУ

2.1 Елементи структури інтелектуальної системи ідентифікації фішингу

Інтелектуальна система ідентифікації фішингових повідомлень і посилань розробляється на основі модульного підходу, що забезпечує її гнучкість, масштабованість і можливість інтеграції з іншими сервісами або аналітичними платформами (див. рис. 2.1). Така структура дозволяє легко оновлювати окремі компоненти – наприклад, змінювати або перенавчати моделі ML, розширювати джерела даних, додавати нові функції користувацької взаємодії чи зворотного зв'язку без порушення роботи всієї системи. Архітектура системи побудована на основі веб-фреймворку Flask (Python), який поєднує серверну логіку з фронтенд-компонентами, реалізованими засобами HTML, CSS та JS. Такий підхід забезпечує зручний інтерфейс користувача, швидкий обмін даними між клієнтською та серверною частинами і можливість гнучкого налаштування обробки запитів.

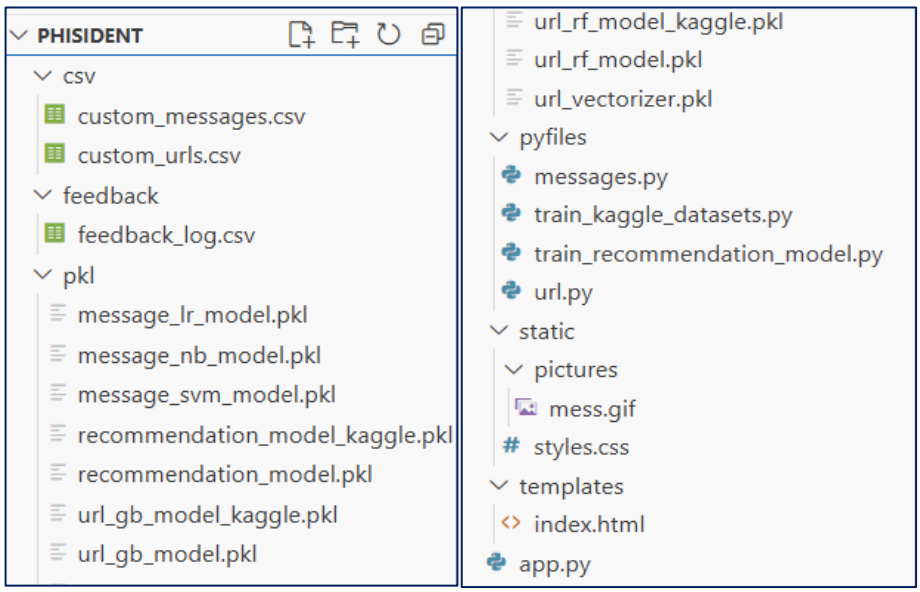

Рисунок 2.1 – Структура системи у середовищі VS Code

Система реалізує два основні режими:

– offline-режим (навчання моделей) – передбачає підготовку, очищення та об'єднання наборів даних, навчання моделей класифікації повідомлень, посилань і рекомендацій, а також збереження натренованих моделей і векторизаторів у відповідних файлах;

– online-режим (інференс) – забезпечує інтерактивну перевірку текстових повідомлень і URL-адрес через вебінтерфейс. У цьому режимі користувач вводить дані для перевірки, сервер обробляє їх за допомогою попередньо навчених моделей і миттєво формує результат класифікації з рекомендаціями.

Загальна структура інтелектуальної системи охоплює повний цикл обробки даних – від введення повідомлення користувачем до виведення результатів аналізу та рекомендацій. У складі системи передбачено певні ключові підсистеми, які розглянуто нижче.

1. Інтерфейс користувача (фронтенд). Інтерфейс створено засобами HTML, CSS і JS (index.html та styles.css). Він забезпечує введення тексту повідомлення та посилання, надсилання запиту до серверу через методи fetch(), відображення результату аналізу (безпечне, фішингове, попередження), формування та надсилання користувацького фідбеку. Також передбачено візуальні індикатори ризику (зміна кольору картки) та блок рекомендацій.

2. Серверна частина (бекенд). Серверна логіка реалізована на основі фреймворку Flask (файл app.py). Вона обробляє HTTP-запити від клієнта, викликає моделі ML, здійснює валідацію введених даних і формує JSON-відповіді для фронтенду. Передбачені окремі маршрути («/check_message», «/check_url», «/feedback»), що відповідають за перевірку повідомлень, посилань і збереження відгуків користувача. Також реалізовано автоматичне завантаження натренованих моделей (.pkl) та відкриття браузера після запуску серверу.

3. Модуль попередньої обробки (препроцесинг). Компоненти попередньої обробки вбудовані безпосередньо у серверну частину (app.py) та у навчальні скрипти (train_recommendation_model.py, train_kaggle_datasets.py, messages.py, url.py). Цей модуль відповідає за перевірку коректності введених даних, виявлення URL у

тексті, фільтрацію спеціальних символів, а також автоматичне визначення мови повідомлення та переклад на англійську для коректної роботи моделей (через бібліотеки langdetect і googletrans).

4. Модулі ML. До складу системи входять окремі модулі навчання та інференсу для повідомлень і URL-адрес: messages.py (тренування моделей класифікації повідомлень за допомогою NB, LR, SVM); url.py (тренування моделей для класифікації посилань (RF, GB); збереження результатів навчання у форматі .pkl (message_lr_model.pkl, url_rf_model.pkl, url_gb_model.pkl, url_vectorizer.pkl). Під час інференсу (app.py) ці моделі завантажуються функцією safe_load() та використовуються для передбачення класу введеного повідомлення або посилання.

5. Модуль рекомендацій і зворотного зв'язку. Модуль рекомендацій поєднує логіку визначення категорії фішингового повідомлення та відображення порад користувачу. Він побудований на основі моделей recommendation_model.pkl та recommendation_model_kaggle.pkl, які формуються у скриптах train_recommendation_model.py та train_kaggle_datasets.py. У випадку відсутності моделі використовується вбудований набір базових рекомендацій (default_recommendations). Зворотний зв'язок користувача реалізовано у фронтенді через текстові поля та обробник /feedback на стороні сервера (app.py).

6. Навчальний модуль. Навчальні процеси реалізовано окремими Python-скриптами: messages.py (базове навчання моделей класифікації повідомлень); url.py (навчання моделей для перевірки посилань); train_recommendation_model.py (побудова моделі рекомендацій на основі поєднаних датасетів); train_kaggle_datasets.py (тренування додаткових моделей і рекомендаційних систем на основі відкритих наборів Kaggle). Результати тренування автоматично зберігаються у директорії pkl/.

7. Сховище даних і моделей. Для збереження моделей, векторизаторів і результатів навчання використовується окрема папка pkl/, що містить файли: message_lr_model.pkl, url_rf_model.pkl, url_gb_model.pkl, url_vectorizer.pkl, recommendation_model.pkl, url_rf_model_kaggle.pkl, url_gb_model_kaggle.pkl,

recommendation_model_kaggle.pkl. Крім того, для кастомних даних користувача застосовуються таблиці custom_messages.csv та custom_urls.csv, які можуть бути доповнені новими прикладами для повторного навчання моделей.

8. Компоненти інтеграції та розгортання системи. Для запуску системи використовується головний файл app.py, який створює Flask-додаток, завантажує моделі, відкриває вебінтерфейс у браузері (webbrowser.open_new()), та забезпечує взаємодію між фронтендом і бекендом. Система може бути розгорнута локально або на хмарних платформах (Heroku, Render, Vercel) із збереженням тієї ж файлової структури.

На рисунку 2.2 подано узагальнену архітектуру інтелектуальної системи ідентифікації фішингових повідомлень і посилань.

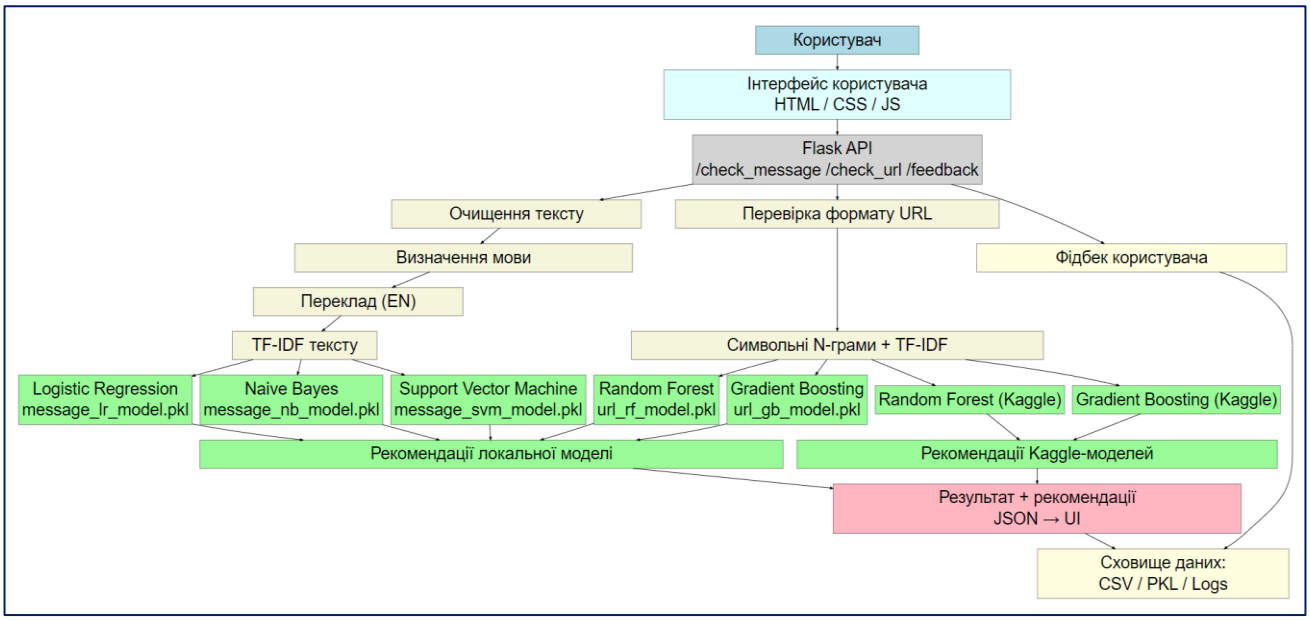

Рисунок 2.2 – Схема структури системи

Таким чином, інтелектуальна система ідентифікації фішингових повідомлень і посилань побудована за модульним принципом, що забезпечує її гнучкість, масштабованість і зручність інтеграції з іншими сервісами. Чітка розмежованість компонентів – від інтерфейсу користувача до модулів ML та сховища даних – дозволяє ефективно виконувати як навчання моделей, так і онлайн-перевірку повідомлень і

URL, а також підтримувати зворотний зв'язок для постійного покращення результатів. Така архітектура забезпечує надійність, прозорість і зрозумілість процесів обробки інформації, створюючи передумови для подальшого розвитку та масштабування системи.

2.2 Огляд методів машинного навчання (LR, NB, SVM, RF, GB)

Логістична регресія (LR) є одним із найпоширеніших та базових методів ML, що використовується для задач бінарної класифікації, зокрема для виявлення фішингових атак (див. рис. 2.3). На відміну від лінійної регресії, яка прогнозує числове значення, LR оцінює ймовірність належності об'єкта до певного класу на основі застосування логістичної (сигмоїдної) функції активації.

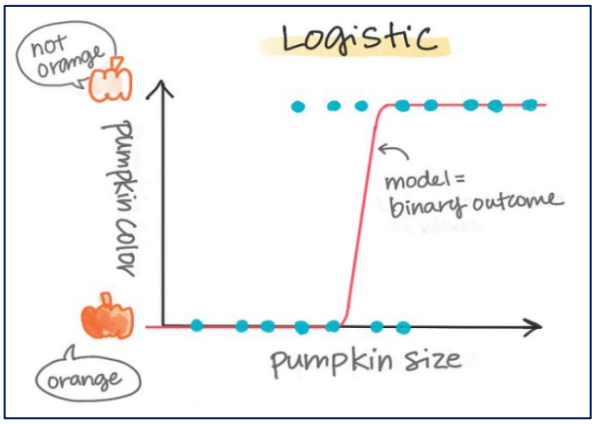

Рисунок 2.3 – Схема LR [10]

Математично модель LR описується наступним чином:

$$P(y = 1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_n x_n)}}, \quad (2.1)$$

де $P(y = 1|x)$ – ймовірність того, що спостереження належить до класу 1 (наприклад, «фішинг»);

β_0 – зміщення;

β_i – вагові коефіцієнти ознак x_i;

e – основа натурального логарифму.

Основні переваги та недоліки методу LR наведено в таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки LR

Переваги	**Недоліки**
Інтерпретованість моделі (коефіцієнти моделі дозволяють оцінити вплив кожної ознаки на результат)	Лінійність (LR передбачає лінійну залежність між ознаками та log-odds, що обмежує її застосування у задачах з нелінійними шаблонами без додаткових перетворень ознак або побудови поліноміальних моделей)
Низькі обчислювальні витрати (метод є швидким та ефективним для великих наборів даних)	
Можливість прогнозування ймовірності (на виході модель видає значення від 0 до 1, що зручно для побудови порогових рішень)	Чутливість до мультиколінеарності (висока кореляція між ознаками може впливати на стабільність моделі та інтерпретацію коефіцієнтів)

У задачах виявлення фішингових повідомлень та URL-адрес LR застосовується як базова модель класифікації, що аналізує текстові характеристики повідомлень або ознаки URL-адрес, такі як довжина, наявність IP-адрес, ключові слова та підозрілі символи. LR ефективно виконує класифікацію у випадках, коли ознаки є незалежними та мають лінійний вплив на результат. Крім того, LR часто використовується як референтна модель при порівнянні з більш складними методами ML (RF, SVM, GB). Таким чином, LR залишається популярним, простим та інтерпретованим методом, який демонструє високу швидкість роботи та достатню точність у задачах кібербезпеки, особливо при використанні як базова модель або частина ансамблевих методів.

З іншого боку, наївний баєсів класифікатор (NB) є популярним методом класифікації у ML, який базується на теоремі Байєса та припущенні незалежності ознак (див. рис. 2.4). Цей метод широко використовується для задач текстової класифікації, спам-фільтрації та виявлення фішингових повідомлень завдяки своїй простоті та високій швидкості роботи.

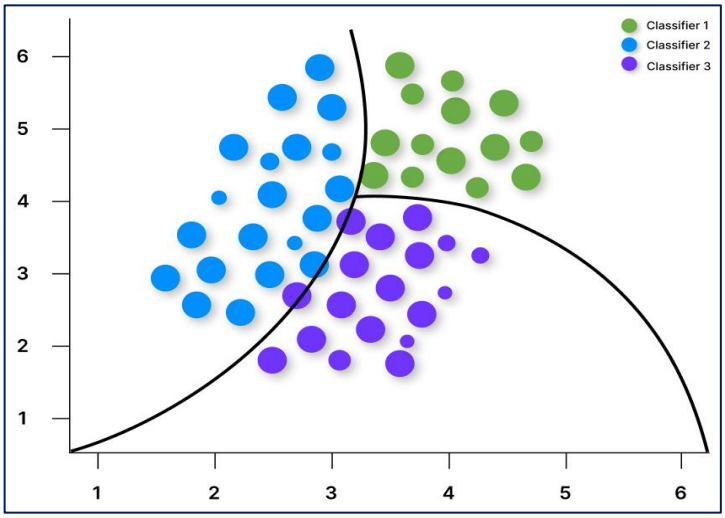

Рисунок 2.4 – Схема NB [11]

Математично модель NB описується наступним чином:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(y)\cdot\prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, \dots, x_n)}, \quad (2.2)$$

де $P(y|x_1, x_2, \dots, x_n)$ – ймовірність того, що спостереження належить до класу y за умови ознак $x_1, x_2, \dots, x_n$;

$P(y)$ – апріорна ймовірність класу y;

$P(x_i|y)$ – ймовірність ознаки x_i при умові класу y;

$P(x_1, x_2, \dots, x_n)$ – спільна ймовірність ознак (нормалізаційний коефіцієнт).

Основні переваги та недоліки методу NB наведено в таблиці 2.2.

Таблиця 2.2 – Переваги та недоліки NB

Переваги	**Недоліки**
Швидкість навчання та прогнозування (метод працює дуже швидко навіть на великих наборах даних завдяки спрощенню розрахунків)	Припущення незалежності ознак (у реальних даних ознаки часто є залежними, що може знижувати точність моделі)
Ефективність при великій кількості ознак (особливо у задачах текстової класифікації, де кількість ознак може сягати десятків тисяч)	Чутливість до нульових ймовірностей (якщо у тренувальній вибірці немає певної комбінації ознак, ймовірність прогнозу буде дорівнювати нулю без застосування методів згладжування, наприклад, Лапласового)
Простота реалізації та інтерпретації (модель легко впроваджується та не потребує значної кількості ресурсів для тренування)	

У задачах виявлення фішингових повідомлень та URL-адрес NB застосовується для класифікації текстів повідомлень або ознак URL-адрес, таких як наявність певних слів, символів або структурних особливостей посилань. Модель показує хороші результати у випадках, коли дані мають велику кількість незалежних текстових ознак, наприклад, при аналізі контенту листів та повідомлень. Тобто, NB залишається одним із найшвидших та найефективніших методів у задачах текстової класифікації та кібербезпеки, демонструючи високу продуктивність та простоту впровадження навіть у ресурсно обмежених системах.

В свою чергу, метод опорних векторів (SVM) є потужним методом класифікації, який широко використовується у ML, зокрема у задачах виявлення фішингових атак, спаму та шкідливих URL-адрес (див. рис. 2.5). Основна ідея SVM полягає у знаходженні оптимальної гіперплощини, яка максимально розділяє дані двох класів з найбільшим можливим зазором між ними.

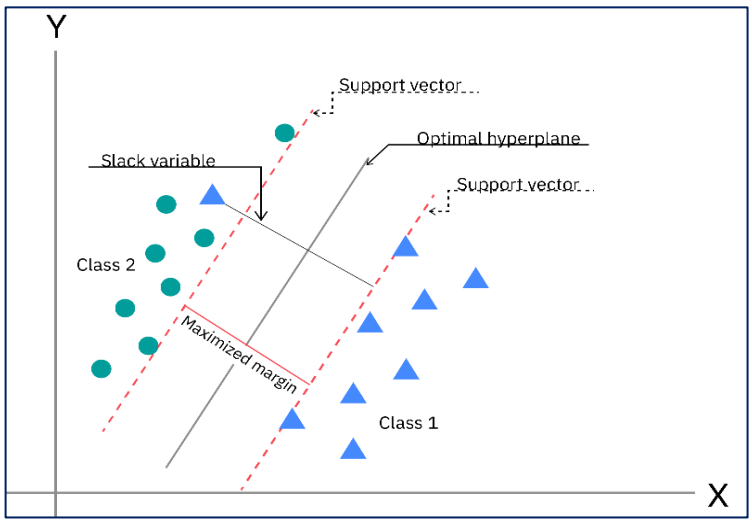

Рисунок 2.5 – Схема SVM [12]

Математично модель SVM описується наступним чином:

$$f(x) = w^T x + b, \tag{2.3}$$

де w – вектор вагових коефіцієнтів;

x – вектор ознак; b – зміщення.

Розв'язок задачі SVM полягає у мінімізації:

$$\min_{w,b} \frac{1}{2}\|w\|^2, \tag{2.4}$$

При умові:

$$y_i(w^T x_i + b) \geq 1, \ \forall_i, \tag{2.5}$$

де y_i – мітка класу спостереження x_i.

Основні переваги та недоліки методу SVM наведено в таблиці 2.3.

Таблиця 2.3 – Переваги та недоліки SVM

Переваги	Недоліки
Висока ефективність у задачах з високою розмірністю ознак (наприклад, у текстовій класифікації або при обробці URL-ознак)	Чутливість до вибору гіперпараметрів (наприклад, C та γ у RBF-ядрі), що потребує ретельної оптимізації
Стійкість до перенавчання, особливо при використанні ядрових функцій для задач з нелінійними залежностями	Складність інтерпретації моделі у випадках з нелінійними ядрами
Можливість побудови як лінійних, так і нелінійних моделей завдяки різним типам ядер (RBF, лінійне, поліноміальне)	Високі обчислювальні витрати при великій кількості спостережень, оскільки навчання масштабується приблизно квадратично за кількістю зразків

У задачах виявлення фішингових повідомлень та URL-адрес SVM застосовується для класифікації текстових ознак повідомлень або структурних характеристик URL, зокрема при використанні TF-IDF-представлення тексту або векторних ознак URL-адрес. Завдяки здатності працювати з великою кількістю ознак та побудові чітких гіперплощин SVM демонструє високу точність у задачах кібербезпеки. Отже, SVM залишається популярним та потужним методом, що забезпечує високу точність класифікації у задачах кібербезпеки, текстового аналізу та виявлення аномалій, особливо при правильному налаштуванні параметрів та виборі ядра.

Крім того, метод випадкових лісів (RF) – це ансамблевий метод ML, який ґрунтується на побудові великої кількості дерев рішень та агрегуванні їх результатів для підвищення точності класифікації або регресії (див. рис. 2.6). Кожне дерево створюється на випадковій підмножині навчальних даних із випадковим підбором ознак, що забезпечує різноманітність і зменшує ризик перенавчання.

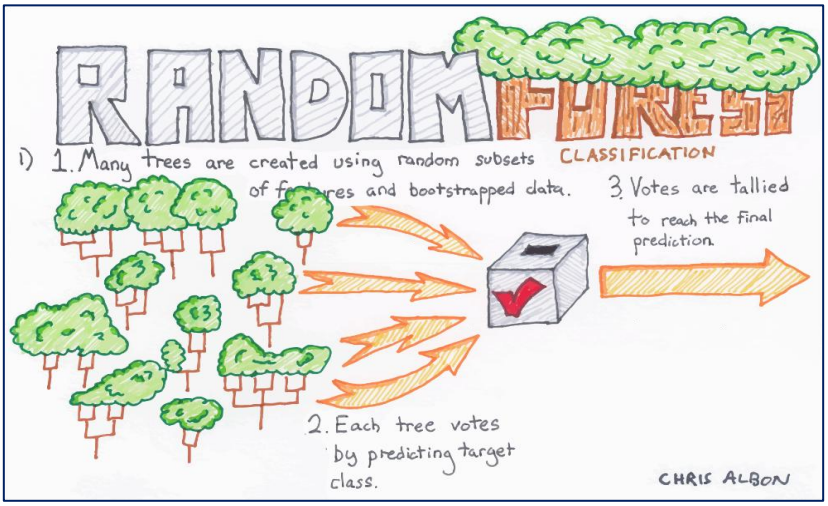

Рисунок 2.6 – Схема RF [13]

Основна ідея RF полягає у поєднанні великої кількості «слабких» моделей (окремих дерев), які разом утворюють «сильну» модель завдяки голосуванню (у класифікації) або усередненню (у регресії). Рішення приймається за більшістю голосів дерев у лісі. Алгоритм побудови RF включає наступні кроки: із навчальної вибірки створюється декілька підвибірок шляхом бутстрепінгу (випадкової вибірки з поверненням); для кожної підвибірки будується дерево рішень, використовуючи випадкову підмножину ознак при кожному розгалуженні; під час прогнозування кожне дерево голосує за клас, а фінальне рішення приймається за більшістю голосів (класифікація) або середнім значенням (регресія).

Основні переваги та недоліки методу RF наведено в таблиці 2.4.

Таблиця 2.4 – Переваги та недоліки RF

Переваги	**Недоліки**
Висока точність моделі завдяки ансамблю незалежних дерев	Більші обчислювальні витрати у порівнянні з простими моделями (наприклад, LR або NB), особливо при великій кількості дерев
Стійкість до перенавчання завдяки використанню бутстрепінгу та випадкових ознак	Менш інтерпретована модель у порівнянні з окремими деревами або лінійними методами
Здатність працювати з великими об’ємами даних і великою кількістю ознак	Схильність до уповільнення при прогнозуванні в реальному часі на слабких пристроях
Можливість визначення важливості ознак	

У задачах виявлення фішингових повідомлень та URL-адрес RF демонструє високу ефективність при класифікації, оскільки здатний обробляти нелінійні залежності між текстовими та структурними ознаками, а також стійкий до шумів у даних. Завдяки підтримці аналізу важливості ознак, RF дозволяє визначати, які саме характеристики повідомлень або URL найбільше впливають на класифікацію, що є цінним для побудови інтерпретованих систем безпеки. Таким чином, RF є надійним та гнучким методом, що демонструє високу точність, стабільність та адаптивність у задачах виявлення фішингових повідомлень і посилань, особливо у випадках, коли дані містять складні залежності та неоднорідні характеристики.

Додатково, градієнтне підсилювання (GB) – це потужний ансамблевий метод ML, що поєднує велику кількість слабких моделей (як правило, дерев рішень) для побудови сильної моделі з високою точністю (див. рис. 2.7). Основна ідея методу полягає у поступовому покращенні якості моделі шляхом послідовного додавання нових дерев, кожне з яких навчається на помилках попередніх.

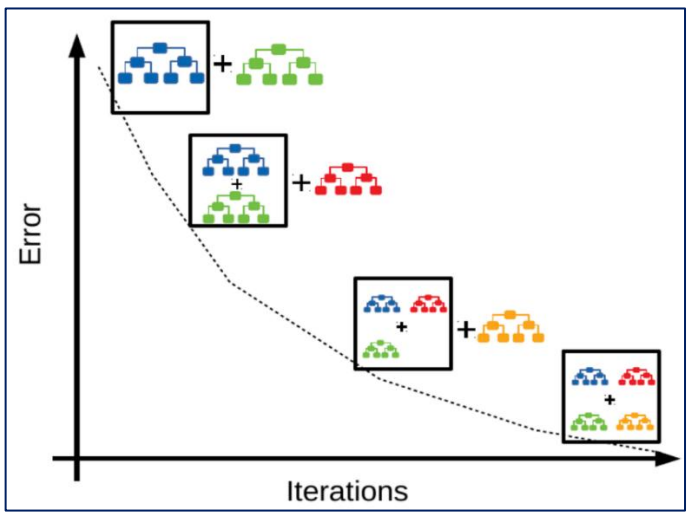

Рисунок 2.7 – Схема GB [14]

На відміну від RF, де дерева створюються незалежно, у GB кожне нове дерево навчається на залишках помилок попередньої моделі. Таким чином, кожен наступний крок градієнтного бустингу намагається зменшити функцію втрат, рухаючись

у напрямку її найшвидшого зменшення (градієнта), звідки й походить назва методу. Математично на кожному кроці модель оновлюється так:

$$F_m(x) = F_{m-1}(x) + \gamma_m \cdot h_m(x), \tag{2.6}$$

де $F_m(x)$ – оновлена модель на ітерації m;

$h_m(x)$ – нове дерево-регресор, побудоване на градієнтах помилок;

γ_m – коефіцієнт навчання, який визначає, наскільки сильно нова модель впливає на загальну.

Основні переваги та недоліки методу GB наведено в таблиці 2.5.

Таблиця 2.5 – Переваги та недоліки GB

Переваги	**Недоліки**
Висока точність класифікації та регресії завдяки покроковому навчанні на помилках	Висока обчислювальна складність (навчання є повільнішим порівняно з RF)
Гнучкість у налаштуванні параметрів (кількість дерев, глибина, швидкість навчання тощо)	Ризик перенавчання при великій кількості дерев або надмірно малому коефіцієнту навчання без належної регуляризації
Здатність моделювати складні нелінійні залежності між ознаками	Складність у налаштуванні (вимагає підбору гіперпараметрів для досягнення оптимальної якості)
Можливість аналізу важливості ознак	

У задачах виявлення фішингових повідомлень та URL-адрес GB показує відмінні результати, особливо при наявності великої кількості неоднорідних ознак, таких як текстові, структурні, статистичні або поведінкові характеристики. Завдяки високій точності та здатності моделювати складні шаблони, GB часто використовується для побудови промислових систем виявлення загроз, у тому числі у поєднанні з TF-IDF, Word Embedding та іншими способами перетворення текстових даних. Тобто, GB є одним із найефективніших методів ML, що забезпечує високу точність і гнучкість, і часто використовується як еталон у задачах класифікації, зокрема у виявленні фішингових повідомлень та URL-адрес.

2.3 Використані інструменти та мови програмування (Python, JS, HTML, CSS, VS Code)

Python – це інтерпретована, високорівнева мова програмування загального призначення з відкритим вихідним кодом, яка стала однією з найпопулярніших мов у галузі ML, обробки даних та кібербезпеки (див. рис. 2.8). Її популярність обумовлена простим синтаксисом, великою кількістю бібліотек та активною спільнотою розробників.

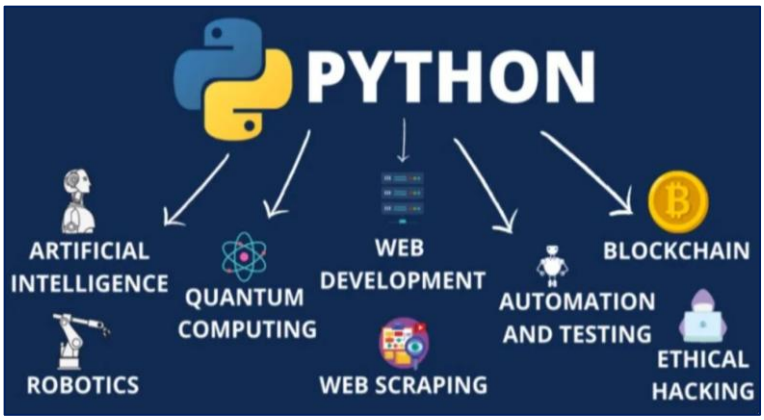

Рисунок 2.8 – Популярні напрями використання Python [15]

Однією з ключових переваг Python є його читабельність і легкість у засвоєнні, що дозволяє швидко створювати прототипи моделей, обробляти великі обсяги даних і реалізовувати методи ML без надмірного програмного перевантаження. Python широко використовується у сфері виявлення фішингу завдяки наявності потужних бібліотек і фреймворків, серед яких:

– scikit-learn – для побудови моделей ML (LR, NB, SVM, RF, GB);

– pandas та numpy – для обробки табличних даних та виконання математичних операцій;

– matplotlib та seaborn – для візуалізації результатів дослідження;

– nltk, re, sklearn.feature_extraction.text – для обробки тексту, токенізації та векторизації даних;

– xgboost, lightgbm – для реалізації високоефективних градієнтних бустинг-моделей;

– joblib, pickle – для збереження та завантаження навчених моделей;

– Flask, FastAPI – для створення API і інтеграції моделей у вебсервіси.

Python також підтримує обробку текстових даних, аналіз URL-адрес, виділення ознак, що є критично важливим у задачах виявлення фішингових повідомлень. Наприклад, з його допомогою легко реалізувати перевірку наявності IP-адреси в URL, виявлення підозрілих символів, токенізацію тексту повідомлень та обчислення TF-IDF. Крім того, Python є кросплатформною мовою, що дозволяє запускати проєкти як на Windows, так і на Linux або в хмарних середовищах. Його екосистема підтримує автоматизацію, інтеграцію з базами даних, DevOps-підходи та розгортання моделей у виробниче середовище.

Завдяки своїй гнучкості та багатофункціональності Python дозволяє реалізовувати повний цикл розробки системи виявлення фішингу – від збору та попередньої обробки даних до побудови моделей, оцінювання їх точності та інтеграції в програмні або вебрішення. Його синтаксис орієнтований на читабельність, що сприяє розробці зрозумілих і підтримуваних рішень у сфері кібербезпеки, де важлива прозорість алгоритмів та ефективна взаємодія в команді. У контексті виявлення фішингових атак Python дозволяє:

– інтегрувати ML з реальними джерелами даних (електронна пошта, вебтрафік, форми зворотного зв'язку);

– автоматизувати збір фішингових повідомлень з відкритих джерел або через API (наприклад, VirusTotal, PhishTank);

– створювати системи моніторингу та сповіщення, які у реальному часі аналізують потенційно шкідливий контент;

– візуалізувати поведінкові та статистичні шаблони фішингових повідомлень, що допомагає у виявленні нових типів атак.

Тобто, Python виступає ключовим інструментом для створення інтелектуальних рішень у сфері протидії фішинговим атакам завдяки своїй універсальності, активній спільноті користувачів та широкому набору спеціалізованих бібліотек. Його адаптивність, простота у використанні та сумісність із сучасними технологіями роблять цю мову підходящою як для швидкого прототипування систем виявлення фішингу, так і для реалізації масштабованих та ефективних рішень у промислових середовищах. Python однаково добре підходить як для дослідницьких експериментів з методами ML, так і для побудови готових продуктів, інтегрованих у реальні системи захисту.

Зі свого боку, JS – це мова програмування, яка виконується на стороні клієнта в веббраузері та забезпечує інтерактивність вебсторінок (див. рис. 2.9). У поєднанні з HTML і CSS, JS дозволяє створювати динамічні інтерфейси користувача, реалізовувати обробку подій, валідацію введених даних, асинхронну взаємодію з сервером (через AJAX або fetch API), а також оновлення вмісту сторінки без її перезавантаження.

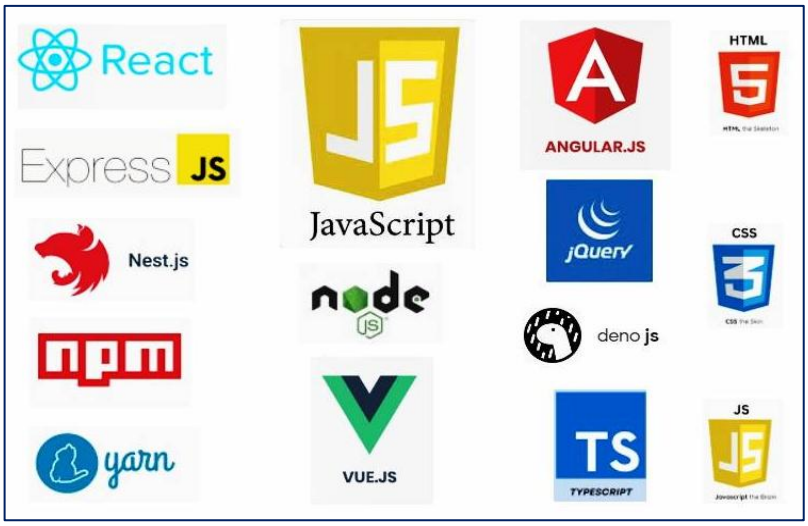

Рисунок 2.9 – Компоненти екосистеми JS [16]

JS є однією з ключових технологій фронтенду, тобто частини вебзастосунку, яка безпосередньо взаємодіє з користувачем. Вона дозволяє реалізовувати складну

логіку на стороні клієнта, включаючи обробку подій, анімацію елементів, формування запитів до серверу в реальному часі та інтеграцію з API. Завдяки своїй гнучкості та широкому розповсюдженню, JS є основою сучасної веброзробки та постійно розвивається завдяки появі нових бібліотек і фреймворків, таких як React, Vue, Angular, Node.js тощо.

Основні переваги та недоліки мови JS наведено в таблиці 2.6.

Таблиця 2.6 – Переваги та недоліки JS

Переваги	**Недоліки**
Швидке виконання (JS інтерпретується напряму у браузері, що забезпечує високу швидкість реакції)	Залежність від браузера (різні браузери можуть по-різному інтерпретувати код)
Інтерактивність (дозволяє створювати зручні та динамічні інтерфейси користувача)	Безпекові ризики (через відкритість коду в браузері можуть виникати XSS-атаки та інші вразливості)
Підтримка усіма сучасними браузерами (не потребує встановлення додаткових плагінів)	
Широка екосистема (велика кількість бібліотек, фреймворків, модулів, що спрощують розробку)	Складність підтримки великого коду (особливо без застосування модульної архітектури або фреймворків)
Універсальність (використовується як на стороні клієнта, так і на стороні сервера)	

Використання JS у проєктах виявлення фішингу полягає у створенні інтуїтивного та адаптивного інтерфейсу користувача для взаємодії з моделлю ML. Наприклад, JS може бути використаний для:

– реалізації форми введення тексту повідомлення або URL;

– динамічного відображення результатів класифікації («фішинг» / «безпечно»);

– асинхронного надсилання даних до серверної частини (через fetch/AJAX) для обробки запиту;

– візуалізації результатів (наприклад, через графіки або кольорові маркери).

У контексті фішингу JS також може слугувати інструментом аналізу поведінки користувача у браузері (наприклад, для виявлення фішингових спроб введення

особистих даних на підозрілих сторінках), або бути частиною навчального інтерфейсу для демонстрації типових прикладів фішингових повідомлень. Таким чином, JS є невід'ємною частиною створення зручного та функціонального інтерфейсу користувача, що дозволяє ефективно взаємодіяти з інтелектуальними системами виявлення фішингу у реальному часі.

Однією з базових технологій, яка використовується разом із JS, є HTML – стандартна мова розмітки, що визначає структуру та вміст вебсторінок (див. рис. 2.10). HTML не є мовою програмування, проте відіграє критично важливу роль у веброзробці, забезпечуючи каркас для всіх елементів інтерфейсу користувача: заголовків, текстів, форм, кнопок, зображень тощо.

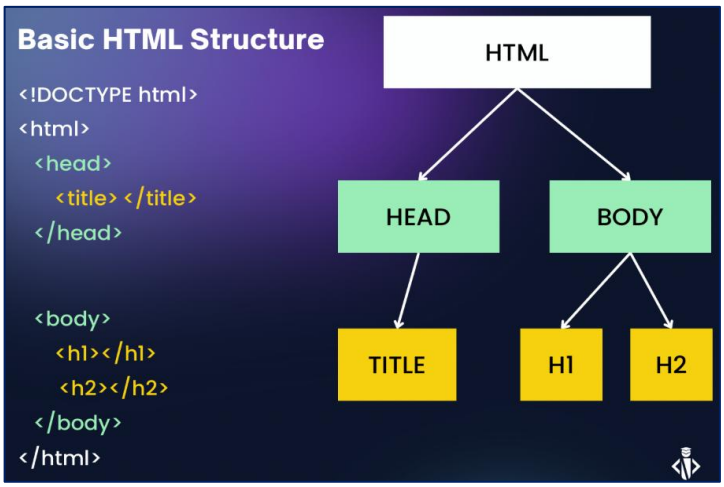

Рисунок 2.10 – Базова структура HTML [17]

HTML працює у тісній зв’язці з CSS (який відповідає за візуальне оформлення) та JS (який додає динаміку). Саме HTML дозволяє визначати поля для введення тексту, форми для взаємодії з користувачем, таблиці результатів, контейнерні блоки тощо. Для побудови систем виявлення фішингових атак HTML слугує основою інтерфейсу, який з’єднує користувача із застосунком. Наприклад, у такому інтерфейсі можна: створити форму для введення повідомлення або URL-адреси; відобразити текст попередження або результат класифікації; вбудувати гра-

фічні елементи чи анімацію для кращого сприйняття результатів. HTML також підтримує семантичну структуру, що важливо для доступності та адаптивності, а в поєднанні з сучасними фреймворками (React, Vue) дозволяє створювати багаторазово використовувані компоненти.

Третім ключовим компонентом створення вебінтерфейсів є CSS – мова стилів, яка використовується для візуального оформлення HTML-елементів (див. рис. 2.11). CSS дозволяє задавати кольори, шрифти, відступи, розташування елементів на сторінці, а також створювати адаптивний дизайн, який підлаштовується під різні розміри екранів і пристроїв.

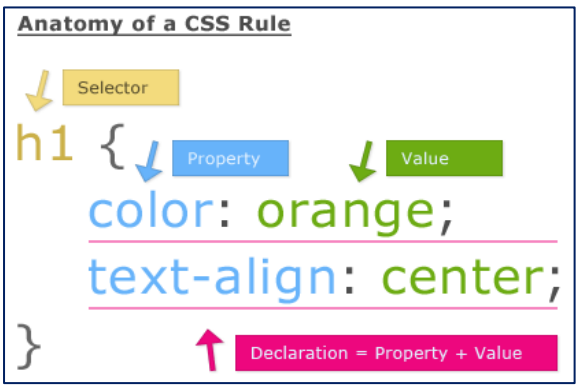

Рисунок 2.11 – Структура CSS [18]

CSS розділяє структуру (HTML) і стиль, що покращує читабельність та підтримку коду. Сучасний CSS підтримує не лише базове оформлення, але й анімації, перехідні ефекти, гнучкі сітки (Flexbox) та адаптивні компонування (Grid Layout), що дозволяє створювати інтуїтивно зрозумілі й привабливі інтерфейси. У контексті систем виявлення фішингу CSS використовується для:

– візуального розділення безпечних і фішингових повідомлень (наприклад, через кольорове кодування);

– оформлення елементів інтерфейсу (форми введення, таблиці результатів, кнопки);

– покращення UX шляхом створення чистого, зрозумілого та адаптивного дизайну;

– реалізації підказок, анімаційних сповіщень або попереджень для користувача.

Також, завдяки фреймворкам типу Bootstrap або бібліотекам TailwindCSS, процес стилізації значно спрощується і пришвидшується, що дозволяє розробникам швидко створювати адаптивні й сучасні інтерфейси користувача (див. табл. 2.7).

Таблиця 2.7 – Порівняльна характеристика HTML, CSS та JS

Компонент	**HTML**	**CSS**	**JS**
Призначення	Структура вебсторінки	Оформлення вебсторінки	Логіка та динаміка
Тип мови	Мова розмітки	Мова стилів	Мова програмування
Основні функції	Створення каркасу вебінтерфейсу: заголовки, форми, поля введення, таблиці	Налаштування зовнішнього вигляду елементів: кольори, розміри, адаптивність, анімації	Реалізація інтерактивності: обробка подій, асинхронні запити, динамічне оновлення даних
Використання в системах виявлення фішингу	Побудова інтерфейсу: форма введення повідомлення/URL, відображення результатів	Візуалізація класифікації («фішинг» / «безпечно»), покращення UX	Надсилання даних до моделі ML, відображення результатів, інтеграція з API

Тобто, HTML, CSS та JS спільно формують основу користувацького інтерфейсу систем виявлення фішингових повідомлень і посилань, забезпечуючи зрозумілу взаємодію з користувачем, ефективну подачу результатів та інтеграцію з інтелектуальними алгоритмами аналізу. Для розробки та налагодження коду, а також для інтеграції клієнтської та серверної частин системи, слід використовувати середовище розробки, таке як VS Code – безкоштовне, кросплатформне IDE, створене компанією Microsoft (див. рис. 2.12). Воно призначене для редагування коду, налагодження, запуску застосунків і роботи з системами контролю версій. VS Code підтримує велику кількість мов програмування, включаючи JS, Python тощо.

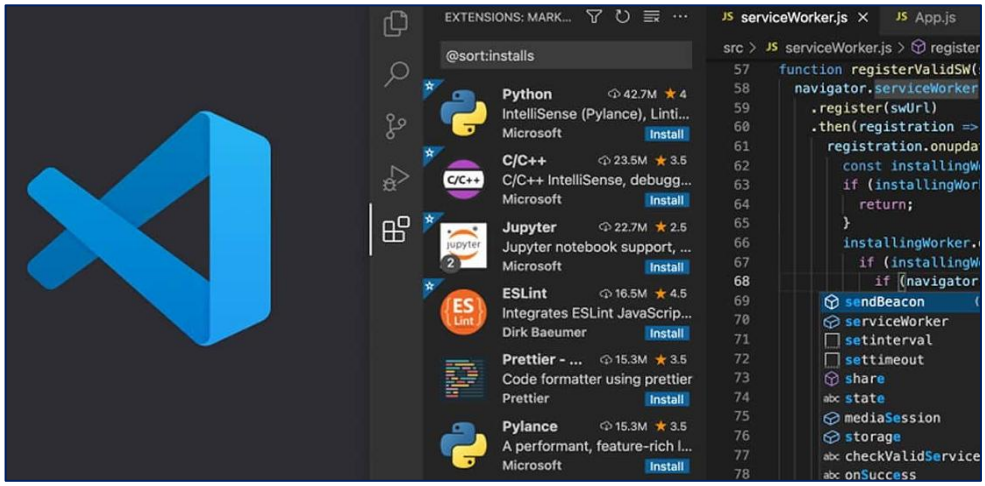

Рисунок 2.12 – VS Code [19]

Основні можливості VS Code:

– підсвічування синтаксису та автодоповнення коду (IntelliSense);

– вбудований термінал для зручної роботи з командним рядком;

– підтримка розширень (дозволяє встановлювати тисячі плагінів для підтримки мов, інструментів і фреймворків);

– інтеграція з Git (дає змогу працювати з системами контролю версій безпосередньо з редактора);

– налагодження коду через breakpoints, call stack, watch тощо;

– легка вага та швидкість запуску, що вигідно відрізняє його від важких IDE.

Переваги використання у розробці вебінтерфейсів:

– плагіни типу Live Server дозволяють миттєво переглядати зміни в браузері без ручного оновлення сторінки;

– підтримка синтаксису HTML, CSS, JS, React, Tailwind тощо;

– інтеграція з платформами для розгортання та тестування, зокрема GitHub, Docker, Firebase;

– можливість одночасно редагувати клієнтську і серверну частину застосунку;

– інтелектуальні підказки та перевірка помилок ще під час написання коду.

У межах реалізації системи виявлення фішингових повідомлень та посилань VS Code використовується як основне середовище для:

- написання коду інтерфейсу користувача (HTML, CSS, JS);
- розробки логіки інтеграції з моделлю ML на Python (через API);
- тестування функціональності застосунку в реальному часі;
- організації структури проєкту у зручному дереві файлів;
- використання системи контролю версій Git для фіксації змін і спільної роботи.

Отже, VS Code є потужним і гнучким інструментом, який значно підвищує продуктивність розробника, спрощує налагодження та забезпечує комфортні умови для повного циклу створення вебзастосунку – від верстки до логіки взаємодії з інтелектуальними методами.

Висновки до розділу 2

У цьому розділі розглянуто структуру інтелектуальної системи ідентифікації фішингових повідомлень і посилань, її основні компоненти та методи ML, що застосовуються для класифікації повідомлень та URL. Встановлено, що модульна архітектура системи забезпечує гнучкість, масштабованість та можливість інтеграції з різними джерелами даних. Вона включає вісім підсистем: інтерфейс користувача, серверну логіку, модулі попередньої обробки, класифікаційні моделі, навчальні скрипти, сховище даних, модуль рекомендацій та зворотного зв’язку, а також інтеграцію та запуск системи.

Offline- та online-режими роботи дозволяють як навчати моделі на відкритих і кастомних наборах даних, так і виконувати інтерактивну перевірку повідомлень та URL у режимі реального часу. Це забезпечує повний цикл обробки даних – від введення інформації до формування рекомендацій для користувача. Крім того, модулі ML включають основні моделі: LR для класифікації повідомлень, RF для кла-

сифікації URL та модель рекомендацій для визначення категорії фішингового повідомлення. Додатково розглядалися NB, SVM та GB на етапі тестування, що дозволяє оцінити точність та ефективність різних методів.

В свою чергу, використані інструменти та мови програмування (Python, JS, HTML, CSS, VS Code) забезпечують повний цикл розробки системи: від збору та обробки даних до інтеграції моделей ML у вебзастосунок. Python відіграє ключову роль у реалізації методів ML, тоді як JS та HTML/CSS забезпечують динамічний і зручний інтерфейс користувача, а VS Code полегшує розробку, тестування та інтеграцію всіх компонентів системи. Загалом, переваги системи включають швидку обробку даних, інтерпретованість результатів, гнучкість у налаштуванні моделей, можливість інтеграції з різними джерелами даних і перспективу використання зворотного зв’язку для донавчання моделей.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

3.1 Python-файли та модулі бекенду

Дана інтелектуальна система реалізовувалась на мові Python із використанням фреймворку Flask, що забезпечує обробку HTTP-запитів, інтеграцію з моделями ML та взаємодію з фронтендом. Основна логіка системи розділена на окремі модулі, які відповідають за різні етапи роботи: від препроцесингу даних та класифікації повідомлень і URL до формування рекомендацій та обробки користувацького фідбеку. Кожен модуль системи виконує чітко визначену функцію, що дозволяє зручно підтримувати код, оновлювати або додавати нові моделі і методи без порушення роботи інших компонентів.

Наприклад, основний файл app.py виконує роль центру керування системою (див. рис. 3.1). Він відповідає за:

– створення Flask-додатку та маршрутизацію HTTP-запитів («/check_message», «/check_url», «/feedback»);

– завантаження натренованих моделей ML для класифікації повідомлень (LR) та URL (RF), а також моделей рекомендацій;

– обробку текстових повідомлень (перевірку на наявність тільки URL, визначення мови, переклад на англійську для коректної роботи моделей, передбачення класу повідомлення);

– формування результатів класифікації та рекомендацій для користувача;

– обробку URL-посилань із перевіркою формату, векторизацією та класифікацією;

– прийом та обробку фідбеку користувача, який зберігається в консолі для подальшого аналізу;

– автоматичне відкриття вебінтерфейсу у браузері після запуску серверу.

```python
from flask import Flask, render_template, request, jsonify
import webbrowser
import threading
import joblib
import re
from langdetect import detect
from googletrans import Translator
from datetime import datetime
import os

app = Flask(__name__)

# Завантаження моделей
def safe_load(path, name):
    try:
        mdl = joblib.load(path)
        print(f"✅ Завантажено {name} з '{path}'")
        return mdl
```

Рисунок 3.1 – Фрагмент коду app.py

Допоміжні функції app.py включають перевірку коректності URL, виявлення лише URL у тексті, безпечне завантаження моделей (safe_load()) та встановлення базових рекомендацій для користувача у випадку відсутності категорії повідомлення у моделі. Тобто, файл app.py виступає центральною ланкою між користувачем та інтелектуальною системою. Через нього фронтенд отримує результати класифікації, а сама система – упорядковані запити та зворотний зв’язок.

На рівні взаємодії з користувачем app.py забезпечує доступність сервісу через єдиний вебінтерфейс, приймаючи введені дані та передаючи їх у модулі класифікації й рекомендацій. З боку системи цей файл виконує роль координатора: отримує запити від користувача, перевіряє їх коректність, активує потрібні моделі, обробляє результати та повертає їх у структурованому та інтерпретованому вигляді. Файл app.py також гарантує стабільність роботи системи за рахунок безпечного завантаження моделей, коректного виклику їх функцій і формування відповідей, що адаптовані до реального сценарію використання. Завдяки цьому модулю система працює цілісно: від моменту введення повідомлення чи URL до виведення класифікації, рекомендацій і можливості врахування зворотного зв’язку для майбутнього покращення.

Окрім app.py, система містить інші модулі Python. Наприклад, модуль pyfiles/messages.py відповідає за підготовку даних, навчання та оцінку моделей ML для класифікації текстових повідомлень (SMS, Email) на безпечні та фішингові (див. рис. 3.2). Основні етапи роботи модуля:

– завантаження датасетів (використовуються два джерела даних: відкритий датасет ealvaradob/phishing-dataset та кастомний файл csv/custom_messages.csv, створений автором, а дані об'єднуються в єдину таблицю для подальшої обробки);

– попередня обробка (створюються ознаки для навчання моделей за допомогою TF-IDF векторизації тексту, тоді як мітки класів (label) приводяться до цілочисельного типу для сумісності з методами ML);

– розподіл на тренувальну та тестову вибірки (дані діляться на навчальну та тестову частини у співвідношенні 80:20 з врахуванням балансування класів);

– навчання моделей (тренуються три моделі: NB, LR та SVM);

– оцінка ефективності (для кожної моделі виводиться звіт класифікації (classification_report) з метриками точності (precision), повноти (recall), F1-міри та підтримки класів (support) на тестовій вибірці);

– збереження моделей (навчені моделі зберігаються у форматі .pkl для подальшого використання в інференсі у Flask-додатку (app.py)).

```
2  from datasets import load_dataset
3  import pandas as pd
4  from sklearn.model_selection import train_test_split
5  from sklearn.feature_extraction.text import TfidfVectorizer
6  from sklearn.naive_bayes import MultinomialNB
7  from sklearn.linear_model import LogisticRegression
8  from sklearn.svm import SVC
9  from sklearn.pipeline import Pipeline
10 from sklearn.metrics import classification_report
11 import joblib
12 import os
13
14 print("🔷 Завантаження основного датасету повідомлень (SMS + Email)...")
15 dataset = load_dataset("ealvaradob/phishing-dataset", "texts", trust_remote_code=True)
16 df_main = dataset['train'].to_pandas()
```

Рисунок 3.2 – Фрагмент коду messages.py

Таким чином, messages.py виконує роль джерела «інтелектуальної основи» для класифікації повідомлень: саме тут формуються моделі, які потім застосовує сервер у процесі інференсу. Отримані й збережені у цьому модулі моделі дозволяють фронтенду одразу отримувати рішення – без очікування додаткового навчання, безпосередньо у момент взаємодії користувача з вебінтерфейсом. Тобто messages.py забезпечує систему статистичною точністю, а також забезпечує повний цикл роботи з текстовими повідомленнями: від завантаження даних і їх препроцесингу до навчання, оцінки та збереження моделей для інтеграції з основним сервером. Завдяки цьому користувач бачить просту і швидку відповідь у браузері, тоді як основна машинна обробка та підготовка моделей прихована всередині messages.py, що робить систему ефективною, масштабованою та узгодженою.

Зі свого боку, модуль pyfiles/url.py реалізує підготовку даних, навчання та оцінку моделей ML для класифікації URL-посилань на фішингові та безпечні (див. рис. 3.3). Його робота багато в чому аналогічна до messages.py, однак орієнтована на особливості текстів URL-адрес. Основні етапи роботи модуля наведено нижче.

1. Завантаження датасетів. Використовуються два джерела: відкритий набір ealvaradob/phishing-dataset (підрозділ «urls») із платформи HuggingFace та власний датасет csv/custom_urls.csv, сформований автором. Обидві вибірки поєднуються в єдину таблицю для подальшої обробки.

2. Підготовка даних. Зі стовпця «text» отримуються URL-рядки, а зі label – мітки класів (0 – безпечні, 1 – фішингові). Для векторизації застосовується TF-IDF із символьними n-грамами (3-5 символів), що дає змогу враховувати структуру адрес і виявляти підозрілі шаблони (наприклад, зайві піддомени або змінені літери).

3. Розподіл вибірки. Дані діляться на тренувальну й тестову частини у співвідношенні 80:20.

4. Навчання моделей. Для класифікації використовуються два методи:

– RF – ансамблева модель на основі множини дерев рішень, яка забезпечує стійкість до шуму та перевіряє велику кількість ознак;

– GB – послідовна побудова слабких моделей, що підвищує точність за рахунок комбінування попередніх результатів.

5. Оцінка ефективності. Для кожної моделі виводиться звіт класифікації (classification_report) з основними метриками: precision, recall, F1-міра та support.

6. Збереження моделей. Після навчання створюються та зберігаються три файли: url_rf_model.pkl (модель RF); url_gb_model.pkl (модель GB); url_vectorizer.pkl (TF-IDF-векторизатор, необхідний для подальшого перетворення нових URL під час інференсу).

```
from datasets import load_dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report
import joblib
import os

print("🔷 Завантаження основного датасету URL...")
url_dataset = load_dataset("ealvaradob/phishing-dataset", "urls", trust_remote_code=True)
df_main = url_dataset['train'].to_pandas()
```

Рисунок 3.3 – Фрагмент коду url.py

Тобто, url.py створює функціональний «модуль аналізу посилань», який працює як ядро моделі для інференсу у Flask-додатку. Завдяки сформованим та збереженим моделям app.py отримує можливість оперативно перевіряти будь-які URL, введені користувачем у вебінтерфейсі, і відразу повертати результат класифікації. Для користувача це виглядає як миттєва оцінка безпечності посилання, тоді як за сценою використовується попередньо натренована статистична логіка з url.py. Саме такий поділ дозволяє забезпечити швидку відповідь у фронтенді та масштабованість серверної частини, де аналіз нових URL стає автоматизованим і формалізованим процесом.

Таким чином, url.py формує окрему підсистему для аналізу веб-посилань, яка після навчання інтегрується в основний серверний додаток Flask (app.py). Вона дозволяє автоматично оцінювати безпечність URL-адрес і є невід’ємною частиною

загальної архітектури системи виявлення фішингових загроз. Крім того, модуль pyfiles/train_kaggle_datasets.py відповідає за завантаження відкритих датасетів з Kaggle, підготовку даних та навчання моделей для класифікації URL-посилань і створення системи рекомендацій для фішингових повідомлень (див. рис. 3.4). Основні етапи роботи модуля викладено нижче.

1. *Завантаження датасетів з Kaggle:*

– «Phishing Email Dataset»: файл phishing_email.csv містить електронні листи з ознаками безпечних та фішингових повідомлень;

– «Phishing URL Dataset»: файл dataset.csv містить URL-посилання з мітками безпечних та фішингових сайтів.

Дані завантажуються через API kagglehub та зберігаються у локальних директоріях для подальшої обробки.

2. *Попередня обробка даних:*

– для email-датасету виділяються лише фішингові повідомлення;

– кожне повідомлення класифікується за категоріями ризику: request_click_link, request_personal_data, account_blocked, payment_alert, default;

– у URL-датасеті мітки -1 замінюються на 0 для узгодженості з методами ML.

3. *Розподіл на тренувальні та тестові вибірки.* Дані поділяються у співвідношенні 80:20 з урахуванням балансування класів для коректного навчання моделей.

4. *Навчання моделей рекомендацій та класифікації:*

– Recommendation Model для email: використовується TF-IDF векторизація тексту та LR для передбачення категорії фішингового повідомлення;

– моделі для URL: навчання RF та GB для класифікації URL на безпечні та фішингові.

5. *Оцінка ефективності.* Для кожної моделі виводяться звіти класифікації (classification_report) з метриками точності, повноти, F1-міри та підтримки класів на тестовій вибірці.

6. Збереження моделей. Навчені моделі зберігаються у форматі .pkl для подальшого використання у Flask-додатку (app.py) під час інференсу.

```
import os
import pandas as pd
import re
import joblib
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import kagglehub

print("🔷 Починаємо завантаження і тренування Kaggle датасетів")

# 1) Download Phishing Email Dataset
print("1) Завантаження Kaggle Phishing Email Dataset...")
email_path = kagglehub.dataset_download("naserabdullahalam/phishing-email-dataset")
print("Файли завантажено у:", email_path)
```

Рисунок 3.4 – Фрагмент коду train_kaggle_datasets.py

Отже, модуль train_kaggle_datasets.py забезпечує інтеграцію відкритих датасетів, формування категорій фішингових повідомлень та навчання моделей для класифікації URL і рекомендацій, які використовуються разом з локальними моделями у системі для підвищення точності і функціональності. У контексті повноцінної системи train_kaggle_datasets.py розширює її можливості завдяки використанню великомасштабних відкритих даних, що підвищує точність класифікації та робить рекомендації більш обґрунтованими. Саме тут формуються моделі, здатні розпізнавати типові шаблони фішингових повідомлень і ризикові характеристики URL, що стає основою для інтерпретованих порад, які бачить користувач у вебінтерфейсі.

Важливо, що результати роботи цього модуля інтегруються з локальними моделями, зменшуючи залежність від одного джерела даних та забезпечуючи більшу стійкість системи. Таким чином, train_kaggle_datasets.py не просто готує моделі, а створює інфраструктуру для більш точного й інформативного аналізу,

який безпосередньо впливає на якість відображення результатів і рекомендацій у фронтенді.

В свою чергу, модуль train_recommendation_model.py реалізує формування та навчання рекомендаційної моделі, що автоматично визначає тип фішингового повідомлення на основі тексту (див. рис. 3.5). Метою модуля є підвищення інформативності результатів класифікації, надаючи користувачу не лише оцінку ризику повідомлення (фішингове/безпечне), а й опис його можливої категорії (типу загрози). Основні етапи роботи наведено нижче.

1. Завантаження та об'єднання датасетів. Модуль використовує два джерела даних:

– відкритий набір ealvaradob/phishing-dataset із платформи HuggingFace, що містить SMS- та email-повідомлення;

– кастомний набір csv/custom_messages.csv, створений автором для доповнення реальними прикладами локальних фішингових текстів.

Після завантаження дані об'єднуються в єдину таблицю та узгоджуються за форматом (мітки класів label приводяться до цілочисельного типу).

2. Присвоєння категорій повідомлень. Для кожного запису визначається категорія на основі змісту тексту та мітки класу. Використовується функція assign_category(), яка здійснює пошук ключових фраз за допомогою регулярних виразів. Передбачено такі категорії: «safe» – безпечні повідомлення; «request_click_link» – містить заклики перейти за посиланням; «request_personal_data» – містить запити на введення персональних або банківських даних; «account_blocked» – повідомлення про блокування акаунта; «payment_alert» – повідомлення про оплату чи рахунок; «default» – інші випадки, не віднесені до вищеописаних.

3. Правило "anti-trigger". З метою зниження кількості помилкових спрацьовувань застосовується додаткове правило, яке перевіряє наявність ключових слів («click», «link», «blocked», «payment» тощо). Якщо в тексті відсутні тригерні слова, категорія повідомлення може бути знижена до default.

4. *Підготовка даних та розподіл вибірок.* Модуль формує ознаки текстів та розподіляє дані на навчальну і тестову вибірки у співвідношенні 80:20, забезпечуючи рівномірне представлення категорій.

5. *Побудова та навчання моделі.* Для навчання застосовується пайплайн із двох компонентів:

– TF-IDF Vectorizer з символьними n-грамами (3–5 символів), що дозволяє враховувати структуру тексту навіть у коротких повідомленнях;

– One-vs-Rest Logistic Regression, яка дозволяє одночасно передбачати кілька категорій класів.

Модель тренується протягом 1500 ітерацій для досягнення стабільної збіжності.

6. *Оцінка результатів.* На тестовій вибірці обчислюються стандартні метрики (precision, recall, F1-score і support) для кожної категорії. Це дає змогу оцінити, наскільки добре модель розрізняє різні типи фішингових повідомлень.

7. *Збереження моделі.* Після навчання створена модель серіалізується за допомогою «joblib» та зберігається у файл pkl/recommendation_model.pkl для подальшого використання у Flask-додатку під час інференсу.

```
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report
import joblib
from datasets import load_dataset
import os

# Завантаження датасетів
print("🔷 Завантаження основного датасету повідомлень (SMS + Email)...")
dataset = load_dataset("ealvaradob/phishing-dataset", "texts", trust_remote_code=True)
df_main = dataset['train'].to_pandas()
```

Рисунок 3.5 – Фрагмент коду train_recommendation_model.py

Тобто, модуль train_recommendation_model.py виконує роль інтелектуального компонента рекомендаційної підсистеми, що доповнює класифікацію повідомлень семантичним аналізом змісту, дозволяючи системі не лише виявляти фішингові тексти, а й пояснювати користувачу тип потенційної загрози. Він відіграє ключову роль у підвищенні інформативності та корисності результатів для користувача. Завдяки йому вебінтерфейс не обмежується простим висновком «фішингове» чи «безпечне», а пропонує уточнення, що фактично перетворює класифікацію на пояснювальний аналіз.

Саме моделі, сформовані у цьому модулі, дозволяють app.py під час інференсу повертати зрозумілу і мотивовану відповідь, яка описує, чому повідомлення може бути небезпечним. Це суттєво покращує взаємодію з користувачем, оскільки система надає не лише результат, а й контекст – ймовірний сценарій атаки. Тобто, train_recommendation_model.py доповнює базову функціональність класифікації і підсилює практичну цінність системи в реальних умовах використання.

Загалом, розроблені Python-модулі формують цілісну архітектуру бекенду інтелектуальної системи ідентифікації фішингу, у якій кожен компонент виконує чітко визначену функцію – від збору та обробки даних до навчання моделей і забезпечення інтерактивної взаємодії з користувачем. Така модульна структура забезпечує гнучкість системи, спрощує її розширення та оновлення, дозволяє швидко інтегрувати нові методи ML або джерела даних без потреби в істотних змінах архітектури. У результаті бекенд виступає надійною основою, що поєднує аналітичні, класифікаційні та рекомендаційні можливості в єдину функціональну платформу для виявлення фішингових загроз.

3.2 Файли фронтенду (HTML, CSS, JS)

Фронтенд системи виконує роль інтерактивного інтерфейсу користувача, який забезпечує зручну взаємодію з бекендом через веббраузер. Його основне завдання – надати користувачу можливість вводити текстові повідомлення або

URL-посилання для перевірки, отримувати результати класифікації, переглядати рекомендації щодо дій і залишати зворотний зв'язок (фідбек).

Сам інтерфейс побудований на базі стандартних вебтехнологій: HTML (структура сторінки), CSS (стилізація елементів) і JS (логіка взаємодії з сервером Flask через API). Завдяки цьому забезпечується простота, швидкодія та універсальна сумісність із будь-якими сучасними браузерами. Наприклад, HTML-файл templates/index.html визначає основну структуру інтерфейсу користувача (див. рис. 3.6).

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8" />
    <title>Phishing Identification</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@700;900&display=swap" rel="stylesheet">
</head>
<body>
<div class="container">
    <div class="header-flex">
        <img src="{{ url_for('static', filename='pictures/mess.gif') }}" alt="Повідомлення" class="gif-mess">
        <h1>Phishing Identification</h1>
        <img src="{{ url_for('static', filename='pictures/mess.gif') }}" alt="Посилання" class="gif-mess">
    </div>
```

Рисунок 3.6 – Фрагмент коду index.html

Сторінка побудована у вигляді двох основних блоків-карток – для перевірки повідомлень та перевірки URL-посилань. У верхній частині сторінки розташовано заголовок із назвою системи "Phishing Identification" та декоративними GIF-зображеннями, які візуально підкреслюють тематику повідомлень. Далі розміщені дві інтерактивні картки (card), оформлені у вигляді блоків із власними заголовками, полями введення, кнопками дій та зонами відображення результатів.

У блоці «Перевірка повідомлення 📧» користувач вводить текст повідомлення у полі «<textarea>» і натискає кнопку «Перевірити повідомлення». Після цього виводиться результат класифікації та рекомендація від системи. Нижче розташована форма для введення фідбеку користувача, який відправляється на

сервер. Більше того, у блоці «Перевірка посилання 🔗» розміщено поле «<input type="text">» для введення URL-адреси, кнопка для запуску перевірки, а також зони відображення результату, рекомендації та форми для зворотного зв'язку.

У нижній частині index.html розміщено JS-код, який виконує роль фронтенд-логіки (файл script.js) (див. рис. 3.7). Основні функції коду:

– checkMessage() – обробляє перевірку текстових повідомлень: зчитує введений текст, надсилає його через fetch()-запит до Flask-маршруту /check_message, отримує JSON-відповідь із результатом класифікації та оновлює інтерфейс (колір картки, текст рекомендації);

– checkURL() – аналогічна функція для перевірки URL-посилань через маршрут /check_url;

– sendFeedback(type) – реалізує відправку відгуку користувача (message або url) на серверний маршрут /feedback, включаючи зміст повідомлення, результат класифікації та текст фідбеку;

– autoResize() – автоматично регулює висоту поля введення тексту залежно від кількості символів для зручності користувача;

– об'єкт recommendations містить наперед визначені шаблони рекомендацій для різних типів фішингових повідомлень.

```
async function checkMessage() {
    const message = document.getElementById("message").value.trim();
    const card = document.getElementById("messageCard");
    const resultEl = document.getElementById("messageResult");
    const recEl = document.getElementById("messageRecommendation");

    card.classList.remove("safe", "phishing", "warning");
    resultEl.classList.remove("warning");
    recEl.innerText = "";

    if (message === "") {
        resultEl.innerText = "⚠️ Введіть текст повідомлення для перевірки";
        resultEl.classList.add("warning");
        card.classList.add("warning");
        return;
```

Рисунок 3.7 – Фрагмент коду JS-модулю у файлі index.html

Таким чином, JavaScript-модуль забезпечує динамічну взаємодію фронтенду з бекендом у режимі реального часу, оновлюючи сторінку без її перезавантаження та створюючи інтерактивний користувацький досвід. В свою чергу, файл стилів static/styles.css відповідає за візуальне оформлення інтерфейсу системи, створюючи приємне, інтуїтивно зрозуміле та адаптивне середовище взаємодії користувача з вебзастосунком (див. рис. 3.8). Основна ідея стилізації полягає у поєднанні простоти, легкості сприйняття та акцентів на безпеці (через кольорові індикатори результатів). Завдяки використанню гнучкої верстки та м'якої кольорової гами, інтерфейс зберігає естетичність і зручність навіть на екранах різного розміру.

```
2  *, *::before, *::after { box-sizing: border-box; }
3
4  body {
5      font-family: 'Roboto', sans-serif;
6      background: linear-gradient(135deg, #bef7ff, #c8eeff, #b9d8ff);
7      background-attachment: fixed;
8      margin: 0;
9      padding: 0;
10 }
11
12 .container {
13     width: 90%;
14     max-width: 1300px;
15     margin: 50px auto;
16 }
```

Рисунок 3.8 – Фрагмент коду styles.css

У файлі визначено базові стилі для всієї сторінки («body», «container»), заголовків («h1»), структурних блоків («.header-flex», «.cards», «.card») та інтерактивних елементів («button», «textarea», «input»). Фон оформлено градієнтом із відтінків блакитного, що асоціюється з технологічністю та довірою. Основні картки «.card» мають округлені кути, напівпрозоре біле тло й тінь, що створює ефект об'ємності. Ключовим елементом є кольорове кодування станів карток: «.safe» – зелений фон для безпечних повідомлень; «.phishing» – червоний відтінок для виявлених фішингових випадків; «.warning» – жовтий колір для

попереджувальних ситуацій. Ці кольори допомагають користувачу миттєво сприймати рівень ризику.

Окремі стилі визначені для текстових полів і кнопок: вони мають згладжені краї, м’які межі, зміну кольору при наведенні, що покращує інтерактивність. Блок «.recommendation» візуально виділяє рекомендації системи, а «.feedback-box» оформлює секцію відгуку користувача – із власним кольором тла, межами та кнопками. Тому CSS-файл забезпечує єдиний візуальний стиль і зрозумілу навігацію інтерфейсу, підсилюючи сприйняття результатів класифікації та рекомендацій. Він поєднує естетику з функціональністю, створюючи зручне середовище для користувача при роботі з системою виявлення фішингових повідомлень і посилань.

Підсумовуючи, фронтенд-система забезпечує інтуїтивно зрозумілий, естетично приємний та динамічний інтерфейс для користувача, який дозволяє швидко вводити дані, отримувати результати класифікації та рекомендації, а також залишати відгуки. Завдяки поєднанню HTML для структури, CSS для стилізації та JS для логіки взаємодії з бекендом, забезпечується висока зручність використання, наочність інформації та інтерактивність, що підвищує ефективність сприйняття ризиків і підтримує безпечну поведінку користувача при роботі з повідомленнями та URL.

3.3 Сховище даних та моделей (CSV, PKL)

Для забезпечення стабільної роботи інтелектуальної системи та відтворюваності результатів усі дані, моделі та проміжні результати зберігаються у відповідних форматах: CSV (табличні дані) та PKL (серіалізовані моделі ML). Такий підхід дозволяє швидко оновлювати або перевчати моделі без необхідності повторного завантаження вихідних датасетів і зберігати історію змін результатів класифікації та фідбеків користувачів.

У структурі проєкту передбачено окремі каталоги для кожного типу даних (див. рис. 3.9). Наприклад, «csv/» – містить вхідні набори даних, створені та доповнені автором:

– «custom_messages.csv» – користувацький набір текстових повідомлень (SMS, Email), який включає приклади безпечних і фішингових текстів, використовується для донавчання моделей класифікації повідомлень;

– «custom_urls.csv» – набір користувацьких URL-посилань, позначених як безпечні або фішингові, застосовується для навчання моделей аналізу посилань.

```
"Ваше замовлення надіслано. Очікуйте доставку протягом 3-5 робочих днів.",0
"Вітаємо! Оплата за послугу успішно прийнята. Дякуємо, що обрали нас!",0
"Ваше замовлення відправлено. Очікуйте доставку протягом 2-3 днів.",0
"Дякуємо за оплату! Чек надіслано на вашу електронну пошту.",0
"Hello!",0
"How are you?",0
"Привіт, як справи?",0
"Дякуємо за оплату замовлення. Очікуйте відправку протягом 1-2 днів.",0
"Привіт! Нагадую про зустріч завтра о 10:00 у нашому офісі.",0
"Змін у розкладі занять не буде. Гарного дня!",0
"Hi! Just a reminder about our meeting tomorrow at 10 AM.",0
"Your order has been shipped and will arrive within 2–3 business days.",0
"Ваше замовлення підтверджено. Дякуємо за покупку.",0
"Доставка очікується завтра.",0
"Чек на оплату сформовано автоматично. Не відповідайте на це повідомлення",0
"Оплата не пройшла. Підтвердьте дані картки, щоб уникнути блокування.",1
"Для підтвердження особи надішліть номер картки та CVV-код.",1
"Thank you for your payment! Your receipt has been sent to your email.",1
"Ваш акаунт заблоковано через підозрілу активність. Для розблокування перейдіть за посиланням.",1
"Your account has been suspended due to suspicious activity. Please click the link below immediately to
"Your bank account will be locked unless you confirm your details immediately at",1
```

Рисунок 3.9 – Фрагмент файлу «custom_messages.csv»

Також є файл «feedback/feedback_log.csv» – файл журналу відгуків користувачів, у якому зберігаються результати класифікацій, введені тексти та коментарі користувачів, що використовуються для подальшого аналізу точності моделей та як потенційні нові приклади для розширення навчальних вибірок (див. рис. 3.10).

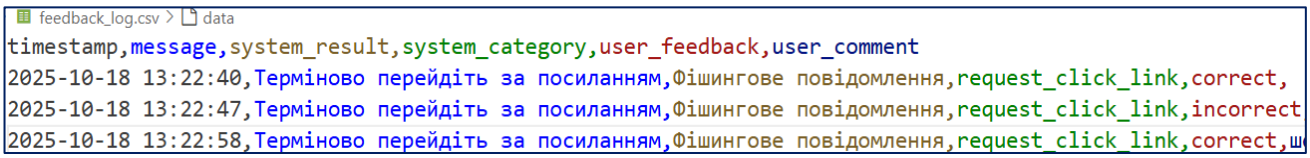

Рисунок 3.10 – Файл «feedback_log.csv»

В свою чергу, «pkl/» – директорія для збереження натренованих моделей і векторизаторів, які використовуються в процесі інференсу у Flask-додатку:

– «message_lr_model.pkl», «message_nb_model.pkl», «message_svm_model.pkl» – моделі класифікації текстових повідомлень, натреновані відповідно з використанням методів LR, NB і SVM;

– «url_rf_model.pkl», «url_gb_model.pkl» – базові моделі класифікації URL-посилань (RF, GB);

– «url_rf_model_kaggle.pkl», «url_gb_model_kaggle.pkl» – додаткові моделі, натреновані на розширених наборах даних із платформи Kaggle для підвищення точності класифікації;

– «url_vectorizer.pkl» – TF-IDF векторизатор для перетворення символьних n-грам URL-адрес у числові ознаки, необхідні для передбачення;

– «recommendation_model.pkl», «recommendation_model_kaggle.pkl» – моделі рекомендацій, які визначають тип фішингових повідомлень і формують пояснення результатів для користувача.

Модулі ML (наприклад, «messages.py», «url.py», «train_recommendation_model.py») зчитують відповідні CSV- та PKL-файли під час процесів навчання та інференсу. Завдяки цьому забезпечується автономність системи: усі компоненти можуть працювати офлайн без потреби у зовнішніх запитах до баз даних або хмарних ресурсів. Крім того, результати класифікації та зворотний зв’язок користувачів зберігаються в окремих CSV-файлах, що дозволяє виконувати періодичний аналіз точності моделей, оновлювати навчальні вибірки та відслідковувати тенденції у типах фішингових повідомлень.

Підсумовуючи, система сховищ даних та моделей виконує ключову роль у підтримці стійкості, гнучкості й відтворюваності інтелектуальної системи. Вона забезпечує централізоване зберігання навчальних даних, результатів класифікацій і користувацьких відгуків, що створює основу для безперервного вдосконалення моделей ML й підвищення якості виявлення фішингових загроз.

Висновки до розділу 3

У даному розділі описано реалізацію інтелектуальної системи виявлення фішингових повідомлень і посилань, що поєднує технології ML, веброзробки та інтерактивної аналітики. На основі архітектурного поділу на бекенд, фронтенд і сховище даних створено повноцінний програмний комплекс, здатний виконувати як автоматичну класифікацію, так і формування рекомендацій для користувача в режимі реального часу.

Бекенд системи реалізовано засобами мови Python із використанням фреймворку Flask, який забезпечує обробку HTTP-запитів і взаємодію з моделями ML. До складу серверної частини входять окремі модулі, кожен із яких виконує специфічну функцію. Завдяки такій модульній структурі система характеризується гнучкістю, масштабованістю та простотою оновлення, що дозволяє безперешкодно інтегрувати нові моделі, джерела даних або алгоритми без зміни основної архітектури.

Фронтенд, реалізований за допомогою HTML, CSS та JS, формує інтуїтивно зрозумілий вебінтерфейс, який забезпечує зручну взаємодію користувача із серверною частиною. Через інтерактивні форми користувач може вводити повідомлення або URL-адреси, миттєво отримувати результати класифікації, рекомендації та залишати зворотний зв’язок. Використання асинхронних запитів і кольорових індикаторів станів створює динамічний та наочний користувацький досвід.

Сховище даних системи, організоване у форматах CSV і PKL, забезпечує централізоване зберігання навчальних вибірок, результатів класифікації, фідбеків і

серіалізованих моделей ML. Це гарантує відтворюваність експериментів, автономність роботи без підключення до зовнішніх ресурсів і можливість періодичного оновлення моделей на основі зібраних користувацьких даних.

Таким чином, реалізована інтелектуальна система є комплексним рішенням для автоматизованого виявлення фішингових загроз. Вона поєднує сучасні методи обробки текстів, аналізу URL і рекомендаційні механізми, утворюючи єдину архітектурну платформу, орієнтовану на підвищення кібербезпеки користувачів. Створена інфраструктура закладає основу для подальшого розвитку системи – зокрема, удосконалення моделей ML, розширення баз даних і впровадження механізмів активного навчання на основі користувацького фідбеку.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Опис наборів даних для тестування

Для проведення тестування та оцінки ефективності розробленої системи виявлення фішингових повідомлень і посилань використовувались кілька відкритих наборів даних, що містять приклади як шкідливих, так і легітимних текстів та URL-адрес. Головним джерелом даних є набір «Phishing Dataset», розміщений на платформі Hugging Face у репозиторії «ealvaradob/phishing-dataset», який забезпечує комплексне охоплення різних типів фішингових об'єктів – повідомлень, посилань, електронних листів і вебсторінок (див. рис. 4.1).

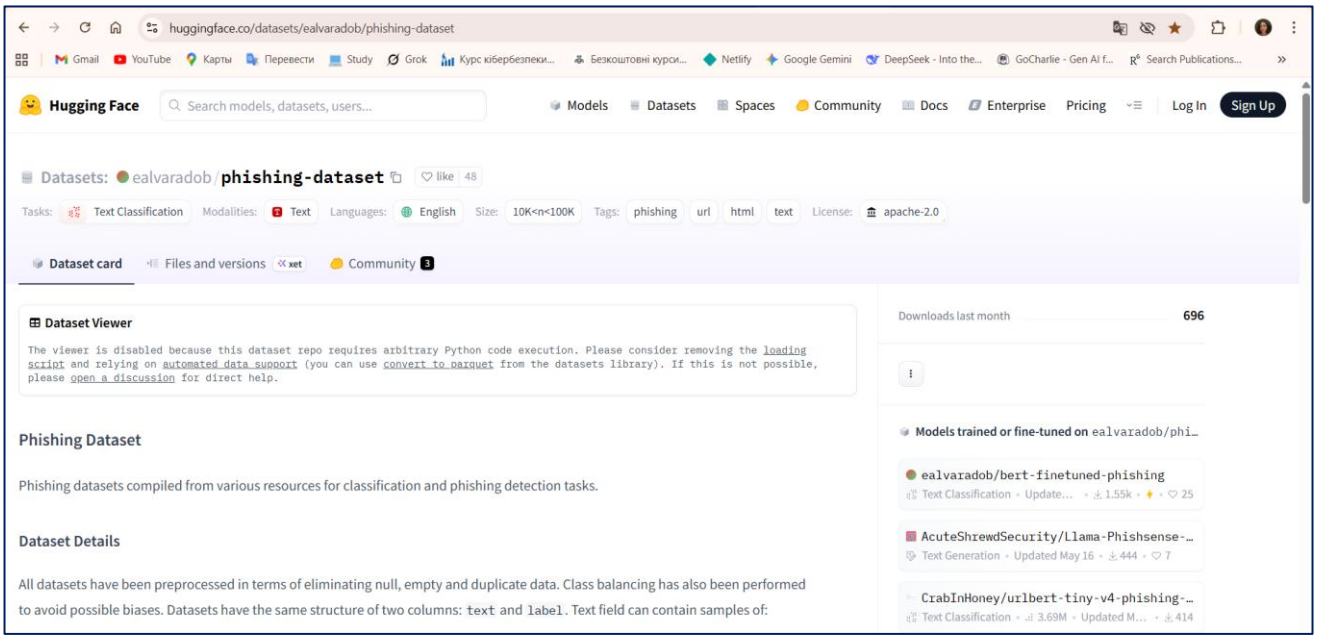

Рисунок 4.1 – Набір даних «Phishing Dataset» [20]

Даний набір призначений для задач класифікації текстів і фішингового виявлення за допомогою методів ML. Він попередньо оброблений – видалено порожні, дубльовані та некоректні записи, а також проведено балансування класів для уникнення зміщень (bias) у моделі. Усі підмножини набору мають єдину структуру з двох стовпців: «text» (текстовий вміст повідомлення, URL, HTML-коду або тіла листа); «label» (цільова змінна, що набуває значень 1 (phishing) або 0

(benign)). Набір є компіляцією даних із чотирьох незалежних джерел, які відрізняються за типом вхідних даних:

– «Mail dataset» – містить понад 18 000 електронних листів, створених співробітниками корпорації Enron, що використовується для задач виявлення фішингових або шахрайських повідомлень у корпоративному середовищі;

– «SMS dataset» – включає 5 971 текстове повідомлення, серед яких 489 є spam, 638 – smishing (SMS phishing), а 4 844 – безпечні (ham), а дані були отримані шляхом перетворення зображень у текстові повідомлення з використанням Python;

– «URL dataset» – найбільший піднабір, який містить понад 800 000 URL-адрес, серед яких близько 52% легітимні, а 47% – фішингові, а дані зібрані з різних відкритих ресурсів, включно з сайтами JPCERT, PhishTank, OpenPhish, GitHub-репозиторіями та базами Kaggle;

– «Website dataset» – набір із близько 80 000 вебсайтів (50 000 легітимних та 30 000 фішингових), кожен із яких представлений URL та HTML-кодом, при чому дані збиралися через Google Search і публічні фішингові бази.

З метою уникнення надмірного домінування URL-записів у загальній вибірці було створено дві комбінації даних: «combined full» (повна версія з усіма URL, де понад 800 тис. рядків) та «combined reduced» (скорочена версія, у якій обсяг URL-записів зменшено на 95% для досягнення балансу між типами даних). Саме «combined reduced» є рекомендованим для навчання та тестування моделей, оскільки він зберігає репрезентативність усіх типів вхідних об’єктів (SMS, email, URL, HTML), не допускаючи переважання одного класу.

Набір оптимізовано для використання з мовними моделями типу BERT, тому він не вимагає традиційної попередньої обробки (lemmatization, stemming, видалення стоп-слів тощо). Це пояснюється тим, що такі моделі використовують контекстне кодування слів, і навіть пунктуація чи службові слова можуть бути важливими ознаками для коректного розпізнавання фішингових патернів. За замовчуванням набір містить лише один спліт «train», тому для розділення на

навчальну та тестову вибірки застосовується розбиття 80/20 із фіксованим випадковим станом (random_state=42).

Тобто, набір даних «ealvaradob/phishing-dataset» є комплексною, збалансованою та репрезентативною колекцією фішингових і легітимних об'єктів, яка охоплює кілька форматів кіберзагроз. Саме його використання дозволяє об'єктивно оцінити точність і надійність розробленої системи, а також забезпечує відтворюваність результатів завдяки відкритій ліцензії Apache 2.0.

Крім того, для підвищення точності та адаптації системи до реальних україномовних і змішаних контекстів було створено власний кастомний набір-доповнення (див. рис. 4.2). Він складається з двох окремих частин: «custom_messages.csv» (для текстових повідомлень) та «custom_urls.csv» (для вебпосилань). Обидва файли мають однакову структуру, що відповідає формату основного датасету – два стовпці: «text» (текст повідомлення або URL-адреса) та «label» (цільова ознака, де 0 – безпечне, 1 – фішингове).

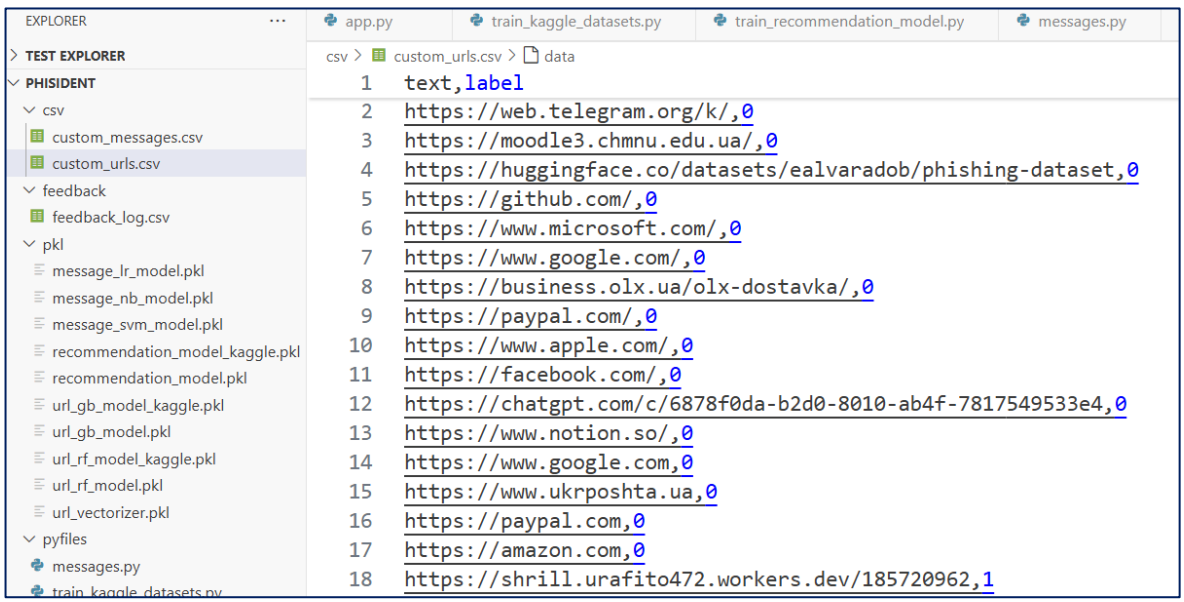

Рисунок 4.2 – Фрагмент файлу «custom_urls.csv»

Щодо файлу «custom_messages.csv», то він містить як англомовні, так і україномовні приклади коротких повідомлень, що відображають типові ситуації з

повсякденного обміну текстами. Безпечні зразки охоплюють нейтральні повідомлення побутового чи робочого характеру (нагадування про зустрічі, підтвердження замовлень, сповіщення про оплату тощо). Фішингові записи, своєю чергою, імітують реалістичні сценарії атак типу smishing та social engineering – повідомлення про «блокування акаунту», «проблеми з оплатою», «підозрілу активність» або «виграші призів», які містять посилання чи заклики виконати дію (наприклад, «підтвердити дані картки», «перейти за посиланням»). Особливістю даного піднабору є наявність україномовних прикладів, що підвищує релевантність моделі для локального використання в умовах україномовного інформаційного простору. Це дозволяє системі краще розпізнавати мовні патерни фішингових повідомлень, характерні для національного сегмента інтернету.

При цьому, другий піднабір «custom_urls.csv» охоплює приклади легітимних і фішингових URL-адрес. Безпечні зразки включають офіційні домени популярних платформ – Google, Microsoft, Apple, GitHub, Notion, Укрпошта тощо. Фішингові ж приклади відтворюють поширені техніки підміни доменів: використання схожих назв («secure-login.google.com», «paypal.verify-account-now.net»), піддоменів або підозрілих структур («workers.dev», «claim-now.co»), що часто застосовуються в атаках типу phishing та credential harvesting. Цей файл постійно доповнюється новими прикладами, зокрема реальними фішинговими доменами, зафіксованими в поточному інтернет-трафіку. Таким чином, він виконує роль динамічного джерела навчальних і тестових даних, адаптованих під сучасні шаблони атак.

Отже, кастомний набір-доповнення дозволяє збалансувати та локалізувати навчальні дані, враховуючи україномовні тексти, сучасні фішингові тенденції та реальні приклади з користувацького середовища. Його інтеграція з основним набором із Hugging Face забезпечує вищу точність класифікації, кращу узагальнюваність моделей і підвищує здатність системи виявляти фішинг у різних мовних контекстах.

Для підвищення якості навчання моделей класифікації повідомлень у системі виявлення фішингу було використано відкритий набір даних «Phishing Email

Dataset» із платформи Kaggle (див. рис. 4.3). Цей датасет є об'єднанням кількох відомих джерел, серед яких Enron, Ling, CEAS, Nazario, Nigerian Fraud та SpamAssassin. Вони містять реальні приклади електронних листів – як легітимних, так і фішингових, що дозволяє забезпечити різноманітність текстових структур і лінгвістичних особливостей.

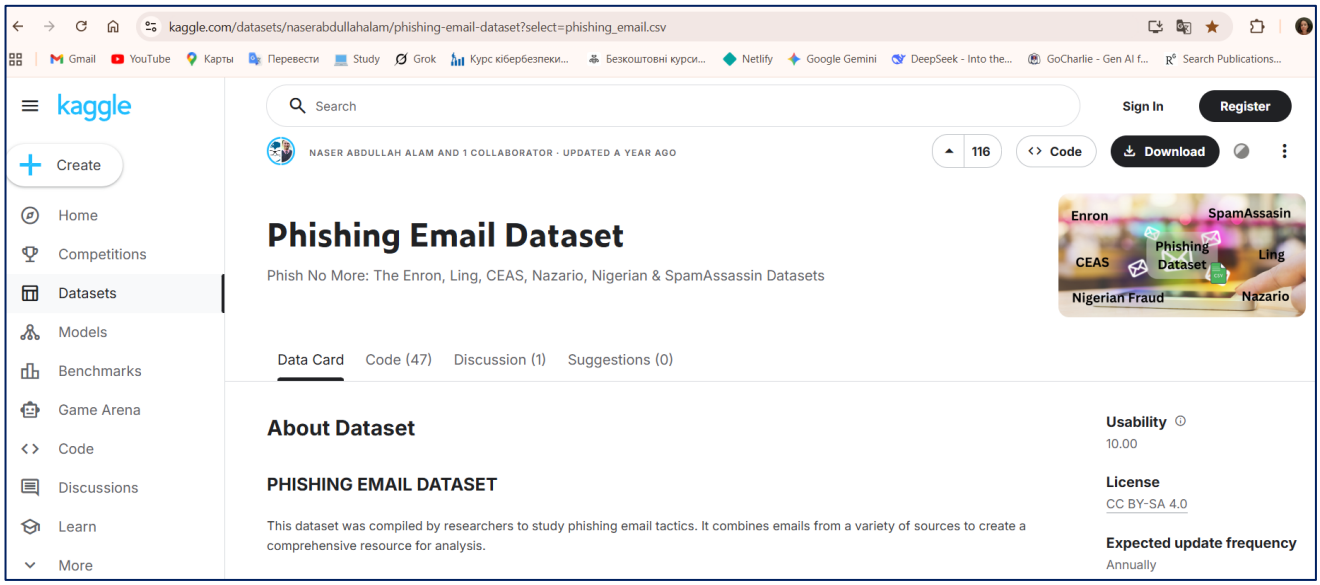

Рисунок 4.3 – Набір даних «Phishing Email Dataset» [21]

У рамках роботи використовувався узагальнений файл phishing_email.csv, який поєднує дані з усіх згаданих джерел. Він містить два основні стовпці: «text_combined» (текстове поле, що включає тему, тіло листа, дату та дані про відправника) та «label» (мітку класу, де 0 – легітимне повідомлення, 1 – фішингове). Цей датасет охоплює понад 82 000 електронних листів, з яких приблизно 43 000 є фішинговими. Завдяки такому співвідношенню класів набір даних є збалансованим для задачі бінарної класифікації.

У процесі побудови системи цей набір даних використовувався для первинного навчання моделей класифікації текстових повідомлень (у модулі «messages.py») та для попереднього порівняння результатів з кастомними наборами, сформованими на основі реалістичних повідомлень українською та англійською мовами. Тобто, використання «Phishing Email Dataset» дало змогу

забезпечити високий рівень узагальнення моделі та створити базову основу для подальшої адаптації під локальні умови за допомогою «custom_messages.csv» і «custom_urls.csv».

Більше того, для навчання моделей класифікації вебпосилань у системі виявлення фішингу було використано набір даних «Phishing URL Classifier Dataset Cleaned», розміщений на платформі Kaggle (див. рис. 4.4). Цей набір містить числові ознаки, отримані в результаті аналізу структури URL-адрес, що дозволяє виконувати класифікацію не за змістом сторінки, а безпосередньо за характеристиками самої адреси.

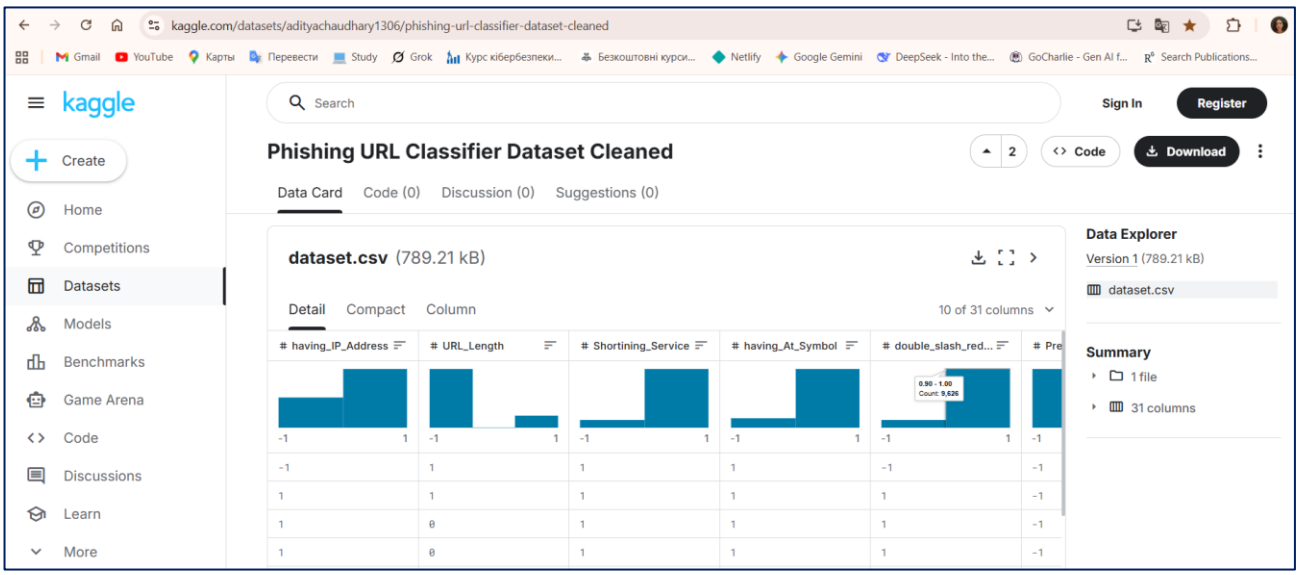

Рисунок 4.4 – Набір даних «Phishing Email Dataset» [22]

Датасет включає 31 атрибут, серед яких:

– having_IP_Address – наявність IP-адреси замість доменного імені;
– URL_Length – довжина URL;
– Shortining_Service – використання сервісів скорочення посилань;
– having_At_Symbol – наявність символу «@»;
– double_slash_redirecting – повторне використання подвійних слешів у середині адреси;

- Prefix_Suffix – використання дефісів між доменом і піддоменом;
- having_Sub_Domain – кількість піддоменів;
- SSLfinal_State – наявність і валідність SSL-сертифікату;
- Domain_registeration_length — тривалість реєстрації домену;
- Favicon – джерело значка сайту (з офіційного або стороннього домену).

Остання колонка, Label, визначає клас вебпосилання: «1» – фішингове посилання, «–1» – легітимне. Загалом набір охоплює близько 11 000 URL-адрес, що забезпечує достатню вибірку для навчання моделей ML, зокрема RF і GB, реалізованих у модулі «url.py».

Цей датасет характеризується належним рівнем очищення й стандартизації ознак, що дозволяє ефективно застосовувати його для дослідницьких і навчальних цілей. Загалом, використання «Phishing URL Classifier Dataset Cleaned» забезпечило можливість формування моделі, здатної визначати фішингові посилання на основі структурних і технічних параметрів без необхідності звернення до контенту вебсторінки.

Підсумовуючи, у процесі створення та тестування системи було використано комбінацію кількох відкритих і кастомних наборів даних, що доповнюють один одного за структурою, змістом і мовними характеристиками. Основний набір «Phishing Dataset» із платформи Hugging Face забезпечив широке охоплення фішингових повідомлень, посилань і вебсторінок, тоді як «Phishing Email Dataset» і «Phishing URL Classifier Dataset Cleaned» із Kaggle надали збалансовані та структуровані вибірки для класифікації текстів і URL відповідно. Власні файли «custom_messages.csv» і «custom_urls.csv» дозволили адаптувати систему до україномовного середовища та реальних шаблонів атак. Сукупне використання цих джерел даних дало змогу сформувати репрезентативну, збалансовану й гнучку навчальну основу, що підвищує точність, надійність і узагальнювальну здатність моделей системи виявлення фішингу.

4.2 Результати роботи моделей ML

У процесі навчання моделей для класифікації повідомлень система виводила докладні метрики якості у консоль, що дозволяє оцінити точність і стабільність роботи методів. Перед тренуванням здійснювалось завантаження основного та кастомного наборів даних (SMS, Email, користувацькі повідомлення), їх об’єднання, а також виведення перших кількох рядків для перевірки правильності структури. Після підготовки даних моделі навчаються на збалансованій вибірці та генерують підсумкові звіти у форматі precision / recall / f1-score / accuracy, що дозволяє порівняти ефективність кожного методу (див. табл. 4.1).

Таблиця 4.1 – Навчання моделей NB, LR та SVM

Naive Bayes				
	Precision	**Recall**	**F1-score**	**Support**
0	0.96	0.96	0.96	2496
1	0.93	0.94	0.93	1539
Accuracy			0.95	4035
Macro average	0.95	0.95	0.95	4035
Weighted average	0.95	0.95	0.95	4035
Logistic Regression				
	Precision	**Recall**	**F1-score**	**Support**
0	0.96	0.97	0.96	2496
1	0.96	0.93	0.94	1539
Accuracy			0.96	4035
Macro average	0.96	0.95	0.95	4035
Weighted average	0.96	0.96	0.96	4035
Support Vector Machine				
	Precision	**Recall**	**F1-score**	**Support**
0	0.97	0.98	0.97	2496
1	0.96	0.94	0.95	1539
Accuracy			0.96	4035
Macro average	0.96	0.96	0.96	4035
Weighted average	0.96	0.96	0.96	4035

Модель NB показала високу стабільність на обох класах, з незначним зниженням точності для фішингових повідомлень (через більшу варіативність текстів). Такий підхід є базовим, швидким у навчанні та використовується як референтна модель для порівняння. При цьому, модель LR продемонструвала найкращу узагальнюючу здатність, з високими показниками precision і recall для обох класів. Завдяки лінійній природі та добре контрольованій регуляризації, цей підхід зберігає баланс між точністю та чутливістю, що робить його надійним для продакшн-використання.

В свою чергу, SVM забезпечила найвищий рівень точності класифікації, особливо у випадках, де фішингові повідомлення містили приховані або змішані мовні структури. Метод ефективно відділяє класи навіть при складній текстовій семантиці, що свідчить про його придатність для задач з високою вимогою до precision. Далі усі три моделі після навчання серіалізовано у вигляді «.pkl» файлів для подальшого використання в онлайн-режимі.

Під час навчання моделей для класифікації URL-адрес у скрипті «url.py» у консоль також виводилися всі основні етапи обробки даних і результати оцінки точності. На початку програма завантажує основний відкритий датасет фішингових посилань та кастомний набір із директорії «csv/custom_urls.csv», після чого об’єднує їх в один масив для подальшої обробки. У консолі відображаються перші кілька рядків з обох джерел, що дозволяє візуально перевірити правильність структури та форматів даних. Підсумкові результати наведено у таблиці класифікації URL (див. табл. 4.2).

Таблиця 4.2 – Результати роботи моделей RF та GB під час класифікації URL

Random Forest				
	Precision	**Recall**	**F1-score**	**Support**
0	0.97	0.98	0.97	89053
1	0.97	0.96	0.97	78093
Accuracy			0.97	167146
Macro average	0.97	0.97	0.97	167146
Weighted average	0.97	0.97	0.97	167146

Продовження таблиці 4.2

Gradient Boosting				
	Precision	**Recall**	**F1-score**	**Support**
0	0.90	0.93	0.92	89053
1	0.92	0.88	0.90	78093
Accuracy			0.91	167146
Macro average	0.91	0.91	0.91	167146
Weighted average	0.91	0.91	0.91	167146

Модель RF показала найвищу точність – 97%, що свідчить про її здатність ефективно виявляти шкідливі посилання навіть у великих і незбалансованих наборах даних. Метод продемонстрував стабільність і надійність завдяки використанню ансамблю дерев рішень, що знижує ризик перенавчання. Зі свого боку, GB досяг точності 91%, показуючи добру узагальнюючу здатність, хоча трохи поступається RF через більшу чутливість до шуму та параметрів навчання. Проте він краще справляється з більш складними патернами URL-адрес і може бути ефективним у сценаріях, де важливо мінімізувати пропущені фішингові випадки. Також всі створені моделі та векторизатор збережено у форматі «.pkl» для подальшої інтеграції з веб-додатком.

Після запуску скрипта «train_kaggle_datasets.py» система автоматично завантажує необхідні набори даних із Kaggle та проводить попереднє навчання моделей класифікації повідомлень і посилань. У консолі відображається послідовність етапів завантаження, кількість записів у кожному датасеті та результати навчання. На рисунку наведено приклад виводу результатів тренування моделі класифікації фішингових електронних листів на наборі «Phishing Email Dataset», що демонструє високі показники точності (accuracy = 0.93) для більшості категорій повідомлень (див. табл. 4.3).

Таблиця 4.3 – Результати тренування recommendation-моделі на Kaggle «Phishing Email Dataset»

Classification report (email)				
	Precision	**Recall**	**F1-score**	**Support**
Account blocked	1.00	0.67	0.80	55
Default	0.93	0.98	0.95	5069
Payment alert	0.93	0.38	0.54	113
Request click link	0.98	0.92	0.95	2237
Request personal data	0.82	0.79	0.80	1105
Accuracy			0.93	8579
Macro average	0.93	0.75	0.81	8579
Weighted average	0.93	0.93	0.93	8579

Після цього відбувається навчання двох моделей для класифікації URL-адрес: RF та GB – на основі набору «Phishing URL Classifier Dataset Cleaned». Як видно з результатів, обидві моделі демонструють стабільно високі значення точності (0.97 та 0.95 відповідно), що свідчить про ефективність методів ML для виявлення фішингових посилань (див. табл. 4.4).

Таблиця 4.4 – Результати тренування моделей RF і GB на «Phishing URL Dataset»

Random Forest report				
	Precision	**Recall**	**F1-score**	**Support**
0	0.98	0.96	0.97	980
1	0.97	0.98	0.98	1231
Accuracy			0.97	2211
Macro average	0.97	0.97	0.97	2211
Weighted average	0.97	0.97	0.97	2211
Gradient Boosting report				
	Precision	**Recall**	**F1-score**	**Support**
0	0.95	0.94	0.95	980
1	0.95	0.96	0.96	1231
Accuracy			0.95	2211
Macro average	0.95	0.95	0.95	2211
Weighted average	0.95	0.95	0.95	2211

На наступному етапі виконувалося тренування моделі рекомендацій на основі змішаного набору даних, який включав як основні повідомлення (SMS та Email), так і користувацькі приклади з файлу «custom_messages.csv». Після об’єднання було сформовано понад 20 тисяч записів, серед яких переважну частину становили безпечні повідомлення (safe), що забезпечує збалансованість моделі для реальних умов використання (див. табл. 4.5).

Таблиця 4.5 – Результати виводу під час завантаження та попередньої обробки даних для тренування recommendation-моделі

Розподіл категорій повідомлень	
Safe	12481
Default	4397
Request click link	2088
Request personal data	1086
Payment alert	104
Accuracy	19
Name	count, dtype: int64

Після завершення навчання система вивела класифікаційний звіт, який показав середню точність моделі 91%. Найкращі результати продемонстровані для категорії «safe» (precision = 0.93, recall = 0.99) та «request_click_link» (precision = 0.91). Нижчі показники для рідкісних типів, як-от «account_blocked» чи «payment_alert», пояснюються невеликою кількістю прикладів у датасеті. Отримана модель була збережена у файлі «recommendation_model.pkl» для подальшого використання у системі PhisIdent (див. табл. 4.6).

Таблиця 4.6 – Результати класифікаційного звіту після тренування recommendation-моделі

Classification report				
	Precision	**Recall**	**F1-score**	**Support**
Account blocked	0.00	0.00	0.00	4
Default	0.84	0.83	0.83	879
Payment alert	1.00	0.10	0.17	21
Request click link	0.91	0.82	0.87	418
Request personal data	0.83	0.57	0.68	217
Safe	0.93	0.99	0.96	2496
Accuracy			0.91	4035
Macro average	0.75	0.55	0.58	4035
Weighted average	0.90	0.91	0.90	4035

Отже, проведене навчання моделей ML продемонструвало високу ефективність підходів, застосованих у системі PhisIdent. Методи класифікації повідомлень (NB, LR, SVM) забезпечили точність понад 90%, що свідчить про їхню придатність для виявлення фішингових текстів різного типу. Моделі для класифікації URL-адрес (RF та GB) показали ще вищі результати – до 97% точності, підтверджуючи надійність ансамблевих методів для аналізу веб-посилань. Додаткове навчання на Kaggle-датасетах дозволило підвищити узагальнюючу здатність моделей і забезпечити адаптивність до різних типів фішингових шаблонів. Рекомендаційна модель, побудована на змішаному наборі SMS, Email та користувацьких повідомлень, досягла 91% загальної точності, ефективно розпізнаючи як фішингові, так і безпечні категорії. Усі натреновані моделі збережено у форматі «.pkl» для подальшої інтеграції у веб-інтерфейс системи та реалізації онлайн-режиму перевірки повідомлень і посилань у реальному часі.

4.3 Демонстрація роботи фронтенду

Робота інтелектуальної системи виявлення фішингових повідомлень і посилань реалізується через вебінтерфейс, створений із використанням технологій HTML, CSS та JS. Головна сторінка сайту системи у браузері представляє зручний

і інтуїтивно зрозумілий інтерфейс, що дозволяє користувачу здійснювати перевірку текстових повідомлень або URL-адрес у режимі реального часу (див. рис. 4.5).

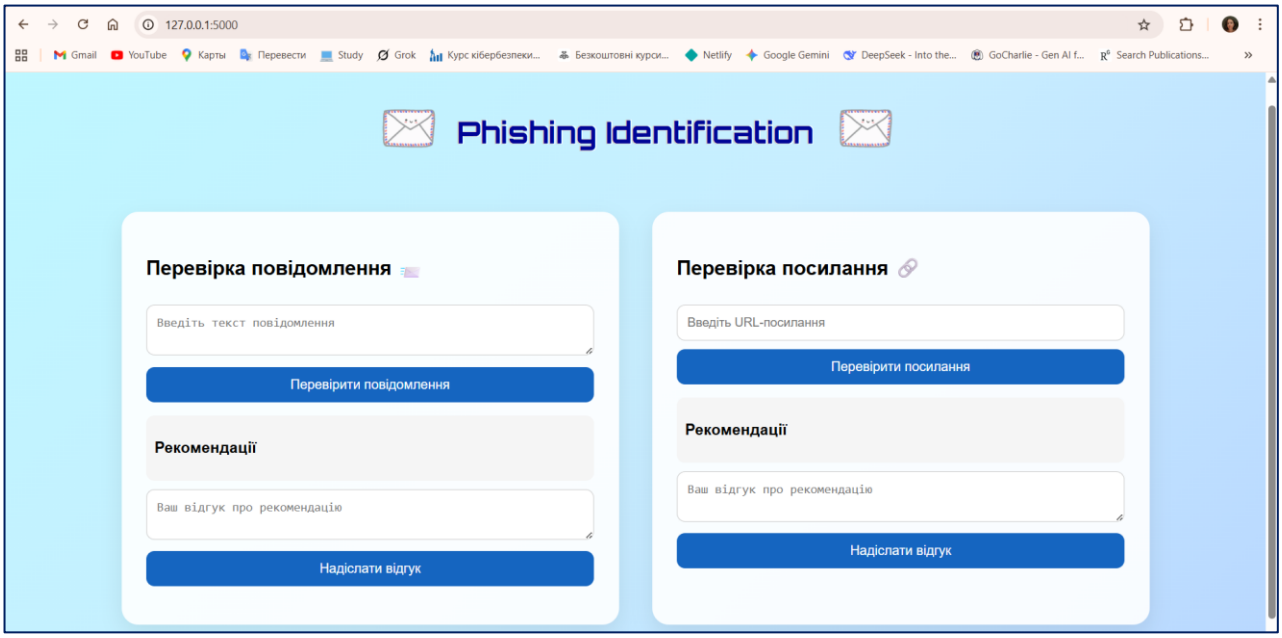

Рисунок 4.5 – Головна сторінка системи у браузері

Основна структура інтерфейсу включає дві окремі форми – для перевірки повідомлень і для перевірки посилань. Кожна форма містить поле введення даних, кнопку «Перевірити», а також блок результатів класифікації, де виводиться висновок моделі у форматі JSON. Після обробки введеного тексту або посилання система миттєво відображає статус (фішингове або безпечне) та рівень ризику. Візуально це реалізовано за допомогою індикатора ризику, який змінює колір картки залежно від результату: зелений – безпечне, червоний – фішингове, жовтий – сумнівне.

Коли система визначає, що повідомлення або URL-адреса не містить ознак фішингу, результат відображається у вигляді зеленої картки з маркером «☑ Безпечне» (див. рис. 4.6). У цьому стані в блоці рекомендацій користувачу пропонується коротке пояснення, наприклад: «Посилання відповідає структурі офіційного домену» або «Повідомлення не містить підозрілих закликів до дії».

Такий колір використовується для позначення низького рівня ризику та підтвердження безпечності взаємодії з об’єктом.

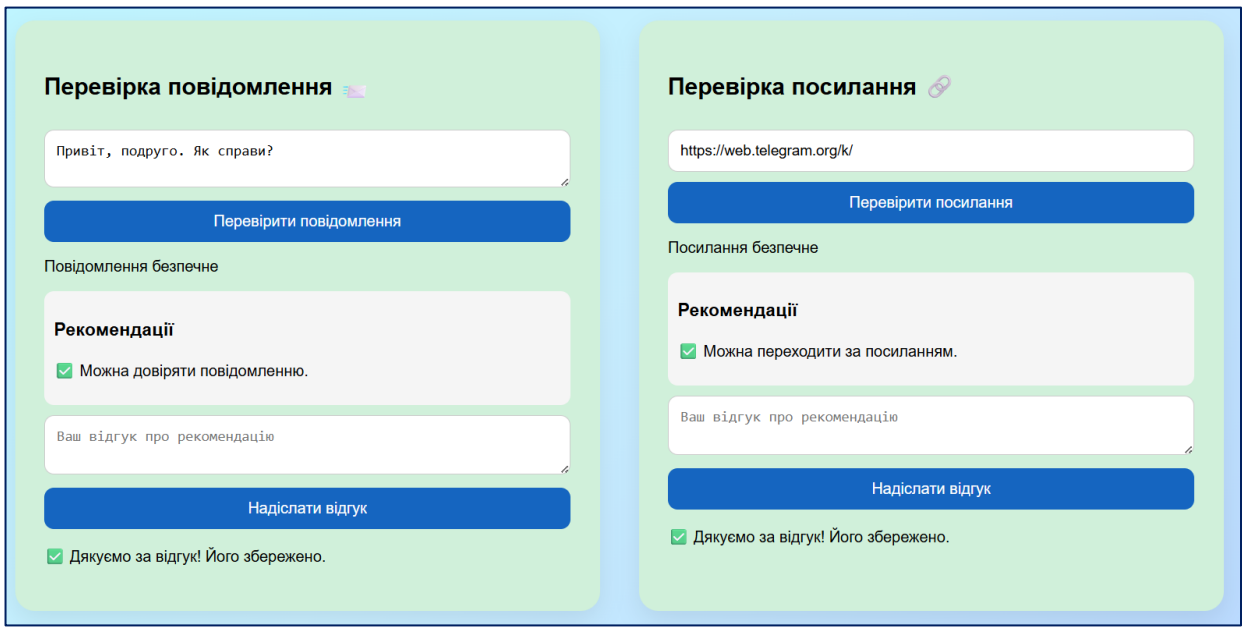

Рисунок 4.6 – Безпечні повідомлення та посилання

У випадку, коли модель класифікації виявляє ознаки фішингової активності, картка результату змінює колір на червоний, а індикатор ризику набуває статусу «⚠ Фішингове повідомлення / посилання» (див. рис. 4.7). Це означає, що введений текст або URL містить типові патерни атак – підозрілі піддомени, заклики до дії, фальшиві бренди, скорочені посилання або вразливі ключові фрази. Під карткою виводиться коротке попередження: «Не переходьте за цим посиланням» або «Не відповідайте на повідомлення, що вимагає особисті дані». Червоний індикатор сигналізує про високий рівень ризику та потребу у негайній обережності з боку користувача.

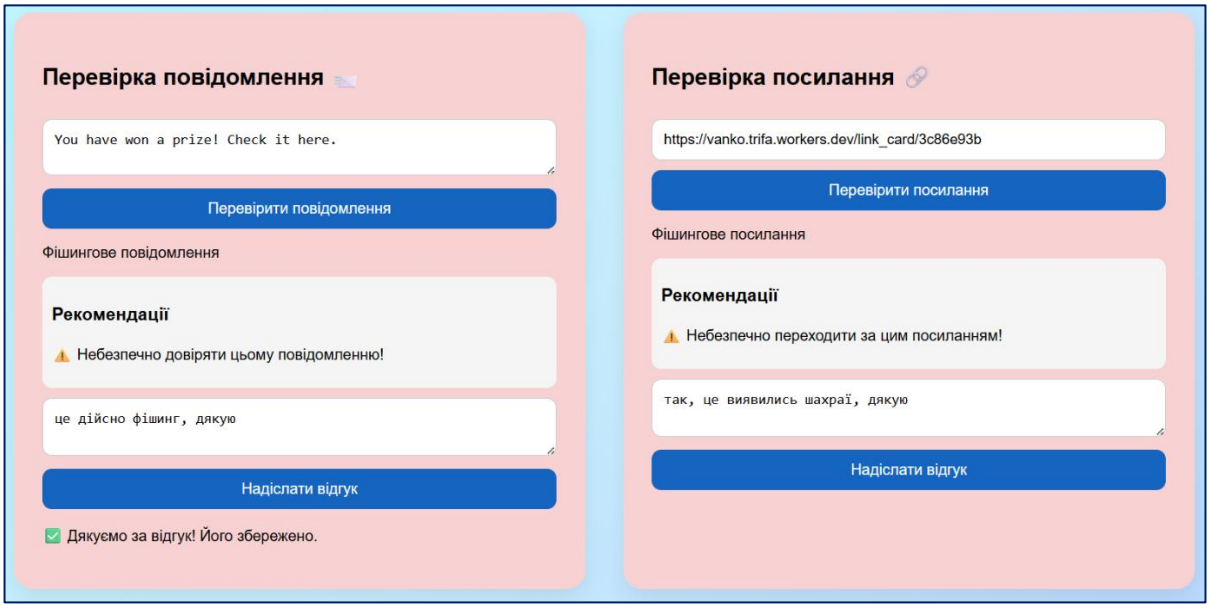

Рисунок 4.7 – Небезпечні повідомлення та посилання

Жовта картка з’являється у випадках, коли система не може з упевненістю класифікувати введений об’єкт, наприклад, якщо текст надто короткий або посилання має неоднозначну структуру (див. рис. 4.8). В такому стані результат позначається як «⚠ Потребує додаткової перевірки», а користувачу пропонується рекомендація перевірити джерело вручну або скористатися іншим сервісом перевірки. Жовтий фон візуально позначає помірний ризик і нагадує про необхідність обережності під час взаємодії з таким контентом.

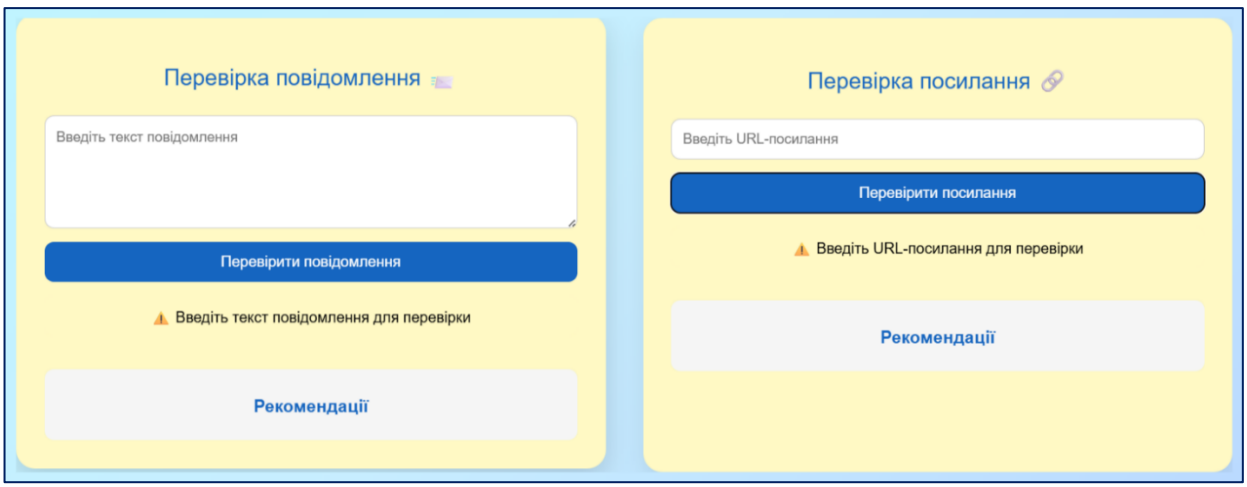

Рисунок 4.8 – Пусті поля для введення повідомлень і посилань

Тобто, кольорова система індикаторів забезпечує швидке інтуїтивне сприйняття рівня загрози, підвищує зрозумілість інтерфейсу та робить процес взаємодії з аналітичними результатами моделі доступним навіть для некваліфікованих користувачів.

Окремим функціональним елементом є блок рекомендацій, у якому користувач отримує пояснення або поради щодо безпечної взаємодії з повідомленням чи вебресурсом. Нижче розташована кнопка «Ваш відгук про рекомендацію» із можливістю залишити фідбек, що дозволяє системі вдосконалювати рекомендаційний компонент на основі користувацьких оцінок. Для покращення зручності взаємодії передбачені спливаючі підказки та повідомлення про помилки введення, які з’являються у випадку порожнього поля або некоректного формату URL. Це підвищує юзабіліті інтерфейсу й забезпечує коректність роботи додатку.

Типовий сценарій використання виглядає так: користувач відкриває сторінку → вводить текст повідомлення або посилання → натискає кнопку «Перевірити» → система виконує класифікацію на бекенді → результат з’являється у відповідному блоці з рекомендаціями та індикатором ризику. Таким чином, фронтенд реалізує не лише базову взаємодію користувача із системою, а й забезпечує візуалізацію результатів роботи моделей ML, що робить процес перевірки фішингового контенту інтуїтивним, зручним і наочним.

4.4 Результати роботи системи щодо фідбеку та інтерактивності

Наведемо приклад роботи механізму збирання користувацьких відгуків (фідбеку) у системі PhisIdent (див. рис. 4.9). Кожен фідбек містить тип перевіреного об’єкта (повідомлення або URL), сам контент, результат класифікації, а також коментар користувача, який підтверджує або спростовує визначення системи. У даному прикладі користувач повідомляє про два фішингові посилання, зазначаючи, що вони належать шахраям. Такі відгуки автоматично зберігаються у локальний

файл формату «.csv» або «.pkl», що дозволяє надалі використовувати їх для оновлення або донавчання моделей.

```
[FEEDBACK] 2025-10-18 13:56:32
Тип: url
Контент: http://microsoft-supports.com/reset-password?id=12345
Результат системи: Фішингове посилання
Відгук користувача: так, це виявились шахраї, дякую

127.0.0.1 - - [18/Oct/2025 13:56:32] "POST /feedback HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2025 13:56:36] "POST /check_url HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2025 13:57:02] "POST /check_url HTTP/1.1" 200 -

[FEEDBACK] 2025-10-18 13:57:24
Тип: url
Контент: https://shrill.urafito472.workers.dev/185720962
Результат системи: Фішингове посилання
Відгук користувача: так, це справді шахраї з онлайн магазину

127.0.0.1 - - [18/Oct/2025 13:57:24] "POST /feedback HTTP/1.1" 200 -
```

Рисунок 4.9 – Приклад обробки користувацького фідбеку в консолі системи

Завдяки цьому реалізовано базову інтерактивність системи – користувач бере участь у процесі підвищення точності класифікації, надаючи реальні приклади нових фішингових повідомлень. Подібний підхід дозволяє поступово адаптувати систему до змін у фішингових шаблонах та забезпечує зворотний зв’язок між користувачем і моделлю, що підвищує якість прогнозів і довіру до результатів.

Висновки до розділу 4

У цьому розділі було проведено тестування, оцінку ефективності та демонстрацію роботи інтелектуальної системи виявлення фішингових повідомлень і посилань PhisIdent. Для оцінки якості роботи системи застосовано кілька відкритих і кастомних наборів даних, що охоплюють різні формати фішингового контенту – від SMS і електронних листів до вебпосилань та HTML-кодів. Основні датасети з платформ Hugging Face і Kaggle забезпечили репрезентативність вибірки, збалансованість класів і можливість адекватного порівняння результатів. Додатково створені кастомні набори дозволили адаптувати систему до

україномовного середовища, що є важливим чинником для локального застосування у вітчизняному кіберпросторі.

Навчання моделей ML показало високу точність класифікації для всіх груп даних. Зокрема, моделі для класифікації текстів (NB, LR, SVM) досягли понад 90% точності, демонструючи стабільність і здатність узагальнювати результати навіть для змішаних мовних прикладів. Моделі для аналізу URL-адрес (RF, GB) продемонстрували ще вищу ефективність – до 97% точності, що підтверджує доцільність використання ансамблевих методів для виявлення фішингових посилань. Рекомендаційна модель, побудована на змішаних датасетах, забезпечила точність близько 91%, ефективно розпізнаючи типові патерни соціальної інженерії.

Розроблений фронтенд інтерфейс довів зручність та інтуїтивність взаємодії користувача із системою. Візуальна система кольорових індикаторів (зелений – безпечне, червоний – фішингове, жовтий – потребує перевірки) спрощує сприйняття результатів класифікації та робить інтерфейс зрозумілим навіть для користувачів без технічної підготовки. Механізм фідбеку, що дозволяє залишати оцінки й коментарі до результатів, реалізує зворотний зв’язок між користувачем і моделлю, підвищуючи динамічну адаптивність системи до нових типів атак.

Отже, результати тестування довели ефективність і надійність системи PhisIdent у виявленні фішингових повідомлень і посилань. Використання комбінованих наборів даних, сучасних методів машинного навчання та інтерактивного вебінтерфейсу забезпечує комплексний підхід до протидії фішингу. Це дозволяє не лише точно ідентифікувати загрози, а й створює основу для подальшого вдосконалення системи – зокрема, шляхом автоматизованого оновлення моделей і розширення підтримки мов та типів контенту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано та протестовано інтелектуальну систему виявлення фішингових повідомлень і посилань PhisIdent, яка поєднує методи ML, обробки природної мови та аналізу вебресурсів. Основною метою роботи було створення ефективного інструменту для автоматичного розпізнавання фішингових загроз у текстах повідомлень та URL-адресах, що дозволяє підвищити рівень інформаційної безпеки користувачів.

У ході дослідження проаналізовано сучасні види фішингових атак, існуючі підходи до їх виявлення та системи, які застосовуються у сфері кіберзахисту. Це дозволило сформулювати вимоги до майбутньої системи та визначити оптимальні методи ML для вирішення поставленої задачі. Зокрема, було обґрунтовано вибір таких методів, як NB, LR, SVM, RF та GB, які забезпечують високу точність класифікації текстів і посилань різних типів.

У процесі розробки створено модульну архітектуру системи, що складається з бекенду (Python), фронтенду (HTML, CSS, JS) і сховища моделей (PKL, CSV). Це забезпечує гнучкість, масштабованість і простоту інтеграції системи в реальні середовища – наприклад, у корпоративні служби кіберзахисту або антифішингові платформи. Інтерфейс системи реалізовано у вигляді вебзастосунку з інтерактивними формами введення, кольоровими індикаторами ризику та механізмом користувацького фідбеку.

Проведене тестування показало, що PhisIdent демонструє високу ефективність у виявленні фішингових повідомлень і URL-адрес. Точність класифікації для більшості моделей перевищила 90-95%, що підтверджує адекватність обраних методів та репрезентативність навчальних наборів даних. Система успішно ідентифікує шаблони соціальної інженерії, фейкові сторінки входу, повідомлення з вимогою введення персональних даних чи натискання на підозрілі посилання.

Важливим елементом проєкту стало впровадження механізму користувацького фідбеку, який дозволяє уточнювати результати класифікації, накопичувати нові дані та вдосконалювати моделі без повного перевчання. Це забезпечує адаптивність системи до нових видів атак і можливість поступового покращення її точності з часом.

Отже, результати дослідження підтвердили досягнення поставленої мети та виконання всіх завдань роботи. Розроблена система PhisIdent може бути використана як практичний інструмент для підвищення кібербезпеки організацій, навчальних установ і приватних користувачів. У подальшому доцільно розширити її функціонал шляхом інтеграції з базами фішингових доменів, додавання аналізу зображень та розроблення механізмів автоматичного оновлення моделей.

Таким чином, виконана робота робить суттєвий внесок у розвиток засобів інтелектуального виявлення фішингових загроз, поєднуючи наукові підходи з практичною реалізацією та підтвердженою ефективністю у тестових умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цифрова трансформація кібербезпеки. (2024). *Державний університет інформаційно-комунікаційних технологій.* URL: https://duikt.edu.ua/uploads/n_12581_11703414.pdf.

2. Rehida, P., Savenko, O. (2025). Method for Detecting Malicious Activity in Infected Programs. *Information Technology Computer Science Software Engineering and Cyber Security, (4)*. URL: https://doi.org/10.32782/IT/2024-4-21.

3. Mehed, D., Tkach, Y., Bazylevych, V. (2019). Research of Influence Technologies and Methods of Countering Fishing. *Ukrainian Information Security Research Journal, 21(4), 246-251.* URL: https://doi.org/10.18372/2410-7840.21.14338.

4. Saleem, H., Fuzail, M., Naeem, A., Aslam, N., Faiz, A. (2025). Model for the Detection of Web Phishing Attacks by using Machine Learning Algorithms. *Kashf Journal of Multidisciplinary Research, 2(06), 223-239*. URL: https://doi.org/10.71146/kjmr501.

5. Hassnain, M., Qureshi, I. A., Haider, A. (2025). Detection and Identification of Novel Attacks in Phishing using AI Algorithms. *International Journal of Computer Science and Mobile Computing, 14(3), 20-27*. URL: https://doi.org/10.47760/ijcsmc.2025.v14i03.003.

6. Oliveira, Í., Wagner, G., Amaral, G., Sales, T. P., Bullée, J.-W. H., Junger, M., Sarmah, D. K., Daneva, M., Guizzardi, G. (2025). An Ontological Model of the Phishing Attack Process. *Enterprise, Business-Process and Information Systems Modeling, 274-289.* URL: https://doi.org/10.1007/978-3-031-95397-2_17.

7. Rajput, J. P. (2025). Phishing Website Detection Using Machine Learning. *Interantional Journal of Scientific Research in Engineering and Management 9(6), 1-9.* URL: https://doi.org/10.55041/IJSREM50689.

8. Mishra, R., Varshney, G. (2025). A Study of Effectiveness of Brand Domain Identification Features for Phishing Detection in 2025. *Cornell University.* URL: https://doi.org/10.48550/arXiv.2503.06487.

9. Swetha, C. V., Shaji, S. (2025). Applying Textual Fusion and Metadata in a Multimodal Transformer-Based Approach to Detect Phishing. *Third International Conference on Recent Trends in Computer Science and Data Analytics*. URL: https://www.researchgate.net/publication/390040540_Applying_Textual_Fusion_and_Metadata_in_a_Multimodal_Transformer-Based_Approach_to_Detect_Phishing.

10. Dewa, S. K. M. (2023). Linear and Logistic Regression: Same Regression but Different Purpose. *Medium*. URL: https://blog.gopenai.com/linear-and-logistic-regression-same-regression-but-different-purpose-f6ff5f93b7ef.

11. Machine Learning: Essence, History, Methods & Techniques. (2024). *Altcraft, LLC*. URL: https://altcraft.com/blog/methods-and-techniques-of-machine-learning.

12. Zhao, A. (2017). Support Vector Machines: A Visual Explanation with Sample Python Code. *YouTube*. URL: https://www.youtube.com/watch?v=N1vOgolbjSc&ab_channel=AliceZhao.

13. Kim, J. (2018). Overview Random Forest. *Computer Vision for Autonomous Driving.* URL: https://gaussian37.github.io/ml-concept-RandomForest/.

14. Hemashreekilari. (2023). Understanding Gradient Boosting. *Medium*. URL: https://medium.com/@hemashreekilari9/understanding-gradient-boosting-632939b98764.

15. Raroque, C. (2024). 9+ Reasons Why Python Is A Popular Programming Language. *AloaLabs LLC*. URL: https://aloa.co/blog/why-python.

16. Javascript Overview – JavaScript Technologies. (2025). *JavaTechOnline*. URL: https://javatechonline.com/javascript-overview-javascript-technologies/.

17. Basic HTML Structure. (2025). *AptLearn*. URL: https://aptlearn.io/courses/html-css-and-javascript-course-for-web-developers/lesson/basic-html-structure/.

18. The Anatomy of HTML, CSS, and JS. (2025). *HedgeDoc*. URL: https://delante.co/definitions/css/.

19. Uspenski, A. (2023). Why VS Code remains a developer favorite, year after year. *ShiftMag*. URL: https://shiftmag.dev/vs-code-171/.

20. Phishing dataset. *Hugging Face*. URL: https://huggingface.co/datasets/ealvaradob/phishing-dataset.

21. Phishing Email Dataset. *Kaggle*. URL: https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset?select=phishing_email.csv.

22. Phishing URL Classifier Dataset Cleaned. *Kaggle*. URL: https://www.kaggle.com/datasets/adityachaudhary1306/phishing-url-classifier-dataset-cleaned.

23. Krishnaiahgari, K. R., Kathrine, G. J. W., Kishan Kumar, D. (2024). Cyber Sentinel: Intelligent Phishing URL Identification System Employing Machine Learning Methods. *IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/10677720.

24. Barraclough, P. A., Fehringer, G., Woodward, J. (2021). Intelligent cyber-phishing detection for online. *Computers & Security, 104.* URL: https://www.sciencedirect.com/science/article/abs/pii/S0167404820303965.

25. Chinta, P. C. R., Moore, C. S., Karaka, L. M., Sakuru, M., Bodepudi, V., Maka, S. R. (2025). Building an Intelligent Phishing Email Detection System Using Machine Learning and Feature Engineering. *European Journal of Applied Science, Engineering and Technology, 3(2), 41-54.* URL: https://doi.org/10.59324/ejaset.2025.3(2).04.

26. Mohd Ariffin, N. H., Mohamed Iqbal, M. I., Yusoff, M., Mohd Zulkefli, N. A. (2025). A Study on the Best Classification Method for an Intelligent Phishing Website Detection System. *ASEAN Artificial Intelligence Journal, 1(1), 20-33.* URL: https://doi.org/10.37934/aaij.1.1.2033.

27. Megha, N., Babu, K. R. R., Sherly, E. (2019). An Intelligent System for Phishing Attack Detection and Prevention. *Institute of Electrical and Electronics Engineers (IEEE), 1577-1582.* URL: https://doi.org/10.1109/icces45898.2019.9002204.

28. Mughaid, A., AlZu'bi, S., Hnaif, A., Taamneh, S., Alnajjar, A., Abu Elsoud, E. (2022). An intelligent cyber security phishing detection system using deep learning techniques. *Cluster Comput, 25(6), 3819-3828.* URL: https://pubmed.ncbi.nlm.nih.gov/35602317/.

ДОДАТОК А

Код файлу app.py

```
from flask import Flask, render_template, request, jsonify
import webbrowser
import threading
import joblib
import re
from langdetect import detect
from googletrans import Translator
from datetime import datetime
import os

app = Flask(__name__)

def safe_load(path, name):
    try:
        mdl = joblib.load(path)
        print(f"☑ Завантажено {name} з '{path}'")
        return mdl
    except Exception as e:
        print(f"⚠ Не вдалося завантажити {name} з '{path}': {e}")
        return None

message_model = safe_load('pkl/message_lr_model.pkl', 'message_model (LR)')
recommendation_model = safe_load('pkl/recommendation_model.pkl', 'recommendation_model (local)')
recommendation_model_kaggle = safe_load('pkl/recommendation_model_kaggle.pkl', 'recommendation_model_kaggle')
url_model = safe_load('pkl/url_rf_model.pkl', 'url_model (RF)')
url_vectorizer = safe_load('pkl/url_vectorizer.pkl', 'url_vectorizer (TF-IDF for url)')
url_model_kaggle = safe_load('pkl/url_rf_model_kaggle.pkl', 'url_model_kaggle (RF)')
url_gb_model_kaggle = safe_load('pkl/url_gb_model_kaggle.pkl', 'url_gb_model_kaggle (GB)')

translator = Translator()

def is_valid_url(url: str) -> bool:
    pattern = re.compile(r'^(https?://)([^\s]+\.)+[^\s]{2,}', re.IGNORECASE)
    return pattern.match(url) is not None

def contains_only_url(text: str) -> bool:
    text = text.strip()
    if re.fullmatch(r'(https?://)?([^\s]+\.)+[^\s]{2,}(\/\S*)?', text, re.IGNORECASE):
        return True
    return False

DEFAULT_RECOMMENDATIONS = {
    "request_click_link": "⚠ Не переходьте за посиланням! Це може бути шахрайство.",
    "request_personal_data": "⚠ Не надавайте свої особисті або банківські дані.",
    "account_blocked": "⚠ Ваш акаунт можуть використовувати шахраї. Не повідомляйте дані та перевірте
офіційний сайт.",
    "payment_alert": "⚠ Не здійснюйте оплату без перевірки достовірності повідомлення.",
    "default": "⚠ Небезпечно довіряти цьому повідомленню!"
}

@app.route("/")
def index():
    return render_template("index.html")
```

```
@app.route("/check_message", methods=["POST"])
def check_message():
    data = request.json or {}
    message = (data.get("message") or "").strip()
    if message == "":
        return jsonify({"result": "⚠ Введіть текст повідомлення для перевірки", "valid_message": False})
    if contains_only_url(message):
        return jsonify({"result": "❌ У полі повідомлення має бути текст, а не лише посилання.", "valid_message": False,
"recommendation": ""})

    try:
        lang = detect(message)
    except Exception:
        lang = None

    if lang and lang != 'en':
        try:
            translation = translator.translate(message, src=lang, dest='en')
            message_en = translation.text
        except Exception as e:
            print(f"[WARN] Translation failed: {e}")
            message_en = message
    else:
        message_en = message

    if message_model is None:
        return jsonify({"result": "❌ Модель для класифікації повідомлень не завантажена.", "valid_message": False})

    try:
        pred = int(message_model.predict([message_en])[0])
    except Exception as e:
        print(f"[ERROR] message_model prediction failed: {e}")
        return jsonify({"result": "❌ Помилка під час передбачення", "valid_message": False})

    if pred == 0:
        return jsonify({"result": "Повідомлення безпечне", "valid_message": True, "recommendation": "☑ Можна
довіряти повідомленню.", "category": None})

    cat = None
    used_rec_model = None
    if recommendation_model_kaggle:
        try:
            cat = recommendation_model_kaggle.predict([message_en])[0]
            used_rec_model = "recommendation_model_kaggle"
        except Exception as e:
            print(f"[WARN] recommendation_model_kaggle failed: {e}")
            cat = None
    if cat is None and recommendation_model:
        try:
            cat = recommendation_model.predict([message_en])[0]
            used_rec_model = "recommendation_model_local"
        except Exception as e:
            print(f"[WARN] recommendation_model (local) failed: {e}")
            cat = None
    if cat is None:
        cat = "default"

    recommendation = DEFAULT_RECOMMENDATIONS.get(cat, DEFAULT_RECOMMENDATIONS["default"])
```

```
    return jsonify({"result": "Фішингове повідомлення", "valid_message": True, "recommendation": recommendation,
"category": cat, "used_recommendation_model": used_rec_model})

@app.route("/check_url", methods=["POST"])
def check_url():
    data = request.json or {}
    url = (data.get("url") or "").strip()
    if url == "":
        return jsonify({"result": "⚠ Введіть URL-посилання для перевірки", "valid_url": False})
    if not is_valid_url(url):
        return jsonify({"result": "❌ Некоректний формат посилання", "valid_url": False})

    if url_vectorizer is None or url_model is None:
        return jsonify({"result": "❌ Модель для перевірки URL не готова на сервері.", "valid_url": False})

    try:
        url_vect = url_vectorizer.transform([url])
        pred = int(url_model.predict(url_vect)[0])
    except Exception as e:
        print(f"[ERROR] URL prediction failed: {e}")
        return jsonify({"result": "❌ Помилка під час аналізу посилання", "valid_url": False})

    if pred == 1:
        return jsonify({"result": "Фішингове посилання", "valid_url": True, "recommendation": "⚠ Небезпечно
переходити за цим посиланням!"})
    else:
        return jsonify({"result": "Посилання безпечне", "valid_url": True, "recommendation": "☑ Можна переходити за
посиланням."})

@app.route("/feedback", methods=["POST"])
def feedback():
    data = request.json or {}
    feedback_type = data.get("type", "unknown")
    content = data.get("content", "")
    system_result = data.get("system_result", "")
    user_opinion = data.get("user_opinion", "")
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    print(f"\n[FEEDBACK] {timestamp}")
    print(f"Тип: {feedback_type}")
    print(f"Контент: {content}")
    print(f"Результат системи: {system_result}")
    print(f"Відгук користувача: {user_opinion}\n")

    return jsonify({"message": "☑ Дякуємо за відгук! Його збережено."})

def open_browser():
    webbrowser.open_new("http://127.0.0.1:5000/")

if __name__ == "__main__":
    threading.Timer(1.25, open_browser).start()
    app.run(debug=True)
```

ДОДАТОК Б

Код файлу messages.py

```
from datasets import load_dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
import joblib
import os

print("◇ Завантаження основного датасету повідомлень (SMS + Email)...")
dataset = load_dataset("ealvaradob/phishing-dataset", "texts", trust_remote_code=True)
df_main = dataset['train'].to_pandas()

print("Перші 5 рядків основного датасету:")
print(df_main.head())

print("\n◇ Завантаження кастомного датасету з csv/custom_messages.csv...")
custom_csv_path = os.path.join(os.path.dirname(__file__), '..', 'csv', 'custom_messages.csv')
df_custom = pd.read_csv(custom_csv_path)

print("Перші 5 рядків кастомного датасету:")
print(df_custom.head())

df = pd.concat([df_main, df_custom], ignore_index=True)
print(f"\nЗагальна кількість записів після об’єднання: {len(df)}")

df['label'] = df['label'].astype(int)

X_train, X_test, y_train, y_test = train_test_split(
    df['text'], df['label'], test_size=0.2, random_state=42, stratify=df['label']
)

X_train, X_test, y_train, y_test = train_test_split(
    df['text'], df['label'], test_size=0.2, random_state=42, stratify=df['label']
)

nb_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=10000)),
    ('clf', MultinomialNB())
])
nb_pipeline.fit(X_train, y_train)
y_pred_nb = nb_pipeline.predict(X_test)
print("\n--- Naive Bayes ---")
print(classification_report(y_test, y_pred_nb))

lr_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=10000)),
    ('clf', LogisticRegression(max_iter=1000))
])
lr_pipeline.fit(X_train, y_train)
y_pred_lr = lr_pipeline.predict(X_test)
print("\n--- Logistic Regression ---")
```

```
print(classification_report(y_test, y_pred_lr))

svm_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=10000)),
    ('clf', SVC(probability=True))
])
svm_pipeline.fit(X_train, y_train)
y_pred_svm = svm_pipeline.predict(X_test)
print("\n--- Support Vector Machine ---")
print(classification_report(y_test, y_pred_svm))

joblib.dump(nb_pipeline, 'pkl/message_nb_model.pkl')
print("☑ Naive Bayes модель збережено як 'message_nb_model.pkl'")
joblib.dump(lr_pipeline, 'pkl/message_lr_model.pkl')
print("☑ Logistic Regression модель збережено як 'message_lr_model.pkl'")
joblib.dump(svm_pipeline, 'pkl/message_svm_model.pkl')
print("☑ Support Vector Machine модель збережено як 'message_svm_model.pkl'")
```

ДОДАТОК В

Код файлу url.py

```
from datasets import load_dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report
import joblib
import os

print("◇ Завантаження основного датасету URL...")
url_dataset = load_dataset("ealvaradob/phishing-dataset", "urls", trust_remote_code=True)
df_main = url_dataset['train'].to_pandas()

print("Перші 5 рядків основного датасету:")
print(df_main.head())

print("\n◇ Завантаження кастомного датасету з csv/custom_urls.csv...")
custom_csv_path = os.path.join(os.path.dirname(__file__), '..', 'csv', 'custom_urls.csv')
df_custom = pd.read_csv(custom_csv_path)

print("Перші 5 рядків кастомного датасету:")
print(df_custom.head())

df = pd.concat([df_main, df_custom], ignore_index=True)
print(f"\nЗагальна кількість записів після об’єднання: {len(df)}")

X = df['text']
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer(analyzer='char_wb', ngram_range=(3,5), max_features=5000)
X_train_vect = vectorizer.fit_transform(X_train)
X_test_vect = vectorizer.transform(X_test)

rf_model = RandomForestClassifier(random_state=42)
gb_model = GradientBoostingClassifier(random_state=42)

print("\n--- Random Forest ---")
rf_model.fit(X_train_vect, y_train)
print(classification_report(y_test, rf_model.predict(X_test_vect)))

print("\n--- Gradient Boosting ---")
gb_model.fit(X_train_vect, y_train)
print(classification_report(y_test, gb_model.predict(X_test_vect)))

joblib.dump(rf_model, 'pkl/url_rf_model.pkl')
print("☑ Random Forest модель збережена як 'url_rf_model.pkl'")

joblib.dump(gb_model, 'pkl/url_gb_model.pkl')
print("☑ Gradient Boosting модель збережена як 'url_gb_model.pkl'")

joblib.dump(vectorizer, 'pkl/url_vectorizer.pkl')
print("☑ Векторизатор збережено як 'url_vectorizer.pkl'")
```

ДОДАТОК Г

Код файлу train_kaggle_datasets.py

```
import os
import pandas as pd
import re
import joblib
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import kagglehub

print("◇ Починаємо завантаження і тренування Kaggle датасетів")

print("1) Завантаження Kaggle Phishing Email Dataset...")
email_path = kagglehub.dataset_download("naserabdullahalam/phishing-email-dataset")
print("Файли завантажено у:", email_path)

email_csv = os.path.join(email_path, "phishing_email.csv")
if not os.path.exists(email_csv):
    raise FileNotFoundError(f"Не знайдено файл phishing_email.csv у {email_path}")

df_email = pd.read_csv(email_csv)
df_email = df_email.rename(columns={'text_combined': 'text'})[['text', 'label']]
print("☑ Завантажено email дані, записів:", len(df_email))

print("\n2) Завантаження Kaggle Phishing URL Dataset...")
url_path = kagglehub.dataset_download("adityachaudhary1306/phishing-url-classifier-dataset-cleaned")
print("Файли завантажено у:", url_path)

url_csv = os.path.join(url_path, "dataset.csv")
if not os.path.exists(url_csv):
    raise FileNotFoundError(f"Не знайдено файл dataset.csv у {url_path}")

df_url = pd.read_csv(url_csv)
print("☑ Завантажено URL дані, записів:", len(df_url))

df_url['Result'] = df_url['Result'].replace({-1: 0}).astype(int)
X_url = df_url.drop(columns=['Result'])
y_url = df_url['Result']

print("\n3) Тренування recommendation model на email...")

df_phish = df_email[df_email['label'] == 1].copy()

def assign_category(text):
    t = str(text).lower()
    if re.search(r"(click|link|посилання|перейдіть|натисніть)", t):
        return "request_click_link"
    if re.search(r"(name|address|credit card|card number|номер картки|персональні дані)", t):
        return "request_personal_data"
    if re.search(r"(blocked|suspended|блоковано|заблоковано)", t):
        return "account_blocked"
    if re.search(r"(payment|invoice|оплата|рахунок|billing)", t):
        return "payment_alert"
```

```
    return "default"

df_phish['category'] = df_phish['text'].apply(assign_category)

X = df_phish['text']
y = df_phish['category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=5000)),
    ('clf', LogisticRegression(max_iter=1000))
])
pipeline.fit(X_train, y_train)

print("\n📊 Classification report (email):")
print(classification_report(y_test, pipeline.predict(X_test)))

os.makedirs('pkl', exist_ok=True)
joblib.dump(pipeline, 'pkl/recommendation_model_kaggle.pkl')
print("☑ recommendation_model_kaggle.pkl збережено")

print("\n4) Тренування URL моделей...")

X_train, X_test, y_train, y_test = train_test_split(X_url, y_url, test_size=0.2, random_state=42, stratify=y_url)

rf = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)
print("\n📊 RandomForest report:")
print(classification_report(y_test, rf.predict(X_test)))
joblib.dump(rf, 'pkl/url_rf_model_kaggle.pkl')
print("☑ url_rf_model_kaggle.pkl збережено")

gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train, y_train)
print("\n📊 GradientBoosting report:")
print(classification_report(y_test, gb.predict(X_test)))
joblib.dump(gb, 'pkl/url_gb_model_kaggle.pkl')
print("☑ url_gb_model_kaggle.pkl збережено")

print("\n🎯 Готово! Усі моделі успішно натреновані та збережені.")
```

ДОДАТОК Д

Код файлу train_recommendation_model.py

```
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report
import joblib
from datasets import load_dataset
import os

print("◇ Завантаження основного датасету повідомлень (SMS + Email)...")
dataset = load_dataset("ealvaradob/phishing-dataset", "texts", trust_remote_code=True)
df_main = dataset['train'].to_pandas()

print("◇ Завантаження кастомного датасету з csv/custom_messages.csv...")
custom_csv_path = os.path.join('csv', 'custom_messages.csv')
df_custom = pd.read_csv(custom_csv_path)

df = pd.concat([df_main, df_custom], ignore_index=True)
df['label'] = df['label'].astype(int)
print(f"Загальна кількість записів після об’єднання: {len(df)}")

def assign_category(text, label):
    text = str(text).lower()

    if label == 0:
        return "safe"

    if re.search(r"(click|link|посилання|перейдіть|натисніть|confirm)", text):
        return "request_click_link"
    elif re.search(r"(name|address|credit card|card number|номер картки|персональні дані)", text):
        return "request_personal_data"
    elif re.search(r"(blocked|suspended|блоковано|заблоковано|тимчасово заблоковано|увійдіть)", text):
        return "account_blocked"
    elif re.search(r"(card|credit|карт|рахунок|платіж|оплат|invoice|billing)", text):
        return "payment_alert"
    elif re.search(r"(номер карт|банківськ|особист|персональн|введіть дані|введіть номер)", text):
        return "request_personal_data"
    else:
        return "default"

df['category'] = df.apply(lambda row: assign_category(row['text'], row['label']), axis=1)

print("\n📊 Розподіл категорій повідомлень:")
print(df['category'].value_counts())

ANTI_TRIGGER_WORDS = ["click", "link", "посилання", "натисніть", "confirm", "blocked", "payment", "invoice",
"card", "credit"]

def apply_anti_trigger(row):
    text = str(row['text']).lower()
    cat = row['category']
    if cat != "safe":
```

```
        if not any(word in text for word in ANTI_TRIGGER_WORDS):
            return "default"  # зменшили вагу фішинговості
    return cat

df['category'] = df.apply(apply_anti_trigger, axis=1)

X = df['text']
y = df['category']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(analyzer='char_wb', ngram_range=(3,5), max_features=7000)),
    ('clf', OneVsRestClassifier(LogisticRegression(max_iter=1500)))
])

print("\n🚀 Навчання моделі...")
pipeline.fit(X_train, y_train)

print("\n📊 Classification report:")
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred, zero_division=0))

os.makedirs('pkl', exist_ok=True)
joblib.dump(pipeline, 'pkl/recommendation_model.pkl')
print("\n☑ Модель рекомендацій збережено як 'pkl/recommendation_model.pkl'")
```

ДОДАТОК Е

Код файлу index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8" />
   <title>Phishing Identification</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
   <link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@700;900&display=swap" rel="stylesheet">
</head>
<body>
<div class="container">
   <div class="header-flex">
      <img src="{{ url_for('static', filename='pictures/mess.gif') }}" alt="Повідомлення" class="gif-mess">
      <h1>Phishing Identification</h1>
      <img src="{{ url_for('static', filename='pictures/mess.gif') }}" alt="Посилання" class="gif-mess">
   </div>

   <div class="cards">
      <div class="card" id="messageCard">
         <h2>Перевірка повідомлення 📧</h2>
         <textarea id="message" maxlength="1000" placeholder="Введіть текст повідомлення"
oninput="autoResize(this)"></textarea>
         <button onclick="checkMessage()">Перевірити повідомлення</button>
         <p id="messageResult"></p>
         <div class="recommendation">
            <h3>Рекомендації</h3>
            <p id="messageRecommendation"></p>
         </div>
         <textarea id="messageFeedback" placeholder="Ваш відгук про рекомендацію"></textarea>
         <button onclick="sendFeedback('message')">Надіслати відгук</button>
         <p id="messageFeedbackStatus"></p>
      </div>

      <div class="card" id="urlCard">
         <h2>Перевірка посилання 🔗</h2>
         <input type="text" id="url" placeholder="Введіть URL-посилання">
         <button onclick="checkURL()">Перевірити посилання</button>
         <p id="urlResult"></p>
         <div class="recommendation">
            <h3>Рекомендації</h3>
            <p id="urlRecommendation"></p>
         </div>
         <textarea id="urlFeedback" placeholder="Ваш відгук про рекомендацію"></textarea>
         <button onclick="sendFeedback('url')">Надіслати відгук</button>
         <p id="urlFeedbackStatus"></p>
      </div>
   </div>
</div>

<script>
const recommendations = {
   "request_click_link": "⚠ Не переходьте за посиланням! Це може бути шахрайство.",
   "account_blocked": "⚠ Ваш акаунт можуть використовувати шахраї. Не повідомляйте свої дані та перевірте на
офіційному сайті.",
   "payment_alert": "⚠ Не здійснюйте оплату, доки не переконаєтесь у достовірності повідомлення.",
```

```
    "default": "⚠ Небезпечно довіряти цьому повідомленню!"
};

async function checkMessage() {
    const message = document.getElementById("message").value.trim();
    const card = document.getElementById("messageCard");
    const resultEl = document.getElementById("messageResult");
    const recEl = document.getElementById("messageRecommendation");
    card.classList.remove("safe", "phishing", "warning");
    resultEl.classList.remove("warning");
    recEl.innerText = "";
    if (message === "") {
        resultEl.innerText = "⚠ Введіть текст повідомлення для перевірки";
        resultEl.classList.add("warning");
        card.classList.add("warning");
        return;
    }
    const res = await fetch("/check_message", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ message })
    });
    const data = await res.json();
    resultEl.innerText = data.result;
    if (data.result.toLowerCase().includes("безпеч")) {
        card.classList.add("safe");
        recEl.innerText = "☑ Можна довіряти повідомленню.";
    } else if (data.result.toLowerCase().includes("фішинг")) {
        card.classList.add("phishing");
        recEl.innerText = recommendations[data.category] || recommendations["default"];
    } else {
        card.classList.add("warning");
        resultEl.classList.add("warning");
    }
}

async function checkURL() {
    const url = document.getElementById("url").value.trim();
    const card = document.getElementById("urlCard");
    const resultEl = document.getElementById("urlResult");
    const recEl = document.getElementById("urlRecommendation");
    card.classList.remove("safe", "phishing", "warning");
    resultEl.classList.remove("warning");
    recEl.innerText = "";
    if (url === "") {
        resultEl.innerText = "⚠ Введіть URL-посилання для перевірки";
        resultEl.classList.add("warning");
        card.classList.add("warning");
        return;
    }
    const res = await fetch("/check_url", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ url })
    });
    const data = await res.json();
    resultEl.innerText = data.result;
    if (data.result.toLowerCase().includes("безпеч")) {
        card.classList.add("safe");
        recEl.innerText = "☑ Можна переходити за посиланням.";
```

```
    } else if (data.result.toLowerCase().includes("фішинг")) {
        card.classList.add("phishing");
        recEl.innerText = "⚠ Небезпечно переходити за цим посиланням!";
    } else {
        card.classList.add("warning");
        resultEl.classList.add("warning");
    }
}

async function sendFeedback(type) {
    const content = type === 'message'
        ? document.getElementById('message').value
        : document.getElementById('url').value;
    const feedbackText = type === 'message'
        ? document.getElementById('messageFeedback').value
        : document.getElementById('urlFeedback').value;
    const resultText = type === 'message'
        ? document.getElementById('messageResult').innerText
        : document.getElementById('urlResult').innerText;
    const statusEl = type === 'message'
        ? document.getElementById('messageFeedbackStatus')
        : document.getElementById('urlFeedbackStatus');
    if (!feedbackText.trim()) {
        statusEl.innerText = "⚠ Напишіть свій відгук.";
        return;
    }
    try {
        const res = await fetch("/feedback", {
            method: "POST",
            headers: {"Content-Type": "application/json"},
            body: JSON.stringify({
                type,
                content,
                system_result: resultText,
                user_opinion: feedbackText
            })
        });
        const data = await res.json();
        statusEl.innerText = data.message;
        if (type === 'message') document.getElementById('messageFeedback').value = "";
        else document.getElementById('urlFeedback').value = "";
    } catch (err) {
        console.error(err);
        statusEl.innerText = "❌ Помилка надсилання відгуку";
    }
}

function autoResize(textarea) {
    textarea.style.height = 'auto';
    textarea.style.height = textarea.scrollHeight + 'px';
}

window.addEventListener('DOMContentLoaded', () => {
    const textarea = document.getElementById('message');
    if (textarea.value.length > 0) autoResize(textarea);
});
</script>
</body>
</html>
```

ДОДАТОК Є

Код файлу style.css

```
*, *::before, *::after { box-sizing: border-box; }
body {
    font-family: 'Roboto', sans-serif;
    background: linear-gradient(135deg, #bef7ff, #c8eeff, #b9d8ff);
    background-attachment: fixed;
    margin: 0;
    padding: 0;
}

.container {
    width: 90%;
    max-width: 1300px;
    margin: 50px auto;
}

.header-flex {
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 20px;
    margin-bottom: 50px;
}

.gif-mess {
    width: 80px;
    height: auto;
}

h1 {
    font-family: 'Orbitron', sans-serif;
    font-size: 36px;
    font-weight: 700;
    color: #000597;
    text-shadow: -1px -1px 0 rgb(209, 209, 209), 1px -1px 0 white, -1px 1px 0 rgb(209, 209, 209), 1px 1px 0 white;
}

.cards {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 40px;
}

.card {
    background: rgba(255, 255, 255, 0.9);
    border-radius: 20px;
    padding: 30px;
    width: 600px;
    box-shadow: 0 10px 25px rgba(0,0,0,0.08);
    transition: transform 0.3s, box-shadow 0.3s;
}
.card.safe { background: #d0f0da; }
.card.phishing { background: #f8d2d2; }
.card.warning { background: #fff9c4; }
```

```
textarea, input[type="text"] {
    width: 100%;
    padding: 12px;
    font-size: 15px;
    margin: 10px 0;
    border-radius: 10px;
    border: 1px solid #ccc;
}

button {
    width: 100%;
    padding: 12px;
    font-size: 16px;
    background: #1565c0;
    color: #fff;
    border: none;
    border-radius: 10px;
    cursor: pointer;
}
button:hover { background: #0d47a1; }

.recommendation {
    margin-top: 15px;
    background: #f5f5f5;
    padding: 10px;
    border-radius: 10px;
}

.feedback-box {
    margin-top: 15px;
    background: #eef3ff;
    padding: 10px;
    border-radius: 12px;
    border: 1px solid #b5c7ff;
}
.feedback-box h4 {
    text-align: center;
    color: #1a3eb1;
    margin: 5px 0 10px 0;
}

.feedback-box input {
    width: 100%;
    padding: 10px;
    border-radius: 8px;
    border: 1px solid #b5c7ff;
}
.feedback-box button {
    background: #0044cc;
    margin-top: 8px;
}

.feedback-box button:hover {
    background: #002b80;
}
.feedback-box p {
    text-align: center;
    font-size: 13px;
    color: #444;
    margin-top: 5px;
```