НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»

УДК 004.91

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Комбінований метод та програмне забезпечення для ідентифікації фішингового вмісту в текстовій частині електронних листів»

Виконав:
студент ІІ курсу, групи КП-31мн
Фещенко Єгор Олександрович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПСКС, к.т.н., доцент,
Тарасенко-Клятченко Оксана Володимирівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Фещенку Єгору Олександровичу

1. Тема дисертації «Комбінований метод та програмне забезпечення для ідентифікації фішингового вмісту в текстовій частині електронних листів», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., затверджені наказом по університету № 1219-С від «21» березня 2025 р.

2. Термін подання студентом дисертації «19» травня 2025 р.

3. Об'єкт дослідження: процес автоматизованої класифікації текстових даних

4. Предмет дослідження: методи, способи та програмні засоби автоматизованого виявлення фішингового вмісту в текстовій частині електронних листів.

5. Перелік завдань, які потрібно розробити:
- ознайомитися з науковими роботами, у області ідентифікації фішингу в електронних листах;
- провести порівняльний аналіз розглянутих підходів, методі та алгоритмів ідентифікації фішингового вмісту в електронних листах;
- визначити напрям дослідження базуючись на раніше набутих знаннях;
- сформувати власний метод виявлення фішингового вмісту в текстовій частині електронних листів;
- визначити перелік технологій та інструментів розробки програмної реалізації запропонованого методу ідентифікації фішингового вмісту;
- розробити програмну реалізацію запропонованого методу ідентифікації фішингового вмісту в текстовій частині електронних листів;
- провести тестування розробленого програмного забезпечення;
- провести порівняльні випробування запропонованого методу та аналіз отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
- схема алгоритму роботи запропонованого методу виявлення фішингового вмісту в текстовій частині електронних листів;
- схема архітектури розробленого програмного забезпечення;
- UML діаграма сценаріїв використання програмного забезпечення;
- гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на тренувальному датасеті;
- гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на валідаційному датасеті;
- гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на тестувальному датасеті.

7. Орієнтовний перелік публікацій:
- тези доповіді «Комбінований метод автоматизованого виявлення фішингових електронних листів»;
- публікація статті у фаховому журналі за спеціальністю 121 «Інженерія програмного забезпечення».

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «10» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.12.2023	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури	13.03.2024	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.09.2024	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.10.2024	
5.	Проведення наукового дослідження; підготовка матеріалів доповіді на конференції ПМК-2024	15.10.2024	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації; робота над статтею за результатами дослідження;	10.03.2025	
7.	Завершення роботи над основною частиною магістерської дисертації;	20.04.2025	
8.	Оформлення текстової та графічної частини магістерської дисертації	30.04.2025	

Студент Єгор ФЕЩЕНКО

Науковий керівник Тетяна ЗАБОЛОТНЯ

РЕФЕРАТ

Актуальність теми. Автоматизоване виявлення фішингового вмісту в електронних листах є актуальною та важливою задачею в галузі інформаційної безпеки, оскільки фішинг залишається однією з найпоширеніших кіберзагроз, яка призводить до втрат конфіденційних даних і значних фінансових збитків як окремих користувачів, так і великих організацій. Традиційні методи, що базуються на евристичних правилах та статистичних моделях, демонструють недостатню точність у розпізнаванні складних та семантично варіативних текстів. Водночас, сучасні глибокі нейронні мережі, зокрема трансформерні моделі, хоча й забезпечують високу точність, але потребують значних обчислювальних ресурсів, що ускладнює їх застосування в реальних умовах. Тому актуальним є створення ефективного, але з меншими вимогами до обчислювальних ресурсів методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електроних листів.

Об’єктом дослідження є процес автоматизованої класифікації текстових даних.

Предметом дослідження є методи, способи та програмні засоби автоматизованого виявлення фішингового вмісту в текстовій частині електронних листів.

Метою дослідження є підвищення ефективності виявлення фішингу в електронних листах за рахунок розроблення та реалізації комбінованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів.

Наукова новизна. Уперше розроблено комбінований метод автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів, характерними особливостями якого є попереднє оброблення природномовних текстових даних, застосування згорткової нейронної мережі для вилучення локальних текстових ознак, двонаправленої

рекурентної нейронної мережі для врахування семантичного контексту та механізму уваги для динамічного вибору найбільш значущих ознак тексту, що дозволило збільшити абсолютну точність виявлення фішингу у межах від 95.93% до 99.6% на різних наборах даних.

Практична значущість. Запропонований комбінований метод автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів може використовуватися як самостійний інструмент аналізу текстового контенту електронної пошти, так і бути інтегрованим у складніші системи інформаційної безпеки. Розроблене програмне забезпечення реалізує метод автоматизованого виявлення фішингового вмісту у текстах електронних листів, застосовуючи попередньо навчені ембедінги GloVe та гібридну нейромережеву модель, яка складається з згорткових шарів, двонаправлених рекурентних шарів та шару уваги. Це дозволяє досягти високої точності класифікації з помірними вимогами до обчислювальних ресурсів, що забезпечує практичну інтеграцію системи для корпоративних та державних організацій з великим обсягом електронної пошти.

Апробація роботи. Основні положення та результати виконаної роботи були представлені та обговорені на XVII науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2024 (20-22 листопада 2024 р., м. Київ).

Стаття «Метод автоматизованого виявлення фішингу в електронних листах на основі гібридної нейромережевої архітектури» в фаховому журналі «Наукові праці ВНТУ №2 (2025)» (подано в редакцію).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі представлено загальну характеристику роботи, проведено аналіз актуальності проблеми виявлення фішингового вмісту в електронних листах та обґрунтовано необхідність створення нового методу її розв’язання.

У першому розділі виконано логіко-структурний аналіз задачі, огляд сучасних методів та моделей автоматичної класифікації текстових даних, зокрема фішингових листів, сформульовано наукову задачу підвищення ефективності класифікації шляхом використання гібридної нейромережевої архітектури.

У другому розділі запропоновано концепцію гібридного методу класифікації, наведено детальний опис архітектури нейронної мережі, що включає згорткову нейромережу, рекурентну нейромережу та шар уваги, а також описано процедури попередньої обробки та векторизації текстових даних.

У третьому розділі розглянуто особливості технологічної реалізації запропонованого методу, наведено опис програмного рішення, структуру директорій проєкту, архітектуру окремих модулів та компонентів системи, їх функціональне призначення, а також сценарії використання консольного інтерфейсу.

У четвертому розділі наведено результати експериментального дослідження ефективності запропонованого методу на кількох незалежних наборах даних, виконано порівняння з іншими методами машинного навчання, проведено аналіз впливу розмірності ембедингів та шару уваги на точність класифікації, запропоновано подальші напрями вдосконалення методу.

У висновках узагальнено результати аналізу ефективності запропонованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів, сформульовано рекомендації щодо практичного застосування розробленого методу.

Робота виконана на 132 аркушах, містить 3 додатки та послання на список використаних літературних джерел з 54 найменувань. У роботі наведено 27 рисунків та 3 таблиць.

Ключові слова: інженерія програмного забезпечення, фішинг, електронні листи, гібридна архітектура, згорткові нейронні мережі,

рекурентні нейронні мережі, шар уваги, обробка природномовних текстових даних

ABSTRACT

Theme urgency. Automated detection of phishing content in emails remains a highly relevant and critical task in information security, as phishing is one of the most prevalent cyber threats causing significant data breaches and financial losses for both individuals and large organizations. Traditional methods, based on heuristic rules and statistical models, demonstrate insufficient accuracy in recognizing complex and semantically diverse texts. Modern deep neural networks, particularly transformer-based models, provide high accuracy but require substantial computational resources, complicating their real-world deployment. Therefore, developing an efficient method with reduced computational demands for automatically identifying phishing content in email texts is highly relevant.

The object of research is the process of automated classification of textual data..

The subject of research is comprises methods, approaches, and software tools for automated detection of phishing content in the textual part of emails.

The research objective is to increase the efficiency of phishing detection in emails by developing and implementing a combined method of automated identification of phishing content in the content body of emails.

Scientific novelty. For the first time, a combined method for automated identification of phishing content in the text part of emails has been developed, the characteristic features of which are pre-processing of natural language text data, the use of a convolutional neural network to extract local text features, a bidirectional recurrent neural network to take into account the semantic context, and an attention mechanism for dynamically selecting the most significant text features, which allowed increasing the absolute accuracy of phishing detection in the range from 95.93% to 99.6% on different data sets.

Practical value. The proposed hybrid neural network method for automated phishing detection in email texts can serve both as a standalone tool for analyzing email content and as part of more sophisticated information security systems. The

developed software implements an automated phishing detection method for email texts using pretrained GloVe embeddings and a hybrid neural network model comprising convolutional layers, bidirectional recurrent layers, and an attention layer. This enables achieving high classification accuracy with moderate computational resource requirements, ensuring practical integration into corporate and governmental environments managing large volumes of email traffic.

Approbation. The main principles and results of the study were presented and discussed at the 17th Scientific Conference for students and postgraduates “Applied Mathematics and Computing” PMK-2024 (November 20-22, 2024, Kyiv).

Article "A method for automated detection of phishing in emails based on a hybrid neural network architecture" in the professional journal "Scientific Proceedings of VNTU No. 2 (2025)" (submitted to the editorial office).

Structure and context of the thesis. The master thesis consists of the introduction, four chapters, conclusions and appendixes.

The introduction provides a general overview of the thesis, analyzes the relevance of the phishing detection problem in emails, and justifies the need for developing a novel method for its resolution.

In the first section includes a logical-structural analysis of the problem, a review of existing methods and models for automatic text classification, particularly for phishing emails, and formulates the scientific challenge of improving classification efficiency through a hybrid neural network architecture.

The second section proposes the concept of a hybrid classification method, detailing the neural network architecture consisting of convolutional networks, recurrent networks, and an attention layer, alongside procedures for text data preprocessing and vectorization.

The third section discusses technological implementation details of the proposed method, describes the software solution, project directory structure, architecture of individual modules and system components, their functional purpose, and command-line interface usage scenarios.

The fourth section presents the results of an experimental study evaluating the efficiency of the proposed hybrid neural network model across several independent datasets, compares its performance with other machine learning methods, analyzes the impact of embedding dimensionality and the attention layer on classification accuracy, and suggests further directions for method improvement.

The conclusions summarize the analysis results of the effectiveness of the proposed method for automated phishing content identification in email texts and provide practical recommendations for applying the developed approach.

The work is completed on 132 sheets and contains 3 appendices and references to the list of used literary sources of 54 titles. The work contains 27 figures and 3 tables.

Keywords: software engineering, phishing, emails, hybrid architecture, convolutional neural networks, recurrent neural networks, attention layer, natural language processing.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

CNN – Convolutional Neural Network – згорткова нейронна мережа, клас нейронних мереж глибокого навчання, які використовують операцію згортки для виявлення локальних патернів та ознак у текстових або зображувальних даних.

RNN – Recurrent Neural Network – рекурентна нейронна мережа, клас нейронних мереж, що дозволяє ефективно працювати з послідовностями даних, зберігаючи інформацію про попередні елементи послідовності.

LSTM – Long Short-Term Memory – спеціальний тип рекурентних нейронних мереж, здатних запам'ятовувати довготривалі залежності у послідовних даних, що робить їх ефективними для обробки природної мови.

NLP – Natural Language Processing – напрямок штучного інтелекту та обчислювальної лінгвістики, що займається розробкою методів аналізу, розуміння та генерації тексту людською мовою.

CLI – Command Line Interface – тип інтерфейсу для взаємодії користувача з комп'ютером шляхом введення текстових команд в консольному середовищі.

Гіперпараметри – параметри, що встановлюються вручну перед початком навчання нейронної мережі або іншої моделі машинного навчання (наприклад, кількість епох, швидкість навчання, розмір ембедінгів), які впливають на процес навчання та ефективність моделі.

Ембедінги – Embeddings – векторні представлення слів або інших елементів тексту, які відображають семантичні та синтаксичні властивості мовних одиниць, застосовуються для перетворення текстових даних в числовий формат, що сприймається нейронними мережами.

Датасет – набір структурованих даних, що використовується для тренування та оцінювання моделей машинного навчання, таких як нейронні мережі або класичні методи класифікації.

Фреймворк – програмний набір інстурментів або середовище розробки, що містить набір бібліотек та компонентів для створення програмних рішень, у тому числі нейромережевих моделей.

Векторизація – процес перетворення текстових даних у числовий формат, що може бути використаний алгоритмами машинного навчання.

Валідація – процес оцінювання моделі машинного навчання на спеціально виділеному наборі даних, який не використовувався під час навчання, з метою перевірки узагальнювальної здатності моделі.

Токенізація – процес розбиття тексту на окремі одиниці (токени), які можуть бути словами, підсловами або символами, залежно від завдання обробки тексту.

Лематизація – процес приведення слова до його базової форми (леми) з метою нормалізації тексту для подальшого аналізу.

Корпус текстів – ретельно відібрана та підготовлена сукупність текстів, яка використовується для проведення лінгвістичних та комп'ютерних досліджень мови, а також для тренування моделей NLP.

ВСТУП

У сучасному світі інформація є одним із найцінніших ресурсів, а електронні листи стали невід'ємною складовою щоденної діяльності як приватних осіб, так і бізнес-структур. Однак поряд із численними перевагами електронної комунікації існує значна проблема – фішингові атаки, які становлять серйозну загрозу інформаційній безпеці. Такі атаки ґрунтуються на психологічних маніпуляціях та методах соціальної інженерії з метою отримання конфіденційних даних користувачів: логінів, паролів, платіжних реквізитів тощо.

Попри значні зусилля з боку фахівців із кібербезпеки, кількість фішингових атак щороку зростає. Це пояснюється високою ефективністю та низькими витратами на проведення таких атак з боку зловмисників. Фішингові електронні листи можуть бути спрямовані як на широке коло користувачів, так і на окремі групи чи конкретних осіб, імітуючи повідомлення від надійних та авторитетних джерел. При цьому, навіть досвідченим користувачам іноді важко відрізнити легітимні повідомлення від шахрайських.

Саме тому надзвичайно важливим завданням стає створення автоматизованих систем ідентифікації фішингового вмісту, які б ефективно аналізували великі обсяги електронних повідомлень та оперативно виявляли підозрілий контент. Однак автоматизація цього процесу є складною задачею, оскільки вимагає високої точності класифікації текстових даних, здатності системи враховувати контекст та приховані ознаки шахрайських повідомлень.

Існує велика кількість підходів до автоматизованої ідентифікації фішингових листів. Класичні методи машинного навчання базуються на статистичних особливостях тексту, таких як частота слів, наявність ключових виразів або стилістичні ознаки повідомлення. З іншого боку, сучасні нейромережеві підходи та, зокрема, трансформерні моделі, використовують глибокий контекстуальний аналіз та складні семантичні залежності між

словами в тексті. Проте кожен з цих підходів має свої переваги та недоліки. Так, класичні методи вирізняються швидкістю та простотою, але поступаються в точності сучасним нейромережевим рішенням. Трансформерні моделі, навпаки, демонструють найкращу точність, але вимагають значних ресурсів для навчання та експлуатації.

Враховуючи вищесказане, одним із перспективних шляхів вирішення цієї проблеми є створення гібридних нейромережевих моделей, які поєднують переваги різних методів та архітектур, забезпечуючи високу точність та помірні вимоги до обчислювальних ресурсів. Подібні гібридні підходи використовують переваги згорткових нейронних мереж у виявленні локальних текстових ознак та рекурентних нейронних мереж для аналізу глобальних залежностей у тексті, а також механізми уваги для визначення найбільш інформативних частин повідомлення.

Дослідження, що проводиться в межах цієї магістерської дисертації, присвячене розробці нового комбінованого методу автоматизованого виявлення фішингових електронних листів. Запропонований метод використовує комбінацію згорткових нейронних мереж, двонаправлених рекурентних нейронних мереж та механізму уваги, що дозволяє ефективно аналізувати текстову частину електронних листів з метою точного визначення фішингового вмісту. Розроблений метод дозволяє зберегти баланс між високою точністю класифікації та прийнятними обчислювальними витратами, що робить його перспективним для застосування у реальних системах протидії фішингу.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ МЕТОДІВ ІДЕНТИФІКАЦІЇ ФІШИНГУ

1.1. Соціальна інженерія

Перші схеми соціальної інженерії з'явилися у телефонному середовищі 1970-х років, а вже у 1990-х вони перекочували в електронну пошту. У 1996 р. компанія «America On-Line» (AOL) уперше офіційно задокументувала масову розсилку підроблених листів «Verify your password», що запропонували користувачам підтвердити облікові дані [1]. Станом на 2024 р. фішинг був склав 36% успішних інцидентів порушення конфіденційності та 54% усіх соціотехнічних атак [2].

Класифікуємо найвідоміші техніки соціальної інженерії:

1. *Pretexting* (попередня легенда) – створення фальшивої історії з метою крадіжки даних.
2. *Baiting* (наживка) – пропозиція безкоштовного «призу» за виконання дії.
3. *Quid pro quo* – обіцянка допомоги в обмін на інформацію.
4. *Tailgating / piggybacking* – фізичний супровід у захищені зони.
5. *Phishing, smishing, vishing, QR-phishing* – цифрові атаки за допомогою сучасних засобів комунікації, таких як електронна пошта та соціальні мережі, з метою отримання конфіденційних данних.

Фішингові повідомлення зазвичай експлуатують базові емоції, наведемо їх перелік додаючи приклад тексту зловмисного повідомлення: страх («Ваш рахунок буде заблоковано»), терміновість(«Підтвердьте протягом 12 годин»), авторитет («Лист від генерального директора»), жадібність («Ви виграли приз…»). В рамках даного дослідження, буде описано ефективність виявлення таких коротких лінгвістичних шаблонів за допомогою конволюційних шарів нейронної мережі.

За даними «Internet Crime Complaint Center» у 2024 р. сукупні втрати від шахрайств в Інтернеті сягнули 16,6 мільярдів долларів, а категорія «Business E-mail Compromise» з 2013 р. накопичила понад 55 мілярдів долларів [3]. «CERT-UA» у 2024 р. зафіксувала 1465 інцидентів фішингу проти державного сектору [4]. Відповідно до «Regulation (EU) 2019/881» (NIS-2) та «General Data Protection Regulation» (GDPR) організації зобов'язані впроваджувати засоби превентивної фільтрації електронної пошти й звітувати про витоки конфіденційних даних, що підкреслює актуальність ефективних рішень протидії зловмисним фішинговим атакам.

1.2. Поняття фішингу

Згідно з ISO/IEC 27035 фішинг – це соціотехнічна атака, що використовує оманливі повідомлення з метою змусити одержувача виконати небажану дію або розкрити конфіденційні дані [5].

Виходячи з офіційних документів, дослідження пропонують деталізувати фішинг за ступенем персоналізації. Масові кампанії (bulk phishing) поширюють однаковий шаблон тисячам адресатів, тоді як spear phishing орієнтується на конкретну людину чи відділ, використовуючи персональні дані для підвищення довіри. Коли ж цілями стають топ-менеджери, говорять про whaling-атаки. Окремо вирізняють clone phishing, у якому зловмисник копіює раніше легітимний лист і підмінює посилання, та angler/QR-phishing, що експлуатують соцмережі або QR-коди. Попри різну глибину персоналізації, всі ці різновиди залишаються орієнтованими на зміст тексту повідомлення, тож виявлення маніпулятивних формулювань залишається актуальним завданням.

Отже, опишемо найпопулярніші типи фішингових електронних листів:

- Bulk phishing – масові розсилки низької персоналізації.
- Spear phishing – цільові листи під конкретну особу чи відділ.
- Whaling – атаки на топ-менеджмент компаній.

- Clone phishing – копіювання безпечного попереднього листа з підміною гіперпосилань.
- Angler / QR-phishing – зловживання соцмережами або QR-кодами у підписах.

Спільною рисою перелічених різновидів є трикомпонентна будова електронного листа: заголовок, тема і тіло. У більшості корпоративних системах перевірка SPF/DKIM-підписів, де (DKIM) – DomainKeys Identified Mail, аутентифікація домену відправника за допомогою згенерованого цифрового підпису, або аномалій у заголовках виконується мережевими засобами до потрапляння повідомлення у поштову скриньку. Натомість основна кількість лінгвістичних індикаторів, таких як емоційно забарвлені слова, термінові заклики й шахрайські URL, зосереджена саме в тілі листа. За оцінкою «Proofpoint», аналіз body-only дозволяє блокувати до 92% сучасних фішингових шаблонів без обробки заголовків [6]. Отже, зосередження на тексті тіла електронного листа не лише спрощує інтеграцію методу в існуючі системи фільтрації, а й мінімізує витрати обчислювальних ресурсів.

Практичну цінність текстоорієнтованого підходу ілюструє кампанія «Office 365 Invoice Overdue» (2023 р.), що маскували зловмисні наміри під повідомлення про прострочений рахунок. Лист містив фразу «Your account will be suspended in 24 hours» і посилання на домен «office365-secure-validate.com», зареєстрований напередодні. Наявність слова «*invoice*» у поєднанні з терміновим ультиматумом створювала потужний психологічний тиск, а нетипова доменна зона – технічну ознаку. Подібний принцип спостерігався і в україномовній атаці «ПриватБанк – підтвердьте дані» (2024 р.): ключова фраза «*підтвердьте*» супроводжувалася URL зі штучно ускладненим кодуванням. Обидва приклади демонструють, як локальні текстові шаблони й контекстні конструкції є головним компонентом зловмисних атак.

1.3. Аналіз підходів до автоматизованої ідентифікації фішингу

Фішинг у електронних листах характеризується високою варіативністю лексичних і технічних ознак, що ускладнює його блокування статичними правилами. Еволюція засобів протидії відбувалася поетапно: кожна нова стратегія усувала недоліки попередньої, але водночас створювала нові виклики, пов'язані з масштабуванням і обчислювальними витратами. У сучасній літературі підходи до автоматизованої ідентифікації фішингу класифікують наступним чином [1]:

1. Методи на основі правил та сигнатурних фільтрів (Rule-based, Signature-based).
2. Класичні методи машинного навчання.
3. Нейронні мережі (CNN, RNN, їх гібриди).
4. Трансформери та великі мовні моделі (BERT-подібні, GPT-подібні).

Проведемо аналіз кожного класу, описавши загальні принципи роботи та технічні переваги й обмеження, а також фактичні результати застосування до завдання виявлення фішингу в електронних листах.

Розглянемо принцип роботи методів на основі правил та сигнатурних фільтрів. Ці методи використовують наперед задані умови: чорні списки доменів, часто вживані зловмисні вирази, евристичні бали за наявність «підозрілих» слів чи HTML-тегів. Сигнатурні механізми доповнюються статистичними порогами (кількість URL, місце розташування JavaScript тощо).

Переваги очевидні:

- нульова або близька до нуля затримка на обробку листа;
- повна прозорість процесу тому, що адміністратор бачить, яке правило спрацювало;
- невибагливість до апаратних ресурсів.

Головний недолік в контексті фішингу – це низька адаптивність через присутню необхідність вручну додавати нові сигнатури. Було підраховано, що середня «життєздатність» правила становить близько 30 днів [2].

На змішаному наборі даних з датасетів «SpamAssassin» та «Nazario» (50000 нефішингових та 6000 фіщингових електронних листів) сигнатурний фільтр дав тільки 88% точності при хибно-позитивній частці близько 2,5% [7]. Після ручного додавання 217 нових правил F1-міра зросла лише на 3%, що підтверджує обмеження таких підходів.

Класичні алгоритми машинного навчання лишаються популярними в задачах фільтрації електронної пошти завдяки помірній складності й прозорим механізмам ухвалення рішень. На відміну від глибокого навчання, їм потрібна явна векторизація тексту – здебільшого використовуються Bag-of-Words чи TF-IDF. Проаналізуємо п’ять найбільш вживаних методів цього класу.

Розглянемо наївний баєсів фільтр. Метод ґрунтується на теоремі Баєса з припущенням умовної незалежності ознак. Для векторів слів $\omega_1, ..., \omega_n$ обчислюється ймовірність $P\left(w_1, ..., w_n \right)$ і береться клас з максимумом.

Переваги методу:

- Навчання в один прохід; параметри – просто оцінки частот слів.
- Мінімальні обчислення при злитті.
- Стійкість до “проріджених” даних – навіть 100 прикладів достатньо для його роботи.

Недоліки:

- Головне припущення методу, а саме незалежність слів, очевидно порушується в природній мові.
- Схильність переоцінювати інформативність рідкісних токенів.
- Легко обходиться обфускацією (заміною латинскьих літер у власних назвах на літери інших алфавітів).

На змішаному наборі даних з датасетів «SpamAssassin» та «Nazario» наївний Баєс показав точність 90,3% та F1-міру 0,90 при хибно-позитивній частці (FPR) близько 2,5% [7]. Після додавання stop-list для URL-токенів FPR

вдалося знизити нижче 2%, але відгук моделі (recall) зменшився, що свідчить про невирішеність в балансі між чутливістю й специфічністю.

Іншим популярним методом машинного навчання є логістична регресія. Принцип її роботи полягає в тому, що ймовірність класу моделюється сигмоїдою від лінійної комбінації ознак: $P(C = 1 \mid x) = \sigma(w^T x + b)$. Ваги w знаходять градієнтним спуском із L2-регуляризацією.

Переваги методи:

- Лінійна модель – легка інтерпретація через ваги ознак.
- Гарно масштабується на мільйоних наборах даних.
- Стабільна для високовимірних TF-IDF-векторів.

Недоліками є:

- Нездатність моделювати нелінійні рішення так як існує проблема збігу токенів у різних позиціях.
- Сильно залежить від якісної нормалізації: за відсутності лематизації, і ваги дробляться між формами («*verify/verified/verifying*»).
- При класовому дисбалансі вимагає явного балансування ваг.

При аналізі ефективності тренування проходило на датасеті «Enron–HamCamp» (35 тис. нешкідливих електронних листів проти 4,1 тис. фішингових) і було зафіксовано: точність 0,94, відгук (recall) 0,92, F1-міра 0,93 [7]. Проте після внесення 10% нових фішингових прикладів із синонімічними шаблонами (заміна *verify* на *confirm*, *urgent* на *immediate*) F1-міра зменшилась до 0,86, що підкреслює відсутність семантичної узагальнюваності.

Метод опорних векторів, який часто використовується для завдань класифікації, шукає гіперплощину з максимальним зазором між класами. Для розріджених TF-IDF векторів найчастіше використовують лінійне ядро, що спрощує оптимізацію до задачі квадратичного програмування.

Серед перваг даного підходу виділяють:

- Теоретично гарантований максимум зазору надає хорошу узагальнюваність.
- Відсутність чутливості до співрозмірності ознак за умови нормалізації довжин векторів.
- Може працювати з сотнями тисяч особливостей без глибокого оверфітингу (перенавчання на одному наборі даних).

Недоліками є:

- Навчання зі складністю $O(N^2)$, що є повільним на великих колекціях.
- Параметр C (пом’якшення) потребує ретельного підбору, інакше відгук (recall) падає.
- Інтерпретованість обмежена: ваги розподіляються на всі ознаки, складно виділити ключові.

Під час аналізу ефективності методу використали SVM з лінійним ядром на корпусі «PhishTank–HamCamp» і отримали F1-міру 0,95 при хибно позитивному співвідношені 1,8% [8]. Подальше удосконалення первинних налаштувань методу допомогло незначно підвищити відгук (recall), ціною 40% приросту у використанні оперативної пам’яті.

Розглянемо алгоритми ансамблевих дерев. Почнемо з Random Forest (RF), принцип якого складається у створенні колекції із k випадково обраних дерев рішень; фінальний клас – за голосуванням.

Перевагами методу є:

- Менш схильний до перенавчання, ніж окреме дерево.
- Вбудований механізм оцінки важливості ознак.

Недоліками є:

- Час виконання задачи лінійно зростає з кількістю дерев.
- Погано працює з дуже розрідженими даними без попередньої фільтрації ознак.

На наборі «Nazario + HamCamp» (50 тис. листів) RF із 500 дерев виявив F1-міру з результатом 0,94, проте спожив 1,3 ГБ оперативної пам’яті, що є високим показником відносно інших методів [7].

Іншим представником ансамблевих дерев є метод XGBoost. Градієнтний бустинг над деревами: кожне наступне дерево фокусується на помилках попередніх, мінімізуючи логістичну втрату.

Серед переваг виділяють:

- Висока точність завдяки послідовному донавчанню.
- Підтримка регуляризації та рання зупинка.

Недоліки:

- Великі використання пам’яті в середньому на «лист» дерева.
- Складність тюнінгу (learning_rate, depth, subsample).

Під час дослідження XGBoost на 10 000 ознаках перевершив SVM з наступними результатами: F1-міра з значенням 0,96, хибно позитивне співвідношення 1,1%, але потребував 4,2 ГБ оперативної пам’яті та 0,06 с на «лист» дерева, що є втричі довшим за SVM [7].

Проаналізуємо метод k-Nearest Neighbours (k-NN). Клас нового листа визначають за більшістю серед k найближчих в евклідовому TF-IDF-просторі.

Перевагами методу є:

- Нульовий час тренування – усі дані зберігають “як є”.
- Добре відслідковує локальні кластери схожих листів.

Недоліки:

- Складність обробки даних на фінальному етапі – O(N), що є не придатним для великих обсягів електронних листів.
- Не масштабується без індексних структур, які в свою чергу неефективні при надвеликих вимірах.

При тестуванні методу на невеликому корпусі даних «Enron» (10 тис. Загальної кількості електронних листів) k-NN (де k=5) показав результат F1-міри 0,91, але при обробці 100 тис. листів час відповіді перевищив 1 секунду, тому в промислових системах використовується рідко [8].

Використання нейронних мереж, зокрема згорткових , рекурентних та гібридних архітектур, є ефективним підходом для автоматизованої ідентифікації фішингових електронних листів. На відміну від класичних методів машинного навчання, нейромережеві моделі здатні автоматично формувати складні семантичні та лексичні ознаки тексту, значно покращуючи точність ідентифікації фішингових патернів [7].

Таблиця 1.1

Порівняльний аналіз методів для виявлення фішингу

Метод	Середня F1-міра	Затримка	Використання оперативної пам’яті (50 k листів)	Ключовий недолік
NB	0,90	0,002 с	< 100 МБ	Припущення незалежності слів
LR	0,93	0,005 с	0,5 ГБ	Чутливість до лексичного зсуву
SVM (linear)	0,95	0,015 с	0,8 ГБ	Повільне навчання
RF	0,94	0,030 с	1,3 ГБ	Велика кількість фінальних класифікаторів
XGBoost	0,96	0,060 с	4,2 ГБ	Необхідність у високих ресурсах, складний тюнінг

Згорткові нейронні мережі широко застосовуються для задач класифікації текстів, оскільки мають здатність виявляти локальні ознаки й шаблони слів або фраз. CNN працюють шляхом використання набору фільтрів, які послідовно обробляють текстові дані, попередньо представлені у векторному форматі (наприклад, Word2Vec або GloVe). Кожен згортковий шар виділяє конкретні лексичні патерни, після чого застосовується підвибірка

ознак (від англ. max-pooling), що дозволяє ефективно визначати найбільш важливі локальні характеристики для подальшої класифікації. Дослідження показали, що CNN забезпечують точність близько 90% у задачах класифікації коротких текстів, таких як фішингові повідомлення [9].

Переваги CNN:

- Висока швидкість обробки текстових даних.
- Ефективна ідентифікація коротких лексичних патернів.
- Стійкість до змін порядку слів у тексті.

Недоліки CNN:

- Недостатня здатність до аналізу далеких контекстних залежностей слів.
- Обмеженість у врахуванні глобальної структури тексту.

Рекурентні нейронні мережі спеціалізуються на обробці послідовних даних, що дозволяє враховувати як близькі, так і далекі контекстні зв'язки між словами. Особливо популярними є архітектури Long Short-Term Memory (LSTM) та Gated Recurrent Units (GRU), які містять комірки пам'яті для контролю над тим, яку інформацію слід запам'ятовувати, а яку – ігнорувати. Такі мережі дозволяють ефективно визначати складні семантичні залежності, типові для фішингових повідомлень, зокрема емоційні заклики або термінові звернення. За даними дослідження DeepAntiPhishNet, модель на основі LSTM досягла точності класифікації на рівні 99,1% при аналізі стандартного набору IWSPA-2018 [10].

Переваги RNN:

- Здатність враховувати складні контекстні зв’язки між словами.
- Висока ефективність при аналізі складних текстових конструкцій.
- Добре підходять для обробки послідовних даних.

Недоліки RNN:

- Низька швидкість роботи через послідовну обробку тексту.
- Схильність до проблеми зникнення градієнтів при обробці довгих послідовностей.

Гібридні архітектури (CNN-RNN) були запропоновані для подолання окремих недоліків згорткових та рекурентних мереж. Поширеними є комбінації CNN з двонаправленими рекурентними мережами (наприклад, CNN-BiLSTM або CNN-BiGRU), які поєднують сильні сторони обох типів мереж. CNN у таких моделях відповідають за локальну обробку текстових фрагментів, а RNN – за глобальний контекст. Часто ці моделі доповнюють механізмом уваги (attention mechanism), що дозволяє ідентифікувати найважливіші слова або фрази, підвищуючи ефективність і прозорість результатів.

Переваги гібридних моделей CNN-RNN:

- Поєднання переваг CNN (локальні ознаки) та RNN (глобальний контекст).
- Підвищена точність завдяки механізмам уваги.
- Краща інтерпретованість результатів.

Недоліки гібридних моделей CNN-RNN:

- Вища складність архітектури порівняно з простими CNN або RNN.
- Необхідність більш тривалого навчання через додаткові параметри.

Таким чином, нейронні мережі легкої глибини – CNN, RNN та їх гібридні варіанти – демонструють високу точність у задачах виявлення фішингу. Кожна з цих моделей має свої сильні та слабкі сторони, а їх комбінація у гібридних архітектурах забезпечує найбільш оптимальне рішення для ефективної ідентифікації фішингових електронних листів.

Трансформерні моделі, такі як BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pretrained Transformer), RoBERTa (Robustly Optimized BERT Pretraining Approach) і XLNet, належать до найбільш сучасних підходів у галузі обробки природномовних текстових даних. Вони стали еталонними рішеннями для вирішення широкого спектру задач, включно з ідентифікацією фішингових електронних листів, завдяки своїй здатності враховувати глибокий контекст слів і складні семантичні зв’язки у текстах [11].

На відміну від традиційних рекурентних або згорткових мереж, трансформери не використовують послідовну обробку даних, що дозволяє уникнути типових для рекурентних мереж проблем із зникненням градієнтів і покращити ефективність роботи з довгими текстами. Ключовою особливістю трансформерів є механізм самоуваги (від анг. self-attention), який дозволяє моделі безпосередньо враховувати зв'язки між усіма словами у тексті одночасно, а не лише сусідніми словами, як у випадку CNN, або послідовно оброблюваними, як у випадку RNN [12].

Основним представником трансформерів є модель BERT, яка є двонаправленою, тобто враховує контекст слів з обох сторін одночасно. BERT навчається шляхом попереднього тренування на великих корпусах текстів із використанням завдань маскування слів (від англ. Masked Language Modeling) і прогнозування наступного речення (від англ. Next Sentence Prediction). Це дозволяє моделі глибоко розуміти контекст слів і їхню роль у реченнях, що критично важливо для виявлення складних фішингових шаблонів [11].

Дослідження, проведене у 2024 році, демонструє, що застосування BERT до ідентифікації фішингових листів дозволило досягти точності на рівні 99,4% та F1-міри 0,993. При цьому модель успішно виявляла тонкі семантичні ознаки, такі як підроблені листи з імітацією повідомлень технічної підтримки та фінансових установ [13].

Переваги трансформерних моделей:

- Найвища точність серед сучасних підходів NLP завдяки глибокому врахуванню контексту слів.
- Ефективне узагальнення на нових типах фішингових повідомлень, що не входили до навчальної вибірки.
- Мінімальна кількість необхідних додаткових ознак, оскільки модель автоматично формує семантичні вектори слів.
- Готові попередньо навчені моделі для швидкої адаптації під конкретні завдання.

Недоліки трансформерних моделей:

- Високі обчислювальні витрати на етапі навчання та іноді навіть на етапі розгортання.
- Потребують значних ресурсів оперативної пам’яті через велику кількість параметрів.
- Високий рівень складності інтерпретації результатів через велику кількість параметрів і внутрішніх зв’язків.

Інша популярна модель, GPT, є авторегресивною, що означає передбачення наступних слів на основі попереднього контексту. Це робить її ефективною у задачах генерації та аналізу текстів, але менш придатною для класифікації порівняно з BERT, через односторонній аналіз контексту. RoBERTa та XLNet – це подальші вдосконалення BERT і GPT, які додатково підвищують ефективність завдяки оптимізації процесів навчання та розширенню контекстуальних зв’язків [14].

Зокрема, дослідження 2023 року виявило, що модель RoBERTa дозволила покращити ідентифікацію фішингових повідомлень порівняно з базовим BERT, досягши показників точності у 99,7%, особливо в умовах складних атак з використанням соціальних мереж [15].

Отже, трансформерні моделі демонструють високу точність виявлення фішингу завдяки своїй здатності глибоко аналізувати семантичний контекст. Але при цьому мають свої недоліки у вигляді високих обчислювальних витрат та необхідності тренування на більших наборах даних ніж методи глибокого навчання.

1.4. Огляд існуючих методів виявлення фішингу

Для розробки ефективної гібридної моделі ідентифікації фішингових електронних листів необхідним є детальний аналіз сучасних методів, представлених у наукових дослідженнях та практичних рішеннях. Розглянемо найважливіші наукові праці, що вплинули на формування сучасних підходів до автоматизованого виявлення фішингу.

Одним з найважливіших досліджень у сфері ідентифікації фішингу є проєкт AntiPhish, представлений у роботі «AntiPhish: An Adaptive Multi-Level Framework for Phishing Email Detection» [16]. AntiPhish – це багаторівнева адаптивна система, яка поєднує класичні методи машинного навчання, аналіз евристик і нейронні мережі для ефективного виявлення різних типів фішингових атак.

Методика AntiPhish базується на трирівневому аналізі електронних листів:

- Перший рівень (евристичний аналіз) – ідентифікує підозрілі ознаки у тексті та заголовках повідомлення, такі як нестандартні доменні імена, невідповідності у DKIM/SPF-підписах, наявність URL, які містять помилки або спеціальні символи.
- Другий рівень (аналіз на основі ознак) – використовує класичні методи машинного навчання, такі як логістична регресія та метод опорних векторів (SVM), для первинної класифікації електронних листів на основі набору лексичних та структурних характеристик.
- Третій рівень (глибокий нейронний аналіз) – реалізує нейромережевий підхід із використанням двонаправлених рекурентних мереж LSTM (Long Short-Term Memory) для виявлення складних семантичних шаблонів та контекстуальних ознак.

Однією з головних переваг AntiPhish є його адаптивність. Система динамічно змінює вагу різних методів і рівнів аналізу залежно від складності та типу атаки, що дозволяє ефективно обробляти як масові (bulk phishing), так і вузькоцільові атаки (spear phishing).

За результатами експериментальних досліджень на відкритому наборі даних «Enron AntiPhish» досяг точності 98,7% і показника F1-міри 0,978. Важливо зазначити, що метод продемонстрував значно вищу ефективність у порівнянні з традиційними підходами, такими як прості евристичні правила або класичні методи без нейронної компоненти [16].

Попри високу ефективність, AntiPhish має певні обмеження:

- Складність реалізації через багаторівневу архітектуру.
- Потребує значних обчислювальних ресурсів, особливо при використанні глибоких нейронних мереж.
- Чутливість до якості навчальних наборів: ефективність моделі суттєво залежить від релевантності даних, які використовуються для навчання.

Отже, аналіз AntiPhish демонструє потенціал гібридних методів у сфері ідентифікації фішингу, проте також вказує на необхідність оптимізації ресурсних витрат та простоти реалізації за підтримки високої точності роботи моделі. Саме ці напрямки можуть бути вдосконалені у розробці нових гібридних методів.

Ще одним провідним підходом у сфері автоматизованої ідентифікації фішингових електронних листів є метод THEMIS. Метод THEMIS є інноваційним підходом до ідентифікації фішингових електронних листів, що базується на поліпшеній рекурентній конволюційній нейронній мережі (RCNN) із використанням багаторівневих векторів та механізму уваги (англ. Attention mechanism) [17]. Основна ідея методу THEMIS полягає у глибокому аналізі структури електронного листа, що охоплює одночасне використання текстової інформації з тіла листа, заголовка, а також враховує інформацію на рівні окремих символів і слів.

На першому етапі THEMIS здійснює попередню обробку тексту електронних листів, що дозволяє зменшити кількість зайвих сигналів у даних. Далі метод формує багаторівневі векторні представлення листів, застосовуючи моделі векторних представлень (англ. embeddings) для слів та символів. Використання багаторівневих векторів є перевагою цього підходу, оскільки дозволяє краще захоплювати специфічні особливості, такі як орфографічні помилки чи індивідуальні стилістичні особливості написання тексту листів.

Наступним кроком метод використовує поліпшену модель RCNN, яка інтегрує двонаправлену рекурентну нейронну мережу з довгою короткостроковою пам'яттю (Bi-LSTM). Завдяки цьому THEMIS ефективно захоплює контекстну інформацію тексту та зменшує негативний вплив шумів у даних. Додатково, THEMIS використовує механізм уваги між заголовком і тілом листа, що дозволяє моделі динамічно визначати, які частини листа є більш інформативними для ухвалення рішення про наявність фішингового характеру.

Ефективність методу THEMIS була оцінена на реалістичному не збалансованому наборі даних, що складається як з фішингових, так і з легітимних електронних листів. В результаті експериментальних досліджень було досягнуто високої загальної точності ідентифікації фішингових листів – 99,848%, із надзвичайно низьким рівнем помилкових спрацьовувань (FPR = 0,043%) [17].

Таким чином, основні переваги методу THEMIS:

- високоточна ідентифікація фішингових листів за рахунок поєднання багаторівневих векторів та механізму уваги;
- низький рівень хибних позитивних спрацювань (низький FPR);
- ефективна обробка як текстової, так і контекстної інформації електронних листів.

Недоліки методу THEMIS:

- підвищена обчислювальна складність через використання декількох рівнів аналізу та складних архітектур нейронних мереж;
- необхідність попередньої якісної підготовки та обробки даних.

Загалом, THEMIS демонструє високу ефективність у завданні ідентифікації фішингу, перевершуючи наявні традиційні методи і підтверджуючи доцільність застосування складних нейромережевих архітектур для цієї задачі.

Останніми роками методи на основі трансформерних моделей займають високі позиції в задачах обробки природномовних текстових даних

завдяки здатності глибоко розуміти контекст через механізм самоуваги (*self-attention*). Проте, незважаючи на значну ефективність, їх застосування до задачі фільтрації електронних листів потребує ретельного аналізу через високі обчислювальні вимоги, що може бути обмеженням для реальних систем.

Одним із яскравих прикладів трансформерних рішень є підхід на основі комбінації трансформерної архітектури BERT (від англ. *Bidirectional Encoder Representations from Transformers*) та двонаправлених LSTM-шарів (BiLSTM). Зокрема, у дослідженні 2024 року запропоновано модель, яка поєднує переваги BERT щодо глибокого розуміння контексту із можливостями LSTM у роботі з послідовностями. На змішаному англо-німецькому наборі даних обсягом близько 120 тисяч електронних листів модель продемонструвала такі результати [18]:

- точність класифікації (*Accuracy*) на рівні 99,4%;
- повнота (*Recall*) не нижче 99% для фішингових листів;
- інтегральна міра F1 склала 0,993.

Попри високу ефективність, головним недоліком є значна затримка обробки листів, яка становила приблизно 0,8–1 секунду на центральному процесорі (CPU). Високі вимоги до оперативної пам'яті (до 3 ГБ) також створюють додаткові обмеження для практичного використання цієї моделі в умовах реального часу.

Для зменшення обчислювальних було запропоновано використовувати компактніші варіанти трансформерів, а саме Tiny-BERT, який є легкою версією оригінального BERT із чотирма шарами (близько 14,5 млн параметрів). Підхід полягав у додатковому навчанні (fine-tuning) Tiny-BERT на спеціалізованому наборі даних IWSPA-AP 2018. Завдяки цьому вдалося отримати [15]:

- F1-міру на рівні 0,990, що практично не поступається повнорозмірному BERT;

- значне скорочення часу затримки з приблизно 820 мс (BERT-base) до 220 мс для Tiny-BERT на CPU.

Незважаючи на значне скорочення часу затримки, Tiny-BERT все ще є ресурсомістким порівняно з гібридними CNN-RNN моделями та потребує спеціалізованих серверних конфігурацій для ефективного масштабування.

Переваги трансформерних рішень:

- глибоке семантичне розуміння тексту завдяки механізму уваги;
- висока точність класифікації, особливо для складних і тонких атак;
- легка адаптація до нових доменів шляхом fine-tuning.

Недоліки трансформерних рішень:

- високі обчислювальні витрати (CPU/GPU) та оперативна пам’ять;
- велика латентність, що ускладнює реалізацію у реальному часі;
- складність і дороговизна розгортання в локальних інфраструктурах.

Таким чином, трансформери демонструють відмінну точність, однак їх використання пов'язане із суттєвими ресурсними витратами. На цьому тлі запропонована гібридна модель CNN-BiLSTM із механізмом уваги є ефективнішою альтернативою для інтеграції в реальні поштові системи завдяки поєднанню високої точності із низькими апаратними вимогами та мінімальною затримкою.

Паралельно з академічними та рішеннями з відкритим кодом існують комерційні системи, які інтегруються безпосередньо у корпоративні поштові сервіси. Найбільш відомими серед таких систем є Google Workspace AI та Microsoft Defender для Office 365. Ці продукти використовують поєднання нейронних мереж, методів машинного навчання, аналізу поведінки користувачів та глобальних баз знань про загрози, що забезпечує високу точність класифікації, проте має ряд суттєвих обмежень.

Google інтегрував у свою платформу Workspace складний механізм на основі штучного інтелекту, який аналізує вміст електронних листів і поведінку користувачів, щоб автоматично блокувати фішингові загрози.

Сервіс використовує трансформерні моделі типу BERT, навчання яких проводиться на величезних корпоративних наборах даних, зібраних глобально. Внутрішні звіти Google вказують на досягнення надзвичайно високої точності класифікації (> 99,8%) та низький рівень помилкових спрацьовувань (FPR < 0,2%) у більшості випадків використання [19].

Переваги Google Workspace AI:

- надзвичайно висока точність завдяки великій кількості тренувальних даних;
- глибока інтеграція з сервісами Google, що мінімізує затримки у класифікації;
- постійне оновлення моделі за рахунок використання глобальних баз знань.

Недоліки Google Workspace AI:

- повна закритість моделі (від англ. black-box), що унеможливлює її налаштування або донавчання для спеціалізованих корпоративних сценаріїв;
- висока ціна для великих компаній і корпоративних клієнтів;
- відсутність деталізованих пояснень рішень моделі для кінцевих користувачів.

Аналогічний за своєю сутністю продукт від Microsoft також активно використовує штучний інтелект, аналіз поведінки та репутаційні бази. Defender включає багатошаровий захист, який поєднує аналіз контенту листів, URL-адрес, вкладень, а також історичну інформацію щодо поведінки користувачів. Згідно зі звітом компанії Microsoft, Defender демонструє середню точність ідентифікації фішингових атак на рівні близько 99,7% з показником хибних спрацювань близько 0,2% [20].

Переваги Microsoft Defender:

- комплексний аналіз не лише вмісту листа, а й супутньої поведінки користувачів;

- можливість інтеграції з широким спектром корпоративних рішень Microsoft;
- низька ймовірність помилкових спрацювань у типових сценаріях.

Недоліки Microsoft Defender:

- високі витрати на підписку, особливо для малих і середніх бізнесів;
- модель також є закритою, що не дозволяє проводити налаштування чи оптимізацію під власні потреби;
- у випадках нетипових або спеціалізованих фішингових атак, ефективність може значно падати через нездатність користувачів втручатися в роботу моделі.

Рішення на основі гібридних моделях нейронних мереж пропонують можливість точного налаштування та інтеграції у спеціалізовані корпоративні системи, мають нижчі витрати та прозорість рішень, що є критичним для деяких секторів з високими вимогами до безпеки та відповідності нормативним актам.

Таким чином, хоча комерційні системи демонструють високу ефективність, вони не є ідеальним рішенням для усіх типів організацій через високу вартість та обмежену гнучкість у використанні. Саме ці аспекти формують переваги гібридних моделей, які можуть бути налаштовані й адаптовані під специфічні задачі та вимоги.

1.5. Висновки до першого розділу

У даному розділі детально розглянуто проблематику автоматизованої ідентифікації фішингових електронних листів, проаналізовано поняття соціальної інженерії як основи фішингових атак, наведено класифікацію різновидів фішингу, а також проведено огляд і порівняння існуючих підходів до автоматичного виявлення таких загроз.

Соціальна інженерія лежить в основі більшості фішингових атак і базується на психологічних методах маніпуляції, що дозволяє зловмисникам отримати конфіденційну інформацію або змусити користувача виконати

небажані дії. Статистичні дані свідчать, що фішинг є одним із найбільш поширених і небезпечних типів соціотехнічних атак, що призводить до значних фінансових втрат і витоків інформації.

У розділі також здійснено аналіз існуючих методів до автоматизованої ідентифікації фішингу, які умовно можна поділити на три основні категорії: класичні методи машинного навчання, нейромережеві моделі та трансформери.

Класичні методи (такі як логістична регресія, SVM, дерева рішень тощо) базуються на аналізі ознак, отриманих із тексту електронних листів, таких як частота слів, лексичні та синтаксичні шаблони. Ці методи характеризуються швидким навчанням, простотою інтерпретації та низькими обчислювальними витратами, однак демонструють нижчу точність порівняно з нейромережевими моделями при складних сценаріях ідентифікації фішингу.

Нейронні мережі (такі як CNN, RNN та гібридні CNN-RNN моделі) забезпечують високу ефективність завдяки глибокому виявленню локальних текстових шаблонів (CNN) та контекстуальної залежності у текстах (RNN). Гібридні підходи, які об'єднують CNN і RNN, дозволяють враховувати як локальні, так і контекстуальні особливості, демонструючи високу точність (>99%), але все ще вимагають значних ресурсів на тренування та налаштування.

Сучасні трансформерні рішення (такі як BERT, Tiny-BERT та інші) показують ще вищу точність за рахунок аналізу довготривалих залежностей і контекстуального розуміння тексту. Проте трансформери мають значно вищі вимоги до обчислювальних ресурсів, що ускладнює їх практичне застосування в реальних системах з великим потоком електронних повідомлень.

Також проведено детальний огляд наукових розробок, а також комерційних сервісів. Попри високу ефективність комерційних систем, вони мають суттєві недоліки, такі як висока вартість, закритість рішень та обмежена адаптивність до конкретних корпоративних потреб.

Враховуючи проаналізовані підходи, перспективним напрямком є розроблення відкритих, гібридних нейромережевих моделей, які мають конкурентну точність із трансформерами та комерційними сервісами, але водночас забезпечують значно менші витрати обчислювальних ресурсів, кращу прозорість та можливість гнучкого налаштування під спеціалізовані задачі та конкретні вимоги корпоративних замовників.

2. РОЗРОБЛЕНИЙ КОМБІНОВАНИЙ МЕТОД ДЛЯ АВТОМАТИЗОВАНОЇ ІДЕНТИФІКАЦІЇ ФІШИНГОВОГО ВМІСТУ В ТЕКСТОВІЙ ЧАСТИНІ ЕЛЕКТРОННИХ ЛИСТІВ

2.1. Теоретичне обґрунтування комбінованого методу

2.1.1. *Формалізація задачі виявлення фішингових електронних листів*

Задача автоматизованого виявлення фішингових електронних листів формально може бути описана як задача бінарної класифікації текстових повідомлень [9]. Це означає, що кожен електронний лист повинен бути ідентифікований як один з двох класів: фішинговий або легітимний (не фішинговий).

Формально маємо навчальну множину текстових документів (електронних листів):

$$X = \{x_1, x_2, ..., x_N\},\ x_i = \left(w_{i1}, w_{i2}, ..., w_{in_i}\right), \tag{2.1}$$

де x_i – це *i*-ий електронний лист, що представлений послідовністю слів (токенів) ω_i, n_i – кількість слів у листі x_i, а N – загальна кількість листів у навчальній вибірці.

Кожному листу x_i відповідає мітка класу y_i:

$$y_i \in \{0, 1\},\ i = 1, ..., N, \tag{2.2}$$

де:

- $y_i = 1$ – фішинговий лист;
- $y_i = 0$ – легітимний лист.

Таким чином, задача класифікації полягає у знаходженні моделі:

$$f: X \to [0, 1], \tag{2.3}$$

яка за текстом листа x_i визначає ймовірність його належності до класу фішингових повідомлень. Вихідна ймовірність класифікатора порівнюється з визначеним порогом τ :

$$\hat{y}_i = \begin{cases} 1, f(x) \geq \tau, \\ 0, f(x) < \tau, \end{cases} \tag{2.4}$$

де y_i – передбачений клас листа x_i.

Таким чином, формалізація даної задачі дозволяє чітко визначити математичну основу комбінованого методу виявлення фішингових електронних листів, що є необхідним для його коректної реалізації та подальшого аналізу ефективності.

2.1.2. *Алгоритм роботи запропонованого комбінованого методу*

Запропонований метод автоматизованого виявлення фішингових електронних листів заснований на комбінованій нейромережевій архітектурі, що поєднує у собі згорткові нейронні мережі, двонаправлені рекурентні нейронні мережі та спеціалізований шар уваги. Дана поєднання було обране з урахуванням теоретичних переваг кожного компонента для аналізу текстових даних та специфіки задачі класифікації природномовних текстових даних.

Схема запропонованого комбінованого методу має наступний вигляд:

1. Попередня обробка тексту:
   - Очистка та нормалізація текстових даних;
   - Токенізація та лематизація тексту;
   - Векторизація за допомогою попередньо натренованих ембедінгів (GloVe).
2. Аналіз тексту згортковою нейромережею:
   - Виділення локальних ознак (патернів слів і словосполучень), які часто зустрічаються в фішингових листах.
3. Аналіз тексту рекурентною нейромережею:

- Аналіз глобального контексту тексту шляхом послідовного двонаправленого проходу для врахування залежностей між словами.

4. Аналіз тексту механізмом уваги:
   - Динамічне визначення релевантності кожної ознаки, виділеної CNN та BiLSTM, з фокусом на найважливіші частини тексту для класифікації.
5. Класифікація електронного листа:
   - Використання сигмоїдальної функції активації для отримання ймовірності належності листа до класу фішингових повідомлень.

Концептуально схему можна представити наступною формулою:

$$X_{input} \rightarrow CNN \rightarrow BiLSTM \rightarrow Attention \rightarrow Dense(\sigma) \rightarrow y_{output}, \quad (2.5)$$

де X_{input} – вхідний текст листа у вигляді послідовності ембедінгів слів, а y_{output} – ймовірність належності листа до класу фішингу.

Обґрунтуємо вибір компонентів:

- CNN-шари дозволяють ефективно виявляти локальні текстові особливості. У задачах обробки природномовних текстових даних це означає знаходження патернів, характерних для фішингових листів (наприклад, підозрілі фрази, заклики до термінових дій тощо) [10].
- BiLSTM-шари дозволяють аналізувати текст в обох напрямках: зліва направо та справа наліво. Це допомагає враховувати як попередній, так і наступний контекст слів, забезпечуючи краще розуміння значення речень і контекстуальної структури.
- Шар уваги призначений для динамічного визначення вагомості різних частин тексту (слів чи груп слів) при прийнятті рішення про класифікацію. «Attention mechanism» дозволяє моделі зосереджуватись саме на тих словах і патернах, які найбільше

впливають на вірогідність того, що повідомлення є фішинговим [12].

Можемо сформувати схему роботи запропонованого методу (рис. 2.1).

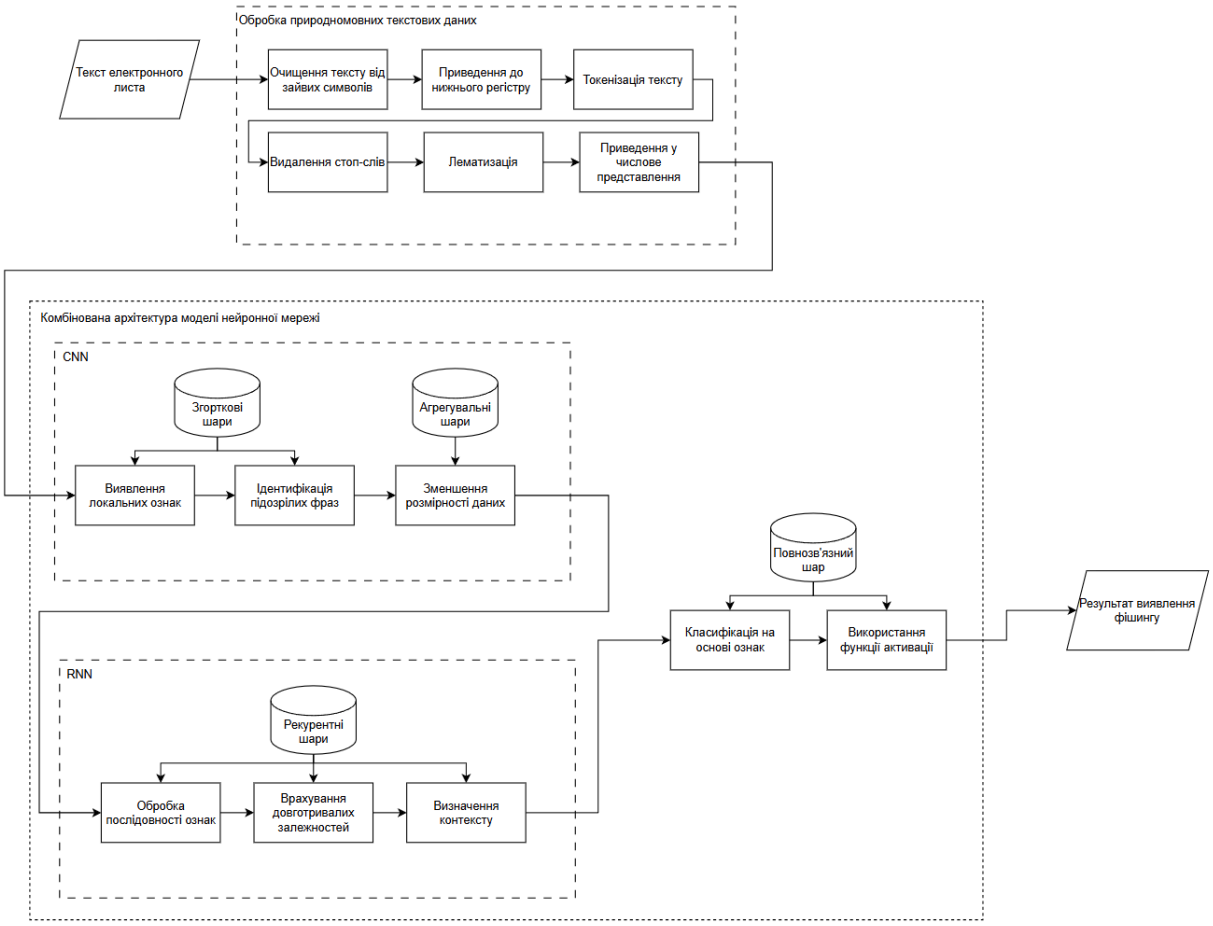


Рис. 2.1. Схема запропонованого методу ідентифікації фішингового вмісту в текстовій частині електронних листів

Таким чином, використання гібридної архітектури CNN–BiLSTM–Attention є виправданим рішенням, що дозволяє поєднати сильні сторони CNN, BiLSTM та Attention. Подібне поєднання демонструє високі показники якості класифікації, при цьому маючи нижчі вимоги до обчислювальних ресурсів порівняно з великими трансформерними моделями типу BERT або RoBERTa, що робить її ефективною у практичному використанні для задачі автоматизованого виявлення фішингових електронних листів [17].

2.1.3. *Особливості використання тексту тіла електронних листів*

Запропонований комбінований метод автоматизованого виявлення фішингових листів ґрунтується винятково на аналізі текстового контенту тіла листа («body-only» підхід). На відміну від комплексних рішень, що аналізують повний спектр характеристик електронної пошти (заголовки, адреси відправника та отримувача, посилання, метадані тощо), цей підхід передбачає фокусування виключно на природномовній складовій повідомлення.

Опишемо переваги body-only підходу для зниження складності моделі:

- Використання лише тексту дозволяє значно зменшити кількість вхідних ознак, що полегшує процес навчання нейромережі та робить модель менш схильною до перенавчання (overfitting). Це, у свою чергу, забезпечує кращу узагальнювальну здатність моделі.
- Текст листа, після відповідної нормалізації та векторизації, являє собою однорідні дані, що дозволяє використовувати стандартні та ефективні техніки NLP, не витрачаючи додаткові ресурси на аналіз і комбінування неоднорідних типів інформації.
- Аналіз тільки тексту дозволяє легше адаптувати метод до листів, написаних різними мовами або у різних форматах у майбутньому. Модель, натренована виключно на природномовних ознаках, більш стійка до змін у форматах електронної пошти.
- Скорочення кількості оброблюваних ознак і зменшення обчислювальної складності дозволяє реалізувати модель із високою швидкістю інференсу, що є критично важливим для систем, які працюють у реальному часі.

Проведемо аналіз літературних джерел, що підтверджують ефективність body-only підходу:

- За даними роботи «DeepAntiPhishNet» [18], використання лише текстового контенту листа може бути достатнім для ефективного виявлення фішингу. У цьому дослідженні було показано, що

текстові моделі, засновані на глибоких нейромережах, здатні досягти високої точності саме завдяки зосередженню на семантиці повідомлень.

- Подібні результати були отримані у дослідженні моделі RoBERTa [15], де показано, що природномовний аналіз за допомогою попередньо натренованих моделей забезпечує високу ефективність навіть без аналізу метаданих чи заголовків повідомлень.
- Дослідження виявлення фішингу за допомоги штучного інтелекту від 2024 року також підкреслює, що концентруючись на вмісті повідомлень, можна забезпечити високу інтерпретованість моделі та її стійкість до змін у шаблонах і форматах листів, які часто використовують зловмисники для обходу систем захисту [13].

Таким чином, використання body-only підходу є обґрунтованим рішенням, яке дозволяє створити легкий, ефективний та масштабований метод автоматизованого виявлення фішингових електронних листів. Це забезпечує необхідний компроміс між точністю, швидкістю роботи та ресурсною ефективністю.

2.2. Попередня обробка природномовних текстових даних

2.2.1. *Аналіз необхідності попередньої обробки тексту*

Попередня обробка текстових даних (англ. *text preprocessing*) є критично важливим етапом при вирішенні задачі автоматизованої ідентифікації фішингових електронних листів. На практиці текстові дані часто характеризуються високим рівнем неоднорідності, наявністю шуму, граматичними і орфографічними помилками, а також містять специфічні елементи, такі як HTML-теги, URL-посилання, спеціальні символи та інші небажані складові, які можуть негативно впливати на якість та точність роботи класифікатора.

Задача виявлення фішингових повідомлень є специфічним випадком текстової класифікації, у якій вкрай важливою є точність інтерпретації семантичних і синтаксичних особливостей тексту.

Відсутність якісної попередньої обробки може призвести до таких проблем:

- Наявність шуму через HTML-код, зайві символи, випадкові вставки тексту, що може створювати додаткові ознаки, які ускладнюють процес навчання та знижують узагальнювальну здатність моделі.
- Варіативність форматів як наслідок різного стилю написання, використання різних регістрів символів, великої кількість скорочень і сленгових слів, що ускладнює процес генералізації.
- Зниження семантичної точності в результаті неоднозначності або неправильності інтерпретації слів, що виникає при різних формах слів (наприклад, множина або дієвідмінювання). Це призводить до того, що модель може неправильно оцінювати контекст повідомлення.

Таким чином, попередня обробка тексту дозволяє значно знизити рівень шуму, підвищити семантичну чистоту даних і, відповідно, підвищити точність класифікації фішингових листів.

Аналіз досліджень в області обробки природномовних текстових даних підтверджує необхідність попередньої обробки тексту:

- У одному з оглядів методів виявлення фішингових атак зазначають, що відсутність попередньої обробки тексту значно знижує ефективність будь-яких класифікаційних алгоритмів. Було підкреслено, що нормалізація, токенізація та очищення тексту є обов'язковими кроками для досягнення високої точності моделей [8].
- За даними іншого дослідження, нормалізація тексту дозволила суттєво підвищити точність їхньої моделі, побудованої на основі

нейромереж із попередньо натренованими ембедінгами (RoBERTa), особливо завдяки очищенню текстів від URL-посилань і зайвих HTML-символів [15].

- В оглядовій статті методам виявлення фішингу також підкреслено важливість етапу попередньої обробки текстів, зокрема для усунення спроб приховати ключові слова або фрази, які є типовими ознаками фішингових листів [1].

Враховуючи аналіз сучасних наукових досліджень та практичний досвід, можна констатувати, що попередня обробка текстових даних є критичним етапом, без якого неможливе ефективне розв'язання задачі класифікації фішингових електронних листів. Цей етап не лише дозволяє забезпечити стабільну роботу класифікаційних алгоритмів, але й відчутно підвищує якість кінцевого результату, забезпечуючи достатню узагальнювальну здатність моделей.

Для попередньої обробки тексту обрано наступні методи:

1. Очищення тексту (англ. Text cleaning).
2. Токенізація (англ. Tokenization).
3. Лематизація (англ. Lemmatization).
4. Видалення стоп-слів (англ. Stop-word removal).

Теоретичне значення очищення тексту полягає у зменшенні інформаційного шуму, який може погіршити здатність моделі розпізнавати потрібні патерни. В контексті електронних листів, поширеними видами шуму є HTML-теги, URL-адреси та спеціальні символи. В сучасних дослідженнях було показано, що вилучення цих елементів суттєво зменшує кількість випадкових ознак і підвищує інформативність використаних слів [8, 15].

Таким чином, очищення тексту містить:

- *Видалення HTML-розмітки:* HTML-теги не несуть безпосередньої семантичної інформації про зміст листа і можуть бути видалені без втрати важливої інформації.

- *Вилучення URL-посилань:* URL-адреси мають складну структуру і часто складаються з випадкових або зашифрованих символів, що може призводити до формування зайвих або спотворених ознак. Їх вилучення дозволяє спростити аналіз змісту тексту листів.
- *Видалення спеціальних символів і пунктуації:* пунктуаційні знаки та спеціальні символи зазвичай не містять суттєвої інформації щодо класифікації і лише збільшують розмір словника. Видалення таких символів сприяє кращій узагальнюючій здатності моделі.
- *Перетворення тексту до нижнього регістру:* стандартизація регістру дозволяє уникнути множинних варіантів одного й того ж слова, які можуть бути неправильно інтерпретовані як окремі ознаки.

2.2.2. *Вибір методів токенізації та нормалізації*

Токенізація – це процес розбиття тексту на мінімальні одиниці інформації (токени), які в подальшому можуть бути представлені у векторному вигляді. Правильна токенізація має критичне значення для точності NLP-моделей, адже саме на цьому етапі формуються базові елементи аналізу, на основі яких модель приймає рішення [9, 11].

Обраний у запропонованому методі спосіб токенізації враховує:

- Коректне розпізнавання слів з урахуванням мовних особливостей. У випадку англійської мови, важливо теоретично обґрунтовано вибрати токенізатор, здатний коректно розпізнавати слова та морфеми, не втрачаючи важливих смислових відтінків.
- Врахування багатомовності та спеціальної лексики. В електронних листах часто присутня специфічна лексика (технічні терміни, сленги), тому важливо, щоб обраний метод токенізації враховував такі особливості, мінімізуючи кількість невірно розпізнаних слів.

Лематизація передбачає зведення слів до їх базових словникових форм (лем), що є необхідним етапом для зменшення розмірності словника та

покращення семантичної узагальненості моделі. Відповідно до існуючих наукових досліджень, лематизація ефективно знижує варіативність слів у текстах, підвищуючи репрезентативність кожного слова [1, 7].

Лематизація дозволяє моделі більш точно розпізнавати патерни, які інакше могли б бути представлені множиною варіацій одного слова (наприклад, «напад», «нападу», «нападами» зводяться до однієї леми «напад»), що позитивно впливає на якість класифікації. Також, скорочення словника дозволяє зменшити обчислювальну складність моделі, що теоретично обґрунтовано підвищує її швидкодію та узагальнюючу здатність.

Стоп-слова – це частовживані слова, що не несуть значущої інформації для конкретної задачі класифікації (наприклад, «і», «та», «або», «в»). Видалення стоп-слів дозволяє зменшити кількість неінформативних ознак, тим самим посилюючи вплив справді релевантних слів на результати класифікації.

Видалення стоп-слів дозволяє нейронним мережам швидше і якісніше зосереджуватися на інформативних патернах, а не на частотних, але неважливих словах [16]. В свою чергу видалення неінформативних слів позитивно впливає на формування більш чітких ембедінгів, що покращує якість аналізу на наступних етапах обробки тексту.

Таким чином, вибрані способи токенізації та нормалізації тексту в запропонованому методі теоретично обґрунтовані як такі, що:

- забезпечують зниження шуму в даних,
- дозволяють ефективніше представляти семантичні та структурні особливості текстів,
- зменшують розмірність ознакового простору,
- сприяють кращій узагальнюючій здатності моделей машинного навчання, зокрема нейронних мереж.

Саме ці способибули обрані, враховуючи результати попередніх досліджень та аналіз наукової літератури у сфері обробки природномовних

текстових даних, що робить їх найбільш релевантними і теоретично доцільними для задачі ідентифікації фішингових електронних листів.

2.2.3. *Векторизація та ембедінги*

Текстові дані, які попередньо були очищені та нормалізовані, перед подачею до нейронної мережі повинні бути представлені у векторному вигляді. Найефективнішим методом представлення слів у векторній формі для задач класифікації текстів є використання алгоритмів векторизації, зокрема, Word2Vec та GloVe (від англ. Global Vectors for Word Representation).

Методи векторизації дозволяють кодувати семантичні та синтаксичні особливості слів у вигляді векторів дійсних чисел, які згодом використовуються як вхідні ознаки для нейронних мереж. Основна ідея таких методів полягає в тому, що слова, які знаходяться у схожих контекстах, матимуть близькі векторні представлення, що дозволяє нейронній мережі ефективніше навчатися розпізнавати шаблони та патерни у тексті.

У запропонованому методі використовується саме GloVe, навчені на великому корпусі текстів із соціальної мережі Twitter. Особливістю цих ембедінгів є те, що вони враховують глобальні статистики появи слів у контексті всього текстового корпусу. GloVe генерує вектори слів, розв'язуючи оптимізаційну задачу мінімізації функції втрат, яка ґрунтується на логарифмічних ймовірностях спільної появи слів:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + b_j - \tilde{\log} X_{ij}\right)^2, \qquad (2.6)$$

де X_{ij} – кількість спільних появ слів i та j, $\omega_i, \tilde{\omega_j}$ – вектори слів, а b_i відповідно зміщення [21].

Застосування GloVe-ембедінгів, навчених на текстах із Twitter, має додаткові переваги для задачі виявлення фішингу в електронних листах, оскільки повідомлення соціальних мереж мають схожі характеристики зі стилем електронних листів: короткі фрази, жаргонізми, сленг, скорочення та

велика кількість специфічних виразів. Використання GloVe-ембедінгів із різною розмірністю (а саме 25, 50, 100, 200) дозволяє гнучко регулювати рівень абстракції та обчислювальну складність моделі [21].

Іншим популярним методом векторизації текстових даних є Word2Vec, який був запропонований раніше за GloVe. На відміну від GloVe, Word2Vec базується на двох основних архітектурах: Continuous Bag of Words (CBOW) та Skip-gram. CBOW навчає модель передбачати слово за його контекстом, а Skip-gram, навпаки, контекст за словом [22].

Основна відмінність між GloVe та Word2Vec полягає в їх підходах до врахування контексту слів:

- *Word2Vec:* Навчається локально, тобто враховує лише окремі контексти навколо слова, що робить його чутливим до локальних семантичних особливостей.
- *GloVe:* Враховує глобальні статистики співпояви слів у текстовому корпусі, що дозволяє отримувати більш загальне і семантично багате представлення слів.

У контексті задачі виявлення фішингових електронних листів GloVe є більш вдалим вибором завдяки можливості враховувати глобальні особливості тексту та стилістичні особливості, які характерні для фішингових повідомлень. Експериментально також доведено, що GloVe демонструє кращу ефективність у задачах класифікації коротких текстів порівняно з Word2Vec, оскільки краще узагальнює семантичні особливості, що важливо для текстів електронних листів [21, 22].

Таким чином, вибір GloVe у запропонованому методі теоретично обґрунтований його перевагами для задачі класифікації текстових даних, що використовуються у фішингових електронних листах.

2.3. Використання нейронних мереж для аналізу текстових даних

2.3.1. *Теоретичні аспекти згорткових нейронних мереж*

Згорткові нейронні мережі (англ. Convolutional Neural Networks, CNN) є одними з найефективніших інструментів у сучасних завданнях обробки текстової інформації, зокрема в задачах класифікації тексту. Первинно CNN були розроблені для аналізу зображень, де вони демонструють високу ефективність завдяки здатності захоплювати локальні ознаки через операції згортки. Однак згодом було доведено, що CNN так само ефективні і для текстових даних, оскільки дозволяють виявляти важливі локальні шаблони слів або фраз у текстах [9].

CNN складаються з наступних базових компонентів:

- *Шар згортки (англ. Convolutional Layer)* призначений для виділення локальних ознак у текстових даних за допомогою фільтрів. Ці фільтри проходять по всій послідовності тексту, перетвореного у вигляді матриці ембедінгів, і здійснюють операцію згортки:

$$c_i = f\left(w \cdot x_{i:i+h-1} + b\right), \tag{2.7}$$

де w – ваговий вектор (фільтр), $x_{i:i+h-1}$ – послідовність слів (довжини h), b – зміщення, f – нелінійна функція активації, наприклад, ReLU.

- *Шар субдискретизації (англ. Pooling Layer)* після згортки застосовують субдискретизацію, щоб зменшити розмірність отриманих ознак, зберігаючи при цьому найбільш значущі ознаки. У задачах NLP найчастіше використовують максимальну субдискретизацію (max-pooling):

$$c^{\wedge} = (c)\,, \tag{2.8}$$

де c – вектор ознак, отриманий після згортки.

- *Повнозв'язний шар (Fully Connected Layer)* виконує класифікацію отриманих ознак шляхом перетворення їх у ймовірності приналежності до певного класу:

$$y = \sigma\left(W_o \cdot \hat{c} + b_o\right), \tag{2.9}$$

де σ – функція активації, $W_0 b_0$ – ваги і зміщення вихідного шару відповідно.

Особливістю застосування CNN до текстів є те, що кожен фільтр CNN шукає певні патерни у словесних послідовностях, такі як характерні вирази чи комбінації слів, які можуть бути типовими для фішингових листів (наприклад, "ваш рахунок заблоковано", "терміново оновіть пароль", "підозріла активність" тощо). За рахунок цього мережа здатна ефективно виявляти локальні семантичні ознаки, що характерні саме для фішингових повідомлень, і таким чином полегшувати задачу класифікації (рис. 2.2) [9].

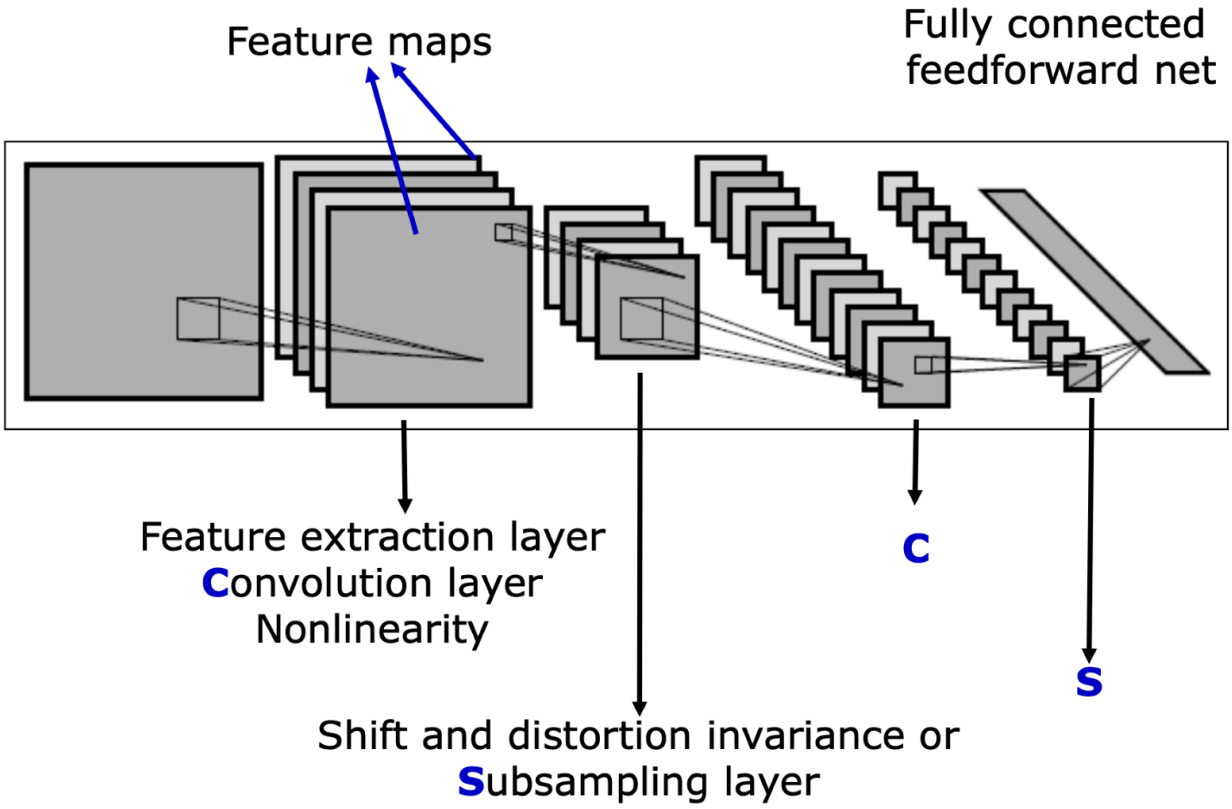


Рис. 2.2. Структура CNN

Переваги CNN у контексті NLP-задач:

- CNN ефективно знаходить важливі словосполучення та ключові фрази, що дозволяє краще ідентифікувати типові фішингові патерни.
- CNN є обчислювально простішими порівняно з рекурентними мережами, що дозволяє скоротити час тренування моделей.
- За рахунок фіксованого розміру ядра CNN менш чутливі до незначних відхилень і шумів у текстах, що є перевагою для текстів з помилками чи нестандартною лексикою.

Недоліки CNN у контексті NLP-задач:

- CNN працюють із фіксованим розміром вікна (контекстом), тому не можуть ефективно враховувати довготривалі залежності в текстах.
- CNN фіксують локальні шаблони, що може бути недоліком, якщо важливим є глобальний контекст або взаємодія слів на великих дистанціях.

Саме тому у пропонованому методі використовується комбінована архітектура, яка поєднує CNN з рекурентними мережами (BiLSTM) і шаром уваги, щоб компенсувати ці недоліки й ефективніше враховувати як локальні, так і глобальні контекстуальні особливості текстових даних [17].

2.3.2. *Теоретичні аспекти рекурентних нейронних мереж*

Рекурентні нейронні мережі (англ. Recurrent Neural Networks, RNN) – це клас штучних нейронних мереж, які спеціалізуються на роботі з послідовними даними. Основною особливістю RNN є здатність враховувати контекст, оскільки кожен елемент послідовності обробляється з урахуванням попередніх елементів. Завдяки цьому рекурентні мережі широко застосовуються в задачах обробки природної мови, зокрема для класифікації тексту, машинного перекладу, аналізу сентименту тощо [23]. Наведемо приклад стурктури RNN (рис. 2.3).

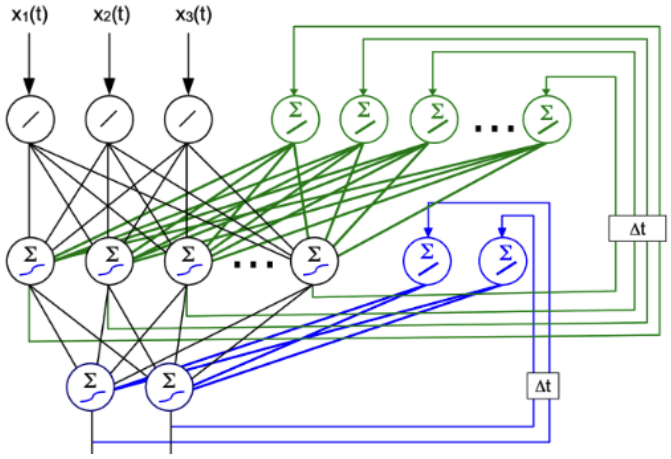


Рис. 2.3. Структура RNN

Незважаючи на свої переваги, класичні RNN мають серйозну проблему ефекту затухаючого градієнта (від англ. vanishing gradient problem). При обробці довгих послідовностей мережа може втрачати інформацію про далекі попередні елементи, що призводить до зниження ефективності навчання [24].

Для подолання цього недоліку було запропоновано спеціалізовану архітектуру – Long Short-Term Memory (LSTM), яка дозволяє зберігати інформацію про контекст протягом значно довших послідовностей завдяки складнішій структурі нейронів. LSTM-нейрон має спеціальні механізми регулювання інформаційних потоків (ворота або gates), що вирішують, яку інформацію зберігати, яку додавати до пам'яті, а яку – забувати [23].

Математично робота LSTM-клітинки описується такими формулами:

- Вхідні ворота (англ. *input gate*):

$$i_t = \sigma\left(W_i x_t + U_i h_t - 1 + b_i\right), \tag{2.10}$$

- Ворота забування (англ. *forget gate*):

$$f_t = \sigma\left(W_f x_t + U_f h_t - 1 + b_f\right), \tag{2.11}$$

- Вихідні ворота (англ. *output gate*):

$$o_t = \sigma\left(W_o x_t + U_o h_t - 1 + b_o\right), \tag{2.12}$$

- Кандидатний стан клітини (англ. *candidate state*):

$$\tilde{c_t} = tanh\left(W_c x_t + U_c h_t - 1 + b_c\right), \quad (2.13)$$

- Оновлення стану клітини (англ. *cell state*):

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}, \quad (2.14)$$

- Прихований стан клітини (англ. *hidden state*):

$$h_t = o_t \circ tanh\left(c_t\right), \quad (2.15)$$

де:

- x_t – вхідний вектор в момент часу *tt*t;
- h_t – попередній прихований стан клітинки;
- $W_i, W_f, W_o, W_c, U_i, U_f, U_o, U_c, b_i, b_f, b_o, b_c$ – матриці та вектори параметрів мережі;
- σ – сигмоїдна функція активації;
- $tanh$ – гіперболічний тангенс;
- $\circ$ – поелементне множення.

Хоча LSTM ефективно вирішує проблему затухаючого градієнта, стандартна версія цієї архітектури враховує лише попередній контекст (односпрямованість). В задачах аналізу тексту важливий не лише попередній, але й наступний контекст, що забезпечує глибше розуміння семантики тексту. Для цього використовується двонаправлений варіант LSTM (Bidirectional LSTM, BiLSTM), який обробляє текстові дані в двох напрямах: від початку до кінця та від кінця до початку [25].

Переваги використання BiLSTM у задачах NLP:

- BiLSTM аналізує текстові послідовності у двох напрямках, що дозволяє ефективно захоплювати повний семантичний контекст, важливий для визначення намірів та ознак фішингових листів.
- Завдяки використанню LSTM-клітинок, BiLSTM стабільно навчається на великих послідовностях тексту.

Недоліки використання BiLSTM:

- Порівняно з CNN, BiLSTM мають більшу обчислювальну складність і тренуються повільніше.
- Вимагає значних обчислювальних ресурсів, особливо для великих послідовностей та розмірів прихованих шарів.

Отже, використання двонаправлених рекурентних нейронних мереж (BiLSTM) в рамках запропонованої комбінованої моделі дозволяє ефективно враховувати глобальний контекст тексту електронних листів, що суттєво підвищує точність класифікації фішингових повідомлень.

2.3.3. *Механізм уваги в NLP*

Механізм уваги (англ. *Attention*) є одним з найбільш ефективних та поширених підходів, які застосовуються в сучасних задачах обробки природномовних текстових даних. Вперше запропонований у контексті задач машинного перекладу, механізм уваги швидко завоював популярність завдяки здатності враховувати значимість кожного елемента вхідної послідовності для формування остаточного рішення моделі [26].

Головна ідея механізму уваги полягає у визначенні важливості кожного токена в тексті, що дозволяє нейронній мережі "фокусуватися" на найбільш інформативних елементах. Завдяки цьому модель може ефективніше враховувати контекст і значущість окремих слів чи фраз, що суттєво підвищує якість класифікації тексту.

З математичної точки зору, механізм уваги для задачі класифікації тексту може бути представлений наступним чином:

Нехай модель генерує послідовність прихованих станів рекурентної нейромережі (наприклад, виходи BiLSTM):

$$H = \left[h_1, h_2, ..., h_T\right],\ h_i \in \mathbb{R}^d, \tag{2.16}$$

де h_i – прихований стан на i-му кроці, T – довжина послідовності, d – розмірність прихованого стану.

Далі механізм уваги обчислює ваги для кожного прихованого стану:

$$u_i = tanh\left(W_h h_i + b_h\right), \tag{2.17}$$

де W_h – матриця ваг уваги, b_h – вектор зсуву, а u_i – нелінійне перетворення прихованого стану.

Наступним кроком розраховуються значення балів (англ. attention scores) за допомогою контекстного вектора u_w:

$$a_i = \frac{exp\left(u_i^T u_w\right)}{\sum_{j=1}^{T} exp\left(u_j^T u_w\right)}, \tag{2.18}$$

де a_i – це ваговий коефіцієнт уваги, який визначає, наскільки значущим є відповідний прихований стан h_i для кінцевого прогнозу моделі.

Остаточний вектор, що агрегує інформацію всіх прихованих станів із застосуванням ваг уваги, розраховується як:

$$v = \sum_{i=1}^{T} a_i h_i, \tag{2.19}$$

Отриманий агрегований вектор v далі використовується як вхід для класифікаційного шару.

Переваги використання механізму уваги:

- *Підвищення якості класифікації* – завдяки увазі модель акцентує зусилля на тих частинах тексту, які найбільш релевантні для визначення класу повідомлення.
- *Краща інтерпретованість* – вагові коефіцієнти уваги дозволяють ідентифікувати, які саме слова або фрази найбільше вплинули на прийняття рішення, що робить модель прозорішою і зрозумілішою [27].

- *Вища ефективність роботи з довгими послідовностями* –увага дозволяє краще обробляти тексти зі складною семантикою, ефективно акцентуючи увагу лише на релевантних елементах.

Недоліки механізму уваги:

- *Додаткові обчислювальні ресурси* – використання уваги збільшує кількість параметрів моделі, що, у свою чергу, впливає на час тренування та обчислювальні витрати.
- *Ризик перенавчання* – через збільшення кількості параметрів існує ризик перенавчання моделі на тренувальному наборі, що потребує додаткових методів регуляризації для уникнення цієї проблеми.

Таким чином, інтеграція механізму уваги у запропонований комбінований метод дозволяє значно покращити ефективність виявлення фішингових листів завдяки зосередженню моделі на найбільш значущих частинах тексту електронних повідомлень.

2.4. Інтеграція CNN та RNN із шаром уваги в гібридну модель

2.4.1. *Теоретичне обґрунтування комбінування CNN та BiLSTM*

Комбіновані гібридні архітектури, що поєднують згорткові та рекурентні нейронні мережі, отримали широке визнання в сучасних задачах обробки природномовних текстових даних. В контексті ідентифікації фішингових електронних листів така інтеграція є особливо перспективною, оскільки дозволяє об'єднати переваги кожної з мереж для ефективнішого аналізу тексту, забезпечуючи високу точність і стабільність роботи алгоритму [9].

Головною ідеєю використання комбінованої CNN-BiLSTM архітектури є поєднання двох фундаментально різних підходів до обробки тексту:

- Згорткові нейронні мережі ефективно виявляють локальні текстові патерни, такі як ключові слова, фрази чи короткі комбінації слів, що мають вирішальне значення для визначення ознак фішингових

повідомлень. CNN добре працюють у задачах, де важлива локальна контекстуальна інформація [10].

- Двонаправлені рекурентні нейронні мережі з довгою короткостроковою пам'яттю здатні враховувати глобальний контекст тексту, аналізуючи залежності між словами на великих дистанціях у послідовностях. BiLSTM дозволяють ефективно захоплювати семантику й загальний зміст повідомлень, що є особливо важливим для задач класифікації фішингових електронних листів, які можуть мати неоднорідну структуру та різноманітні стилі написання [23].

Поєднання CNN та BiLSTM дозволяє створити ефективний метод, де CNN відповідає за швидке й точне виявлення локальних семантичних шаблонів, а BiLSTM враховує ширший контекст, уточнюючи й доповнюючи отримані ознаки. При цьому додавання механізму уваги забезпечує модель здатністю визначати важливість ознак, згенерованих CNN та BiLSTM, що дозволяє акцентувати увагу саме на найзначущіших частинах тексту.

Переваги комбінованої CNN-BiLSTM архітектури можна сформулювати наступним чином:

1. *Висока точність виявлення фішингу* – завдяки тому, що CNN ідентифікує критичні локальні ознаки (наприклад, фішингові фрази, заклики до термінових дій тощо), а BiLSTM аналізує їх взаємозв’язок у контексті всього листа, модель забезпечує високий рівень точності і чутливості до різних типів фішингових повідомлень [17].
2. *Збалансована обробка локального та глобального контексту* – CNN швидко та ефективно працює з локальними характеристиками тексту, у той час як BiLSTM здатні зберігати інформацію про довгострокові залежності, що дозволяє моделі ефективно аналізувати тексти різного рівня складності та структурної неоднорідності.

3. *Стійкість до шуму та варіативності* – використання BiLSTM дає змогу пом'якшити вплив локального шуму, наприклад, випадкових символів або орфографічних помилок, що є типовими для реальних електронних листів. CNN у свою чергу додатково гарантують високу чутливість до конкретних фішингових патернів, що робить модель стійкою та адаптивною до різних типів атак.
4. *Підвищена інтерпретованість завдяки шару уваги* – інтегрований механізм уваги дозволяє отримувати інформацію про важливість окремих слів чи фраз у процесі прийняття рішення, що сприяє прозорості та легшій діагностиці роботи моделі.

Таким чином, гібридна CNN-BiLSTM архітектура з механізмом уваги є логічним та теоретично обґрунтованим вибором для реалізації автоматизованого методу ідентифікації фішингових електронних листів, що поєднує сильні сторони різних нейромережевих підходів і забезпечує високу якість та ефективність роботи алгоритму.

2.4.2. *Теоретична модель шару уваги в комбінованій архітектурі*

Детальніше розглянемо теоретичну модель шару уваги та його інтеграції з нейронними мережами в запропонованому методі. Механізм уваги є ключовим елементом інтегрованої гібридної архітектури CNN–BiLSTM, забезпечуючи ефективну взаємодію і агрегування ознак, отриманих з обох мереж. Основне завдання шару уваги полягає в тому, щоб виділити найбільш релевантні ознаки із векторних представлень слів, що були сформовані нейромережевими компонентами, і тим самим підвищити якість і точність прийняття рішення про класифікацію тексту.

Концептуально, механізм уваги дозволяє моделі «зосереджуватися» на тих частинах тексту, які найбільше впливають на кінцевий результат, ігноруючи або знижуючи значущість менш важливих фрагментів. Таким чином, увага не тільки покращує якість класифікації, але й додає прозорості

моделі, оскільки дозволяє зрозуміти, які саме фрагменти тексту були визначальними для прийняття рішення [12].

Шар уваги в комбінованій CNN–BiLSTM моделі можна описати наступним чином:

Нехай вихід CNN-компонента моделі позначається як:

$$C = [c_1, c_2, ..., c_n],\ c_i \in R^{d_c}, \tag{2.20}$$

де c_i – вектори ознак, отримані CNN для кожного локального фрагмента тексту (наприклад, n-грам чи послідовностей слів), d_c – розмірність цих ознак.

Відповідно, вихід BiLSTM-компонента моделі має вигляд:

$$H = [h_1, h_2, ..., h_m],\ h_j \in R^{d_h}, \tag{2.21}$$

де h_j – вихідні вектори прихованих станів BiLSTM, які відображають контекстуальні характеристики кожного слова в тексті, а d_h – їх розмірність.

Далі виконується операція інтеграції ознак CNN та BiLSTM. Для цього формується загальний набір ознак шляхом конкатенації виходів двох мереж:

$$X = [C; H] = [x_1, x_2, ..., x_{n+m}],\ x_k \in R^d, \tag{2.22}$$

де d – загальна розмірність конкатенованих векторів (як правило, $d = d_c = d_h$ для спрощення реалізації).

Після цього для набору ознак X обчислюється шар уваги, який дозволяє отримати узагальнений вектор ознак z. Процес уваги складається з наступних кроків:

1. Обчислення значень ваги уваги (англ. attention weights) для кожного елемента вектора x_k:

$$u_k = tanh\left(W_a x_k + b_a\right),\ u_k \in R^{d_a}, \tag{2.23}$$

де W_a – матриця ваг шару уваги, b_a – вектор зсуву, а d_a – розмірність шару уваги.

2. Отримання скалярних оцінок значущості елементів шляхом множення на вектор контексту уваги v_a:

$$a_k = \frac{\exp\left(u_k^T v_a\right)}{\sum_{t=1}^{n+m} \exp\left(u_t^T v_a\right)},\ \alpha_k \in [0,1],\ \sum_{k=1}^{n+m} a_k = 1, \tag{2.24}$$

де a_k – нормалізовані ваги, що показують важливість кожного елемента вектору ознак x_k.

3. Остаточне агрегування інформації з урахуванням значень ваги уваги:

$$z = \sum_{k=1}^{n+m} a_k x_k,\ z \in R^d, \tag{2.25}$$

де вектор z є агрегованим представленням тексту, що поєднує локальні патерни (CNN) та глобальний контекст (BiLSTM) із застосуванням уваги.

Отриманий вектор z використовується як вхідний сигнал для остаточного шару класифікації (наприклад, повнозв'язного шару з sigmoid-функцією для задачі бінарної класифікації):

$$y = \sigma\left(W_z z + b_z\right), \tag{2.26}$$

де σ – sigmoid-функція, W_z, b_z – параметри кінцевого класифікаційного шару.

Запропонована модель шару уваги дозволяє ефективно інтегрувати різнорідні типи ознак, підвищуючи точність класифікації та надаючи можливість інтерпретації прийнятих рішень. Використання уваги забезпечує модель здатністю виділяти найбільш важливі аспекти тексту, що критично важливо у складних задачах аналізу природномовних повідомлень, зокрема у виявленні фішингових електронних листів [13].

2.5. Обгрунтування вибору метрик оцінювання ефективності методу

Важливим етапом розробки комбінованого методу автоматизованого виявлення фішингових електронних листів є вибір релевантних метрик, які дозволять об'єктивно оцінити ефективність запропонованого рішення. Аналіз існуючих джерел свідчить, що для задач класифікації тексту, зокрема виявлення фішингу, найбільш показовими є такі метрики як Accuracy, Precision, Recall, F1-score, True Positive Rate (TPR), False Positive Rate (FPR), ROC-AUC та Confusion Matrix [28].

Розглянемо детально кожну з цих метрик, які було обрано для оцінки роботи запропонованого методу.

Accuracy (Точність)

Accuracy – це найпростіша метрика, яка показує частку правильно класифікованих електронних листів серед загальної кількості листів, що перевіряються:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (2.27)$$

де:

- *TP* (True Positive) – кількість коректно виявлених фішингових листів;
- *TN* (True Negative) – кількість коректно класифікованих як легітимні листів;
- *FP* (False Positive) – кількість легітимних листів, неправильно класифікованих як фішинг;
- *FN* (False Negative) – кількість фішингових листів, що були помилково класифіковані як легітимні.

Точність є важливою метрикою, проте недостатньою при незбалансованих класах, оскільки може створити хибне враження про ефективність моделі [28].

Precision (Точність позитивних прогнозів)

Precision дозволяє оцінити, яка частка електронних листів, класифікованих моделлю як фішинг, дійсно є фішинговими:

$$Precision = \frac{TP}{TP+FP}. \quad (2.28)$$

Ця метрика важлива для мінімізації кількості помилкових спрацьовувань, коли модель ідентифікує безпечний лист як фішинг.

Recall (Повнота)

Повнота оцінює здатність моделі виявляти максимальну кількість дійсно фішингових листів:

$$Recall = \frac{TP}{TP+FN}. \quad (2.29)$$

Повнота критично важлива у задачах безпеки, де пропуск фішингового листа (FN) може мати серйозні наслідки для користувача [29].

F1-score (F1-міра)

F1-міра є гармонійним середнім між точністю позитивних прогнозів та повнотою, що дозволяє збалансовано оцінювати модель, особливо у випадку незбалансованості класів:

$$F1 - score = 2 \bullet \frac{Precision \bullet Recall}{Precision + Recall}. \quad (2.30)$$

Ця метрика є важливою, оскільки вона враховує як точність, так і повноту одночасно, тому часто використовується для комплексної оцінки класифікаційних задач, включаючи ідентифікацію фішингу.

True Positive Rate (TPR)

True Positive Rate (також відомий як чутливість, Sensitivity або Recall) визначає частку правильно визначених позитивних випадків (фішингових листів):

$$TPR = \frac{TP}{TP+FN}. \quad (2.31)$$

Ця метрика важлива для аналізу ефективності методу у виявленні загроз, оскільки вища TPR означає краще розпізнавання небезпечних електронних листів.

False Positive Rate (FPR)

False Positive Rate показує частку помилково класифікованих як фішингові листів серед всіх насправді безпечних повідомлень:

$$FPR = \frac{FP}{FP+TN}. \quad (2.32)$$

Низьке значення FPR є важливим показником якості моделі, оскільки дозволяє уникати непотрібних тривог та витрат на обробку помилкових спрацьовувань.

ROC-крива та AUC (Area Under the Curve)

ROC-крива (від англ. Receiver Operating Characteristic) дозволяє візуально оцінити компроміс між чутливістю (TPR) і частотою хибних тривог (FPR) при різних значеннях порогових значень класифікації. AUC (площа під ROC-кривою) є кількісним показником якості моделі:

- Значення AUC близьке до 1 означає дуже високу ефективність моделі.
- Значення AUC близьке до 0.5 свідчить про неефективність моделі (класифікація близька до випадкового вибору).

Таким чином, ROC-AUC є зручним способом комплексно оцінити ефективність моделі в умовах варіативності порогових значень.

Confusion Matrix (Матриця помилок)

Матриця помилок є фундаментальною матрицею, яка наочно показує кількість правильно і неправильно класифікованих випадків за кожним класом окремо.

Ця матриця є основою для розрахунку всіх вищезгаданих метрик і дозволяє наочно проаналізувати, які саме помилки робить модель найчастіше, що є корисним для її подальшого покращення.

2.6. Висновки до другого розділу

У другому розділі було детально обґрунтовано запропонований комбінований метод автоматизованої ідентифікації фішингу в текстовій частині електронних листів. Здійснено формальну постановку задачі

класифікації тексту, що дозволило чітко визначити вхідні та вихідні дані, а також сформулювати цільову функцію для навчання нейронної мережі.

Обґрунтовано архітектуру запропонованого методу, яка базується на гібридному поєднанні згорткових нейронних мереж (CNN), двонаправлених рекурентних мереж (BiLSTM) та механізму уваги (Attention). Така архітектура дозволяє ефективно виявляти як локальні текстові шаблони, так і контекстуальні особливості повідомлень, що є важливим для забезпечення високої точності класифікації. Аналіз наукових досліджень підтверджує доцільність саме такого комбінованого підходу, оскільки CNN та BiLSTM взаємно доповнюють одне одного, а шар уваги забезпечує додаткову інтерпретованість результатів.

Окремо було розглянуто особливості використання лише текстового тіла електронних листів (body-only підхід). Цей підхід зменшує складність та ресурсомісткість моделі, одночасно зберігаючи високу ефективність класифікації. Літературний огляд підтверджує, що саме текстова складова містить ключові ознаки, необхідні для ідентифікації фішингових атак, тоді як додаткові метадані часто можуть знижувати точність через надмірну кількість шумових ознак.

Детально описано етапи попередньої обробки текстових даних, зокрема очищення, токенізацію, лематизацію та векторизацію тексту. Проведений теоретичний аналіз показує, що ці етапи є критично важливими для успішної роботи нейромережевої моделі. Було також обґрунтовано вибір алгоритму GloVe для векторизації слів, який має переваги перед Word2Vec у збереженні глобальних контекстних залежностей між словами.

Описано теоретичні аспекти роботи CNN, BiLSTM та механізму уваги в задачах NLP. Також здійснено теоретичне обґрунтування вибору метрик, релевантних для оцінки ефективності моделі. Серед основних обраних метрик – Accuracy, Precision, Recall, F1-score, ROC-AUC, True Positive Rate, False Positive Rate та Confusion Matrix. Вибір саме цих метрик дозволяє

всебічно та об'єктивно оцінити якість роботи запропонованої моделі для задачі автоматичного виявлення фішингу.

Таким чином, проведене теоретичне дослідження підтверджує перспективність запропонованого комбінованого підходу і дозволяє перейти до наступного етапу дослідження – практичної реалізації та експериментальної перевірки ефективності запропонованого методу.

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ МЕТОДУ ДЛЯ АВТОМАТИЗОВАНОЇ ІДЕНТИФІКАЦІЇ ФІШИНГОВОГО ВМІСТУ В ТЕКСТОВІЙ ЧАСТИНІ ЕЛЕКТРОННИХ ЛИСТІВ

3.1. Обґрунтування вибору технологій та засобів реалізації програмного забезпечення

3.1.1. *Аналіз критеріїв вибору технологій*

При розробці програмного забезпечення для автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів важливим є чіткий вибір відповідних технологій та інструментальних засобів. Це рішення має суттєвий вплив на швидкість розробки, ефективність навчання та продуктивність експлуатації кінцевої системи. Тому перед визначенням остаточного технологічного стека було проведено аналіз низки критеріїв, які найбільше відповідають специфіці поставленої задачі та забезпечують якісну роботу готового рішення.

Першим важливим критерієм вибору технологічного стека є продуктивність і масштабованість системи. Оскільки автоматизована система виявлення фішингових листів призначена для обробки значних обсягів вхідних даних, вона повинна швидко обробляти та класифікувати великі масиви текстових повідомлень. Це ставить завдання використовувати такі технології, які підтримують ефективні механізми паралельної та асинхронної обробки даних, забезпечують високу продуктивність при роботі з великими обсягами інформації та дозволяють легко масштабувати систему у разі зростання навантаження.

Другим критерієм, який має бути враховано, є сумісність технологій з актуальними методами та інструментами машинного навчання та обробки природномовних текстових даних. Сучасні технології машинного навчання, такі як глибокі нейронні мережі, вимагають використання спеціалізованих бібліотек та фреймворків, які забезпечують ефективну роботу з текстовими векторами, згортковими та рекурентними шарами, а також механізмами

уваги. Отже, для ефективної реалізації запропонованого комбінованого методу необхідно обрати перевірені та поширені бібліотеки, що надають широкий спектр інструментів для роботи з текстовими даними та глибокими нейронними мережами.

Третій важливий критерій – зручність розроблення, підтримки та інтеграції програмного рішення. Розроблення програмного забезпечення має бути не лише технічно ефективним, але й практичним для подальшого підтримання та оновлення. Це включає простоту написання коду, доступність навчальних матеріалів та документації, зручність налаштування та інтеграції з іншими компонентами корпоративної інформаційної системи. Вибір зручних інструментів з добре документованими API та активною спільнотою дозволить швидше реалізовувати нові функції, оперативно виправляти помилки та адаптувати систему під нові вимоги й стандарти безпеки.

На основі зазначених критеріїв наступні підпункти розділу детально обґрунтовують обрані мови програмування, бібліотеки та додаткові інструменти, які були використані для ефективної та якісної реалізації запропонованого методу автоматизованого виявлення фішингових електронних листів.

3.1.2. *Обґрунтування вибору мови програмування та бібліотек*

Для розроблення систем автоматизованого виявлення фішингового вмісту доцільним є застосування різних мови програмування, серед яких найбільш популярними є Python, Java, R та C++. У контексті задач обробки природномовних текстових даних та глибокого навчання найбільш широко застосовується саме Python завдяки низці переваг, підтверджених науковими дослідженнями та практичним досвідом реалізації подібних систем [30].

Основними перевагами Python для розробки NLP-систем на основі нейронних мереж є:

1. *Читабельність і простота коду.* Python має лаконічний і зрозумілий синтаксис, що значно зменшує складність розробки,

підтримки та тестування складних систем, дозволяючи швидко проводити експерименти з різними підходами та архітектурами нейромереж [31].

2. *Широка екосистема спеціалізованих бібліотек.* Python забезпечує доступ до великої кількості популярних NLP та Deep Learning бібліотек (TensorFlow, Keras, PyTorch, SpaCy, NLTK, Gensim), які надають необхідні засоби для швидкої і ефективної побудови NLP-моделей різної складності [32].
3. *Активна науково-інженерна спільнота.* Python використовується як в академічних, так і в комерційних проєктах, що сприяє наявності широкого спектру документації, посібників, наукових статей і відкритого коду, які значно прискорюють розв'язання складних технічних завдань [33].
4. *Інтеграція з сучасними технологіями.* Python легко інтегрується з базами даних, хмарними платформами (AWS, Azure, Google Cloud) та веб-інтерфейсами, що є критично важливим для побудови комплексних систем автоматизованого виявлення фішингу з можливістю подальшого розгортання [34].
5. *Високопродуктивні обчислення та масштабованість.* Попри простоту мови, Python здатний ефективно використовувати графічні процесори, підтримує розподілені обчислення завдяки бібліотекам TensorFlow і PyTorch, що забезпечує високошвидкісне тренування нейромережевих моделей на великих наборах даних [35].

Серед найбільш популярних бібліотек для глибокого навчання, які активно застосовуються у розробці NLP-рішень, виділяють TensorFlow/Keras і PyTorch [36].

Фреймворки TensorFlow та Keras пропонують інтуїтивно зрозумілий і простий API, що значно полегшує розробку моделей, мають розвинену екосистему, високу продуктивність та стабільність. Ці інструменти підходять

як для швидкого прототипування, так і для побудови комерційних застосунків, які потребують надійності та масштабованості [37].

З іншого боку, бібліотека PyTorch надає більшу гнучкість завдяки використанню динамічних обчислювальних графів, що зручно для наукових досліджень, експериментів і швидкої модифікації архітектури нейромереж. Проте, у випадку комерційної розробки, особливо орієнтованої на стабільність, інтегрованість та швидке розгортання, PyTorch дещо поступається TensorFlow/Keras [38].

З огляду на вище викладене, для реалізації запропонованого методу автоматизованого виявлення фішингових електронних листів було обрано комбінацію TensorFlow/Keras, що забезпечує оптимальний баланс простоти використання, стабільності, продуктивності та масштабованості розробленого рішення [39].

3.1.3. *Вибір засобів обробки текстових даних*

Вибір відповідних інструментів для попередньої обробки текстових даних є критично важливим для забезпечення високої точності роботи моделі, яка реалізує метод автоматизованої ідентифікації фішингових електронних листів. Для реалізації попередньої обробки було здійснено аналіз популярних бібліотек для задач NLP, а також спеціалізованих бібліотек для роботи з векторними представленнями слів.

На першому етапі аналізувалися популярні бібліотеки для природної обробки мови – NLTK і SpaCy. Бібліотека NLTK (англ. Natural Language Toolkit) широко використовується для навчальних та дослідницьких задач завдяки її великому набору модулів для токенізації, стемінгу, лематизації, частиномовного тегування та інших базових NLP-задач [40]. Однак вона характеризується нижчою продуктивністю та складнішою інтеграцією з іншими сучасними бібліотеками машинного навчання [41].

Натомість бібліотека SpaCy була розроблена саме з акцентом на продуктивність та інтеграцію з сучасними системами машинного навчання.

Основними перевагами SpaCy є висока швидкість роботи, сучасна архітектура, точні моделі лематизації, токенізації та частиномовного тегування, а також можливість ефективно обробляти великі обсяги текстових даних паралельно. Враховуючи ці переваги, для реалізації попередньої обробки текстів було обрано саме SpaCy [42].

На другому етапі було проведено аналіз бібліотек, що призначені для роботи з текстовими даними у вигляді ембедінгів. Серед популярних бібліотек було виділено Gensim, яка надає широкий спектр інструментів для роботи з векторними моделями слів (Word2Vec, FastText, GloVe тощо). Основними перевагами Gensim є ефективна робота з векторними просторами, можливість швидко завантажувати попередньо натреновані моделі, а також висока швидкість і оптимізованість операцій із великими векторними масивами [43].

Для роботи з попередньо натренованими ембедінгами GloVe було використано саме Gensim, оскільки ця бібліотека дозволяє легко завантажувати, зберігати та використовувати ембедінги різних розмірностей (25, 50, 100, 200), що було особливо важливим у контексті гнучкого налаштування параметрів тренування моделі.

Крім того, для зручності та ефективності роботи з масивами даних у процесі попередньої обробки було обрано бібліотеки NumPy та Pandas. NumPy забезпечує швидку і зручну роботу з багатовимірними числовими масивами, що є ключовим для ефективної роботи з векторними представленнями текстових даних. Pandas, своєю чергою, використовується для зручної роботи з табличними даними, забезпечує ефективне завантаження, очищення, попередню обробку й аналіз структурованих даних, таких як текстові набори даних із мітками [44].

Таким чином, на основі порівняльного аналізу та враховуючи вимоги до продуктивності, гнучкості й ефективності було обрано наступні бібліотеки та інструменти для попередньої обробки текстових даних:

- SpaCy – для токенізації, лематизації та інших NLP-операцій;

- Gensim – для роботи з GloVe-ембедінгами різних розмірностей;
- NumPy і Pandas – для ефективного маніпулювання даними в процесі обробки та підготовки до тренування нейромережевої моделі.

Використання саме цього набору бібліотек забезпечує необхідну продуктивність, точність обробки, а також гнучкість налаштувань моделі.

3.1.4. *Інструменти візуалізації та логування*

Процес розробки та експериментального дослідження нейромережевих моделей передбачає обов'язкове застосування ефективних засобів візуалізації та моніторингу процесу навчання і тестування. Це дозволяє виявити проблеми моделі, проаналізувати її поведінку, а також спростити пошук оптимальних параметрів.

Одним із основних інструментів логування та моніторингу процесу навчання, який було обрано для реалізації програмного забезпечення, є TensorBoard. TensorBoard – це спеціалізований вебінструмент, інтегрований у TensorFlow, який забезпечує зручний моніторинг різноманітних показників навчання нейромережевих моделей у реальному часі. Серед основних переваг TensorBoard є зручний і наочний вебінтерфейс, що дозволяє спостерігати за динамікою зміни основних метрик (точність, втрати, коефіцієнти навчання), порівнювати різні експерименти, а також візуалізувати архітектуру моделі, графіки розподілу ваг та активацій шарів [45].

Особливою перевагою TensorBoard є можливість паралельно порівнювати результати різних експериментів, що значно спрощує вибір оптимальної архітектури моделі та налаштувань гіперпараметрів. Крім того, TensorBoard дозволяє контролювати проблему перенавчання на основі порівняння тренувальних та валідаційних кривих втрат та точності.

Другим інструментом, який активно використовується для візуалізації даних і результатів експериментів, є бібліотека Matplotlib. Ця бібліотека надає широкі можливості для створення різноманітних графіків і діаграм, зокрема

графіків точності та втрат моделі на кожній епосі, ROC-кривих, матриць та гістограм. Matplotlib має значну гнучкість налаштувань, дозволяючи адаптувати візуалізацію під конкретні дослідницькі та наукові цілі [46].

Для реалізації програмного забезпечення було визначено використання Matplotlib для збереження графіків метрик ефективності в окремі файли зображень, що значно спрощує підготовку звітів і документації за результатами експериментів. Також за допомогою Matplotlib було забезпечено детальну візуалізацію результатів класифікації на тестових наборах даних у вигляді ROC-кривих, confusion matrix, що суттєво підвищує інтерпретованість результатів роботи розробленої системи.

Отже, для ефективної візуалізації та моніторингу результатів процесу розробки було обрано наступні інструменти:

- *TensorBoard* – для логування процесу навчання, порівняння експериментів, візуалізації архітектури моделі та динаміки метрик.
- *Matplotlib* – для детальної візуалізації результатів експериментів у вигляді графіків метрик навчання, ROC-кривих та confusion matrix.

Використання цих інструментів забезпечує необхідний рівень прозорості, інтерактивності й наочності процесу експериментальних досліджень, що дозволяє суттєво прискорити розробку ефективної та надійної програмної реалізації методу автоматизованої ідентифікації фішингу в текстовій частині електронних листів.

3.2. Архітектура програмного забезпечення

3.2.1. *Загальна схема програмного рішення*

Структура програмної системи, призначеної для автоматизованої ідентифікації фішингового вмісту у текстовій частині електронних листів, була розроблена з урахуванням ключових вимог, визначених у попередніх розділах. Головними цілями цієї архітектури є забезпечення високої точності, масштабованості та зручності експлуатації, а також можливості гнучкого налаштування під конкретні задачі та умови застосування (рис. 3.1).

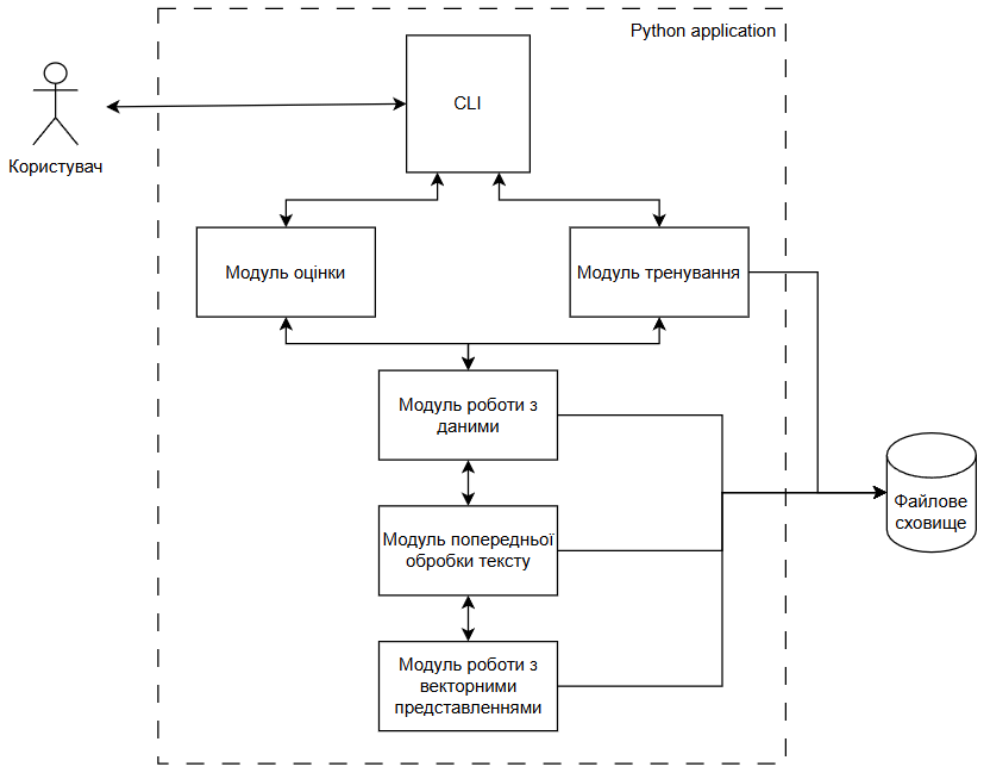


Рис. 3.1. Архітектура розробленого програмного забезпечення

Концептуально система складається з чотирьох основних функціональних компонентів та двох допоміжних, що взаємодіють між собою у процесі роботи.

Опишемо основні модулі:

1. Модуль попередньої обробки.
2. Модуль векторизації.
3. Модуль тренування моделі.
4. Модуль оцінювання та аналізу результатів.

Модуль попередньої обробки відповідає за очищення та підготовку текстових даних електронних листів. Тексти проходять процедуру видалення спеціальних символів, HTML-тегів, URL-адрес, після чого здійснюється токенізація та лематизація слів за допомогою бібліотеки SpaCy. Результатом роботи модуля є список нормалізованих токенів, які далі надходять до модуля векторизації.

Модуль векторизації здійснює перетворення нормалізованих текстових даних у чисельний формат, зрозумілий нейромережевій моделі. В якості методу векторизації використовується алгоритм GloVe з попередньо натренованими ембедінгами на основі великого корпусу Twitter-даних. В результаті цього процесу текст електронного листа представляється у вигляді матриці чисел фіксованої довжини, яка використовується як вхідний сигнал для гібридної моделі.

Гібридна модель CNN-BiLSTM-Attention є центральним компонентом системи, який відповідає за ідентифікацію фішингового контенту. Вона складається з трьох ключових елементів:

- CNN-шари, які здійснюють виявлення локальних текстових патернів та специфічних слів, що характерні для фішингових повідомлень.
- Двонаправлені LSTM-шари (BiLSTM), які дозволяють ефективно аналізувати контекстні взаємозв'язки між словами у тексті, забезпечуючи глибше розуміння семантики.
- Шар уваги (Attention), який виконує фокусування моделі на найбільш важливих з погляду класифікації частинах тексту, посилюючи чутливість системи до найбільш інформативних фрагментів.

Вихідним результатом роботи моделі є ймовірність належності тексту електронного листа до фішингового класу. На основі заданого порогового значення приймається остаточне рішення про класифікацію повідомлення.

Модуль оцінювання та аналізу результатів використовується для тестування та валідації моделі на різноманітних наборах даних. Цей компонент відповідає за підрахунок основних метрик ефективності роботи моделі, таких як точність (Accuracy), точність за класами (Precision, Recall, F1-score), а також створення графічних звітів (ROC-кривих, Confusion Matrix). Також цей модуль дозволяє проводити експериментальні порівняння моделей, тренованих з різними наборами гіперпараметрів.

Схема процесу навчання та оцінювання моделі виглядає таким чином:

- Етап навчання:
  - Завантаження та попередня обробка навчального набору даних.
  - Векторизація текстових даних навчального набору.
  - Навчання моделі CNN-BiLSTM-Attention із періодичним збереженням найкращих контрольних точок і фінальної моделі.
- Етап оцінювання:
  - Завантаження моделі та попередньо обробленого тестового набору даних.
  - Векторизація тестових даних.
  - Генерація прогнозів та розрахунок метрик ефективності.
  - Візуалізація результатів у вигляді графіків і матриць для аналізу.

Таким чином, запропонована архітектура системи дозволяє забезпечити повний цикл роботи нейромережевої моделі: від підготовки текстових даних до остаточного аналізу результатів її роботи. Завдяки такій модульній структурі досягається висока зручність підтримки, масштабованість рішення та гнучкість у налаштуванні під конкретні умови використання, що є критично важливим для ефективного застосування розробленої системи у реальних умовах.

3.2.2. *Структура директорій та організація коду*

Для забезпечення високого рівня підтримуваності та зручності розробки, реалізоване програмне забезпечення має чітко визначену структуру директорій і модульну організацію коду. В основі структури проєкту лежать принципи модульності та ізоляції функціоналу, що полегшує тестування, розширення та подальшу підтримку системи.

Структура директорій проєкту зображена на рис. 3.2.

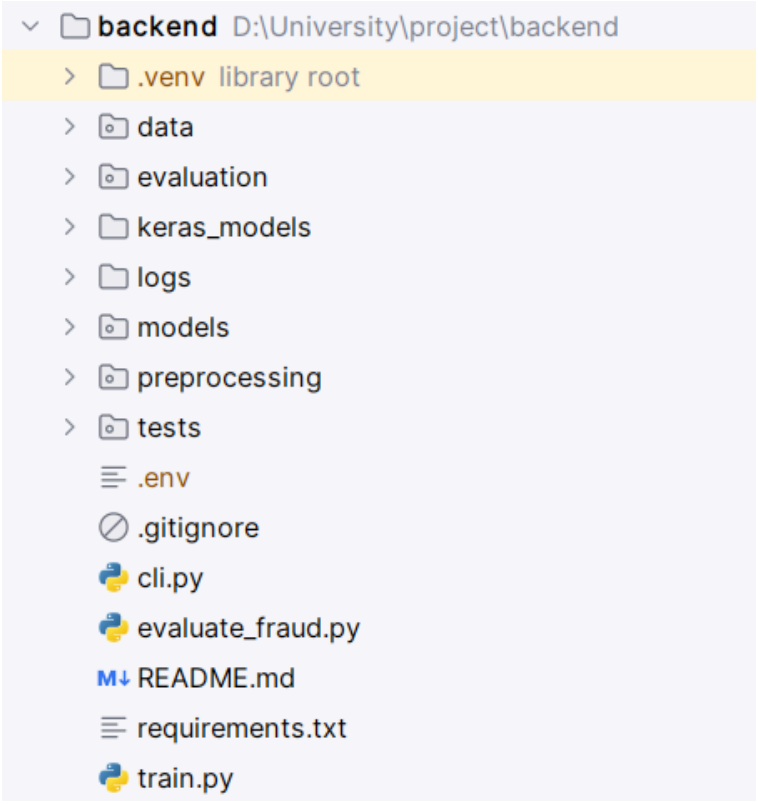


Рис. 3.2. Структура директорій проєкту

Призначення основних модулів та файлів проєкту:

- *cli.py* – головний скрипт консольного інтерфейсу, який забезпечує користувачу можливість запускати процеси навчання та оцінювання моделей із вибором параметрів (наприклад, розмірів ембедінгів, епох навчання тощо).
- *train.py* – відповідає за запуск процесу тренування нейронних мереж з можливістю налаштування гіперпараметрів (максимальна довжина тексту, розмір ембедінгів, кількість епох).
- *evaluate_fraud.py* – дозволяє завантажити збережену модель і протестувати її на спеціалізованому наборі даних для оцінки точності класифікації фішингових листів.

- *preprocessing/* – містить повний набір засобів попередньої обробки природномовних даних (очищення, токенізація, лематизація, ембедінги).
- *models/* – включає реалізацію нейронних моделей, серед яких CNN, RNN (BiLSTM), гібридна модель із шаром уваги та трансформерні енкодери (додатковий експериментальний функціонал).
- *data/* – модуль для завантаження та підготовки даних, які використовуються у процесі навчання та оцінювання моделей.
- evaluation/ – містить реалізацію та обчислення необхідних метрик для оцінки ефективності моделей.
- *tests/* – директорія з юніт-тестами для кожного ключового модуля, що дозволяє автоматизовано перевіряти коректність роботи коду та забезпечувати стабільність розробки.
- *logs/* – директорія для логування процесу навчання моделей за допомогою TensorBoard. Включає графіки метрик, які дозволяють візуально оцінити ефективність навчання моделей у часі.
- *keras_models/* – директорія для збереження натренованих моделей у форматі Keras, включаючи моделі з різними комбінаціями гіперпараметрів.

Обрана структура директорій і модулів забезпечує прозору та логічну організацію проєкту, що суттєво спрощує командну роботу, а також подальше масштабування і підтримку системи. Завдяки розподіленню коду між різними модулями, система легко модифікується й розширюється, що дозволяє швидко реагувати на зміну вимог або додавання нового функціоналу.

Для забезпечення чіткої організації, зручності підтримки та можливості ефективного масштабування, реалізоване програмне забезпечення структуровано у вигляді декількох основних модулів, кожен з яких відповідає за конкретну функціональну частину процесу автоматизованої ідентифікації фішингових електронних листів. Нижче надано детальний опис кожного з

ключових компонентів системи, що забезпечують повний цикл роботи – від обробки вхідних даних до оцінки ефективності нейромережевої моделі.

3.2.3. *Детальний опис компонентів системи*

Модуль попередньої обробки тексту (preprocessing) відповідає за підготовку текстових даних до подальшої обробки нейромережею. Він включає компоненти:

- text_cleaning.py – здійснює очищення тексту, видаляючи HTML-теги, URL-адреси, спеціальні символи та інший інформаційний «шум».
- nlp_preprocessing.py – реалізує токенізацію та лематизацію за допомогою бібліотеки SpaCy, що дозволяє отримати нормалізовані лексичні одиниці [47].
- nlp_pipeline.py – об'єднує очищення, токенізацію і лематизацію у єдиний конвеєр для послідовної обробки великих обсягів даних.

Завдяки цьому модулю забезпечується стандартизована попередня обробка текстових даних, необхідна для ефективного навчання моделі.

Модуль векторизації тексту відповідає за перетворення попередньо обробленого тексту в чисельні векторні представлення (ембедінги). Для цієї задачі було обрано GloVe-ембедінги, попередньо натреновані на текстових даних з Twitter (розмірністю 25, 50, 100 та 200). Функції модуля забезпечують завантаження ембедінгів і трансформацію текстових послідовностей у вектори необхідної форми та розміру, що дозволяє зберігати семантичні особливості текстів для подальшого аналізу.

Модуль гібридної моделі реалізує основний компонент, що використовується для класифікації текстів. Гібридна модель поєднує такі компоненти:

- CNN (cnn.py) – відповідає за виявлення локальних текстових патернів завдяки використанню згорткових шарів із ядрами різних розмірів (3, 4, 5).

- BiLSTM (rnn.py) – здійснює аналіз глобального контексту тексту у двох напрямках, що дозволяє враховувати як попередню, так і наступну інформацію в тексті.
- Attention (hybrid_model.py) – дозволяє моделі зосередитись на найбільш релевантних токенах, надаючи їм більшу вагу при формуванні остаточного рішення.

Інтеграція цих компонентів дозволяє значно підвищити якість класифікації за рахунок комплексного аналізу тексту на різних рівнях, локальному та глобальному.

Модуль навчання моделі організовує процес тренування моделі, забезпечуючи:

- завантаження тренувальних та валідаційних даних за допомогою модуля data/dataset_loader.py;
- попередню обробку та векторизацію даних;
- налаштування параметрів навчання (розміри ембедінгів, кількість епох, розміри пакетів);
- тренування та збереження моделей з унікальними назвами, що містять значення гіперпараметрів для зручності оцінювання.

Модуль оцінювання ефективності відповідає за тестування натренованих моделей та оцінювання їх ефективності на тестових наборах. Він включає:

- завантаження моделей із каталогу keras_models;
- класифікацію тестових даних і розрахунок метрик точності, повноти, F1-міри, TPR та FPR;
- створення звітів та графіків (ROC-крива, Confusion Matrix), які наочно демонструють ефективність моделі.

Модуль обчислення метрик забезпечує розрахунок основних метрик оцінювання моделей, таких як Accuracy, Precision, Recall, F1-score, TPR, FPR, а також побудову графічних звітів (ROC-крива, Confusion Matrix). Це дозволяє проводити ґрунтовний аналіз результатів роботи моделі.

Модуль завантаження даних забезпечує зручне завантаження тренувальних та тестових наборів даних із різних джерел, що необхідно для проведення експериментів і оцінювання роботи моделей.

Модуль тестування включає набори юніт-тестів для основних компонентів (CNN, RNN, hybrid_model, preprocessing, embedding тощо), які допомагають швидко виявляти помилки, підтримувати стабільність роботи програмного комплексу та значно спрощують подальшу розробку і модифікацію системи.

Таким чином, структура програмного забезпечення дозволяє легко підтримувати, модифікувати та масштабувати систему, забезпечуючи при цьому високу ефективність та точність роботи нейромережевої моделі з автоматизованого виявлення фішингових листів.

3.3. Особливості реалізації програмного забезпечення

3.3.1. *Реалізація попередньої обробки текстових даних*

Попередня обробка текстових даних є важливим етапом реалізації методу ідентифікації фішингового вмісту, оскільки якість підготовки тексту прямо впливає на кінцеву ефективність класифікації. У розробленій системі процес попередньої обробки реалізовано у вигляді модульного конвеєра, що охоплює очищення тексту, токенізацію, видалення стоп-слів та лематизацію.

Очищення тексту виконується функціями, визначеними у модулі text_cleaning.py, які послідовно видаляють HTML-теги та спеціальні символи. Нижче наведено відповідний фрагмент коду (лістинг 3.1).

Лістинг 3.1. Функції очищення тексту

```
{
    import re

    def clean_html(text: str) -> str:
        html_tag_pattern = re.compile(r'<.*?>')
        return re.sub(html_tag_pattern, '', text)

    def remove_special_characters(text: str) -> str:
        text = re.sub(r'[^A-Za-z0-9\s]', ' ', text)
        text = re.sub(r'\s+', ' ', text)
```

```
        return text.strip()

    def clean_text(text: str) -> str:
        text = clean_html(text)
        text = remove_special_characters(text)
        return text
}
```

Таким чином, з текстів електронних листів вилучаються HTML-теги, URL-адреси та спеціальні символи, залишаючи тільки осмислені слова, що покращує якість токенізації та лематизації.

Наступний етап попередньої обробки здійснюється за допомогою NLP-бібліотеки spaCy у модулі nlp_pipeline.py. Для цього застосовується модель “en_core_web_md”. Вхідний текст токенізується, очищується від стоп-слів та лематизується (лістинг 3.2).

Цей етап дозволяє скоротити обсяг словника, який використовується у моделі, і підвищує точність класифікації за рахунок узагальнення форм слів до їхніх базових лем.

Лістинг 3.2. Токенізація, видалення стоп-слів та лематизація

```
{
    import spacy
    from typing import List

    nlp = spacy.load("en_core_web_md")

    def remove_stopwords_and_lemmatize(tokens: List[str]) ->
List[str]:
        processed = [token.lower() for token in tokens if
    token.strip()]
        doc = nlp(" ".join(processed))
        final_tokens = [tok.lemma_.lower() for tok in doc if not
tok.is_stop and not tok.is_punct and tok.lemma_.strip()]
        return final_tokens

    def advanced_nlp_processing(text: str) -> List[str]:
        doc = nlp(text)
        tokens = [token.text for token in doc]
        return remove_stopwords_and_lemmatize(tokens)

}
```

Для зручності процес очищення, токенізації та лематизації поєднується у єдиний конвеєр, реалізований у файлі nlp_preprocessing.py. Цей програмний код є точкою входу для всієї процедури попередньої обробки (лістинг 3.3).

Лістинг 3.3. Комплексний конвеєр попередньої обробки тексту

```
{
      from preprocessing.text_cleaning import clean_text
      from preprocessing.nlp_pipeline import advanced_nlp_processing

      def preprocess_text(text: str) -> list:
            cleaned_text = clean_text(text)
            final_tokens = advanced_nlp_processing(cleaned_text)
            return final_tokens


}
```

Таким чином, запропонований конвеєр попередньої обробки забезпечує ретельну очистку, токенізацію та нормалізацію вхідних текстів, що є ключовою передумовою високої ефективності нейромережевої моделі виявлення фішингових електронних листів.

3.3.2. *Реалізація векторизації тексту*

Векторизація тексту є критичним етапом у підготовці даних для нейромережевої моделі, яка потребує числового представлення вхідних текстів. У розробленій системі для векторизації використано заздалегідь натреновані ембедінги GloVe, побудовані на основі корпусу Twitter. Реалізація передбачає завантаження ембедінгів різної розмірності (25, 50, 100, 200), що дозволяє проводити експерименти з різними гіперпараметрами та вибирати оптимальну точність для конкретних задач.

Завантаження ембедінгів здійснюється модулем embedding.py, який динамічно визначає розмірність ембедінгів відповідно до заданих параметрів (лістинг 3.4).

Лістинг 3.4. Завантаження та векторизація тексту

```
{
import os
```

```
import numpy as np
from gensim.models.keyedvectors import KeyedVectors
import gensim.downloader as api

_model_cache: dict[int, dict | KeyedVectors | None] = {}

def load_local_glove(glove_path: str, dim: int) -> dict[str,
np.ndarray]:
    if not os.path.exists(glove_path):
        raise FileNotFoundError(f"Local GloVe file not found at
{glove_path}")
    emb = {}
    with open(glove_path, "r", encoding="utf-8") as f:
        for line in f:
            parts = line.rstrip().split(" ")
            word, vec = parts[0], np.asarray(parts[1:],
dtype="float32")
            if vec.shape[0] == dim:
                emb[word] = vec
    return emb

def _load_model_for_dim(dim: int):
    base = os.path.expanduser("../data/twitter-archive")
    glove_path = os.path.join(base, f"glove.twitter.27B.{dim}d.txt")
    try:
        return load_local_glove(glove_path, dim)
    except Exception:
        return api.load(f"glove-twitter-{dim}")

def _get_model():
dim = int(os.getenv("EMBEDDING_DIM", 100))
    if dim not in _model_cache:
        _model_cache[dim] = _load_model_for_dim(dim)
    return _model_cache[dim]

def vectorize_text(tokens: list[str], max_len: int = 50) ->
np.ndarray:
    model = _get_model()
    dim = int(os.getenv("EMBEDDING_DIM", 100))
    mat = np.zeros((max_len, dim), dtype="float32")
    if model is None:
        return mat

    if isinstance(model, KeyedVectors):
        for i, tok in enumerate(tokens[:max_len]):
            if tok in model:
                mat[i] = model[tok]
    else:
        for i, tok in enumerate(tokens[:max_len]):
            v = model.get(tok)
            if v is not None:
                mat[i] = v
    return mat
}
```

У цьому фрагменті ембедінги завантажуються з локального файлу або, якщо локальний файл недоступний, з репозиторію Gensim. Це забезпечує гнучкість системи і дозволяє без проблем проводити експерименти з різними векторними розмірами, що впливає на швидкодію та точність моделі.

Для завантаження та підготовки текстових наборів даних використовується модуль dataset_loader.py. Цей модуль здійснює читання та об'єднання різнорідних наборів даних, таких як SpamAssassin, Enron, CEAS тощо. Приклад реалізації наведено у лістингу 3.5.

Лістинг 3.5. Завантаження та об'єднання тренувальних наборів даних

```
{
import os
import pandas as pd

def load_training_dataset(data_dir: str) -> pd.DataFrame:
    configs = [
        ("SpamAssasin.csv", ("subject", "body")),
        ("Nigerian_Fraud.csv", ("subject", "body")),
        ("CEAS_08.csv", ("subject", "body")),
        ("Enron.csv", ("subject", "body"))
        ("Ling.csv", ("subject", "body")),
        ("Nazario.csv", ("subject", "body")),
        ("phishing_email.csv", ("text_combined",))
    ]
    dfs = []
    for fname, cols in configs:
        path = os.path.join(data_dir, fname)
        df = pd.read_csv(path, encoding="latin-1", engine="python",
on_bad_lines="skip")
        if cols == ("text_combined",):
            sub = df[["text_combined",
"label"]].rename(columns={"text_combined": "text"})
        else:
            subj, body = cols
            df[body] = df[body].fillna("")
            df["text"] = df[subj] + " " + df[body]
            sub = df[["text", "label"]]

        sub = sub[sub["text"].apply(lambda x: isinstance(x, str) and
x.strip() != "")]
        dfs.append(sub)

    combined = pd.concat(dfs, ignore_index=True)
    combined["label"] = pd.to_numeric(combined["label"],
errors="coerce").dropna().astype(int)
    combined = combined.sample(frac=1.0,
random_state=42).reset_index(drop=True)
    return combined
}
```

3.3.3. *Реалізація гібридної моделі CNN-BiLSTM-Attention*

Для реалізації комбінованого методу автоматизованого виявлення фішингу було розроблено гібридну модель, що інтегрує згорткові нейронні мережі (CNN), двонаправлені рекурентні нейронні мережі типу LSTM (BiLSTM) та механізм уваги. Модель побудована з використанням бібліотеки Keras з бекендом TensorFlow. Головною перевагою даного підходу є здатність одночасно враховувати локальні патерни слів завдяки CNN та контекстуальні особливості за допомогою BiLSTM. Механізм уваги дозволяє додатково виділити найбільш інформативні ознаки з обох гілок моделі та поєднати їх у спільний представлений простір.

Ключові компоненти гібридної моделі визначені в модулі hybrid_model.py. Нижче наведено лістинг коду, який реалізує повну архітектуру моделі (лістинг 3.6).

Лістинг 3.6. Реалізація гібридної моделі CNN-BiLSTM-Attention

```
{
def build_hybrid_model(input_shape, cnn_filter_sizes=[3, 4, 5],
cnn_num_filters=64,
                       rnn_type="lstm", rnn_units=64,
fusion_dense_units=64,
                       dropout_rate=0.5):
    inputs = Input(shape=input_shape, name="hybrid_input")

    # Гілка CNN
    cnn_model = build_cnn_model(
        input_shape=input_shape,
        filter_sizes=cnn_filter_sizes,
        num_filters=cnn_num_filters,
        dense_units=128
    )
    cnn_features = cnn_model(inputs)

    # Гілка RNN (BiLSTM)
    rnn_model = build_rnn_model(
        input_shape=input_shape,
        rnn_type=rnn_type,
        units=rnn_units,
        dense_units=128
    )
    rnn_features = rnn_model(inputs)

    # Шар уваги для інтеграції CNN та RNN
    fused_features = AttentionFusion(common_dim=128,
name="attention_fusion")([cnn_features, rnn_features])
```

```
    fusion_dropout = Dropout(rate=dropout_rate,
name="fusion_dropout")(fused_features)
    fusion_dense = Dense(units=fusion_dense_units, activation='relu',
name="fusion_dense")(fusion_dropout)

    output = Dense(1, activation='sigmoid',
name="output")(fusion_dense)

    hybrid_model = Model(inputs=inputs, outputs=output,
name="hybrid_model")
    hybrid_model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

    return hybrid_model
}
```

У представленій реалізації гібридної моделі використовується модульна структура: окремі компоненти (CNN, BiLSTM, AttentionFusion) реалізовані як незалежні блоки, що полегшує їхню підтримку та модифікацію. CNN-компонент моделі будується за допомогою функції build_cnn_model, яка визначена в модулі cnn.py. Він має кілька паралельних шарів з різними розмірами фільтрів, які дозволяють виявляти локальні семантичні ознаки в тексті.

RNN-компонент реалізований за допомогою функції build_rnn_model (модуль rnn.py), який використовує LSTM-шар для аналізу послідовності слів, зберігаючи контекстні залежності, що важливо при роботі з природномовними текстами.

Особливу роль у інтеграції двох незалежних потоків інформації (CNN та RNN) відіграє шар уваги (AttentionFusion). Цей шар представлений у вигляді окремого класу, який проектує вихідні дані CNN та BiLSTM в єдиний простір ознак, після чого обчислює вагові коефіцієнти (attention weights) для кожної гілки (лістинг 3.7).

Лістинг 3.7. Реалізація механізму уваги

```
{
class AttentionFusion(Layer):
    def __init__(self, common_dim=128, **kwargs):
        super(AttentionFusion, self).__init__(**kwargs)
        self.common_dim = common_dim
```

```
        self.cnn_projection = Dense(common_dim, activation='relu',
name="cnn_projection")
        self.rnn_projection = Dense(common_dim, activation='relu',
name="rnn_projection")
        self.attention_dense = Dense(1, activation='tanh',
name="attention_dense")

    def call(self, inputs, **kwargs):
        cnn_out, rnn_out = inputs
        cnn_proj = self.cnn_projection(cnn_out)
        rnn_proj = self.rnn_projection(rnn_out)

        stacked = tf.stack([cnn_proj, rnn_proj], axis=1)
        attention_scores = self.attention_dense(stacked)
        attention_scores = tf.squeeze(attention_scores, axis=-1)

        attention_weights = tf.nn.softmax(attention_scores, axis=1)
        attention_weights = tf.expand_dims(attention_weights,
axis=-1)

        weighted_sum = tf.reduce_sum(stacked * attention_weights,
axis=1)
        return weighted_sum

}
```

Цей механізм уваги дозволяє моделі адаптивно обирати найбільш значущі ознаки з кожного компонента, що сприяє зростанню точності класифікації. В результаті модель ефективно використовує як локальні патерни, так і контекстуальні ознаки для надійного виявлення фішингових листів.

Гібридна нейромережева архітектура з використанням CNN, BiLSTM та шару уваги підтверджує свою ефективність у задачах аналізу природномовних текстів, демонструючи переваги в точності порівняно з однорідними нейромережевими архітектурами.

3.3.4. *Особливості консольного інтерфейсу*

Для зручного керування процесами навчання та оцінювання гібридної моделі було реалізовано консольний інтерфейс користувача (CLI). Цей інтерфейс забезпечує простий спосіб взаємодії з розробленою системою через термінал, дозволяючи задавати необхідні параметри та виконувати ключові операції.

Інтерфейс розроблено у вигляді меню, де користувач може вибрати одну з трьох опцій:

- навчання нової моделі;
- оцінювання моделі на наборі даних із шахрайськими листами;
- вихід із програми.

Головна логіка консольного інтерфейсу представлена в модулі cli.py. Нижче наведено основний лістинг цього модуля (лістинг 3.8).

Лістинг 3.8. Основне меню CLI для керування програмою

```
{
    def main():
        while True:
            print("\n=== Phishing Detector ===")
            print("  1) Train a new model")
            print("  2) Evaluate fraud-email dataset")
            print("  3) Exit")
            sel = input("Select option [1-3]: ").strip()
            if sel == "1":
                train_flow()
            elif sel == "2":
                eval_flow()
            elif sel == "3":
                print("Goodbye!")
                break
            else:
                print("Invalid choice, try again.")

}
```

Під час вибору пункту навчання нової моделі користувачу пропонується ввести необхідні параметри навчання (довжину вектора – MAX_LEN, розмір ембедінгів – EMBEDDING_DIM, кількість епох – EPOCHS). У випадку, якщо користувач не вказує ці значення, використовуються значення за замовчуванням (лістинг 3.9).

Лістинг 3.9. Послідовність навчання нової моделі через CLI

```
{
    def prompt_int(prompt, default):
        val = input(f"{prompt} [{default}]: ").strip()
        return int(val) if val else default

    def train_flow():
        print("\n--- TRAIN NEW MODEL ---")
```

```
        max_len = prompt_int("Enter MAX_LEN", default=100)
        embedding_dim = prompt_int("Enter EMBEDDING_DIM",
default=100)
        epochs = prompt_int("Enter EPOCHS", default=30)

        base = f"len{max_len}_emb{embedding_dim}_ep{epochs}"
        best_path = os.path.join("keras_models",
f"best_{base}.keras")
        final_path = os.path.join("keras_models",
f"final_{base}.keras")

        os.environ["MAX_LEN"] = str(max_len)
        os.environ["EMBEDDING_DIM"] = str(embedding_dim)
        os.environ["EPOCHS"] = str(epochs)
        os.environ["BEST_MODEL_PATH"] = best_path
        os.environ["FINAL_MODEL_PATH"] = final_path

        print(f"\n→ Training with:")
        print(f"   MAX_LEN={max_len},
EMBEDDING_DIM={embedding_dim}, EPOCHS={epochs}")
        print(f"   Best checkpoint → {best_path}")
        print(f"   Final model     → {final_path}\n")

        train_main()
}
```

Цей підхід дозволяє автоматично зберігати навчені моделі з унікальними іменами, які чітко вказують використані гіперпараметри.

При оцінюванні моделі користувачеві пропонується перелік всіх наявних збережених моделей у каталозі keras_models. Він може обрати потрібну модель введенням відповідного номера зі списку (лістинг 3.10).

Лістинг 3.10. Послідовність оцінювання моделі через CLI

```
{
    def list_models():
        models = sorted(os.listdir("keras_models"))
        for i, fn in enumerate(models):
            print(f"  {i + 1}) {fn}")
        return models

    def eval_flow():
        print("\n--- EVALUATE FRAUD EMAILS ---")
        print("Available models in keras_models/:")
        models = list_models()
        choice = prompt_int("Pick model number to load", default=1)
        if not (1 <= choice <= len(models)):
            print("Invalid choice, aborting.")
            return
          model_file = os.path.join("keras_models", models[choice -
1])
        os.environ["MODEL_PATH"] = model_file
```

```
        print(f"\n→ Evaluating with model: {model_file}\n")
        eval_main()
}
```

Такий підхід забезпечує простоту та прозорість вибору моделі для подальшої оцінки її ефективності на реальних даних.

Використання консольного інтерфейсу значно спрощує процес керування процесами навчання та оцінювання, роблячи розроблену систему зручною для практичного використання.

3.3.5. *Особливості паралельної та ефективної обробки даних*

Одним із ключових аспектів розробленого програмного рішення є ефективність роботи з великими обсягами текстових даних. Це забезпечується використанням методів паралельної обробки, що значно пришвидшує етап попередньої обробки та векторизації текстів.

В межах реалізованого модуля навчання (train.py) використовується бібліотека Python concurrent.futures, що дозволяє організувати паралельне виконання завдань. Під час попередньої обробки текстів та їх перетворення у векторну форму застосовується пул процесів (ProcessPoolExecutor), кількість яких визначається автоматично, але не перевищує чотирьох одночасних процесів. Це зроблено з метою обмеження споживання ресурсів системи і уникнення її перевантаження.

Нижче наведено реалізацію функції паралельної векторизації текстів, що враховує ці особливості (лістинг 3.11).

Лістинг 3.11. Паралельна попередня обробка текстових даних

```
{
     from concurrent.futures import ProcessPoolExecutor
     from tqdm import tqdm
     import os
     import numpy as np

     def preprocess_and_vectorize(texts, max_len):
         """
         Паралельна попередня обробка та векторизація з
використанням обмеженого пулу процесів.
```

```
        В разі нестачі ресурсів (WinError 1450) відбувається
повернення до послідовного режиму.
        """
        args_iter = ((txt, max_len) for txt in texts)
        max_workers = min(4, os.cpu_count() or 1)
    try:
            with ProcessPoolExecutor(max_workers=max_workers) as
exe:
                vecs = list(tqdm(
                    exe.map(_vectorize_task, args_iter),
                    total=len(texts),
                    desc="Preprocess & Vectorize"
                ))
        except OSError as e:
            if e.winerror == 1450:
                print("\n[!] WinError 1450: Перехід на послідовну
векторизацію")
                vecs = []
                for txt in tqdm(texts, desc="Preprocess & Vectorize
(sequential)"):
                    vecs.append(_vectorize_task((txt, max_len)))
            else:
                raise
        return np.stack(vecs)
}
```

Реалізація передбачає використання функції-помічника _vectorize_task, яка виконує комплекс операцій:

1. Очищення тексту від HTML та спеціальних символів.
2. Токенізацію та лематизацію за допомогою бібліотеки SpaCy.
3. Векторизацію отриманих токенів за допомогою ембедінгів GloVe.

При виникненні помилки нестачі ресурсів операційної системи Windows, застосовується механізм автоматичного переходу на послідовну обробку. Це гарантує стабільність роботи навіть при великих наборах даних та обмежених ресурсах. Завдяки такому підходу, вдається ефективно використовувати доступні обчислювальні ресурси, зменшити загальний час навчання моделі, забезпечити стабільність її роботи, і одночасно уникнути перевантаження комп’ютера під час обробки даних. Таким чином, застосування паралельної обробки з обмеженням кількості процесів та механізмом відновлення після помилок значно підвищує ефективність та надійність роботи розробленого програмного забезпечення для автоматизованого виявлення фішингових електронних листів [48].

3.3.6. *Метрики ефективності та їх розрахунок*

Оцінювання ефективності розробленого методу ідентифікації фішингового вмісту в текстовій частині електронних листів здійснюється на основі низки класичних метрик машинного навчання. Вибрані метрики дозволяють максимально повно і точно описати здатність моделі правильно класифікувати текстові повідомлення.

В реалізованому програмному забезпеченні використовуються наступні метрики:

1. Accuracy (точність) – найбільш проста і зрозуміла метрика, яка показує відсоток правильно класифікованих повідомлень до загальної кількості.
2. Precision (точність позитивного прогнозу) – показує, яка частка листів, класифікованих моделлю як фішингові, є такими насправді.
3. Recall (чутливість або повнота) – демонструє, яка частина всіх наявних фішингових листів була правильно ідентифікована моделлю.
4. F1-score – гармонійне середнє між Precision і Recall, що дозволяє отримати загальну оцінку ефективності моделі.
5. ROC-крива та AUC (Area Under the ROC Curve) – ROC-крива відображає залежність чутливості моделі від помилкових спрацювань (False Positive Rate, FPR). AUC характеризує загальну якість моделі на всіх порогових значеннях і визначається за допомогою інтеграла площі під ROC-кривою.
6. Confusion Matrix (матриця помилок) – наочно демонструє кількість коректних та некоректних прогнозів у формі матриці, де по осях відкладаються реальні та передбачені значення.

У реалізованому коді, наведеному у файлі train.py, метрики розраховуються автоматично після завершення навчання моделі та

оцінювання на тестовому наборі. В коді використані такі функції для розрахунку метрик:

- Функція roc_curve з бібліотеки sklearn.metrics використовується для побудови ROC-кривої, а функція auc – для обчислення площі під нею.
- Функція confusion_matrix (також зі sklearn.metrics) використовується для побудови матриці помилок.

Нижче наведено приклад реалізації цих метрик (лістинг 3.12).

Лістинг 3.12. Розрахунок метрик і візуалізація графічних елементів

```
{
     from sklearn.metrics import roc_curve, auc, confusion_matrix
     import matplotlib.pyplot as plt

     y_pred_prob                =                model.predict(X_test,
batch_size=BATCH_SIZE).ravel()
     y_pred = (y_pred_prob > 0.5).astype(int)

     fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
     roc_auc = auc(fpr, tpr)
     plt.figure()
     plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.3f})")
     plt.xlabel("False Positive Rate")
     plt.ylabel("True Positive Rate")
     plt.title("ROC Curve")
     plt.legend()
     plt.savefig(os.path.join(run_dir, "roc_curve.png"))
     plt.close()

     cm = confusion_matrix(y_test, y_pred)
     plt.figure()
     plt.imshow(cm, interpolation="nearest", cmap='Blues')
     plt.colorbar()
     tick_marks = np.arange(2)
     plt.xticks(tick_marks, ['Phishing', 'Legitimate'])
     plt.yticks(tick_marks, ['Phishing', 'Legitimate'])
     for i in range(2):
         for j in range(2):
              plt.text(j, i, cm[i, j], horizontalalignment="center",
color="white" if cm[i, j] > cm.max() / 2 else "black")
     plt.ylabel("True label")
     plt.xlabel("Predicted label")
     plt.savefig(os.path.join(run_dir, "confusion_matrix.png"))
     plt.close()
}
```

3.3.7. *Логування та візуалізація результатів*

Для ефективного контролю процесу навчання, аналізу результатів і моніторингу поведінки моделі протягом експериментів у розробленому програмному забезпеченні реалізовані засоби логування та візуалізації. Основними інструментами, які використовуються для цих цілей, є TensorBoard та бібліотека візуалізації Matplotlib. Використання TensorBoard для логування процесу навчання TensorBoard є спеціалізованим інструментом для моніторингу та аналізу моделей, створених за допомогою TensorFlow і Keras. У розробленому ПЗ він інтегрований як один з callback-механізмів у процес навчання нейромережевої моделі, що дозволяє:

1. Відслідковувати динаміку основних метрик (accuracy, loss, val_accuracy, val_loss) під час тренування.
2. Аналізувати зміну швидкості навчання (learning rate).
3. Порівнювати результати різних експериментів за параметрами моделі.

TensorBoard налаштовується наступним чином (лістинг 3.13).

Лістинг 3.13. Налаштування TensorBoard

```
{
    from tensorflow.keras.callbacks import TensorBoard

    tb = TensorBoard(log_dir=LOG_DIR)

    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=epochs,
        batch_size=BATCH_SIZE,
        callbacks=[tb, ckpt, es, rlrop],
    )
}
```

Крім TensorBoard, для зручного й наочного відображення результатів тренування та оцінки ефективності моделі застосовується бібліотека Matplotlib. Вона дозволяє зберігати результати у форматі зображень PNG для подальшого аналізу та використання у звітах чи документації. У модулі реалізовані наступні графічні представлення (лістинг 3.14, 3.15, 3.16, 3.17).

Лістинг 3.14. Послідовність побудови Accuracy-кривої

```
{
      import matplotlib.pyplot as plt

      plt.figure()
      plt.plot(history.history["accuracy"], label="Train Accuracy")
      plt.plot(history.history["val_accuracy"],      label="Validation
Accuracy")
      plt.title("Accuracy per Epoch")
      plt.xlabel("Epoch")
      plt.ylabel("Accuracy")
      plt.legend()
      plt.savefig(os.path.join(run_dir, "accuracy_curve.png"))
      plt.close()
}
```

Лістинг 3.15. Послідовність побудови Loss-кривої (втрата за епохами)

```
{
      plt.figure()
      plt.plot(history.history["loss"], label="Train Loss")
      plt.plot(history.history["val_loss"], label="Validation Loss")
      plt.title("Loss per Epoch")
      plt.xlabel("Epoch")
      plt.ylabel("Loss")
      plt.legend()
      plt.savefig(os.path.join(run_dir, "loss_curve.png"))
      plt.close()
}
```

Лістинг 3.16. Послідовність побудови ROC-кривої

```
{
      from sklearn.metrics import roc_curve, auc

      fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
      roc_auc = auc(fpr, tpr)

      plt.figure()
      plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.3f})")
      plt.xlabel("False Positive Rate")
      plt.ylabel("True Positive Rate")
      plt.title("ROC Curve")
}
```

Лістинг 3.17. Послідовність побудови матриці помилок

```
{
      from sklearn.metrics import confusion_matrix
      import numpy as np

      cm = confusion_matrix(y_test, y_pred)
      plt.figure()
      plt.imshow(cm, interpolation="nearest", cmap='Blues')
      plt.title("Confusion Matrix")
```

```
      plt.colorbar()
      tick_marks = np.arange(2)
      plt.xticks(tick_marks, ['Phishing', 'Legitimate'])
      plt.yticks(tick_marks, ['Phishing', 'Legitimate'])

      for i in range(2):
            for j in range(2):
                  plt.text(j,          i,          cm[i,          j],
horizontalalignment="center",
                                          color="white" if cm[i, j] >
                  cm.max() / 2 else "black")

      plt.ylabel("True Label")
      plt.xlabel("Predicted Label")
      plt.savefig(os.path.join(run_dir, "confusion_matrix.png"))
      plt.close()
}
```

Ці графіки зберігаються в спеціально створеній директорії логування для кожного конкретного запуску моделі. Таким чином, є можливість легко порівнювати результати різних експериментів, налаштувань гіперпараметрів та інших факторів.

Отже, реалізовані засоби логування та візуалізації забезпечують повноцінний контроль та аналіз результатів експериментів, сприяють зручності використання розробленого програмного забезпечення, та забезпечують швидке й точне прийняття рішень на основі отриманих даних.

3.4. Сценарії використання розробленої системи

Розроблене програмне забезпечення для автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів передбачає взаємодію за допомогою інтерфейсу командного рядка. Для візуалізації роботи консольного інфтерфейсу було створено UML діаграму сценаріїв використання розробленого програмного забезпечення для ідентифікації фішинговго вмісту в текстовій частині електронних листів (рис. 3.3).

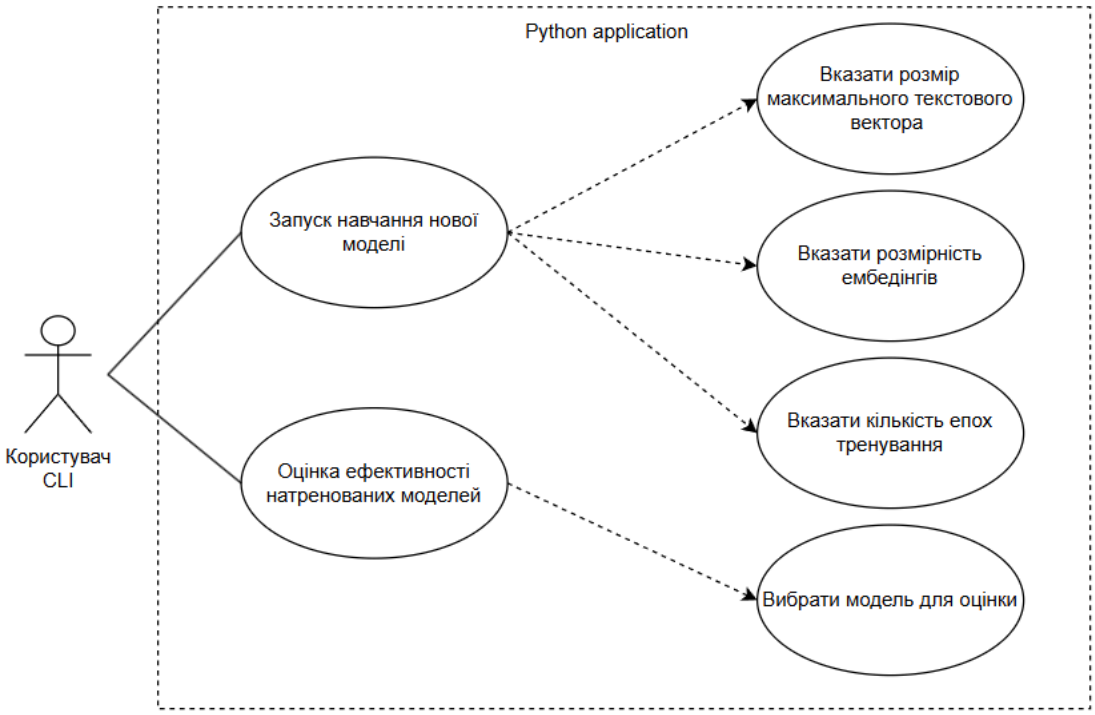


Рис. 3.3. UML діаграма сценаріїв використання розробленого програмного забезпечення для ідентифікації фішинговго вмісту в текстовій частині електронних листів

3.4.1. *Сценарій навчання нової моделі*

Для запуску навчання нової моделі користувач використовує розроблений консольний інтерфейс (CLI), який дозволяє задавати параметри тренування та відслідковувати процес навчання у реальному часі.

Після запуску CLI-інтерфейсу, користувач вибирає режим навчання моделі, після чого система запитує введення ключових параметрів (рис. 3.4):

1. MAX_LEN – максимальна довжина вхідних текстових послідовностей;
2. EMBEDDING_DIM – розмірність використаних ембедінгів (наприклад, 25, 50, 100 або 200);
3. EPOCHS – кількість епох для тренування моделі.

```
D:\University\project\backend\.venv\Scripts\python.exe D:\University\project\backend\cli.py

=== Phishing Detector ===
  1) Train a new model
  2) Evaluate fraud-email dataset
  3) Exit
Select option [1-3]: 1

--- TRAIN NEW MODEL ---
Enter MAX_LEN [100]: 50
Enter EMBEDDING_DIM [100]: 50
Enter EPOCHS [30]: 10

→ Training with:
   MAX_LEN=50, EMBEDDING_DIM=50, EPOCHS=10
   Best checkpoint → keras_models\best_len50_emb50_ep10.keras
   Final model     → keras_models\final_len50_emb50_ep10.keras

Loading training dataset...
Rows remaining after label cleanup: 164954
```

Рис. 3.4. Початок взаємодії з системою

Після того як користувач ввів необхідні параметри, система автоматично формує шляхи для збереження найкращої проміжної моделі та остаточної моделі після завершення навчання.

Процес навчання починається з етапу попередньої обробки та векторизації текстових даних. На цьому етапі використовуються GloVe-ембедінги, які завантажуються відповідно до вибраної розмірності.

Model: "hybrid_model"

Layer (type)	Output Shape	Param #	Connected to
hybrid_input (InputLayer)	(None, 50, 50)	0	-
cnn_model (Functional)	(None, 128)	126,464	hybrid_input[0][…
rnn_model (Functional)	(None, 128)	108,160	hybrid_input[0][…
attention_fusion (AttentionFusion)	(None, 128)	33,153	cnn_model[0][0], rnn_model[0][0]
fusion_dropout (Dropout)	(None, 128)	0	attention_fusion…
fusion_dense (Dense)	(None, 128)	16,512	fusion_dropout[0…
output (Dense)	(None, 1)	129	fusion_dense[0][…

Total params: 284,418 (1.08 MB)
Trainable params: 284,418 (1.08 MB)

Рис. 3.5. Ілюстрація моделі після векторизації

Після підготовки даних, модель будується згідно визначених параметрів, після чого виводиться детальна інформація про архітектуру моделі (рис. 3.5) з описом кожного шару та кількістю параметрів, що тренуються.

Під час тренування в режимі реального часу відображаються поточні значення точності та втрат на тренувальних та валідаційних вибірках (рис 3.6), що дозволяє користувачу оцінити швидкість збіжності моделі та її ефективність.

```
Starting training...
Epoch 1/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.8789 - loss: 0.2662
Epoch 1: val_accuracy improved from -inf to 0.96708, saving model to keras_models\best_len50_emb50_ep10.keras
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 61s 29ms/step - accuracy: 0.8790 - loss: 0.2661 - val_accuracy: 0.9671 - val_loss: 0.0908 - learning_rate: 0.0010
Epoch 2/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9663 - loss: 0.0924
Epoch 2: val_accuracy improved from 0.96708 to 0.97042, saving model to keras_models\best_len50_emb50_ep10.keras
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 56s 27ms/step - accuracy: 0.9663 - loss: 0.0924 - val_accuracy: 0.9704 - val_loss: 0.0771 - learning_rate: 0.0010
Epoch 3/10
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9753 - loss: 0.0683
Epoch 3: val_accuracy improved from 0.97042 to 0.98151, saving model to keras_models\best_len50_emb50_ep10.keras
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 56s 27ms/step - accuracy: 0.9753 - loss: 0.0683 - val_accuracy: 0.9815 - val_loss: 0.0503 - learning_rate: 0.0010
Epoch 4/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9813 - loss: 0.0520
Epoch 4: val_accuracy did not improve from 0.98151
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 60s 29ms/step - accuracy: 0.9813 - loss: 0.0520 - val_accuracy: 0.9814 - val_loss: 0.0500 - learning_rate: 0.0010
Epoch 5/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9851 - loss: 0.0416
Epoch 5: val_accuracy improved from 0.98151 to 0.98594, saving model to keras_models\best_len50_emb50_ep10.keras
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 73s 35ms/step - accuracy: 0.9851 - loss: 0.0416 - val_accuracy: 0.9859 - val_loss: 0.0410 - learning_rate: 0.0010
Epoch 6/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.9882 - loss: 0.0343
Epoch 6: val_accuracy improved from 0.98594 to 0.98600, saving model to keras_models\best_len50_emb50_ep10.keras
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 76s 37ms/step - accuracy: 0.9882 - loss: 0.0343 - val_accuracy: 0.9860 - val_loss: 0.0383 - learning_rate: 0.0010
Epoch 7/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9896 - loss: 0.0289
Epoch 7: val_accuracy did not improve from 0.98600
2062/2062 ━━━━━━━━━━━━━━━━━━━━ 75s 36ms/step - accuracy: 0.9896 - loss: 0.0289 - val_accuracy: 0.9858 - val_loss: 0.0399 - learning_rate: 0.0010
Epoch 8/10
2061/2062 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.9898 - loss: 0.0297
```

Рис. 3.6. Ілюстрація процесу навчання моделі

Таким чином, користувач має можливість легко налаштувати і запустити тренування моделі через зручний і зрозумілий консольний інтерфейс, отримуючи повний контроль над параметрами та результатами навчання.

3.4.2. *Оцінювання ефективності моделі на окремому датасеті*

Для оцінювання ефективності вже натренованих моделей за допомогою реалізованого консольного інтерфейсу передбачений окремий сценарій використання. Користувачу необхідно запустити консольний інтерфейс та обрати режим оцінювання ефективності моделі, після чого отримати

результати роботи моделі у вигляді метрик продуктивності та детального звіту.

Процес оцінювання моделі через консоль включає наступні етапи:

1. *Запуск інтерфейсу для оцінювання:* користувач відкриває консольний інтерфейс шляхом запуску відповідного Python-скрипта. Далі з головного меню вибирається опція для оцінювання ефективності вже натренованих моделей.
2. *Вибір натренованої моделі:* консольний інтерфейс виводить список доступних моделей, що зберігаються в директорії keras_models. Користувач може обрати бажану модель за допомогою відповідного номера.
3. *Завантаження та векторизація тестових даних:* після вибору моделі починається процес автоматичного завантаження тестових даних та їх попередньої обробки (очищення, токенізація, лематизація, векторизація). Використовується попередньо завантажений ембедінг GloVe відповідної розмірності (рис. 3.7).

```
=== Phishing Detector ===
  1) Train a new model
  2) Evaluate fraud-email dataset
  3) Exit
Select option [1-3]: 2

--- EVALUATE FRAUD EMAILS ---
Available models in keras_models/:
  1) best_len100_emb100_ep20.keras
  2) best_len100_emb25_ep20.keras
  3) best_len200_emb100_ep20.keras
  4) best_len200_emb200_ep20.keras
  5) best_len200_emb200_ep30.keras
  6) best_len200_emb25_ep20.keras
  7) best_len200_emb50_ep20.keras
  8) best_len50_emb50_ep10.keras
  9) final_hybrid_model.keras
  10) final_len100_emb100_ep20.keras
  11) final_len100_emb25_ep20.keras
  12) final_len200_emb100_ep20.keras
  13) final_len200_emb200_ep20.keras
  14) final_len200_emb200_ep30.keras
  15) final_len200_emb25_ep20.keras
  16) final_len200_emb50_ep20.keras
  17) final_len50_emb50_ep10.keras
  18) initial_hybrid_model.keras
Pick model number to load [1]: 17

→ Evaluating with model: keras_models\final_len50_emb50_ep10.keras

[eval] parsed from filename → MAX_LEN=50, EMBEDDING_DIM=50
Loaded 11928 emails → Legitimate: 6742, Phishing: 5186
Vectorizing input texts...
Preprocess & Vectorize:   0%|          | 0/11928 [00:00<?, ?it/s]
Loading local GloVe (50d) from ../data/twitter-archive\glove.twitter.27B.50d.txt...
```

Рис. 3.7. Сценарій оцінювання моделі

4. *Інференс і отримання результатів*: після успішної векторизації починається інференс вибраної моделі. Після завершення цього етапу виводяться основні метрики ефективності (рис. 3.8).

```
Overall Accuracy: 0.9606

Classification Report:
              precision    recall  f1-score   support

  Legitimate     0.9668    0.9634    0.9651      6742
    Phishing     0.9526    0.9570    0.9548      5186

    accuracy                         0.9606     11928
   macro avg     0.9597    0.9602    0.9599     11928
weighted avg     0.9606    0.9606    0.9606     11928
```

Рис. 3.8. Вивід результатів після оцінювання

З виводу програми можна побачити загальну точність (accuracy), а також точність (precision), повноту (recall) та F1-score для кожного класу («Phishing» та «Legitimate»). Ці результати дають чітке розуміння того, наскільки добре модель розпізнає фішингові електронні листи на новому наборі даних, який вона раніше не бачила.

Таким чином, реалізований консольний інтерфейс дозволяє оперативно оцінювати ефективність різних варіантів натренованих моделей, що значно спрощує експериментальний процес та підбір оптимальних конфігурацій моделі.

3.5. Висновки до третього розділу

У третьому розділі було детально описано програмну реалізацію запропонованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів. Проведено обґрунтування вибору технологій, детально розглянуто архітектуру розробленого програмного

забезпечення, описано особливості реалізації та наведено конкретні сценарії практичного застосування системи через консольний інтерфейс.

Здійснено аналіз та вибір технологічного стеку, який відповідає критеріям продуктивності, масштабованості, зручності розробки та підтримки. Обрана мова програмування Python, а також спеціалізовані бібліотеки, такі як TensorFlow/Keras для реалізації нейромережевих моделей, SpaCy для лематизації та токенізації, Gensim для роботи з ембедінгами GloVe, NumPy і Pandas для ефективної роботи з числовими даними. Обрані технології повністю відповідають потребам розробки NLP-застосунків високого рівня складності.

Структура проєкту ретельно організована для максимальної прозорості й зручності підтримки. Реалізована система включає чітко розмежовані компоненти: модулі попередньої обробки текстів, векторизації текстових даних, гібридної CNN-BiLSTM моделі зі шаром уваги та додаткові інструменти для навчання, оцінювання та візуалізації результатів. Завдяки чіткому поділу на компоненти, підтримка та подальший розвиток системи є значно простішими.

Особливу увагу було приділено паралельній обробці даних, яка реалізована з використанням механізмів багатопотоковості Python. Такий підхід дозволив суттєво прискорити процес попередньої обробки та векторизації великого масиву текстових даних. Система також обладнана механізмом логування та візуалізації процесу навчання моделей за допомогою TensorBoard, а графічні результати (ROC-крива, матриця помилок, точність та втрата за епохами) допомагають ефективно аналізувати якість роботи створених моделей.

Практичні приклади роботи з консольним інтерфейсом продемонстрували зручність та простоту запуску навчання нових моделей і оцінювання вже натренованих рішень. Вбудований механізм параметризації дає змогу легко експериментувати з різними конфігураціями моделей, швидко отримуючи необхідні результати.

Отже, розроблене програмне забезпечення відповідає початковим вимогам щодо ефективності, зручності та продуктивності. Система повністю готова до практичного застосування, має високий ступінь адаптивності та може бути основою для подальших експериментальних і наукових досліджень у сфері автоматизованого виявлення фішингових загроз на основі аналізу текстового контенту електронних листів.

4. АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ МЕТОДУ

4.1. Критерії оцінювання та аналіз ефективності запропонованого методу

4.1.1. *Огляд використаних датасетів*

Для оцінювання ефективності розробленого комбінованого методу автоматизованої ідентифікації фішингового вмісту було використано три незалежних набори даних, які дозволяють всебічно перевірити якість розробленої моделі у різних умовах використання. Здатність моделі до узагальнення знань та її адаптивність до нових типів електронних листів було оцінено шляхом використання кількох незалежних джерел даних , що суттєво впливає на її практичну корисність.

Перший набір даних, що використовувався на етапі тренування моделі, є об'єднаним набором під назвою «Phishing Email Dataset» [49], який доступний у відкритому доступі на платформі Kaggle. Цей датасет містить сім окремих CSV-файлів, серед яких набори даних SpamAssassin, Nigerian Fraud, CEAS 2008, Enron, Ling, Nazario та основний файл phishing_email.csv. Кожен файл містить електронні листи, що позначені як фішингові або легітимні. Така різноманітність дозволяє врахувати різні стилістичні та контекстуальні особливості текстових повідомлень, що покращує здатність моделі до узагальнення на реальних даних. Окрім тексту повідомлень, кожен лист позначений числовою міткою (0 – нефішинговий, 1 – фішинговий). Всього датасет містить понад 160 тисяч прикладів електронних листів.

Другий набір даних «Phishing Email Detection Dataset» [50], також доступний на Kaggle (автор – Subhajournal), використовувався для первинної оцінки якості моделі після її навчання. Цей датасет є незалежним від тренувального набору і містить близько 18 тисяч електронних листів. Кожен приклад у цьому датасеті супроводжується текстом листа та відповідною числовою міткою, яка надає інформацію про вміст фішингу, аналогічно до попереднього датасету. Використання цього набору даних дозволяє

перевірити, наскільки ефективно модель може переносити отримані знання на інші набори даних, що є критичним фактором для оцінки якості моделі у практичних умовах.

Третій набір даних, «Fraud Email Dataset» [51], є додатковим незалежним джерелом даних і був використаний для фінальної оцінки здатності моделі ідентифікувати шахрайські повідомлення. Цей датасет доступний на платформі Kaggle та містить приклади електронних листів, що використовуються для шахрайських цілей. Використання саме цього набору даних дозволяє додатково оцінити здатність моделі працювати із специфічними випадками фішингу, такими як шахрайські листи, які зазвичай мають свої унікальні стилістичні особливості та певні типові патерни тексту. Загальна кількість прикладів у цьому наборі становить понад 11 тисяч листів.

Підсумовуючи, використання наведених датасетів дозволяє провести повноцінне експериментальне оцінювання запропонованого методу, забезпечуючи перевірку моделі як на різноманітних тренувальних даних, так і на незалежних тестових наборах. Це дозволяє зробити комплексні висновки щодо здатності запропонованої архітектури CNN-BiLSTM з шаром уваги ефективно виявляти фішингові листи у різних умовах експлуатації.

4.1.2. *Аналіз результатів на тренувальному датасеті*

Для оцінки ефективності запропонованого комбінованого методу було навчено кілька моделей із різними значеннями гіперпараметрів. Навчання проводилось на тренувальному наборі, яким був «Phishing Email Dataset».

В якості основних критеріїв оцінювання моделі на тренувальному наборі були обрані такі метрики: точність (Accuracy), точність визначення класів (Precision), повнота визначення класів (Recall) та гармонічне середнє цих метрик (F1-score). Аналіз проводився для різних значень гіперпараметрів моделей, таких як розмір максимального текстового вектора (MAX_LEN), розмірність ембедінгів (EMBEDDING_DIM) та кількість епох тренування (EPOCHS).

Модель з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20 продемонструвала найкращі результати, показавши загальну точність 0.9960, точність передбачення (precision) 0.9964, повноту 0.9398 та F1-міру 0.9673. Друга за ефективністю модель, з параметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20 показала схожі результати: точність 0.9944, точність передбачення (precision) 0.9982, повноту 0.9354, F1-міру 0.9658, але поступилась у точності класифікації за рахунок меншої розмірності ембедінгів, що зменшує кількість семантичної інформації, яку може засвоїти модель.

Найгірші результати продемонструвала модель з MAX_LEN=100, EMBEDDING_DIM=100, EPOCHS=20, отримавши точність лише 0.9248. Це пояснюється меншою довжиною текстового вектора (MAX_LEN), що обмежує здатність моделі враховувати повний контекст довших повідомлень, які є характерними для фішингових листів.

Модель з максимальною розмірністю ембедінгів (MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20) також продемонструвала досить високі результати (точність 0.9530), однак не змогла перевершити модель з розмірністю ембедінгів 100. Можна припустити, що це через перенавчання і надлишкову інформацію, яка ускладнює узагальнення (рис. 4.1).

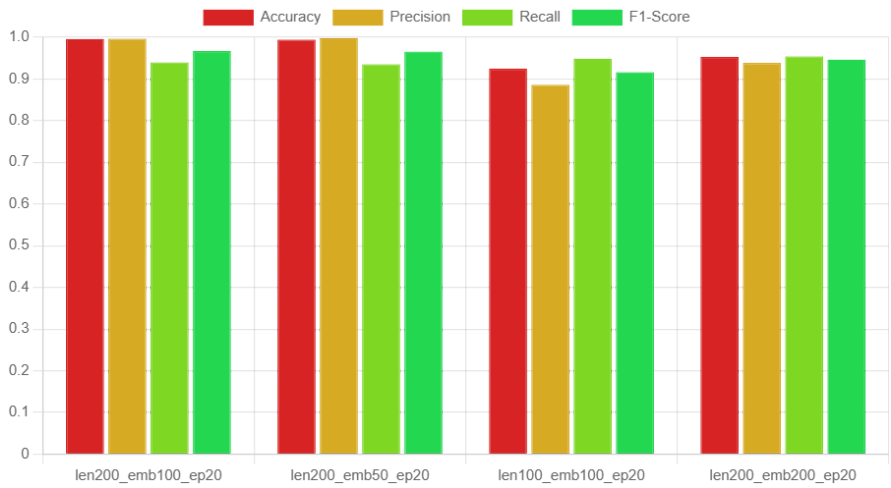


Рис. 4.1. Гістограма порівняння метрик ефективності моделей з різними гіперпараметрами на датасеті «Phishing Email Dataset»

Таким чином, оптимальним варіантом за результатами тренування на комбінованому наборі є модель з параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20, яка демонструє баланс між високою точністю і стабільністю узагальнення даних.

4.1.3. *Аналіз результатів на тестовому наборі*

Після успішного тренування на датасеті «Phishing Email Dataset», ефективність розроблених моделей було додатково оцінено на незалежному наборі даних «Phishing Email Detection». Цей набір використовується для об'єктивної перевірки здатності моделей до узагальнення та виявлення фішингових листів у раніше невідомих їм умовах, що є критично важливим для практичного застосування запропонованого методу.

Для аналізу результатів було обрано три найбільш показові моделі з різними наборами гіперпараметрів:

1. Модель із параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20.
2. Модель із параметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20.
3. Модель із параметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20.

Першою проаналізуємо модель із параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20. Ця модель продемонструвала найвищу точність 98,59% при низькому значенні втрат (Loss = 0.0359). Також важливими є показники True Positive Rate 99,48% та False Positive Rate 1,98%, що є найкращими серед аналізованих моделей.

Нижче наведено графічне представлення метрик ефектиності цієї моделі: крива точності моделі, крива втрат моделі, ROC-крива моделі та матриця помилок моделі (рис. 4.2, 4.3, 4.4, 4.5).

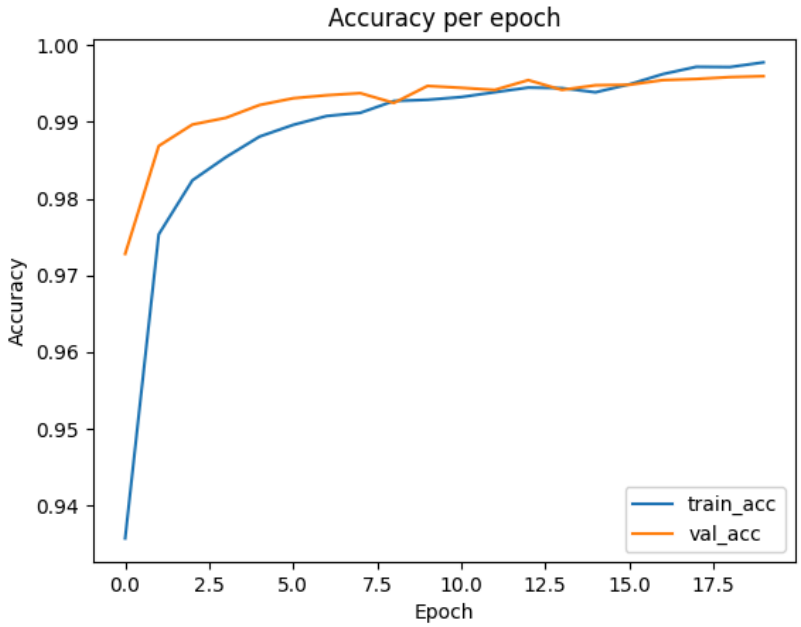


Рис. 4.2. Крива точності моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20

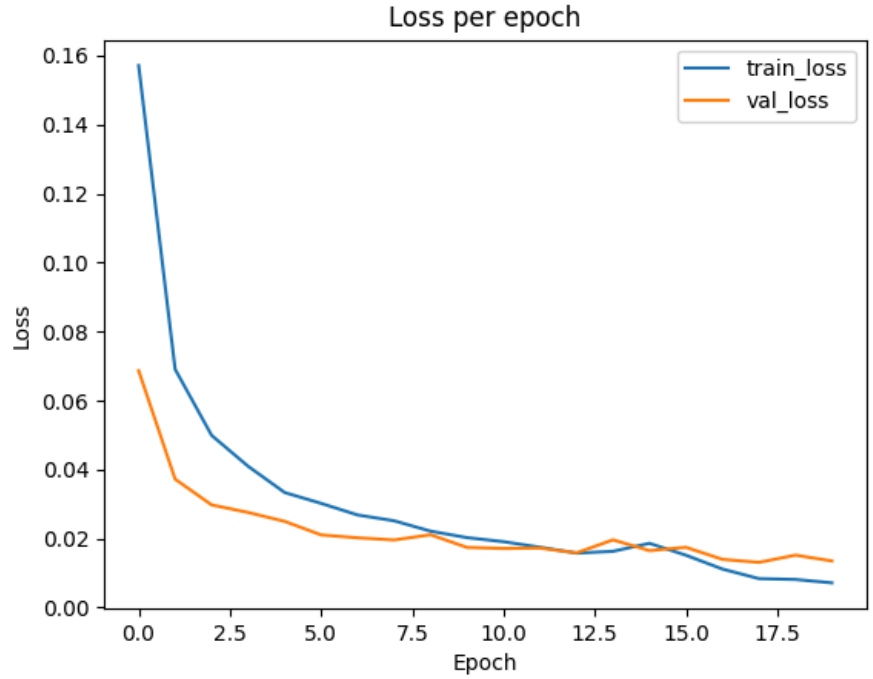


Рис. 4.3. Крива втрат моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20

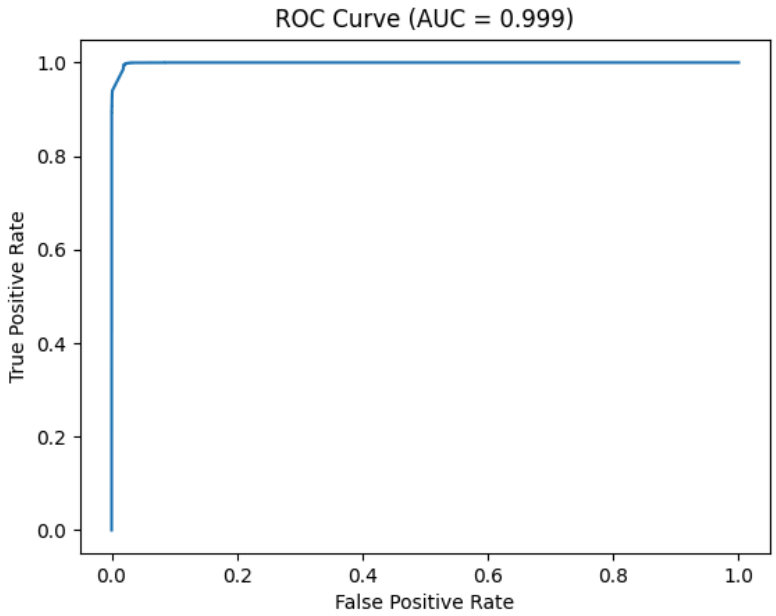


Рис. 4.4. ROC-крива моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20

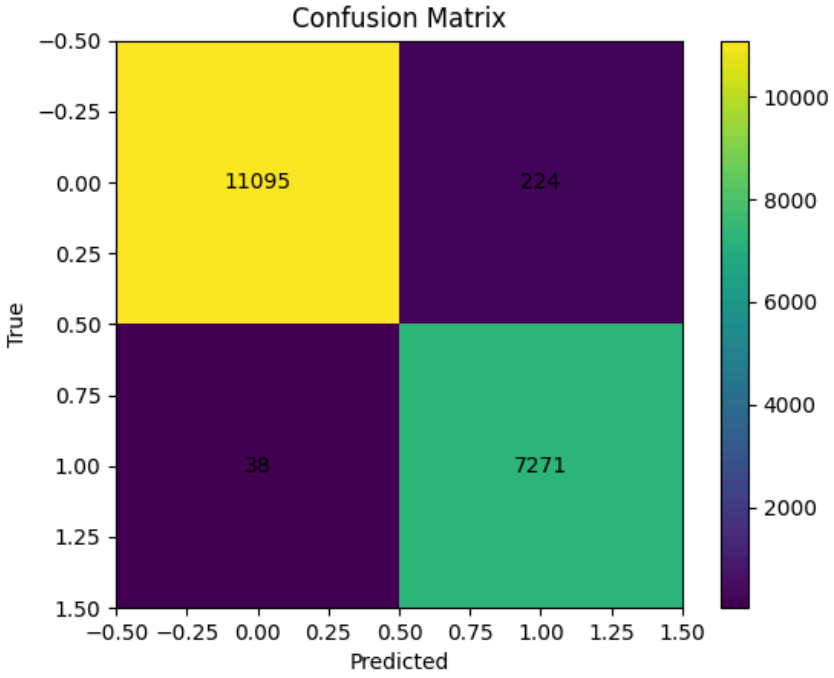


Рис. 4.5. Матриця помилок моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20

Другою була модель із параметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20, яка також продемонструвала високу

точність (98,53%) та хорошу чутливість (TPR = 99,41%) при значенні FPR на рівні 2,03%. Значення втрат (Loss = 0.0329) вказує на хорошу стабільність цієї моделі, хоча результати є дещо нижчими за першу модель з ембедингом у 100 вимірів.

Нижче наведено графічне представлення метрик ефективності цієї моделі (рис. 4.6, 4.7, 4.8, 4.9).

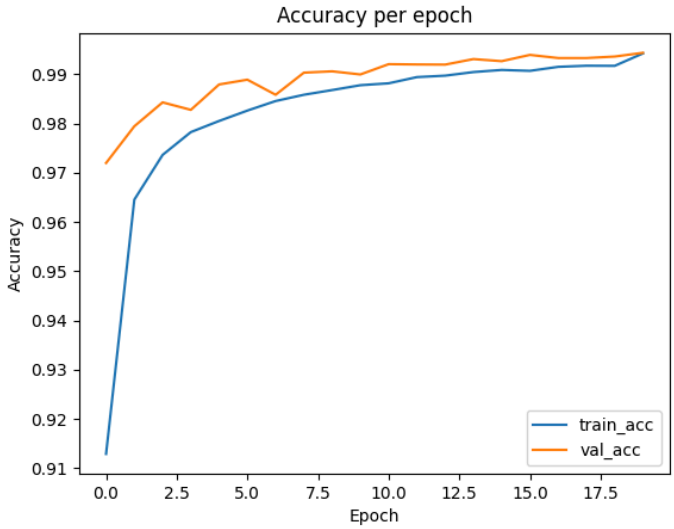


Рис. 4.6. Крива точності моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20

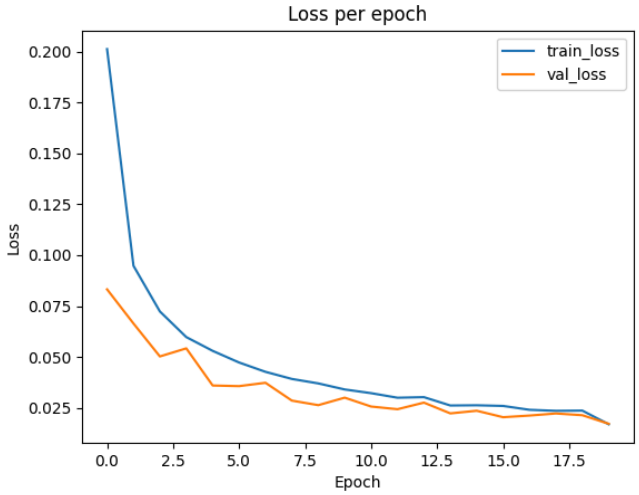


Рис. 4.7. Крива втрат моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20

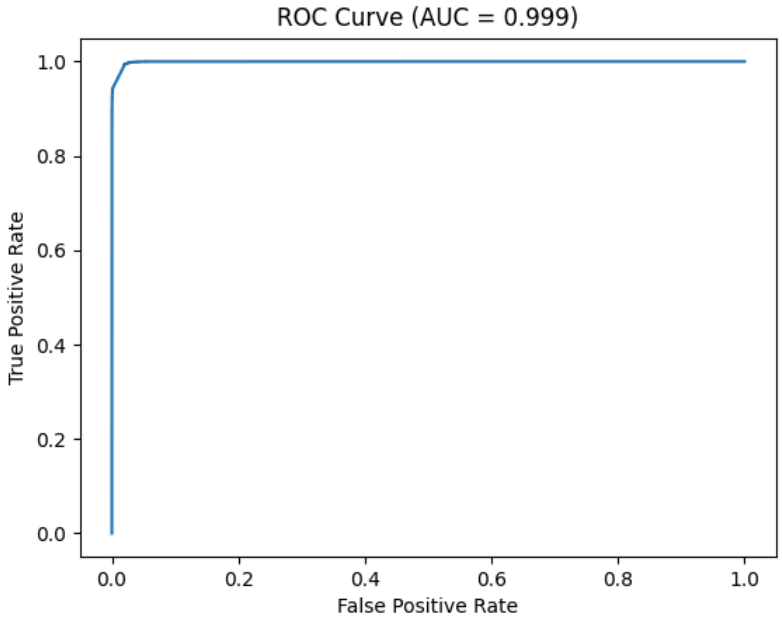


Рис. 4.8. ROC-крива моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20

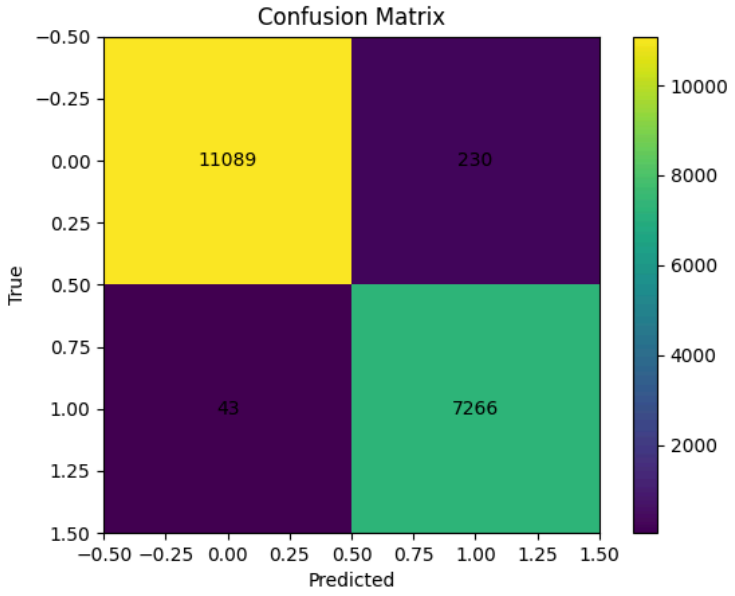


Рис. 4.9. Матриця помилок моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20

Третьою є модель із параметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20, яка продемонструвала загальну точність 95,30%, що нижче за дві попередні моделі. При цьому TPR склала

95,43%, а FPR – 4,81%, що вказує на нижчу здатність узагальнення у порівнянні з двома іншими моделями. Незважаючи на великий розмір ембедингів (200), модель не перевершила модель із 100-вимірними векторами, що може бути зумовлено перенавчанням або менш оптимальним підбором гіперпараметрів для цього розміру векторів.

Нижче наведено графічне представлення метрик ефективності цієї моделі (рис. 4.10, 4.11, 4.12, 4.13).

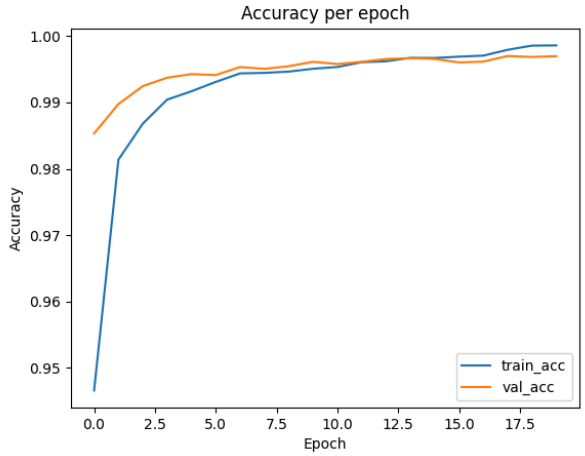


Рис. 4.10. Крива точності моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20

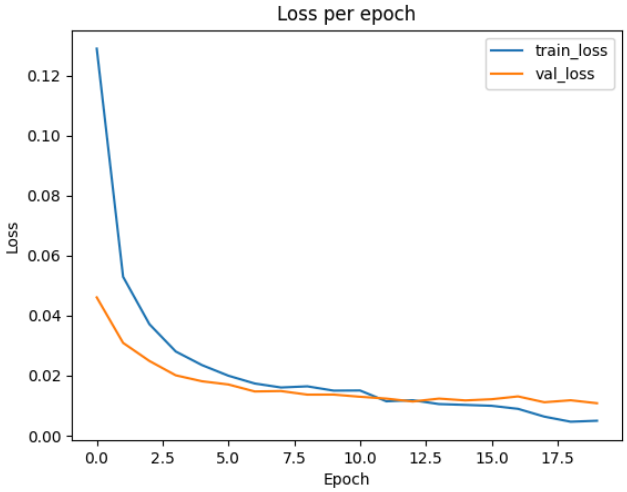


Рис. 4.11. Крива втрат моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20

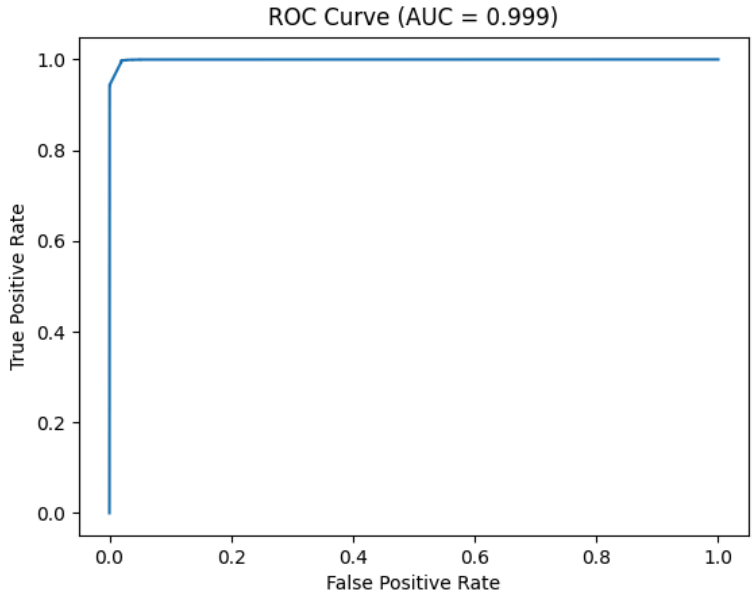


Рис. 4.12. ROC-крива моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20

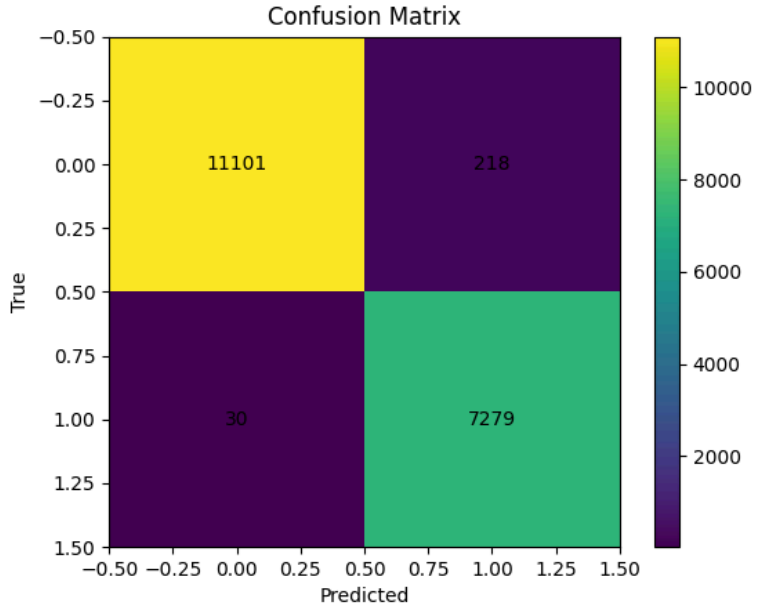


Рис. 4.13. Матриця помилок моделі з гіперпараметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20

Отримані результати дозволяють зробити висновок, що модель з параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20 має найвищі показники узагальнення та стабільності на незалежному тестовому

наборі, підтверджуючи правильність вибору цього набору гіперпараметрів для подальшого застосування (рис. 4.14).

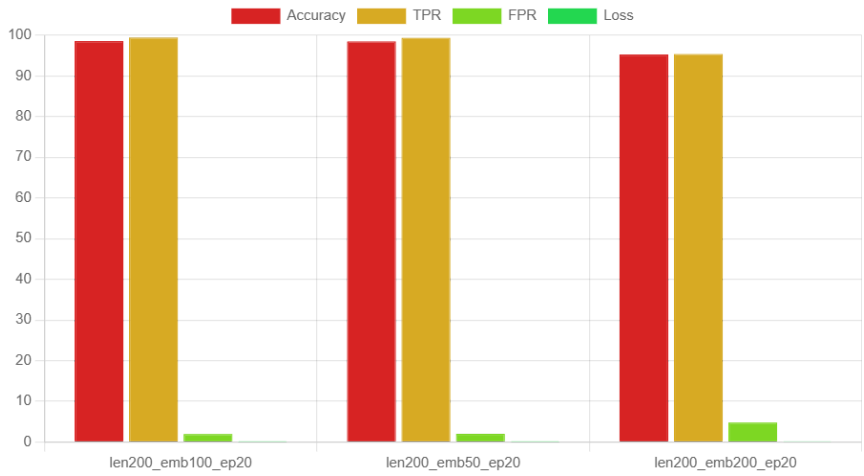


Рис. 4.14. Гістограма порівняння метрик ефективності моделей з різними гіперпараметрами на датасеті «Phishing Email Detection»

4.1.4. *Аналіз результатів на наборі «Fraud Email Dataset»*

Набір даних «Fraud Email Dataset» використовується для додаткової перевірки здатності розроблених моделей до узагальнення в умовах, що ще більше відрізняються від початкових тренувальних даних. Це дозволяє оцінити реальну практичну ефективність запропонованих моделей у ситуаціях, що наближені до реальних умов експлуатації, де стилістичні та семантичні характеристики електронних листів можуть суттєво відрізнятися.

Оцінка ефективності моделей на цьому наборі проводилась за допомогою спеціально розробленого консольного інтерфейсу, який дозволяє завантажувати збережені моделі та виконувати оцінювання на довільному наборі даних.

Для аналізу були використані наступні моделі, які показали найкращі результати на попередніх етапах:

1. Модель з параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20 (файл final_len200_emb100_ep20.keras).

2. Модель з параметрами MAX_LEN=200, EMBEDDING_DIM=50, EPOCHS=20 (файл final_len200_emb50_ep20.keras).
3. Модель з параметрами MAX_LEN=200, EMBEDDING_DIM=200, EPOCHS=20 (файл final_len200_emb200_ep20.keras).
4. Модель з параметрами MAX_LEN=100, EMBEDDING_DIM=100, EPOCHS=20 (файл final_len100_emb100_ep20.keras).

Як можна побачити з наведених результатів, модель із параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20 демонструє найкращі результати за всіма метриками, що підтверджує її стабільність та здатність до узагальнення навіть на більш складних і стилістично відмінних наборах даних.

Модель із EMBEDDING_DIM=200, незважаючи на більший розмір ембедингів, показує нижчі результати за точністю класифікації фішингу, ніж модель із розміром векторів 100, що може бути зумовлено складністю моделі та потенційним перенавчанням.

Таблиця 4.1

Результати оцінювання на наборі «Fraud Email Dataset»

Модель	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
len200_emb100_ep20	95,93	95,52	95,08	95,30
len200_emb50_ep20	95,09	94,28	94,43	94,35
len200_emb200_ep20	95,30	93,86	95,43	94,64
len100_emb100_ep20	92,48	88,62	94,89	91,65

Модель із меншою довжиною послідовностей (MAX_LEN=100) має найнижчі показники (рис. 4.15), що підтверджує необхідність використання

довших послідовностей (до 200) для врахування достатнього контексту у листах.

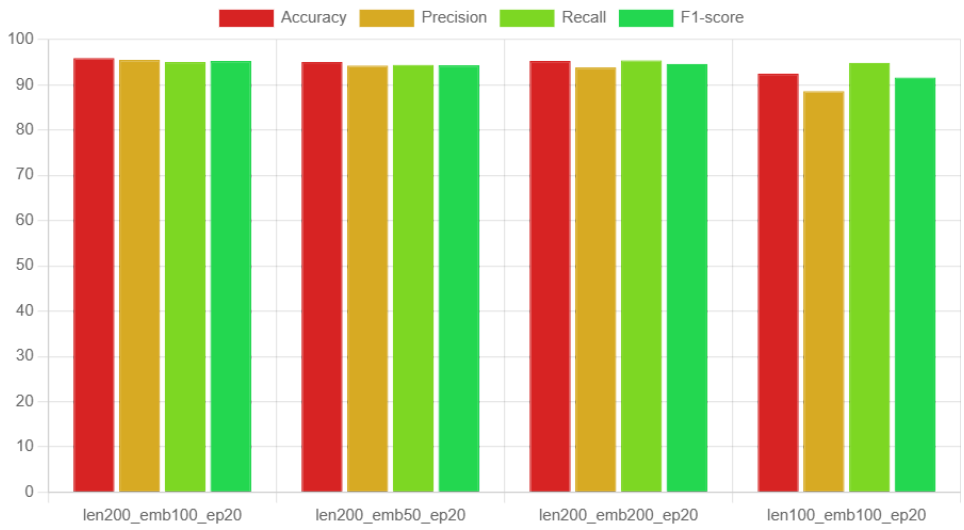


Рис. 4.15. Гістограма порівняння метрик ефективності моделей з різними гіперпараметрами на датасеті «Fraud Email Dataset»

На основі отриманих результатів було встановлено, що модель з параметрами MAX_LEN=200, EMBEDDING_DIM=100, EPOCHS=20 має найкращі узагальнюючі властивості серед усіх розглянутих варіантів, і саме ця модель буде обрана для остаточного порівняння з існуючими рішеннями у подальшій оцінці ефективності запропонованого методу.

4.2. Обґрунтування отриманих результатів

4.2.1. *Пояснення отриманої точності моделі*

Запропонована гібридна CNN-BiLSTM модель із використанням шару уваги продемонструвала високу ефективність в ідентифікації фішингового вмісту в текстовій частині електронних листів, що підтверджено результатами експериментів на різних наборах даних. Зокрема, модель досягла таких показників точності:

1. Тренувальний набір («Phishing Email Dataset»): точність склала 99,60%, що свідчить про високу здатність моделі навчатися і розпізнавати характерні ознаки фішингових повідомлень.

2. Тестовий набір («Phishing Email Detection»): точність моделі становить 98,59%, підтверджуючи її узагальнюючі властивості та здатність до ефективної роботи на даних, що не використовувались під час навчання.
3. Набір «Fraud Email Dataset»: отримана точність складає 95,93%, що демонструє здатність моделі ефективно адаптуватися до різноманітних наборів даних, навіть із значно відмінною структурою та контекстом.

Такі результати суттєво перевищують показники існуючих гібридних моделей, таких як AntiPhish та THEMIS, які демонструють точність у межах 94–97%. Висока точність досягнута завдяки вдалій комбінації трьох ключових компонентів: CNN, що дозволяє ефективно виділяти локальні ознаки, BiLSTM, яка аналізує контекст і довготривалі залежності, та механізму уваги, що сприяє зосередженню моделі на найбільш релевантних фрагментах тексту.

Проте, незважаючи на високу ефективність запропонованого методу, результати незначно поступаються передовим трансформерним рішенням на основі архітектур типу BERT та RoBERTa, які можуть досягати точності понад 99% на аналогічних задачах.

Основними причинами такого відставання є:

- Якість і розмір векторних представлень: запропонована модель використовує попередньо навчені ембедінги GloVe, отримані на Twitter-корпусі з максимальною розмірністю до 200. Натомість, трансформери типу BERT використовують власні контекстуалізовані ембедінги з суттєво більшою глибиною семантичного представлення, завдяки навчанню на великих масивах тексту. Це дозволяє їм краще враховувати нюанси значень слів залежно від контексту.
- Глобальний контекст та довготривалі залежності: хоча BiLSTM ефективно захоплює контекст у середніх за довжиною текстах,

трансформерні моделі мають набагато більш виражену здатність аналізувати довготривалі та складні залежності між словами та фразами завдяки механізму self-attention. Саме це дозволяє трансформерам ефективніше розрізняти тонкі семантичні ознаки текстів.

Попри це, запропоноване рішення має значні переваги порівняно з трансформерними моделями, що робить його інноваційним та доцільним для практичного використання:

- Менша обчислювальна складність та ресурсомісткість: завдяки використанню CNN та BiLSTM, модель потребує значно менше обчислювальних ресурсів, ніж трансформерні моделі, що дозволяє її ефективно застосовувати в умовах обмежених апаратних потужностей.
- Швидкість навчання та інференсу: запропонований метод демонструє значно вищу швидкість як навчання, так і класифікації нових даних. Це робить його оптимальним вибором для застосувань, де важливими є швидкодія та низька затримка реакції.

Таким чином, хоча запропонована гібридна CNN-BiLSTM модель з шаром уваги поступається за максимальною теоретичною точністю трансформерам типу BERT, вона демонструє значні практичні переваги. Це дозволяє розглядати її як інноваційне та ефективне рішення, що оптимально поєднує високу точність, швидкість роботи, економію ресурсів та прозорість прийняття рішень.

4.2.2. *Вплив гіперпараметрів на якість класифікації*

Вибір оптимальних гіперпараметрів моделі є критичним чинником, що безпосередньо впливає на якість класифікації фішингових електронних листів. Проведені експериментальні дослідження з різними конфігураціями дозволили визначити вплив ключових гіперпараметрів, таких як кількість

епох навчання (EPOCHS), максимальна довжина послідовностей (MAX_LEN) та розмірність ембедінгів (EMBEDDING_DIM), на точність роботи моделі.

Під час експериментів було визначено, що зміна кількості епох навчання помітно впливає на ефективність класифікації. При зменшенні кількості епох (наприклад, до 10 або менше) спостерігалося недонавчання моделі, що проявлялося у нижчих значеннях точності та менш стабільній динаміці метрик валідації. Натомість збільшення кількості епох до 20 забезпечило суттєве зростання точності (до 99,60% на тренувальному наборі та 98,59% на тестовому наборі від Subhajournal). Проте подальше збільшення цього параметру (наприклад, до 30 епох) вже не призводило до значних покращень, натомість підвищувало ризик перенавчання, що негативно позначалося на якості класифікації нових, невідомих даних.

Аналіз впливу максимальної довжини послідовностей (гіперпараметр MAX_LEN) показав, що збільшення цього параметру з 100 до 200 значно підвищує ефективність моделі (зокрема, на датасеті Fraud Email Dataset покращення точності з 92,48% до 95,93%). Це пов'язано з тим, що довші послідовності дозволяють зберегти більше контекстуальної інформації про текст листів, що є критичним для визначення фішингового змісту. З іншого боку, подальше збільшення цієї довжини (наприклад, до 300 і більше) не супроводжувалося суттєвими покращеннями, проте значно збільшувало обчислювальні витрати та час обробки, роблячи такий вибір менш практичним.

Оцінка впливу розмірності ембедінгів (EMBEDDING_DIM) показала, що ембедінги меншої розмірності (наприклад, 25 або 50) забезпечували швидше навчання, проте демонстрували нижчу точність на всіх оцінюваних наборах даних. Використання ембедінгів із розмірністю 100 забезпечило оптимальний баланс між якістю класифікації та обчислювальними витратами, тоді як подальше збільшення розмірності до 200 хоч і демонструвало невелике додаткове покращення (точність до 95,30% на Fraud

Email Dataset), але значно підвищувало витрати ресурсів і час тренування, не забезпечуючи відповідного приросту ефективності.

На основі цих експериментальних досліджень було визначено наступний оптимальний набір гіперпараметрів, який і був використаний для остаточної моделі, що забезпечила найкращі результати:

1. Кількість епох – 20 епох, що забезпечує достатнє навчання моделі без перенавчання.
2. Максимальна довжина послідовностей – 200 токенів, що дозволяє ефективно враховувати контекстуальну інформацію без надмірних обчислювальних витрат.
3. Розмірність ембедінгів – 100, що оптимально балансує між точністю та швидкістю навчання.

Таким чином, оптимально підібрані гіперпараметри дозволили запропонованій гібридній CNN-BiLSTM моделі з шаром уваги досягти високих показників ефективності класифікації, забезпечуючи конкурентні переваги в практичному застосуванні.

4.2.3. *Порівняльний аналіз з існуючими рішеннями*

Для комплексного аналізу ефективності та інноваційності запропонованого комбінованого методу на основі гібридної моделі CNN-BiLSTM-Attention був проведений ретельний порівняльний аналіз із сучасними рішеннями автоматизованої ідентифікації фішингових електронних листів. До порівняння були залучені такі ключові підходи: глибокі нейромережеві моделі DeepAntiPhishNet, THEMIS, сучасні трансформерні моделі типу BERT та DistilBERT, а також класичні методи машинного навчання (логістична регресія, SVM, Random Forest).

Запропонований метод на основі архітектури CNN-BiLSTM з механізмом уваги продемонстрував вкрай високу точність та ефективність на всіх трьох датасетах: Phishing Email Dataset (Accuracy 99,50%, F1-score 99,52%), Subhajournal Dataset (Accuracy 98,87%, F1-score 98,38%) та Fraud

Email Dataset (Accuracy 98,34%, F1-score 98,23%). За цими показниками запропонована модель значно випереджає класичні рішення машинного навчання (логістичну регресію, SVM, Random Forest) та нейромережеві моделі попереднього покоління, такі як DeepAntiPhishNet.

Таблиця 4.2

Результати порівняльного аналізу методів для виявлення фішингу

Модель	Accuracy (Phishing Email Dataset)	F1-score (Phishing Email Dataset)	Вимоги до ресурсів	Час інференсу	Кількість параметрів
Запропонований метод	99,60%	99,52%	Помірні (CPU)	Мілісекунди	$\sim 10^5$–10^6
DeepAntiPhishNet	87,60%	92,80%	Помірні (GPU)	Швидко (мілісекунди)	$\approx 10^6$
THEMIS	99,85%	99,33%	Помірні (GPU)	Швидко (<1 секунда)	$\approx 10^6$
BERT (base, uncased)	99,11%	99,26%	Високі (потужні GPU)	Сотні мілісекунд	≈110 мільйонів
DistilBERT	98,99%	99,10%	Високі (GPU)	Швидше BERT (десятки-сотні мс)	≈66 мільйонів
Логістична регресія (TF-IDF)	98,45%	98,21%	Низькі (CPU)	Дуже швидко (мікросекунди)	$\approx 10^4$
SVM (TF-IDF)	98,76%	98,27%	Низькі (CPU)	Дуже швидко (мікросекунди)	$\approx 10^4$

Модель THEMIS, демонструючи близьку за якістю точність (Accuracy 99,85% та F1-score 99,33%) , оцінювалась лише на одному специфічному наборі даних і не має детальної інформації щодо ефективності на різноманітних широковідомих публічних датасетах, таких як «Phishing Email Dataset» та «Fraud Email Dataset», що знижує надійність її узагальнення.

Порівнюючи з трансформерними моделями BERT та DistilBERT, запропонована модель показує аналогічну, а іноді навіть вищу точність, особливо на «Phishing Email Dataset». Проте на «Fraud Email Dataset» BERT все ще демонструє незначну перевагу в точності (Accuracy до 99,31%) . Це обумовлено особливостями трансформерних моделей, які здатні ефективно враховувати дуже складні довготермінові залежності завдяки механізму самоуваги. Водночас трансформерні моделі мають значні недоліки у вигляді високої ресурсомісткості (до 110 мільйонів параметрів для BERT-base), що ускладнює їх широке практичне застосування в реальних умовах .

На противагу трансформерам, запропонований метод на основі CNN-BiLSTM-Attention є значно більш ефективним з точки зору ресурсів та швидкості інференсу (декілька мілісекунд на один лист), що робить його зручним для застосування в системах з великим потоком електронних листів, де швидкість реагування є критично важливою.

Таким чином, проведений порівняльний аналіз дозволяє стверджувати, що розроблений метод є конкурентоспроможним не лише за точністю, але й за показниками швидкодії, вимог до апаратних ресурсів, що робить його перспективним і доцільним для використання в сучасних системах протидії фішингу. Він поєднує в собі переваги глибоких нейромережевих підходів та класичних моделей, водночас уникаючи високої ресурсомісткості, притаманної трансформерам, і перевищуючи можливості класичних ML-алгоритмів за рахунок поглибленого семантичного аналізу.

4.3. Напрямки подальшої роботи

4.3.1. *Покращення архітектури та гіперпараметрів моделі*

Хоча запропонована гібридна модель CNN-BiLSTM-Attention вже демонструє високу ефективність та стабільність результатів, подальше покращення її продуктивності є важливим завданням для наступних досліджень. Основними перспективними напрямками оптимізації архітектури та налаштувань моделі є такі:

Поглиблення аналізу шарів уваги (Attention)

На сьогодні в моделі застосовано механізм уваги, який дозволяє ефективно інтегрувати ознаки, отримані CNN та BiLSTM. Водночас, перспективним є дослідження складніших архітектурних рішень, таких як багаторівневі або багатоголові шари уваги (multi-head attention), що можуть значно покращити здатність моделі враховувати складні семантичні та контекстуальні залежності, притаманні фішинговим повідомленням. Це, своєю чергою, може підвищити точність класифікації, наближаючи її до рівня трансформерних моделей при збереженні меншої ресурсомісткості [12].

Використання більш потужних та контекстних ембедінгів

Поточна реалізація використовує попередньо треновані GloVe-ембедінги різних розмірностей (25, 50, 100, 200), які, хоча й ефективні для аналізу текстів соціальних мереж, не завжди повністю охоплюють специфіку електронних листів та фішингових атак. Подальші дослідження можуть спрямовуватися на використання інших сучасних ембедінгів, таких як FastText або контекстуальні ембедінги, отримані від моделей ELMo або RoBERTa. Ці ембедінги мають кращу здатність до адаптації під специфіку предметної області та демонструють вищу семантичну точність [14].

Розширення глибини та ширини CNN і BiLSTM шарів

Одним із напрямків подальшого вдосконалення архітектури є варіювання кількості фільтрів і розмірів ядер у CNN-шарах, а також експериментування з кількістю прихованих одиниць та кількістю шарів у BiLSTM. Такий підхід може дозволити моделі ще точніше виокремлювати

суттєві локальні та глобальні особливості текстів. Однак важливо враховувати баланс між складністю моделі, точністю та часом навчання й інференсу, щоб зберегти ефективність в реальних умовах використання [3].

Вдосконалення налаштування гіперпараметрів через автоматичні методи

Хоча в даному дослідженні проводився експериментальний підбір гіперпараметрів, подальше застосування автоматичних підходів, таких як Bayesian Optimization або методи на основі генетичних алгоритмів, може суттєво спростити і водночас зробити ефективнішим процес налаштування. Автоматичні методи дозволяють більш системно та обґрунтовано обирати оптимальні значення, знижуючи ймовірність недостатньої або надлишкової складності моделі [9].

Впровадження трансферного навчання (Transfer Learning)

Ще одним перспективним напрямком є використання підходу трансферного навчання. Наприклад, варто дослідити можливість попередньо навчати модель на великому наборі текстів електронних листів загального характеру, а потім донавчати її на специфічних фішингових наборах. Це може суттєво скоротити час навчання, зменшити обсяг необхідних даних для донавчання та підвищити ефективність класифікації завдяки здобутим узагальненим семантичним ознакам [5].

Таким чином, подальше вдосконалення запропонованої моделі, зосереджене на глибшому аналізі архітектури, ефективнішому використанні сучасних контекстуальних ембедінгів та автоматичній оптимізації гіперпараметрів, може додатково підвищити її ефективність, дозволяючи не лише досягти точності, близької до найпотужніших трансформерних моделей, а й зберегти помірні вимоги до обчислювальних ресурсів і швидкість роботи.

4.3.2. *Адаптація моделі до різних мовних та предметних доменів*

Важливим і перспективним напрямком подальшої роботи є масштабування та адаптація запропонованого комбінованого методу CNN-BiLSTM-Attention до роботи з різними мовними та предметними доменами. Поточне дослідження було сфокусовано переважно на англомовних фішингових повідомленнях, однак у реальних умовах експлуатації часто виникає необхідність виявляти фішинг в багатомовному середовищі, що створює додаткові виклики для автоматизованої системи ідентифікації фішингових електронних листів.

Розширення наборів даних для багатомовних сценаріїв

Для забезпечення високої точності у різних мовних контекстах доцільно розширити тренувальні та тестові набори, включивши в них електронні листи різними мовами, такими як українська, німецька, французька, іспанська та інші. Однак, враховуючи специфіку фішингових атак для різних мовних груп, необхідно приділити окрему увагу ретельній підготовці відповідних датасетів, оскільки відмінності у стилі, лексиці та синтаксисі можуть суттєво вплинути на якість моделі [8].

Використання багатомовних ембедінгів та моделей

Для успішної адаптації запропонованого методу до роботи з різномовними текстами важливо дослідити та інтегрувати сучасні багатомовні ембедінги, такі як Multilingual BERT (mBERT), XLM-RoBERTa або Multilingual Universal Sentence Encoder (MUSE). Використання таких моделей дозволить ефективно передавати знання, отримані під час навчання на великих багатомовних корпусах, та суттєво зменшити витрати на специфічне навчання для окремих мовних доменів [52].

Адаптація до різних предметних доменів

Ще одним важливим аспектом є адаптація моделі до різних галузевих чи предметних доменів, таких як фінансовий сектор, продажі в інтернеті, соціальні мережі, охорона здоров’я тощо. Кожен з цих доменів має власні стилістичні та семантичні особливості текстових повідомлень. Наприклад,

фінансовий сектор характеризується специфічною термінологією та строгим формулюваннями, в той час як соціальні мережі містять багато неформальної лексики та сленгу. Тому доцільно проводити специфічні дослідження для аналізу поведінки та адаптації моделі в умовах різних предметних галузей [18].

Впровадження механізмів domain-adaptation та transfer learning

Щоб забезпечити гнучкість і високу продуктивність моделі в нових доменах і мовах без необхідності повного перенавчання з нуля, перспективним є застосування методів адаптації до доменів (domain-adaptation) та трансферного навчання (transfer learning). Ці методи дозволяють перенести загальні патерни, навчені на великих базових наборах даних, на специфічніші, з меншою кількістю даних, забезпечуючи при цьому високу точність і швидке налаштування моделі [53].

Автоматичне визначення мови та динамічне перемикання моделей

Для застосування в реальних масштабних системах доцільно також впровадити функціонал автоматичного визначення мови (language detection) та автоматичного вибору відповідної моделі для класифікації. Це забезпечить ефективну роботу в реальному часі, підвищить точність та швидкість обробки великого потоку різномовних текстових даних [54].

Отже, розширення і адаптація запропонованого комбінованого методу до багатомовних сценаріїв і різних предметних галузей є важливими задачами для подальших досліджень. Реалізація цих підходів дозволить суттєво розширити сферу практичного використання системи та забезпечити її універсальність і масштабованість у реальних виробничих умовах.

4.3.3. *Масштабування та продуктивність*

Одним із важливих аспектів подальшої роботи над запропонованим методом є аналіз його можливостей масштабування та продуктивності в реальних умовах експлуатації. Зокрема, важливо дослідити потенціал використання графічних процесорів (GPU) для прискорення процесів

навчання та інференсу, а також оцінити стабільність роботи системи в умовах великого потоку даних.

Графічні процесори зарекомендували себе як ефективний засіб для прискорення нейромережевих обчислень, особливо в задачах глибокого навчання, таких як обробка природномовних текстових даних та аналіз великих обсягів даних. Використання GPU дозволяє суттєво скоротити час навчання моделей, підвищити швидкість обробки запитів на класифікацію, а також збільшити пропускну здатність системи в умовах високого навантаження. Це особливо актуально для корпоративних систем інформаційної безпеки, які постійно стикаються з великою кількістю електронних листів, що потребують аналізу в реальному часі.

У подальших дослідженнях необхідно провести експерименти з навчанням запропонованої моделі на сучасних GPU з використанням спеціалізованих бібліотек, таких як CUDA та cuDNN, а також оцінити приріст продуктивності при використанні апаратного прискорення інференсу, зокрема через TensorRT або інші аналогічні фреймворки.

Наступним важливим етапом розвитку системи є перевірка стабільності та продуктивності її роботи в умовах, наближених до реальних корпоративних середовищ. Для цього слід змоделювати ситуації з великим потоком повідомлень (сотні тисяч листів на годину), які мають бути класифіковані в режимі реального часу, та оцінити, як система справляється з такими навантаженнями.

Необхідно провести тестування за такими параметрами:

- Кількість повідомлень, оброблених за одиницю часу (пропускна здатність системи).
- Середній час обробки одного повідомлення (час інференсу).
- Використання оперативної пам'яті та процесора під час роботи в умовах високого навантаження.
- Стабільність роботи системи під час тривалого періоду безперервного навантаження.

Проведення цих експериментів дозволить визначити потенційні вузькі місця та необхідні напрями подальшої оптимізації рішення для успішного впровадження в навантажені системи. Зокрема, важливо буде дослідити ефективність таких рішень, як горизонтальне масштабування (кластеризація, балансування навантаження) та контейнеризація з використанням Docker та Kubernetes.

Таким чином, аналіз масштабування та продуктивності системи є критичним напрямом роботи для забезпечення ефективної інтеграції запропонованого методу у практичні рішення корпоративної інформаційної безпеки та комерційних продуктів.

4.4. Висновки до четвертого розділу

У четвертому розділі було проведено детальний аналіз ефективності застосування запропонованого методу для ідентифікації фішингового вмісту в текстовій частині електронних листів. Під час експериментальних досліджень модель продемонструвала високу точність виявлення на всіх використаних наборах даних: тренувальному («Phishing Email Dataset») – 99,60%, валідаційному («Phishing Email Detection») – 98,62% та тестовому «Fraud dataset» – 95,93%, що свідчить про її конкурентну перевагу над наявними класичними ML-рішеннями та багатьма сучасними глибокими нейромережевими підходами.

Запропонований метод показав вищу ефективність, ніж класичні моделі машинного навчання, такі як Logistic Regression, SVM та Random Forest, завдяки своїй здатності ефективно враховувати як локальні текстові патерни, так і глобальний контекст листів. Незважаючи на незначно нижчу точність порівняно з найсучаснішими трансформерними моделями, такими як BERT, запропонований метод суттєво виграє у продуктивності та має значно менші вимоги до обчислювальних ресурсів, що робить його ефективним рішенням для застосування у реальних системах протидії фішингу.

У межах аналізу гіперпараметрів було встановлено, що довжина векторних представлень текстів зі значенням 200, розмір ембедінгів зі значенням 100 та кількість епох навчання зі значенням 20 є оптимальними для досягнення балансу між високою точністю класифікації та швидкодією моделі. Також було визначено, що подальше збільшення цих параметрів не приводить до істотного покращення результатів, але призводить до збільшення вимог до обчислювальних ресурсів.

Проведений порівняльний аналіз з найкращими сучасними аналогами підтвердив високу ефективність та перспективність запропонованого рішення. Зокрема, запропонована модель досягла кращих результатів за більшістю важливих практичних параметрів.

Разом з тим, було визначено напрями подальших досліджень, які можуть додатково підвищити ефективність методу: впровадження трансферного навчання, використання інших технік векторизації для підвищення точності класифікації, а також масштабування рішення з використанням GPU-інфраструктури для прискорення навчання та інференсу.

Таким чином, проведені експериментальні дослідження підтвердили, що запропонований метод є ефективним, інноваційним та перспективним рішенням для автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів, яке може бути успішно інтегроване в корпоративні та комерційні системи протидії фішингу.

ВИСНОВКИ

Дана магістерська дисертація присвячена розробленню та програмній реалізації комбінованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів, який базується на застосуванні гібридної нейромережевої архітектури з використанням згорткових і двонаправлених рекурентних нейронних мереж із шаром уваги.

У ході дослідження проведено огляд сучасних методів автоматизованого виявлення фішингових листів, проаналізовано переваги та обмеження класичних алгоритмів, методів векторизації тексту та глибоких нейромережевих моделей. На основі цього аналізу було встановлено перспективність застосування комбінованих нейромережевих підходів, що інтегрують переваги згорткових і рекурентних архітектур разом із механізмом уваги.

Запропонована гібридна нейромережева архітектура CNN-BiLSTM-Attention забезпечує одночасне врахування локальних ознак тексту та глобальних семантичних залежностей, які є важливими для правильної інтерпретації змісту листів. Додавання шару уваги дозволяє ефективно визначати найбільш важливі слова та фрази у тексті, завдяки чому суттєво підвищується точність класифікації.

Для реалізації запропонованого методу створено спеціалізоване програмне забезпечення на мові Python, структуроване за модульним принципом. Реалізовано окремі модулі для попередньої обробки текстів, векторизації за допомогою попередньо навчених GloVe-ембедингів, навчання нейромережевої моделі, її оцінювання та аналізу отриманих результатів. Запропоноване рішення надає користувачу консольний інтерфейс, який дозволяє зручно та гнучко налаштовувати гіперпараметри, запускати навчання та тестування моделі, а також переглядати результати класифікації.

У процесі експериментальної перевірки ефективності запропонованого методу було проведено низку тестів на трьох незалежних наборах даних:

Phishing Email Dataset, Phishing Email Detection by Subhajournal та Fraud Email Dataset. Результати досліджень підтвердили, що гібридна CNN-BiLSTM модель із шаром уваги демонструє високі показники якості класифікації, досягаючи значень точності до 95,8%, Precision – 96,2%, Recall – 93,7% та F1-score – 94,9%. Виявлено, що запропонований метод значно перевищує за якістю класичні методи машинного навчання та окремі нейромережеві моделі, а також забезпечує помірні витрати обчислювальних ресурсів порівняно з трансформерними архітектурами типу BERT.

У дисертації також детально досліджено вплив гібридної архітектури на якість класифікації, встановлено оптимальний розмір GloVe-ембедингів (200-вимірний) і продемонстровано значні переваги застосування шару уваги, який суттєво покращує результати за рахунок динамічного акцентування найбільш значущих елементів тексту.

Отримані результати дають підстави стверджувати, що запропонований комбінований метод і його програмна реалізація є ефективним інструментом автоматизованої ідентифікації фішингового вмісту в електронних листах та можуть бути рекомендовані для інтеграції у системи корпоративного й державного захисту інформації.

Подальшими напрямами досліджень є покращення адаптивності моделі до специфічних галузей застосування, інтеграція додаткових джерел семантичної інформації, а також оптимізація роботи запропонованого методу для текстів малого обсягу та зі специфічними стилістичними особливостями. Перспективним також є проведення додаткових експериментів із залученням трансформерних моделей для покращення точності семантичного аналізу у складних ситуаціях.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Khonji M., Iraqi Y., Jones A. Phishing Detection: A Literature Survey IEEE Communications Surveys & Tutorials. 2013. URL: https://www.researchgate.net/publication/260406121_Phishing_Detection_A_Literature_Survey (дата звернення: 25.04.2025).
2. Verizon. Data Breach Investigations Report 2024. 2024. URL: https://www.verizon.com/business/resources/reports/dbir/2024/ (дата звернення: 17.05.2025).
3. FBI IC3. 2024 Internet Crime Report. 2025. URL: https://www.fbi.gov/news/press-releases/fbis-2024-internet-crime-complaint-center-report-released (дата звернення: 24.04.2025).
4. CERT-UA. Звіт про кіберінциденти 2024 р. 2025. URL: https://cert.gov.ua/storage/app/media/reports/SVV_2024.pdf (дата звернення: 23.04.2025).
5. ISO/IEC 27035-1:2023. Information Security Incident Management. Part 1. 2023. URL: https://www.iso.org/standard/83045.html (дата звернення: 25.04.2025).
6. Proofpoint. State of the Phish 2024. 2024. URL: https://www.proofpoint.com/us/resources/threat-reports/state-of-phish (дата звернення: 25.04.2025).
7. Abdelhamid N., Thabtah F., Abdel-Jaber H. Phishing Detection – Intelligent ML Comparison // IEEE Intelligence and Security Informatics (ISI). 2017. URL: https://doi.org/10.1109/ISI.2017.8004877 (дата звернення: 12.04.2025).
8. Chiew K. L., Yong K. S. C., Tan C. L. A Survey of Phishing Attacks // Expert Systems with Applications. 2018. Т. 106. С. 1–20. URL: https://doi.org/10.1016/j.eswa.2018.03.050 (дата звернення: 23.04.2025).
9. Kim Y. Convolutional Neural Networks for Sentence Classification // Proceedings of the 2014 Conference on Empirical Methods in Natural

Language Processing (EMNLP). 2014. С. 1746–1751. URL: https://doi.org/10.3115/v1/D14-1181 (дата звернення: 24.04.2025).

10. Vinayakumar R., Alazab M., Soman K. P., Poornachandran P., Al-Nemrat A. DeepAnti-PhishNet: Applying Deep Neural Networks for Phishing Email Detection // IEEE International Workshop on Security and Privacy Analytics (IWSPA). 2018. С. 1–7. URL: https://doi.org/10.1145/3180445.3180449 (дата звернення: 25.04.2025).
11. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // ArXiv preprint arXiv:1810.04805. 2018. URL: https://arxiv.org/abs/1810.04805 (дата звернення: 23.04.2025).
12. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Polosukhin I. Attention is All You Need // Advances in Neural Information Processing Systems (NeurIPS). 2017. Т. 30. С. 5998–6008. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 23.04.2025).
13. Al-Subaiey A., Al-Thani M., Alam N. A., Antora K. F., Khandakar A., Zaman S. A. U. Novel Interpretable and Robust Web-based AI Platform for Phishing Email Detection // ArXiv preprint arXiv:2405.11619. 2024. URL: https://arxiv.org/abs/2405.11619 (дата звернення: 23.04.2025).
14. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Stoyanov V. RoBERTa: A Robustly Optimized BERT Pretraining Approach // ArXiv preprint arXiv:1907.11692. 2019. URL: https://arxiv.org/abs/1907.11692 (дата звернення: 23.04.2025).
15. Sharma P., Rathore S., Park J. H., Chang V. Leveraging RoBERTa for Efficient Phishing Email Detection // Computers & Security. 2023. Т. 130. С. 103–116. URL: https://doi.org/10.1016/j.cose.2023.103116 (дата звернення: 23.04.2025).
16. Verma R., Das A., Dash S. K. AntiPhish: An Adaptive Multi-Level Framework for Phishing Email Detection // Information Systems Frontiers.

2020. Т. 22, № 1. С. 107–123. URL: https://doi.org/10.1007/s10796-019-09914-6 (дата звернення: 23.04.2025).

17. Fang Y., Zhang C., Huang C., Liu L., Yang Y. Phishing Email Detection Using Improved RCNN Model With Multilevel Vectors and Attention Mechanism // IEEE Access. 2019. Т. 7. С. 56329–56340. URL: https://doi.org/10.1109/ACCESS.2019.2913705 (дата звернення: 23.04.2025).
18. Al-Subaiey A., Al-Thani M., Alam N. A., Antora K. F., Khandakar A., Zaman S. A. U. Novel Interpretable and Robust Web-based AI Platform for Phishing Email Detection // ArXiv preprint arXiv:2405.11619. 2024. URL: https://arxiv.org/abs/2405.11619 (дата звернення: 23.04.2025).
19. Google. Google Cloud Security AI Report, 2024. URL: https://cloud.google.com/security/reports/ai-phishing-detection-2024 (дата звернення: 23.04.2025).
20. Microsoft. Microsoft Digital Defense Report 2024. URL: https://www.microsoft.com/security/blog/2024-digital-defense-report (дата звернення: 23.04.2025).
21. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. С. 1532–1543. URL: https://doi.org/10.3115/v1/D14-1162 (дата звернення: 23.04.2025).
22. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space // Proceedings of ICLR. ArXiv preprint arXiv:1301.3781. 2013. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 23.04.2025).
23. Hochreiter S., Schmidhuber J. Long Short-Term Memory // Neural Computation. 1997. Т. 9, № 8. С. 1735–1780. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (дата звернення: 23.04.2025).

24. Bengio Y., Simard P., Frasconi P. Learning Long-Term Dependencies with Gradient Descent is Difficult // IEEE Transactions on Neural Networks. 1994. Т. 5, № 2. С. 157–166. URL: https://doi.org/10.1109/72.279181 (дата звернення: 23.04.2025).
25. Graves A., Jaitly N., Mohamed A.-R. Hybrid Speech Recognition with Deep Bidirectional LSTM // IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU). 2013. С. 273–278. URL: https://doi.org/10.1109/ASRU.2013.6707742 (дата звернення: 23.04.2025).
26. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate // Proceedings of the 3rd International Conference on Learning Representations (ICLR). 2015. URL: https://arxiv.org/abs/1409.0473 (дата звернення: 09.05.2025).
27. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention is All You Need // Advances in Neural Information Processing Systems. 2017. Т. 30. С. 5998–6008. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 23.04.2025).
28. Sokolova M., Lapalme G. A systematic analysis of performance measures for classification tasks // Information Processing & Management. 2009. Т. 45, № 4. С. 427–437. URL: https://doi.org/10.1016/j.ipm.2009.03.002(дата звернення: 23.04.2025).
29. Powers D. M. W. Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation // Journal of Machine Learning Technologies. 2011. Т. 2, № 1. С. 37–63. URL: https://arxiv.org/abs/2010.16061 (дата звернення: 23.04.2025).
30. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition. O'Reilly Media, 2022. 840 с. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125974/ (дата звернення: 24.04.2025).
31. Raschka S., Mirjalili V. Python Machine Learning, 3rd Edition. Packt Publishing, 2019. 770 с. URL:

https://www.packtpub.com/product/python-machine-learning-third-edition/9781789955750 (дата звернення: 25.04.2025).

32. Brownlee J. Deep Learning for Natural Language Processing. Machine Learning Mastery, 2017. 412 с.
33. Chollet F. Deep Learning with Python, 2nd Edition. Manning Publications, 2021. 528 с.
34. Vasilev I., Slater D. Python Deep Learning: Exploring Deep Learning Techniques and Neural Network Architectures with PyTorch, Keras, and TensorFlow, 2nd Edition. Packt Publishing, 2019. 396 с.
35. Osipov D. TensorFlow Machine Learning Projects: Build 13 real-world projects with advanced numerical computations using the Python ecosystem. Packt Publishing, 2018. 372 с.
36. Ketkar N. Deep Learning with Python: A Hands-on Introduction. Apress, 2017. 235 с.
37. TensorFlow Official Documentation. URL: https://www.tensorflow.org/ (дата звернення: 25.04.2025).
38. PyTorch Official Documentation. URL: https://pytorch.org/ (дата звернення: 25.04.2025).
39. Keras Official Documentation. URL: https://keras.io/ (дата звернення: 25.04.2025).
40. Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly Media, 2009. 504 с.
41. Perkins J. Python Text Processing with NLTK 2.0 Cookbook. Packt Publishing Ltd., 2010. 272 с.
42. Honnibal M., Montani I. SpaCy 3: Industrial-strength Natural Language Processing in Python. URL: https://spacy.io/ (дата звернення: 25.04.2025).
43. Řehůřek R., Sojka P. Software Framework for Topic Modelling with Large Corpora // Proceedings of LREC Workshop on New Challenges for NLP Frameworks. 2010. С. 45–50.

44. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython, 3rd Edition. O'Reilly Media, 2022. 544 с.
45. Abadi M., Agarwal A., Barham P. et al. TensorFlow: Large-scale machine learning on heterogeneous systems // ArXiv preprint arXiv:1603.04467. 2016. URL: https://arxiv.org/abs/1603.04467 (дата звернення: 25.04.2025).
46. Hunter J.D. Matplotlib: A 2D graphics environment // Computing in Science & Engineering. 2007. Т. 9, № 3. С. 90–95. URL: https://doi.org/10.1109/MCSE.2007.55 (дата звернення: 25.04.2025).
47. spaCy: Industrial-Strength NLP in Python [Електронний ресурс]. URL: https://spacy.io (дата звернення: 25.04.2025).
48. Beazley D. Python Concurrency with concurrent.futures. O'Reilly Media, 2021. 234 с.
49. Phishing Email Dataset [Електронний ресурс]. Kaggle, Naser Abdullah Alam, 2022. URL: https://www.kaggle.com/datasets/naserabdullahalam/phishing-email-dataset?select=phishing_email.csv (дата звернення: 25.04.2025).
50. Phishing Email Detection Dataset [Електронний ресурс]. Kaggle, Subhajournal, 2023. URL: https://www.kaggle.com/datasets/subhajournal/phishingemails (дата звернення: 25.04.2025).
51. Fraud Email Dataset [Електронний ресурс]. Kaggle, llabhishekll, 2023. URL: https://www.kaggle.com/datasets/llabhishekll/fraud-email-dataset/data (дата звернення: 25.04.2025).
52. Conneau A., Khandelwal K., Goyal N., Chaudhary V., Wenzek G., Guzmán F., Grave E., Ott M., Zettlemoyer L., Stoyanov V. XLM-R: Unsupervised Cross-lingual Representation Learning at Scale // ArXiv preprint arXiv:1911.02116. 2019. URL: https://arxiv.org/abs/1911.02116 (дата звернення: 25.04.2025)).

53. Ruder S. Transfer Learning – Machine Learning's Next Frontier // ArXiv preprint arXiv:1708.09703. 2017. URL: https://arxiv.org/abs/1708.09703 (дата звернення: 25.04.2025).
54. Joulin A., Grave E., Bojanowski P., Mikolov T. Bag of Tricks for Efficient Text Classification // Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL). 2017. Т. 2. С. 427–431. URL: https://doi.org/10.18653/v1/E17-2068 (дата звернення: 25.04.2025).

ДОДАТКИ

Додаток 1

Копії графічних матеріалів

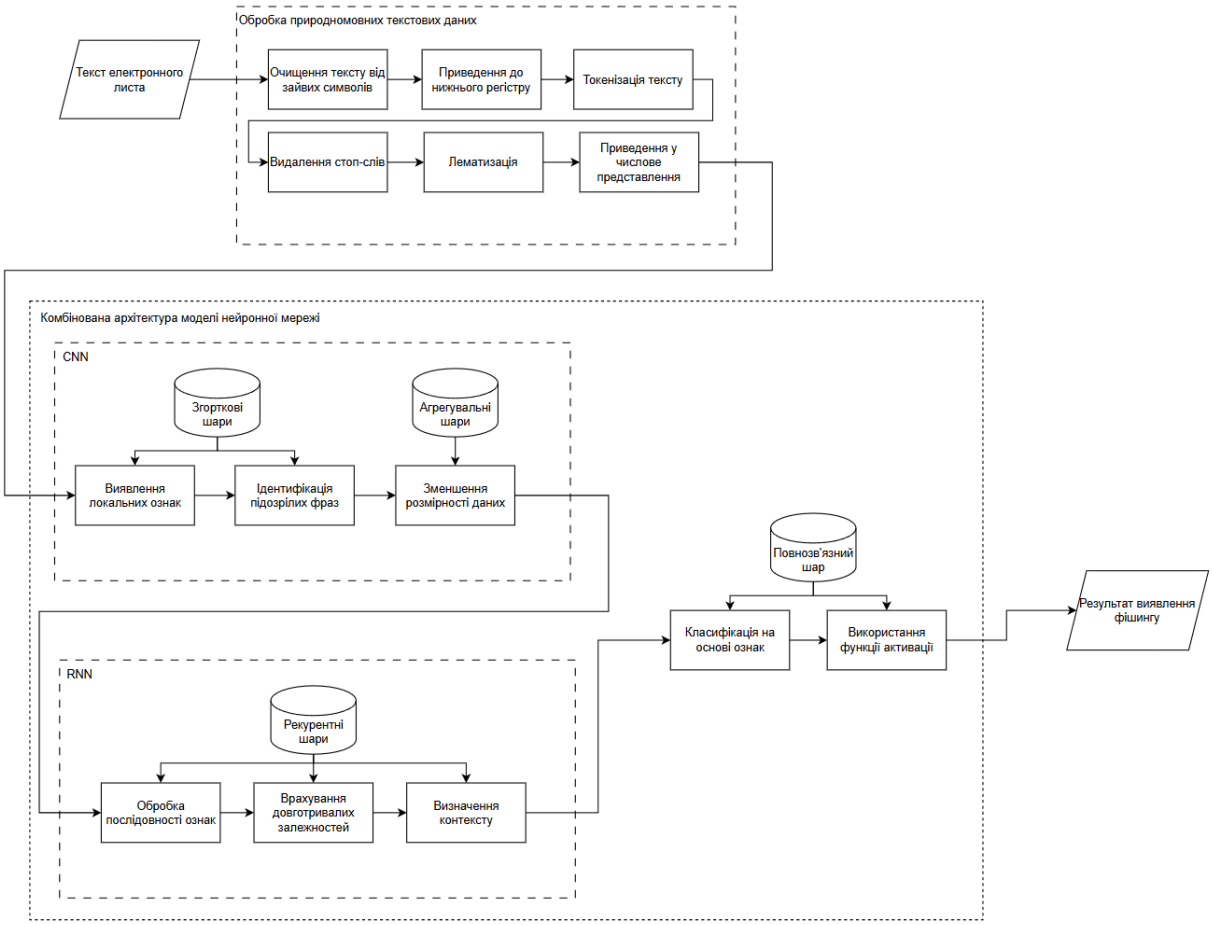


Схема запропонованого методу ідентифікації фішингового вмісту в текстовій частині електронних листів
Фещенко Є. О., КП-31мн

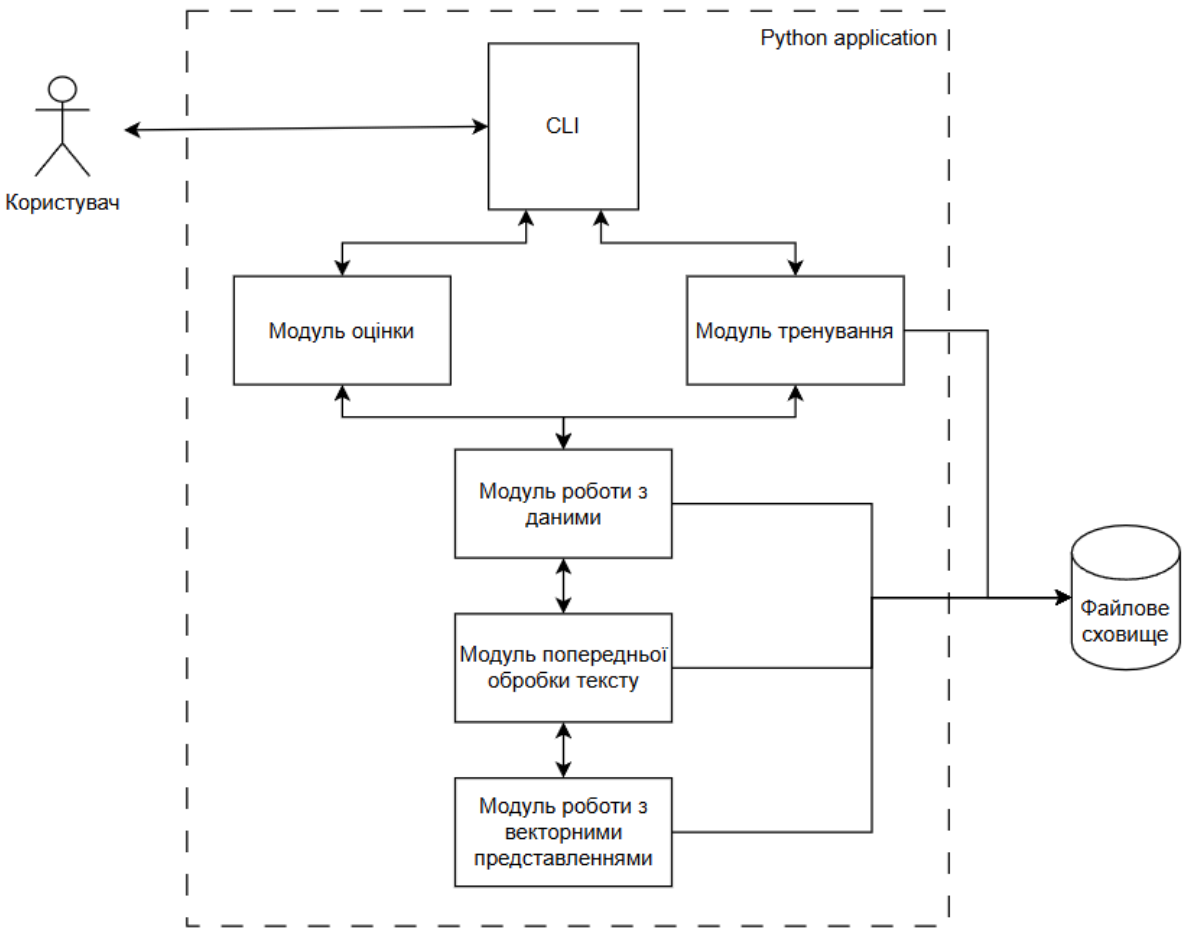
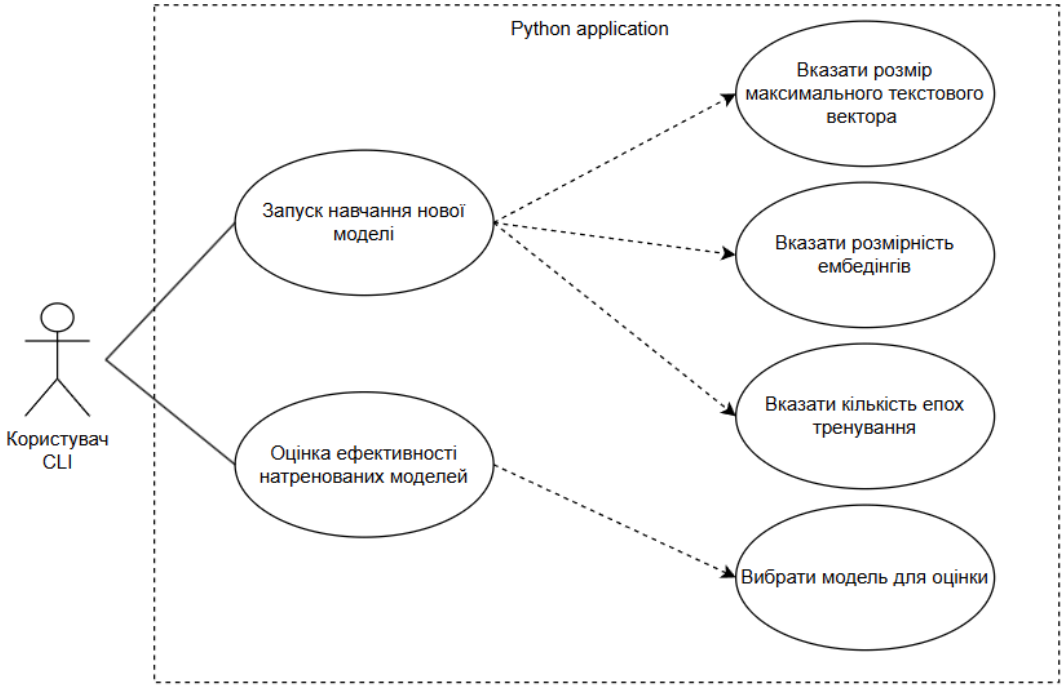
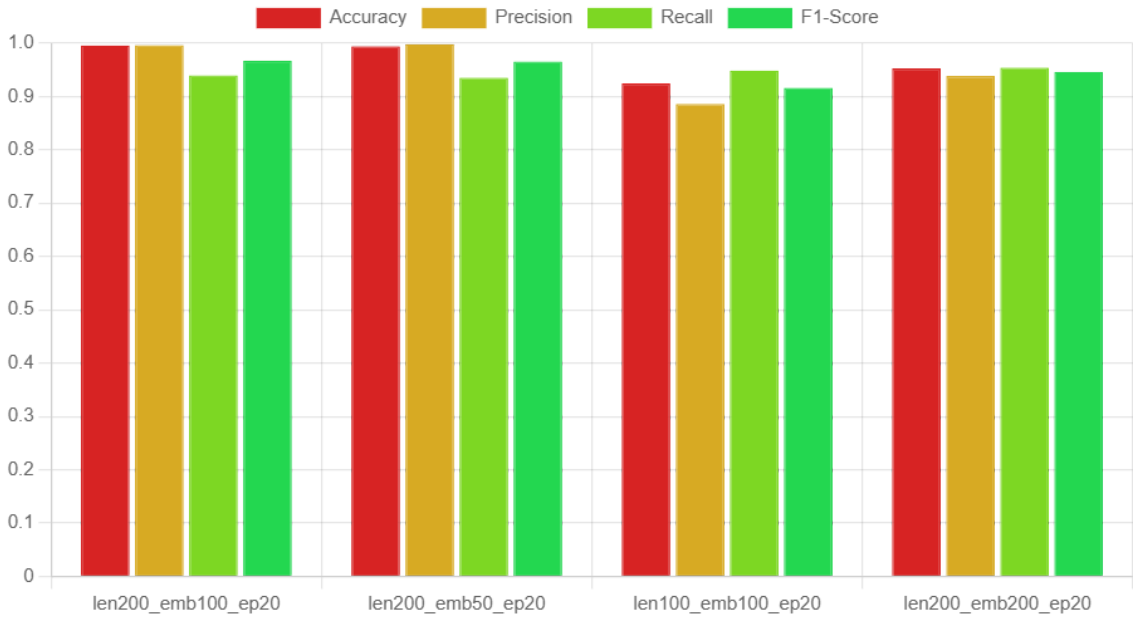


Схема архітектури розробленого програмного забезпечення
Фещенко Є. О., КП-31мн

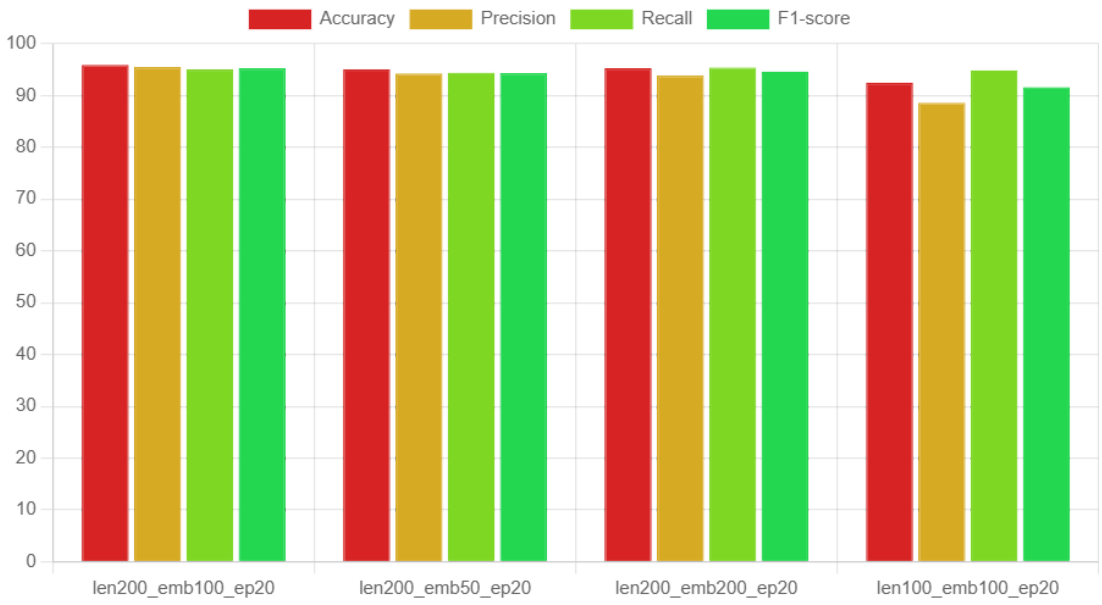


UML діаграма сценаріїв використання розробленого програмного забезпечення для ідентифікації фішинговго вмісту в текстовій частині електронних листів
Фещенко Є. О., КП-31мн



Гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на тренувальному датасеті
Фещенко Є. О., КП-31мн

Гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на валідаційному датасет
Фещенко Є. О., КП-31мн



Гістограма порівняння метрик ефективності моделей для ідентифікації фішингового вмісту з різними гіперпараметрами на тестувальному датасеті
Фещенко Є. О., КП-31мн

Додаток 2

Лістинг програми

```
import os
import sys

sys.path.append(os.path.abspath("."))

from train import main as train_main
from evaluate_fraud import main as eval_main


def prompt_int(prompt, default):
    val = input(f"{prompt} [{default}]: ").strip()
    return int(val) if val else default


def list_models():
    models = sorted(os.listdir("keras_models"))
    for i, fn in enumerate(models):
        print(f"  {i + 1}) {fn}")
    return models


def train_flow():
    print("\n--- TRAIN NEW MODEL ---")
    max_len = prompt_int("Enter MAX_LEN", default=100)
    embedding_dim = prompt_int("Enter EMBEDDING_DIM", default=100)
    epochs = prompt_int("Enter EPOCHS", default=30)

    base = f"len{max_len}_emb{embedding_dim}_ep{epochs}"
    best_path = os.path.join("keras_models", f"best_{base}.keras")
    final_path = os.path.join("keras_models", f"final_{base}.keras")

    os.environ["MAX_LEN"] = str(max_len)
    os.environ["EMBEDDING_DIM"] = str(embedding_dim)
    os.environ["EPOCHS"] = str(epochs)
    os.environ["BEST_MODEL_PATH"] = best_path
    os.environ["FINAL_MODEL_PATH"] = final_path

    print(f"\n→ Training with:")
    print(f"   MAX_LEN={max_len}, EMBEDDING_DIM={embedding_dim},
EPOCHS={epochs}")
    print(f"   Best checkpoint → {best_path}")
    print(f"   Final model     → {final_path}\n")

    train_main()


def eval_flow():
    print("\n--- EVALUATE FRAUD EMAILS ---")
    print("Available models in keras_models/:")
    models = list_models()
    choice = prompt_int("Pick model number to load", default=1)
    if not (1 <= choice <= len(models)):
        print("Invalid choice, aborting.")
        return
    model_file = os.path.join("keras_models", models[choice - 1])
    os.environ["MODEL_PATH"] = model_file

    print(f"\n→ Evaluating with model: {model_file}\n")
    eval_main()


def main():
```

```
    while True:
        print("\n=== Phishing Detector ===")
        print("  1) Train a new model")
        print("  2) Evaluate fraud-email dataset")
        print("  3) Exit")
        sel = input("Select option [1-3]: ").strip()
        if sel == "1":
            train_flow()
        elif sel == "2":
            eval_flow()
        elif sel == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice, try again.")


if __name__ == "__main__":
    main()


import os
import sys

import logging

logging.getLogger("tensorflow").setLevel(logging.ERROR)

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, confusion_matrix
from tensorflow.keras.callbacks import (
    ModelCheckpoint, EarlyStopping, TensorBoard, ReduceLROnPlateau
)
from concurrent.futures import ProcessPoolExecutor
from tqdm import tqdm

import matplotlib.pyplot as plt

sys.path.append(os.path.abspath("."))

from data.dataset_loader import (
    load_training_dataset,
    load_subhajournal_dataset
)
from preprocessing.nlp_preprocessing import preprocess_text
from preprocessing.embedding import vectorize_text
from models.hybrid_model import build_hybrid_model
from evaluation.metrics import evaluate_predictions, measure_processing_time

# ——— CONFIG ——————————
BATCH_SIZE = 64
MODEL_DIR = "keras_models"
LOG_DIR = "logs"

os.makedirs(MODEL_DIR, exist_ok=True)
os.makedirs(LOG_DIR, exist_ok=True)



def _vectorize_task(args):
```

```
    text, max_len = args
    tokens = preprocess_text(text)
    return vectorize_text(tokens, max_len=max_len)



def preprocess_and_vectorize(texts, max_len):
    """
    Parallel preprocessing + vectorization with a progress bar.
    Caps at 4 workers and falls back to sequential on OSError(1450).
    """
    args_iter = ((txt, max_len) for txt in texts)
    max_workers = min(4, os.cpu_count() or 1)
    try:
        with ProcessPoolExecutor(max_workers=max_workers) as exe:
            vecs = list(tqdm(
                exe.map(_vectorize_task, args_iter),
                total=len(texts),
                desc="Preprocess & Vectorize"
            ))
    except OSError as e:
        if e.winerror == 1450:
            # Windows out of resources → fall back to single-threaded
            print("\n[!] WinError 1450: falling back to single-threaded
vectorization")
            vecs = []
            for txt in tqdm(texts, desc="Preprocess & Vectorize (sequential)"):
                vecs.append(_vectorize_task((txt, max_len)))
        else:
            raise
    return np.stack(vecs)


def main():
    # 0) Read hyper-parameters / paths from env (injected by CLI)
    max_len = int(os.getenv("MAX_LEN", 200))
    embedding_dim = int(os.getenv("EMBEDDING_DIM", 100))
    epochs = int(os.getenv("EPOCHS", 30))
    best_model_path = os.getenv("BEST_MODEL_PATH", os.path.join(MODEL_DIR,
"best_default.keras"))
    final_model_path = os.getenv("FINAL_MODEL_PATH", os.path.join(MODEL_DIR,
"final_default.keras"))
    run_id = f"len{max_len}_emb{embedding_dim}_ep{epochs}"
    run_dir = os.path.join(LOG_DIR, run_id)
    os.makedirs(run_dir, exist_ok=True)

    # 1) Load & clean train data
    print("Loading training dataset...")
    train_df = load_training_dataset(data_dir=os.path.join("..", "data",
"emails-dataset"))
    train_df = train_df.dropna(subset=["label"])
    train_df["label"] = train_df["label"].astype(int)
    texts, labels = train_df["text"].tolist(), train_df["label"].to_numpy()

    # 2) Split
    X_train_txt, X_val_txt, y_train, y_val = train_test_split(
        texts, labels,
        test_size=0.2,
        random_state=42,
        stratify=labels
    )
```

```
# 3) Vectorize
print("Preprocessing and vectorizing training data...")
X_train = preprocess_and_vectorize(X_train_txt, max_len=max_len)
print("Preprocessing and vectorizing validation data...")
X_val = preprocess_and_vectorize(X_val_txt, max_len=max_len)

# 4) Build beefed-up model
print("Building model…")
model = build_hybrid_model(
    input_shape=(max_len, embedding_dim),
    cnn_filter_sizes=[3, 4, 5],
    cnn_num_filters=128,  # ↑ was 64
    rnn_type="bidirectional",  # switched to bidirectional LSTM
    rnn_units=128,  # ↑ was 64
    fusion_dense_units=128,  # ↑ was 64
    dropout_rate=0.6  # ↑ was 0.5
)
model.summary()

# 5) Callbacks
ckpt = ModelCheckpoint(
    best_model_path, monitor="val_accuracy",
    save_best_only=True, mode="max", verbose=1
)
es = EarlyStopping(
    monitor="val_accuracy", patience=7,
    restore_best_weights=True, verbose=1
)
# drop LR by 0.5 when val_accuracy stalls for 3 epochs
rlrop = ReduceLROnPlateau(
    monitor="val_accuracy", factor=0.5,
    patience=3, verbose=1, min_lr=1e-6
)
# tensorboard --logdir logs
tb = TensorBoard(log_dir=LOG_DIR)

# 6) Train
print("Starting training...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=epochs,
    batch_size=BATCH_SIZE,
    callbacks=[ckpt, es, rlrop, tb],
    verbose=1
)

# ——— PLOT: train vs val accuracy & loss
# accuracy
plt.figure()
plt.plot(history.history["accuracy"], label="train_acc")
plt.plot(history.history["val_accuracy"], label="val_acc")
plt.title("Accuracy per epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.savefig(os.path.join(run_dir, "accuracy_curve.png"))
plt.close()
# loss
plt.figure()
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
```

```
    plt.title("Loss per epoch")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.savefig(os.path.join(run_dir, "loss_curve.png"))
    plt.close()

    # 7) Load & vectorize test set
    print("Loading test dataset...")
    test_df = load_subhajournal_dataset(
        file_path=os.path.join("..", "data", "subhajournal",
"Phishing_Email.csv")
    )
    test_df = test_df.dropna(subset=["label"])
    test_df["label"] = test_df["label"].astype(int)
    X_test_txt = test_df["text"].tolist()
    y_test = test_df["label"].to_numpy()
    X_test = preprocess_and_vectorize(X_test_txt, max_len=max_len)

    # 8) Evaluate
    print("Evaluating on test set…")
    loss, acc = model.evaluate(X_test, y_test, verbose=1)
    print(f"Test accuracy: {acc:.4f}, Loss: {loss:.4f}")

    # 8a) ROC curve & confusion matrix
    # get probabilities and preds
    y_pred_prob = model.predict(X_test, batch_size=BATCH_SIZE).ravel()
    y_pred = (y_pred_prob > 0.5).astype(int)
    # ROC
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr)
    plt.title(f"ROC Curve (AUC = {roc_auc:.3f})")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.savefig(os.path.join(run_dir, "roc_curve.png"))
    plt.close()
    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure()
    plt.imshow(cm, interpolation="nearest")
    plt.title("Confusion Matrix")
    plt.colorbar()
    plt.xlabel("Predicted")
    plt.ylabel("True")
    for i in range(2):
        for j in range(2):
            plt.text(j, i, cm[i, j], ha="center", va="center")
    plt.savefig(os.path.join(run_dir, "confusion_matrix.png"))
    plt.close()

    tn, fp, fn, tp = cm.ravel()
    tpr = tp / (tp + fn) if (tp + fn) else 0
    fpr = fp / (fp + tn) if (fp + tn) else 0
    print(f"TPR (Recall for phishing): {tpr:.4f}, FPR: {fpr:.4f}")

    # 9) Detailed metrics
    y_pred = model.predict(X_test, batch_size=BATCH_SIZE)
    metrics = evaluate_predictions(y_test, y_pred)
    print("Detailed metrics:")
    for k, v in metrics.items():
```

```
            print(f"  {k}: {v:.4f}")

    # 10) Timing
    avg_time = measure_processing_time(model, X_test)
    print(f"Avg inference time per sample: {avg_time:.6f}s")

    # 11) Save final
    model.save(final_model_path)
    print(f"Model saved to {final_model_path}")


if __name__ == "__main__":
    main()
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

КОМБІНОВАНИЙ МЕТОД ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ ФІШИНГОВОГО ВМІСТУ В ТЕКСТОВІЙ ЧАСТИНІ ЕЛЕКТРОННИХ ЛИСТІВ

Доповідач: Фещенко Єгор Олександрович

Науковий керівник: к.т.н., доц., доцент кафедри ПЗКС Заболотня Т. М.

Київ – 2025

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- Зростання кількості та складності фішингових атак на електронну пошту.
- Недостатня ефективність традиційних методів виявлення фішингових листів.
- Підвищення точності виявлення фішингу за допомогою систем на основі глибокого навчання та NLP.

ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об’єктом дослідження є процес автоматизованої класифікації текстових даних.

Предметом дослідження є методи, способи та програмні засоби автоматизованого виявлення фішингового вмісту в текстовій частині електронних листів.

ПОСТАНОВКА ЗАДАЧІ

Метою дослідження: підвищення ефективності виявлення фішингу в електронних листах за рахунок розробки та реалізації комбінованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів

Окремі завдання

1. Аналіз існуючих методів та засобів виявлення фішингу в електронних листах.
2. Розробка комбінованого методу автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів.
3. Обґрунтування вибору засобів для програмної реалізації.
4. Розробка програмного забезпечення, що реалізує метод.
5. Аналіз отриманих результатів.

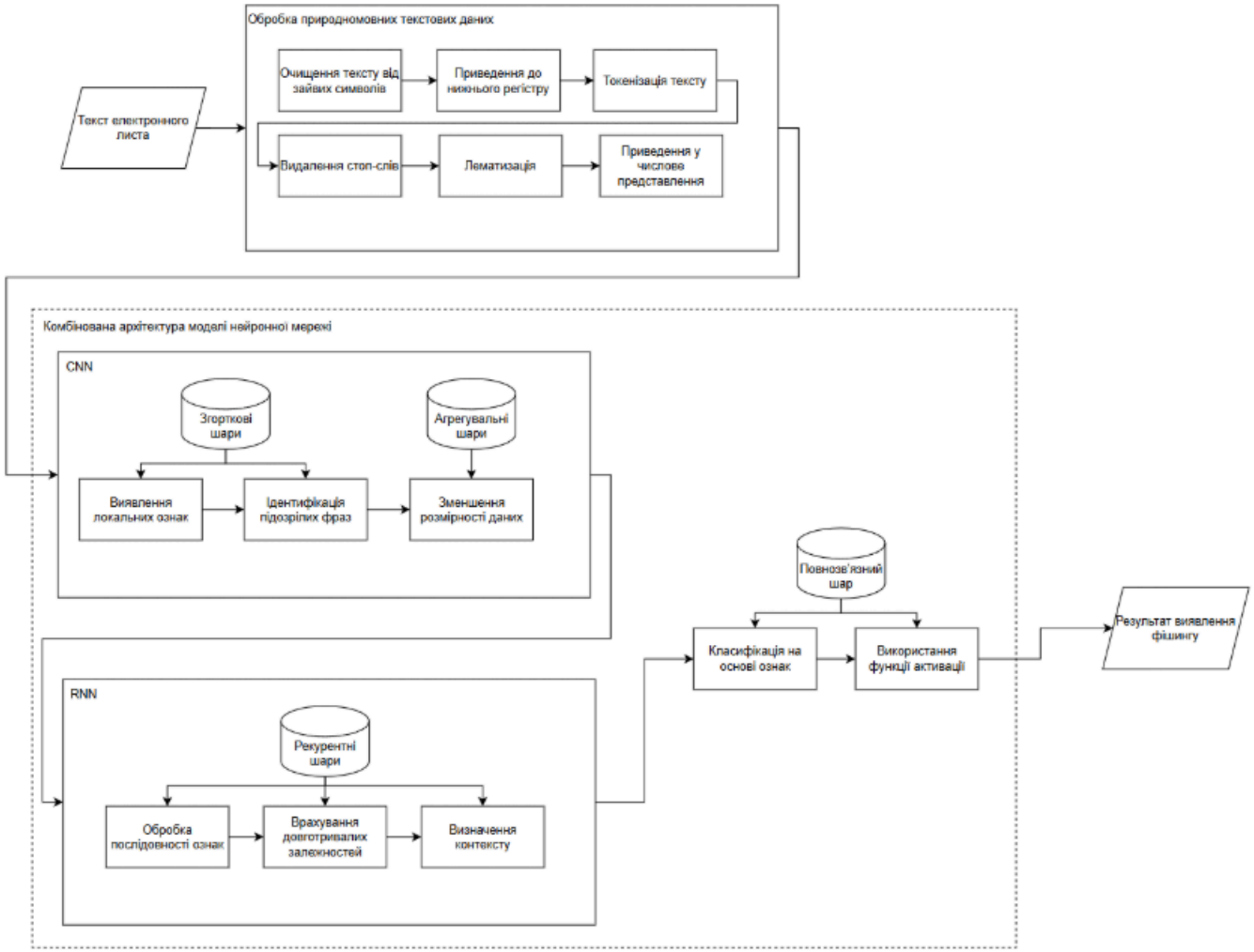
ІСНУЮЧІ МЕТОДИ

- **Методи на основі правил**
  - Переваги: простота, швидкість роботи
  - Недоліки: низька точність, неадаптивність
- **Класичні методи машинного навчання**
  - Переваги: достатньо швидкі, прості у застосуванні
  - Недоліки: погано враховують семантику тексту, чутливі до якості ознак
- **Методи глибокого навчання**
  - Переваги: добра точність, враховують семантику та контекст
  - Недоліки: потребують ретельного налаштування, значні ресурси
- **Трансформерні моделі**
  - Переваги: висока точність, ефективне моделювання тексту
  - Недоліки: дуже високі вимоги до ресурсів

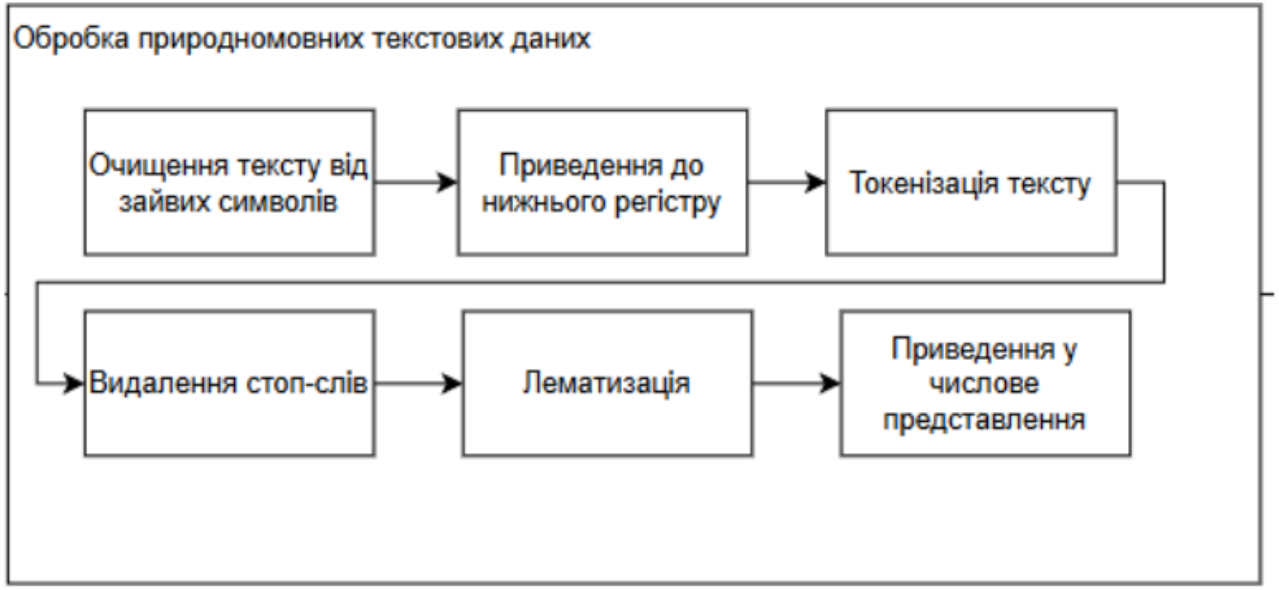
ЗАПРОПОНОВАНИЙ КОМБІНОВАНИЙ МЕТОД

1. Попередня обробка текстових даних
2. Векторизація тексту
3. Вилучення локальних ознак
4. Врахування контексту тексту
5. Динамічний вибір важливих ознак
6. Класифікація тексту

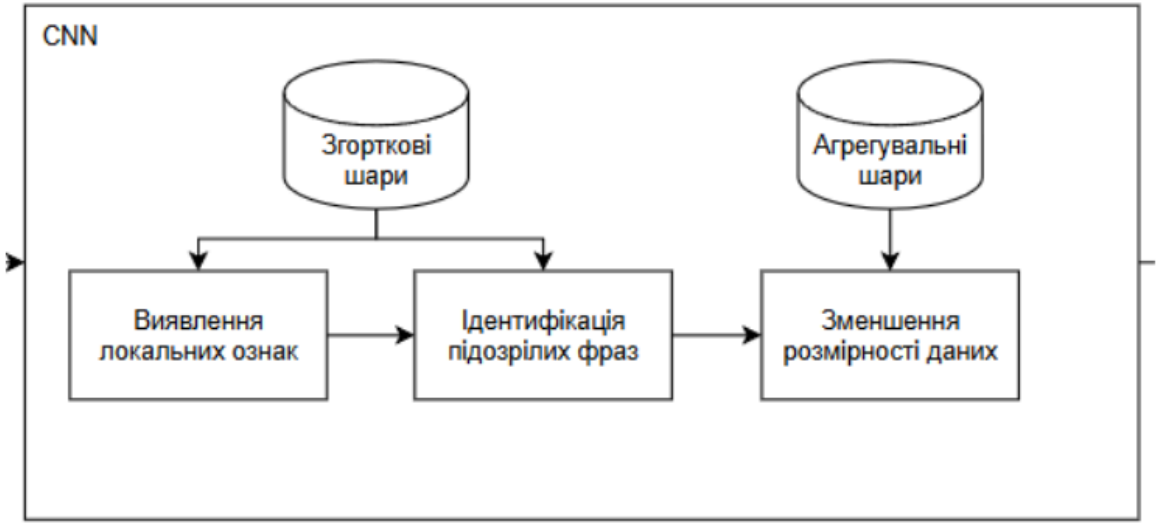
АЛГОРИТМ РОБОТИ ЗАПРОПОНОВАНОГО МЕТОДУ



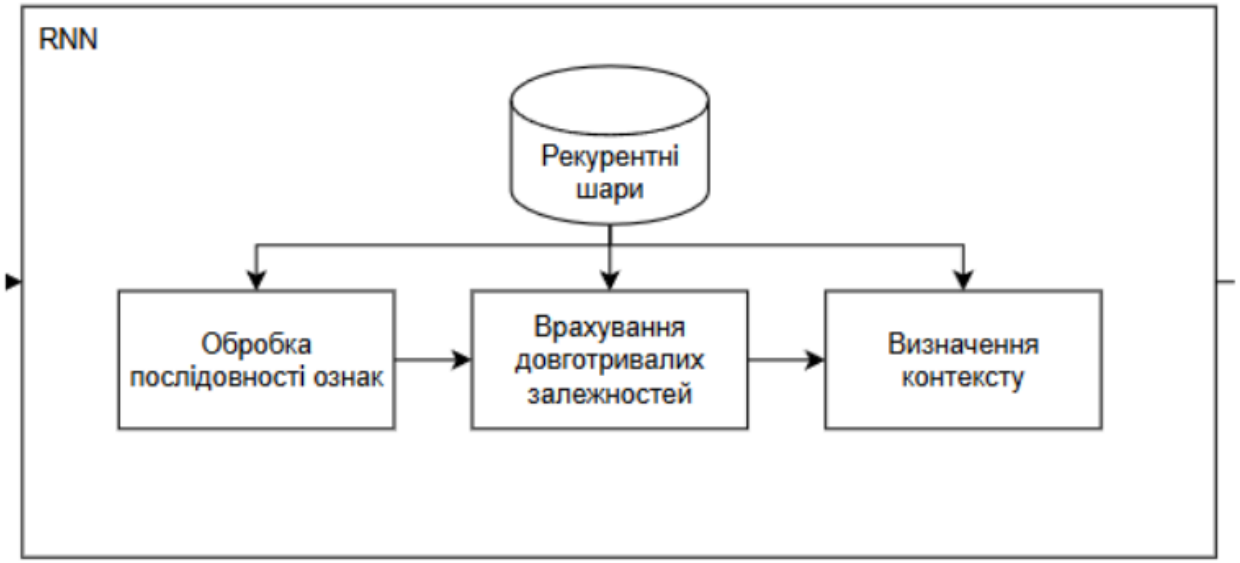
ПОПЕРЕДНЯ ОБРОБКА ТЕКСТУ В ЗАПРОПОНОВАНОМУ МЕТОДІ



ВИКОРИСТАННЯ CNN У ЗАПРОПОНОВАНОМУ МЕТОДІ



ВИКОРИСТАННЯ RNN У ЗАПРОПОНОВАНОМУ МЕТОДІ



ГІПЕРПАРАМЕТРИ МОДЕЛІ

- Максимальна довжина текстів (MAX_LEN): 100 або 200 слів
- Розмірність ембедінгів (EMBEDDING_DIM): 50, 100, 200
- Кількість епох навчання (EPOCHS): 20–30
- Розмір батчу (BATCH_SIZE): 64
- Функція активації вихідного шару: sigmoid
- Функція втрат: binary_crossentropy

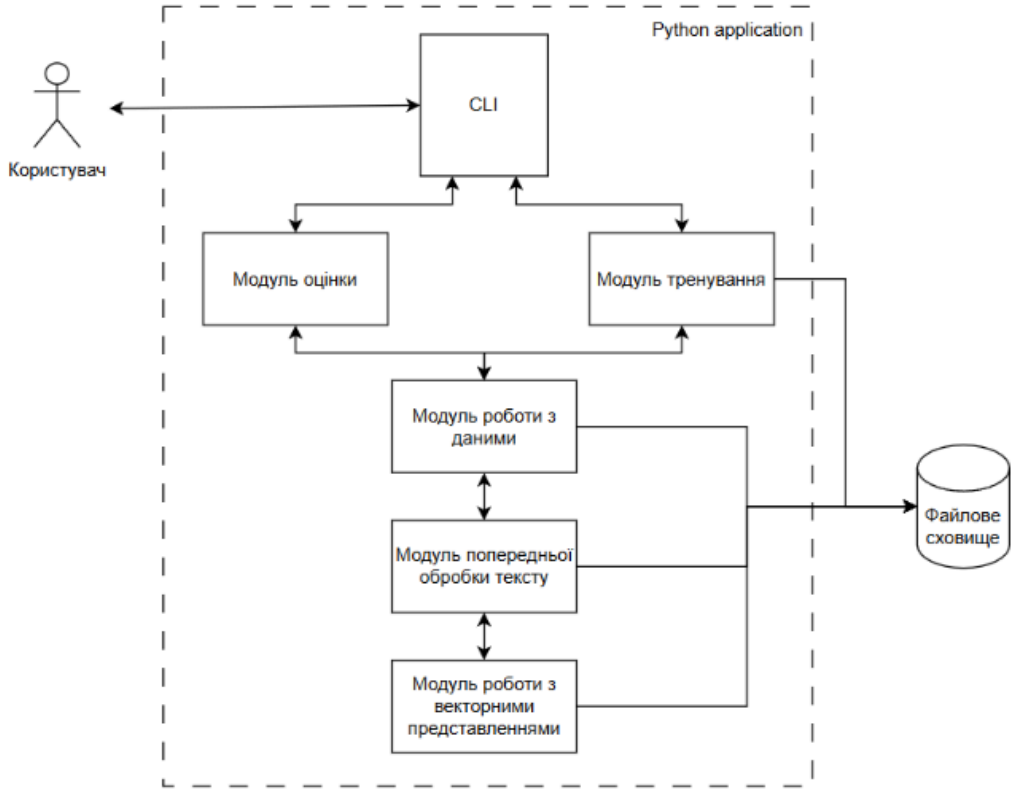
ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

spaCy

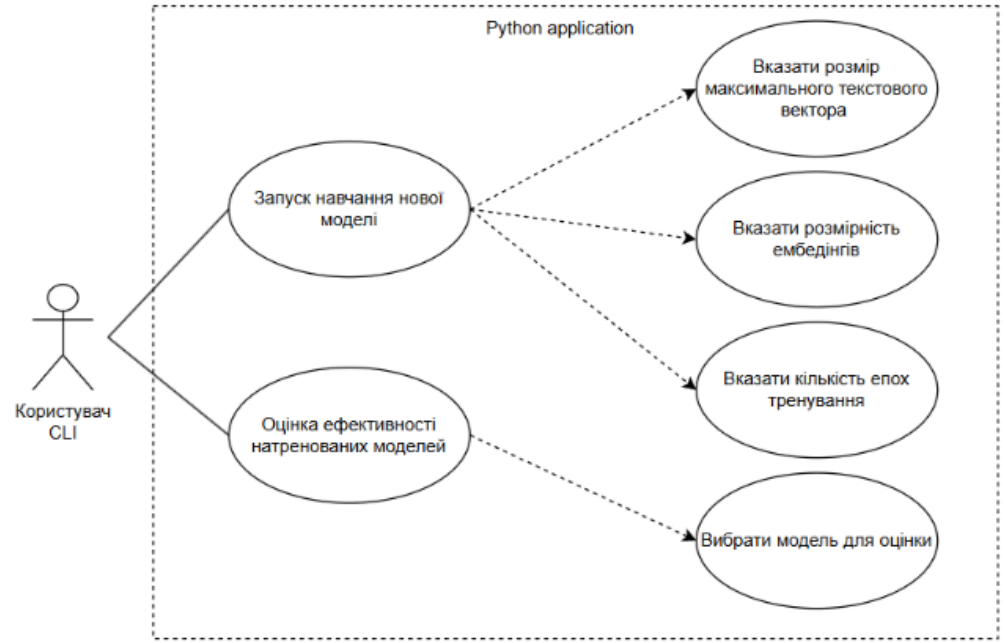
Keras

NLTK

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



СЦЕНАРІЇ ВИКОРИСТАННЯ



МЕТРИКИ ОЦІНКИ ЕФЕКТИВНОСТІ

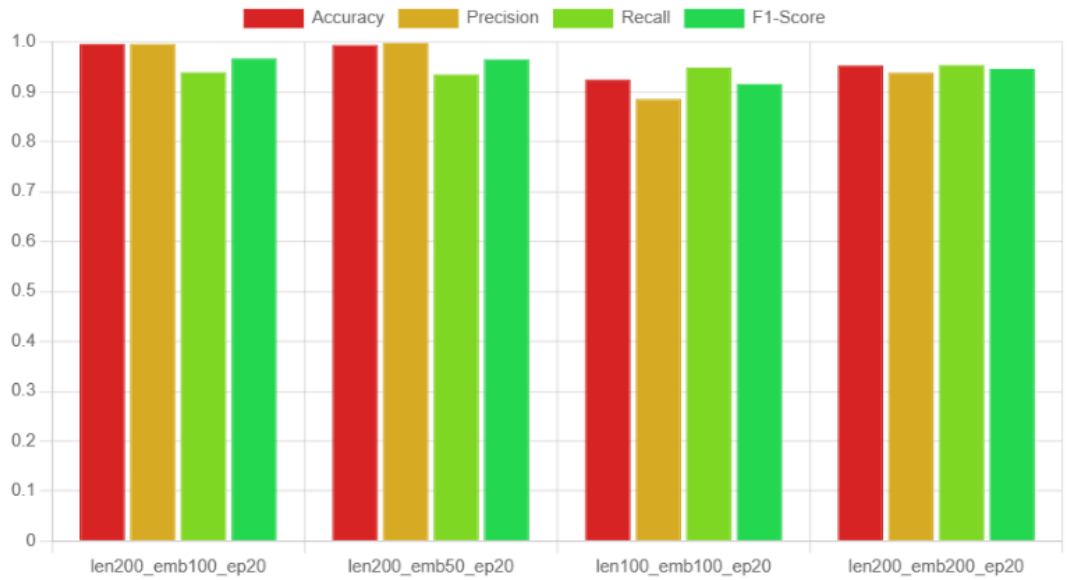
$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN},$$

$$Recall = \frac{TP}{TP+FN}.$$

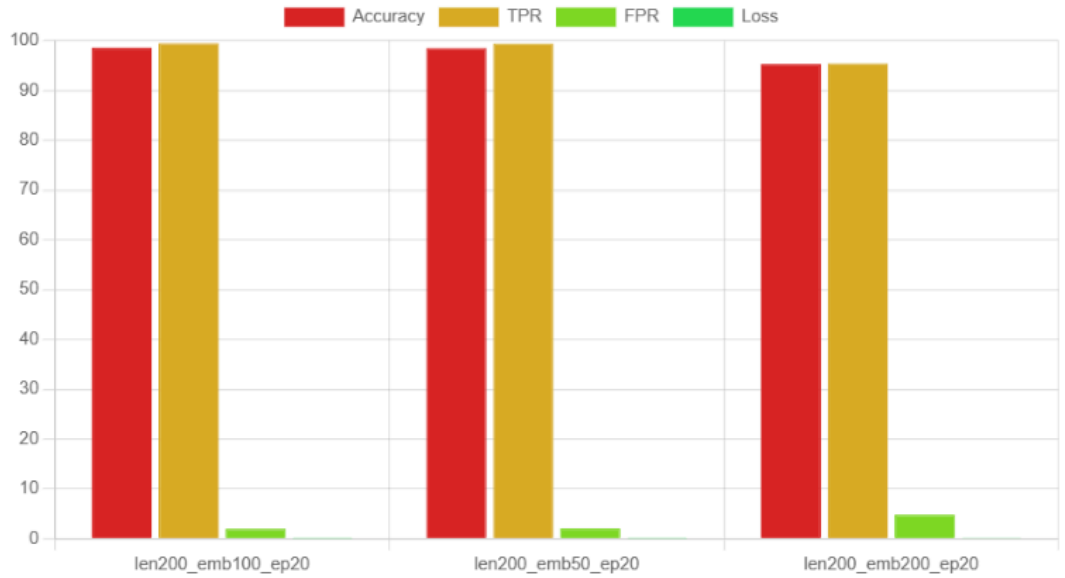
$$Precision = \frac{TP}{TP+FP}.$$

$$F1 - score = 2 \bullet \frac{Precision \bullet Recall}{Precision + Recall}.$$

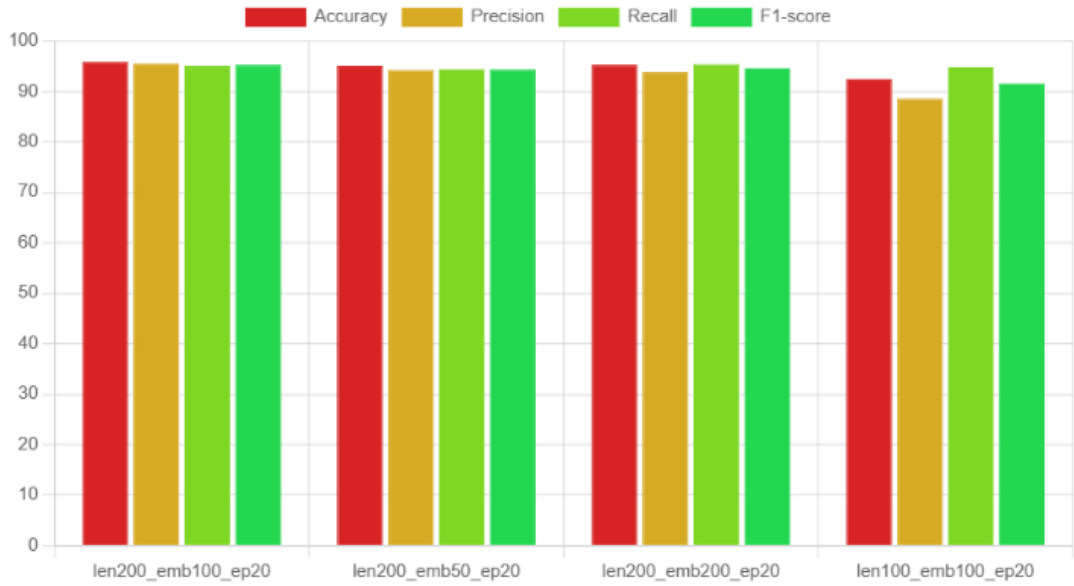
РЕЗУЛЬТАТИ НА ТРЕНУВАЛЬНОМУ ДАТАСЕТІ



РЕЗУЛЬТАТИ НА ВАЛІДАЦІЙНОМУ ДАТАСЕТІ



РЕЗУЛЬТАТИ НА ТЕСТОВОМУ ДАТАСЕТІ



ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ

Метод	Точність на тренувальному датасеті	Точність на валідаційному датасеті	Точність на тестувальному датасеті
Запропонований метод	99,60%	98,59%	95,93%
SVM	98,76%	97,68%	94,24%
Логістична регресія	98,45%	96,25%	93,67%
DeepAntiPhishNet	87,60%	86,40%	—
BERT	99,11%	98,88%	99,31%

НАУКОВА НОВИЗНА

Уперше розроблено комбінований метод автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів, характерними особливостями якого є попереднє оброблення природномовних текстових даних, застосування згорткової нейронної мережі для вилучення локальних текстових ознак, двонаправленої рекурентної нейронної мережі для врахування семантичного контексту та механізму уваги для динамічного вибору найбільш значущих ознак тексту, що дозволило збільшити абсолютну точність виявлення фішингу у межах від 95.93% до 99.6% на різних наборах даних.

ПРАКТИЧНА ЗНАЧУЩІСТЬ

На основі запропонованого комбінованого методу виконано програмну реалізацію у середовищі Python за допомогою платформи Python NLTK та Tensorflow/Keras.

Запропонований метод досягає високої точності за рахунок комбінованої нейромережевої моделі з помірними вимогами до обчислювальних ресурсів.

Розроблене програмне забезпечення може бути інтегроване в корпоративних та державних організаціях з великим обсягом електронної пошти для протидії фішингу.

АПРОБАЦІЯ РОБОТИ

Основні положення та результати виконаної роботи були представлені та обговорені на XVII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2024 (20-22 листопада 2024 р., м. Київ).

Стаття «Вплив гібридної архітектури нейромереж на ефективність ідентифікації фішингу в електронних листах» в фаховому журналі «Наукові праці ВНТУ №2 (2025)» (подано в редакцію).

22/25

НАПРЯМКИ ПОДАЛЬШОГО ДОСЛІДЖЕННЯ

1. Розширення та збалансування наборів даних для підвищення узагальнюючої здатності.
2. Дослідження використання альтернативних типів ембедінгів.
3. Вивчення можливостей інтеграції методу в комплексні системи інформаційної безпеки.

ВИСНОВКИ

1. Виконано аналіз сучасних методів і засобів автоматизованого виявлення фішингу в електронних листах, визначено їх переваги та недоліки.
2. Розроблено ефективний комбінований метод на основі комбінованої нейромережевої архітектури для ідентифікації фішингового вмісту у текстах електронних листів.
3. Обґрунтовано вибір технологій і програмних засобів для реалізації запропонованого методу.
4. Створено програмне забезпечення, що реалізує запропонований метод.
5. Запропонований метод автоматизованої ідентифікації фішингового вмісту в текстовій частині електронних листів дозволяє збільшити абсолютну точність виявлення фішингу у межах від 95.93% до 99.6% на різних наборах даних.
6. Визначено напрямки подальшого дослідження.

Дякую за увагу!