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

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВИКОРИСТАННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ АНАЛІЗУ ТЕКСТУ НА ПРЕДМЕТ ІНФОРМАЦІЙНО-ПСИХОЛОГІЧНИХ ОПЕРАЦІЙ В ІНТЕРНЕТІ»

Виконала: здобувачка 4 курсу

Рівень бакалавр

Галузь знань 12 «Інформаційні технології»,

спеціальність 125 «Кібербезпека»

Долінко Ксенія Валентинівна

Керівник: д.т.н., професор

Соколов Артем Вікторович

Рецензент: доктор філософії

Слатвінська Валерія Миколаївна

Одеса – 2025

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

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

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачці Долінко Ксенії Валентинівні

1. Тема роботи: «Використання рекурентних нейронних мереж для аналізу тексту на предмет інформаційно-психологічних операцій в інтернеті»
Керівник роботи: д.т.н, професор Соколов Артем Вікторович
затверджені наказом НУ ОЮА від «18» березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Отримання та узгодження завдання на кваліфікаційну роботу	Вересень-жовтень 2024	
2.	Збір та аналіз інформації, огляд літературних джерел	Листопад 2024	
3.	Підготовка теоретичної частини роботи (Розділ 1)	Грудень 2024	
4.	Проектування та реалізація рекурентної нейронної мережі (Розділ 2)	Січень-лютий 2025	
5.	Тестування розробленої моделі (Розділ 3)	Лютий 2025	
6.	Оформлення кваліфікаційної роботи та додатків	Березень 2025	
7.	Подання роботи на перевірку керівнику та рецензенту	Травень 2025	
8.	Підготовка до захисту (презентація, доповідь)	Травень-06.2025	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Використання рекурентних нейронних мереж для аналізу тексту на предмет інформаційно-психологічних операцій в інтернеті» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека, освітньою програмою: Кібербезпека, містить 10 рисунків, 2 таблиці, 2 додатки, 22 літературних джерел за переліком посилань. Робота виконана на 79 сторінках загального тексту та 48 сторінках основного тексту.

Кваліфікаційна робота присвчена автоматичному виявленню інформаційно-психологічних операцій (ІПСО) в інтернет-текстах. Основною метою роботи є розробка та тестування рекурентної нейронної мережі (РНМ) на базі LSTM для бінарної класифікації текстів.

У рамках роботи проаналізовано особливості ІПСО та існуючі ШІ-підходи до їх виявлення, а також вивчено основи обробки природної мови (НЛП) та архітектури РНМ. Розроблено методику передобробки даних та реалізовано LSTM-модель, яка інтегрує текстові вектори та показник надійності джерела. Створено механізм для динамічного оновлення цього показника в базі даних SQLite.

Модель навчено на підготовленому наборі даних, а її ефективність оцінено за стандартними метриками. Розроблено програмний інтерфейс для взаємодії моделі із зовнішніми системами, зокрема, для інтеграції з Telegram-ботом, розробленим у рамках суміжного проекту. Практична цінність роботи полягає у створенні функціонального прототипу нейромережевого класифікатора ІПСО, який враховує репутацію джерела, слугуючи основою для систем моніторингу інформаційного простору.

ІНФОРМАЦІЙНО-ПСИХОЛОГІЧНІ ОПЕРАЦІЇ (ІПСО), НЕЙРОННІ МЕРЕЖІ, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ (РНМ), LSTM, АНАЛІЗ ТЕКСТУ, ОБРОБКА ПРИРОДНОЇ МОВИ (НЛП), БІНАРНА КЛАСИФІКАЦІЯ ТЕКСТУ, ГЛИБОКЕ НАВЧАННЯ, ЗВАЖУВАННЯ КЛАСІВ, KERAS, TENSORFLOW.

ABSTRACT

Qualification Paper on the topic "Using Recurrent Neural Networks for Text Analysis for Information-Psychological Operations on the Internet" for obtaining the first (bachelor's) level of higher education in specialty 125 Cybersecurity, educational program: Cybersecurity, includes 10 figures, 2 tables, 2 appendices, and 22 literary sources in the list of references. The paper comprises 79 pages of general text and 48 pages of main text.

The qualification paper is dedicated to the automatic detection of information-psychological operations (IPSO) in internet texts. The main goal of the work is the development and testing of a recurrent neural network (RNN) based on LSTM for binary text classification.

The work analyzed the peculiarities of IPSO and existing AI approaches to their detection, as well as studied the fundamentals of natural language processing (NLP) and RNN architectures. A data preprocessing methodology was developed, and an LSTM model was implemented that integrates text vectors and a source reliability indicator. A mechanism for dynamically updating this indicator in an SQLite database was created.

The model was trained on a prepared dataset, and its effectiveness was evaluated using standard metrics. A software interface was developed for the model's interaction with external systems, specifically for integration with a Telegram bot developed as part of a related project. The practical value of the work lies in creating a functional prototype of an IPSO neural network classifier that considers source reputation, serving as a basis for information space monitoring systems.

KEYWORDS: INFORMATION-PSYCHOLOGICAL OPERATIONS (IPSO), NEURAL NETWORKS, RECURRENT NEURAL NETWORKS (RNN), LSTM, TEXT ANALYSIS, NATURAL LANGUAGE PROCESSING (NLP), BINARY TEXT CLASSIFICATION, DEEP LEARNING, CLASS WEIGHTING, KERAS, TENSORFLOW.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 7
ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 10
1.1 Інформаційно-психологічні операції: сутність, класифікація 10
1.2 Огляд підходів до автоматичного виявлення дезінформації та ІПСО 11
1.3 Теоретичні основи обробки природної мови (НЛП) 14
1.4 Рекурентні нейронні мережі (РНМ) для аналізу послідовностей 17
2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ АНАЛІЗУ ІПСО 21
2.1 Аналіз та підготовка даних для навчання моделі 21
2.2 Розробка моделі рекурентної нейронної мережі 25
2.3 Реалізація механізму управління надійністю джерела 27
2.4 Процес навчання та налаштування моделі 30
2.5 Розробка інтерфейсу для інтеграції моделі 32
3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ МОДЕЛІ 36
3.1 Середовище проведення експериментів 36
3.2 Метрики оцінки якості класифікації 37
3.3 Аналіз процесу навчання моделі 39
3.4 Оцінка якості бінарної класифікації на тестовій вибірці 42
3.5 Розробка інструменту для інтерактивного тестування моделі (test_interactive.py) 45
3.6 Аналіз впливу показника надійності джерела 49
3.7 Аналіз помилок моделі: приклади хибних спрацювань 54
ВИСНОВКИ 58

ПЕРЕЛІК ПОСИЛАНЬ 60

Додаток А. Лістинг програмного коду модуля аналізу ІПСО (ipso_model.py) 63

Додаток Б. Лістинг програмного коду скрипта для інтерактивного тестування моделі (test_interactive.py) 76

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІПСО – інформаційно-психологічна операція

ШІ – штучний інтелект

NLP – Natural Language Processing

ШІ – штучний інтелект

RNM – Recurrent Neural Network

LSTM – Long Short-Term Memory

GRU – Gated Recurrent Unit

API – Application Programming Interface

TP – True Positive

TN – True Negative

FP – False Positive

FN – False Negative

ВСТУП

Сучасний інформаційний простір, особливо соціальні мережі та месенджери, активно використовується для проведення інформаційно-психологічних операцій (ІПСО) та поширення дезінформації з метою маніпулювання суспільною думкою, дестабілізації соціально-політичної ситуації та завдання шкоди окремим особам чи групам. Масштаби та швидкість розповсюдження такого контенту роблять ручний моніторинг та аналіз практично неможливими, що зумовлює гостру потребу в розробці ефективних автоматизованих засобів протидії. Існуючі підходи до автоматичного виявлення ІПСО та дезінформації часто базуються на традиційних методах машинного навчання, які мають обмеження в розумінні контексту та різних тактик маніпуляції. Більш сучасні методи на основі глибокого навчання, зокрема рекурентні нейронні мережі (РНМ), демонструють кращі результати в аналізі семантики тексту.

Однак, багато існуючих нейромережевих рішень фокусуються виключно на аналізі змісту повідомлення, ігноруючи важливий аспект – репутацію джерела. Ефективність ІПСО часто залежить від довіри до каналу поширення, а недобросовісні джерела схильні до систематичного розповсюдження маніпулятивного контенту. Тому підхід, що інтегрує аналіз текстового контенту за допомогою потужних моделей, таких як РНМ з довгою короткочасною пам'яттю (LSTM), здатною вловлювати довгострокові залежності в тексті, з динамічною оцінкою надійності джерела, є перспективним напрямком. Обґрунтуванням необхідності проведення даного дослідження є саме відсутність комплексних рішень, які б поєднували глибокий семантичний аналіз тексту та динамічне врахування репутації джерела, що потенційно може підвищити точність виявлення ІПСО. Розробка та дослідження такої системи становить актуальну науково-технічну задачу, вирішенню якої присвячено дану роботу.

Метою роботи дослідження ефективності використання рекурентної нейронної мережі для виявлення ІПСО у текстових повідомленнях, за умови інтеграції механізму динамічного оновлення надійності джерела інформації.

Об'єктом дослідження виступає процес аналізу текстових даних в інтернет-середовищі на наявність ознак інформаційно-психологічних операцій. Предметом дослідження є рекурентні нейронні мережі архітектури LSTM, методи обробки природної мови та алгоритми врахування динамічної надійності джерела для класифікації тексту за категоріями ІПСО.

Практичне значення отриманих результатів полягає у створенні програмного прототипу нейромережевої моделі, здатної автоматично аналізувати текстові повідомлення на предмет належності до ІПСО, враховуючи репутацію джерела. Розроблена модель та код можуть бути використані як інтелектуальне ядро для систем моніторингу інформаційного простору та виявлення дезінформаційних кампаній. Зокрема, розроблений програмний інтерфейс моделі дозволяє її інтеграцію у прикладні рішення. В рамках суміжного дипломного проекту, модель була інтегрована у Telegram-бот ("Розробка бота для аналізу безпеки текстових та графічних даних на основі нейронних мереж"), що демонструє безпосередню практичну застосовність отриманих результатів для підвищення інформаційної гігієни та протидії маніпулятивним впливам в інтернеті. Результати дослідження також можуть бути корисними для подальших наукових розробок у сфері НЛП та виявлення шкідливого контенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Інформаційно-психологічні операції: сутність, класифікація

Інформаційно-психологічні операції (ІПСО) є невід'ємною складовою сучасних конфліктів. Під ІПСО розуміють комплекс скоординованих заходів, спрямованих на вплив на психологічний стан, емоції, мотивацію, об'єктивне мислення та, як наслідок, поведінку цільової аудиторії (урядів, організацій, груп населення, окремих осіб) шляхом поширення спеціально підготовленої інформації та дезінформації. Ключовими характеристиками ІПСО є їх плановий характер, наявність конкретних цілей та визначеної цільової аудиторії, використання маніпулятивних технік для прихованого чи явного впливу.

Для досягнення своїх цілей суб'єкти ІПСО використовують різноманітні методи впливу. З точки зору автоматичного аналізу, важливими є ті методи, що залишають лінгвістичні або семантичні маркери, які може розпізнати модель обробки природної мови. До таких методів належать:

– Емоційне маніпулювання - використання емоційно забарвленої лексики (як позитивної, так і негативної), апеляція до страху, гніву, патріотизму, використання мови ворожнечі;

– Маніпуляція фактами - подання напівправди, виривання цитат з контексту, зміщення акцентів;

– Використання логічних хитрощів - застосування софізмів, хибних дихотомій ("хто не з нами, той проти нас"), переходів на особистості;

– Створення та просування наративів - формування стійких сюжетних ліній, які вигідно інтерпретують події;

– Заклики до дії - прямі чи приховані спонукання до певних дій, від поширення повідомлення до участі в акціях чи насильстві;

– Дискредитація опонентів та джерел - підрив довіри до альтернативних точок зору або джерел інформації.

Розуміння сутності та методів ІПСО є критично важливим для розробки ефективних засобів їх автоматичного виявлення. Постійна еволюція методів

впливу вимагає застосування потужних інструментів аналізу, здатних вловлювати складні паттерни в текстових даних. Подальші розділи роботи будуть присвячені саме аналізу та реалізації такого інструменту із використанням нейронних мереж.

1.2 Огляд підходів до автоматичного виявлення дезінформації та ІПСО

Задача автоматичного виявлення дезінформації та ІПСО є складною і багатокомпонентною, що зумовило появу різноманітних підходів, які можна умовно розділити на традиційні методи машинного навчання та сучасні методи на основі глибокого навчання.

На ранніх етапах досліджень для виявлення шкідливого контенту активно використовувалися класичні алгоритми машинного навчання, такі як:

1) Метод опорних векторів (Support Vector Machines, SVM)

Ефективний для задач класифікації з високою розмірністю простору ознак, що характерно для текстових даних.

2) Наївний баєсівський класифікатор (Naive Bayes)

Простий імовірнісний класифікатор, що добре працює на текстових даних, особливо при використанні моделі "мішка слів" (Bag-of-Words).

3) Логістична регресія (Logistic Regression)

Статистичний метод, що моделює ймовірність належності об'єкта до певного класу.

4) Дерева рішень (Decision Trees) та випадкові ліси (Random Forests)

Ансамблеві методи, здатні моделювати нелінійні залежності.

Ці підходи зазвичай вимагають ретельного етапу інженерії ознак (feature engineering), де дослідник вручну визначає та витягує з тексту характеристики, які, на його думку, є релевантними для виявлення ІПСО. До таких ознак можуть належати лексичні ознаки (частота вживання певних слів (Bag-of-Words), n-грами (послідовності слів або символів), наявність слів з емоційно забарвлених словників), стилістичні ознаки (довжина речень, середня довжина слова, використання великих літер, розділових знаків, показники читабельності тексту),

синтаксичні ознаки (частиномовні теги (POS-tags), структура речень), семантичні ознаки (використання тематичних моделей (наприклад, LDA) для визначення основних тем документа), метадані (інформація про автора, час публікації, кількість репостів/лайків), тощо [3].

Недоліками традиційних підходів є:

– Висока трудомісткість та суб'єктивність процесу вибору ознак.

– Обмежена здатність вловлювати складні семантичні зв'язки, контекст та приховані маніпуляції.

– Чутливість до незначних змін у формулюваннях, які можуть бути використані для обходу детекторів.

– Низька ефективність при аналізі коротких повідомлень (характерних для месенджерів та соцмереж) та текстів з неформальною лексикою, помилками, емодзі.

Поява та розвиток глибокого навчання кардинально змінили підходи до обробки природної мови [11] та виявлення дезінформації. Моделі глибокого навчання здатні автоматично вивчати ієрархічні представлення даних (ознаки) безпосередньо з сирого тексту, що дозволяє їм вловлювати значно складніші патерни та залежності. До основних архітектур, що застосовуються в цій галузі, належать:

1) Конволюційні нейронні мережі (Convolutional Neural Networks, CNN)

Хоча CNN первинно розроблялися для обробки зображень, їх адаптовані версії успішно застосовуються для класифікації текстів. Вони ефективно виявляють локальні патерни (подібно до n-грам) у тексті за допомогою згорткових фільтрів.

2) Рекурентні нейронні мережі (Recurrent Neural Networks, RNN)

Спеціалізовані для обробки послідовних даних, якими є текст. РНМ мають внутрішню пам'ять, що дозволяє враховувати попередній контекст при обробці поточного слова. Однак базові РНМ страждають від проблеми зникаючих/вибухаючих градієнтів при навчанні на довгих послідовностях.

3) LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit)

Це вдосконалені типи РНМ зі спеціальними вентильними механізмами (gates), які дозволяють ефективно контролювати потік інформації та навчатися довгостроковим залежностям у тексті. Саме ці архітектури показали високу ефективність у багатьох задачах НЛП, включаючи класифікацію тексту, і є предметом дослідження даної роботи [4].

4) Трансформери (Transformers)

Сучасна архітектура, що базується на механізмах уваги (attention mechanisms), які дозволяють моделі зважувати важливість різних слів у послідовності незалежно від їхньої відстані. Моделі на основі Трансформерів (наприклад, BERT, GPT, RoBERTa) демонструють найвищу якість у багатьох задачах НЛП, але вимагають значних обчислювальних ресурсів та великих обсягів даних для навчання [12].

Перевагами методів глибокого навчання є їхня здатність до автоматичного вилучення ознак, врахування контексту та семантики, а також вища точність на складних завданнях порівняно з традиційними підходами. Однак вони потребують більших обсягів розмічених даних та значних обчислювальних ресурсів для навчання [2].

Аналіз контенту в соціальних мережах та месенджерах (як Telegram) має свої особливості, що впливають на вибір та ефективність методів виявлення ІПСО:

– Короткі тексти: Повідомлення часто є короткими, що ускладнює виявлення контексту та застосування методів, що потребують довгих текстів.

– Неформальна мова: Широке використання сленгу, скорочень, емодзі, навмисних помилок ускладнює роботу традиційних НЛП-інструментів.

– Швидкість поширення: Інформація (включаючи ІПСО) розповсюджується дуже швидко, вимагаючи оперативних методів детектування.

– Мультимодальність: Часто текст супроводжується зображеннями, відео, посиланнями, що створює додатковий контекст (хоча аналіз мультимодальності виходить за рамки даної роботи).

– Динамічність джерел: З'являються та зникають нові канали, змінюється їхня репутація, що актуалізує необхідність врахування надійності джерела як динамічного параметра.

Таким чином, для ефективного виявлення ІПСО в середовищі месенджерів доцільним є використання моделей глибокого навчання, зокрема РНМ (LSTM/GRU), які добре працюють з послідовностями та здатні вловлювати семантичні нюанси, з додатковим врахуванням контекстуальної інформації, такої як динамічна надійність джерела, що і стало основою для підходу, розробленого в даній дипломній роботі.

1.3 Теоретичні основи обробки природної мови (НЛП)

Обробка природної мови (Natural Language Processing, NLP або НЛП) – це міждисциплінарна галузь на перетині інформатики, штучного інтелекту та лінгвістики, що займається розробкою алгоритмів і систем, здатних аналізувати, розуміти та генерувати людську мову в текстовій або усній формі. Для того, щоб нейронна мережа могла ефективно аналізувати текстові повідомлення на предмет ІПСО, необхідно застосувати ряд фундаментальних технік НЛП для перетворення сирого тексту у формат, придатний для машинного навчання. Основні етапи та концепції включають передобробку тексту та його векторизацію.

Сирі текстові дані, отримані з інтернет-джерел, зазвичай містять багато "шуму" та варіативності, які можуть заважати навчанню моделі. Передобробка спрямована на стандартизацію та очищення тексту. Ключові етапи, що часто застосовуються (повністю або частково, залежно від задачі та даних):

1) Очищення тексту (Text Cleaning): Видалення елементів, що не несуть суттєвого семантичного навантаження для даної задачі, таких як HTML-теги, URL-адреси, символи @ та # (якщо згадки та хештеги не є важливими ознаками), спеціальні символи, іноді – пунктуація. В даній роботі для цього використовувались регулярні вирази (бібліотека re в Python).

2) Токенізація (Tokenization): Процес розбиття тексту на окремі смислові одиниці – токени. Найчастіше токенами є слова, але можуть бути й речення, символи або суб-слова (subwords), залежно від обраного підходу. В даній роботі використовувалась токенізація на рівні слів за допомогою інструмента Tokenizer з бібліотеки Keras [10].

3) Нормалізація (Normalization): Приведення токенів до єдиної форми для зменшення розміру словника та уникнення дублювання семантично близьких слів. Основні методи:

4) Приведення до нижнього регістру (Lowercasing): Перетворення всіх символів на малі літери (наприклад, "Операція", "операція", "ОПЕРАЦІЯ" стають "операція").

5) Стемінг (Stemming): Спрощений процес відсікання закінчень та суфіксів для отримання основи слова (стему). Працює швидко, але основа не завжди є реальним словом (наприклад, "операці" для "операція", "операційний").

6) Лематизація (Lemmatization): Більш складний процес приведення слова до його словникової форми (леми) з урахуванням морфологічного аналізу та частини мови. Дає лінгвістично коректніші результати (наприклад, "операція" для "операцією", "операціями"), але потребує більше ресурсів та лінгвістичних словників.

7) Видалення стоп-слів (Stop Word Removal): Виключення з тексту часто вживаних слів, які зазвичай не несуть значного смислового навантаження в контексті конкретної задачі (наприклад, прийменники, сполучники, частки: "і", "в", "на", "але", "як"). Списки стоп-слів є специфічними для кожної мови. Цей крок є опціональним і не завжди покращує результат, особливо для моделей, що враховують контекст (як РНМ).

Результатом передобробки є послідовність очищених та нормалізованих токенів, готова до наступного етапу – векторизації.

Нейронні мережі та інші алгоритми машинного навчання працюють з числовими даними. Тому текстові послідовності необхідно перетворити на вектори чисел. Існують різні підходи:

– Модель "Мішок слів" (Bag-of-Words, BoW)

Представляє текст як невпорядкований набір слів (токенів), ігноруючи їх порядок. Кожен документ кодується вектором, де кожна позиція відповідає слову зі словника всього корпусу текстів, а значення на цій позиції відображає частоту появи слова в документі (або просто його наявність/відсутність).

– TF-IDF (Term Frequency-Inverse Document Frequency)

Модифікація BoW, яка зважує частоту слова (TF) у документі оберненою частотою цього слова у всьому корпусі документів (IDF). Це дозволяє зменшити вагу дуже частих слів (як стоп-слова) і збільшити вагу більш рідкісних, інформативних слів.

Недоліком BoW та TF-IDF є те, що вони повністю втрачають інформацію про порядок слів у тексті, що є критично важливим для розуміння сенсу, особливо при аналізі ІПСО, де маніпуляція часто криється саме в побудові фраз та речень. Крім того, ці методи призводять до створення дуже розріджених векторів високої розмірності, особливо при великих словниках.

Сучасним та ефективним підходом до векторизації, який зберігає семантичну інформацію та добре підходить для моделей глибокого навчання (включаючи РНМ), є векторні представлення слів або embeddings. Ідея полягає у тому, щоб представити кожне слово у вигляді щільного (dense) вектора фіксованої, відносно невеликої розмірності (наприклад, 100, 128, 300), де семантично близькі слова мають близькі вектори у цьому векторному просторі (наприклад, вектори слів "король" і "цар" будуть ближчими один до одного, ніж до вектора слова "стіл").

Існують різні алгоритми для навчання таких векторних представлень:

– Word2Vec (Mikolov et al., 2013) - навчає представлення на основі контексту слова (моделі CBOW та Skip-gram) [13].

– GloVe (Pennington et al., 2014) - використовує глобальну статистику співвустрічальності слів у корпусі [6].

– FastText (Bojanowski et al., 2017) - розглядає слова як набори n-грам символів, що дозволяє генерувати вектори навіть для слів, яких не було в навчальній вибірці [15].

Важливою перевагою є можливість використання наперед навчених (pre-trained) векторів, отриманих на великих текстових корпусах, що може покращити якість моделі, особливо при обмеженій кількості власних даних.

Однак, у моделях глибокого навчання, таких як ті, що використовуються в даній роботі, часто застосовується Embedding шар (Embedding Layer). Цей шар є частиною самої нейронної мережі. Він приймає на вхід послідовність цілочисельних індексів слів (отриманих від токенізатора) і перетворює кожен індекс на відповідний йому вектор фіксованої розмірності (EMBEDDING_DIM у коді). Важливо, що ваги цього шару (тобто самі векторні представлення слів) навчаються спільно з іншими параметрами мережі в процесі навчання на конкретній задачі (класифікації ІПСО). Це дозволяє отримати представлення, оптимально адаптовані саме для розрізнення цільових категорій у даному корпусі текстів.

Отже, застосування технік передобробки та векторизації слів за допомогою Embedding шару є ключовими кроками НЛП, які дозволяють підготувати текстові дані для ефективної обробки рекурентними нейронними мережами, що детально розглядається у наступному підрозділі.

1.4 Рекурентні нейронні мережі (РНМ) для аналізу послідовностей

Текстові дані є послідовностями, де порядок слів або символів несе ключову семантичну інформацію. Традиційні нейронні мережі прямого поширення (Feedforward Neural Networks) обробляють вхідні дані незалежно один від одного, що робить їх неоптимальними для завдань, де важливий контекст та історія попередніх елементів. Для ефективної обробки таких послідовних даних були розроблені рекурентні нейронні мережі (Recurrent Neural Networks, RNN).

Основна ідея РНМ полягає у наявності рекурентного зв'язку (петлі), який дозволяє виходу мережі на попередньому часовому кроці впливати на її обчислення на поточному кроці. На кожному часовому кроці t РНМ приймає на вхід поточний елемент послідовності *x_t* (наприклад, вектор слова) та прихований стан *h_{t-1}* з

попереднього кроку. На основі цих даних вона обчислює новий прихований стан *h_t* та, зазвичай, вихід *y_t*. Прихований стан *h_t* діє як своєрідна "пам'ять" мережі, акумулюючи інформацію про попередні елементи послідовності.

Математично це можна описати (спрощено) так:

$$h_t = f(W_{hh} * h_{t-1} + W_{xh} * x_t + b_h) \quad (1.1)$$

$$y_t = g(W_{hy} * h_t + b_y), \quad (1.2)$$

де W_{hh}, W_{xh}, W_{hy} – матриці ваг для рекурентного зв'язку, вхідного зв'язку та вихідного зв'язку відповідно; b_h, b_y – вектори зсувів; f та g – функції активації (часто *tanh* або *ReLU* для f, і *softmax* для g у задачах класифікації). Важливо, що матриці ваг є спільними для всіх часових кроків, що дозволяє РНМ обробляти послідовності змінної довжини та узагальнювати вивчені патерни.

Незважаючи на елегантність ідеї, навчання простих РНМ на довгих послідовностях стикається з серйозними проблемами:

– Проблема зникаючих градієнтів (Vanishing Gradients)

Під час зворотного поширення помилки градієнти, що передаються через багато часових кроків, можуть ставати експоненційно малими. Це призводить до того, що ваги, відповідальні за довгострокові залежності, практично не оновлюються, і мережа "забуває" інформацію з далекого минулого.

– Проблема вибухаючих градієнтів (Exploding Gradients)

Зворотна ситуація, коли градієнти стають експоненційно великими, що призводить до нестабільного навчання та великих стрибків у значеннях ваг. Цю проблему частково можна вирішити за допомогою технік, таких як обрізання градієнтів (gradient clipping).

Проблема зникаючих градієнтів є більш фундаментальною і суттєво обмежує здатність простих РНМ моделювати довгострокові залежності, що є критично важливим для розуміння сенсу багатьох текстів, де ключова інформація може знаходитись на значній відстані від поточного слова.

Для подолання обмежень простих РНМ були запропоновані більш складні архітектури, найвідомішою з яких є мережа з довгою короткочасною пам'яттю (Long Short-Term Memory, LSTM), розроблена Хохрайтером та Шмідхубером у 1997 році. LSTM була спеціально спроектована для ефективного запам'ятовування інформації на довгих проміжках часу.

Ключовим елементом LSTM є комірка пам'яті (memory cell), яка має власний стан *C_t* (cell state), що може зберігати інформацію протягом тривалого часу. Потік інформації до та з комірки пам'яті контролюється трьома спеціальними вентилями (gates) – нейронними мережами з сигмоїдною функцією активації, вихід якої лежить у діапазоні [0, 1] (0 – повністю блокувати інформацію, 1 – повністю пропустити):

– Вентиль забування (Forget Gate): Вирішує, яку інформацію потрібно викинути зі стану комірки *C_{t-1}*. Він аналізує попередній прихований стан *h_{t-1}* та поточний вхід *x_t*.

– Вентиль входу (Input Gate): Визначає, яка нова інформація буде збережена у стані комірки. Він складається з двох частин: сигмоїдний шар, що вирішує, які значення оновити, та tanh шар, що створює вектор кандидатів на додавання до стану.

– Вентиль виходу (Output Gate): Визначає, яка частина поточного стану комірки *C_t* буде використана для формування виходу мережі (нового прихованого стану *h_t*). Стан комірки спочатку пропускається через *tanh* (для нормалізації значень від -1 до 1), а потім множиться на вихід сигмоїдного шару вентиля виходу.

Ця вентильна структура дозволяє LSTM вибірково зберігати, оновлювати та забувати інформацію, що робить її значно стійкішою до проблеми зникаючих градієнтів і дозволяє ефективно моделювати довгострокові залежності в послідовностях [1].

Іншою популярною архітектурою, що вирішує проблеми простих РНМ, є керований рекурентний блок (Gated Recurrent Unit, GRU), запропонований Чо та іншими у 2014 році. GRU є певним спрощенням LSTM, маючи меншу кількість параметрів.

Основні відмінності GRU від LSTM:

– Об'єднання стану комірки та прихованого стану: GRU не має окремого стану комірки C_t.

– Два вентилі замість трьох:

– Вентиль оновлення (Update Gate): Аналог вентилів забування та входу в LSTM. Визначає, яка частина попереднього стану збережеться, а яка буде оновлена новою інформацією.

– Вентиль скидання (Reset Gate): Визначає, наскільки попередній стан впливатиме на обчислення стану-кандидата.

Завдяки простішій структурі, GRU може бути обчислювально ефективнішим за LSTM, при цьому часто демонструючи порівнянну якість на багатьох задачах.

Виявлення ІПСО в тексті часто потребує аналізу не лише окремих слів, але й контексту, в якому вони вживаються, синтаксичних конструкцій, тональності та потенційно прихованих смислів, які можуть розкриватися лише при розгляді значних фрагментів тексту. Наприклад, маніпулятивний наратив може будуватися поступово протягом кількох речень.

Саме здатність LSTM та GRU ефективно моделювати довгострокові залежності та зберігати контекстну інформацію робить їх потужним інструментом для цієї задачі, значно перевершуючи можливості традиційних підходів. Можливість навчати векторні представлення слів (Embedding) разом з рекурентними шарами дозволяє моделі адаптувати ці представлення для кращого розрізнення специфічних патернів, характерних для ІПСО.

У даній роботі перевага була віддана LSTM як класичній та добре дослідженій архітектурі для обробки тексту, яка продемонструвала високу ефективність у багатьох схожих задачах.

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ АНАЛІЗУ ІПСО

У даному розділі детально описано процес підготовки даних, проектування архітектури рекурентної нейронної мережі, реалізацію механізму врахування надійності джерела, процес навчання моделі та розробку інтерфейсу для її використання в задачах бінарної класифікації повідомлень на предмет наявності ознак ІПСО. Всі етапи реалізовано з використанням мови програмування Python та ключових бібліотек для машинного навчання та обробки даних. Повна реалізація моделі нейронної мережі наведена у Додатку А.

2.1 Аналіз та підготовка даних для навчання моделі

Якість та репрезентативність даних є критично важливими факторами для успішного навчання будь-якої моделі машинного навчання, особливо моделей глибокого навчання. Оскільки поставлена задача полягає у бінарній класифікації (виявлення наявності ознак ІПСО проти безпечного контенту), процес підготовки даних був спрямований на формування відповідного навчального набору.

Вихідними даними для дослідження слугували три основні файли:

1) messages/messages.jsonl: Файл у форматі JSON Lines, що містить основний корпус текстових повідомлень. Кожен рядок представляє окреме повідомлення з полями source_id (ідентифікатор джерела), message_id (ідентифікатор повідомлення), text (текст повідомлення) та published_at (час публікації).

Приклад запису з файлу messages.jsonl:

```
{'source_id': '12345', 'message_id': 6789, 'text': 'Приклад тексту повідомлення...', 'published_at': 1678886400}
```

2) train.csv: CSV-файл, що містить інформацію про частину джерел (source_id, source_url), для яких наявна вихідна багатокласова мітка (source_category), що включає як різні типи ІПСО, так і безпечний контент.

Приклад запису з файлу train.csv:

```
source_id,source_url,source_category
38619,https://t.me/readovkaru,AGGRESSIVE_INFORMATION
```

3) test.csv: CSV-файл, що містить інформацію про іншу частину джерел (source_id, source_url), але без початкової класифікації.

Приклад запису з файлу test.csv:

```
source_id,source_url
241835,https://t.me/ViktorVikarchuk
```

Для підготовки даних до навчання бінарного класифікатора було виконано наступні кроки з використанням бібліотеки Pandas:

1) Завантажено дані з messages.jsonl у DataFrame df_messages. Проведено базову обробку: перетворення published_at у формат datetime та видалення рядків з відсутніми значеннями в ключових полях (text, source_id, message_id).

2) Завантажено дані про джерела з train.csv та test.csv.

3) Сконсолідовано інформацію про джерела шляхом об'єднання (pd.concat) даних з обох CSV-файлів в єдиний DataFrame df_sources, видаливши дублікати джерел за source_id.

4) Створено цільову бінарну мітку: У DataFrame df_sources було додано нову колонку label. Правило присвоєння мітки базувалося на значенні в колонці source_category: якщо оригінальна мітка була SAFE_CONTENT_LABEL ("SAFE_CONTENT"), то значення в label залишалося таким же; якщо оригінальна мітка була будь-якою іншою (відповідала будь-якій категорії ІПСО) і була наявною (не NaN), то значення в label встановлювалося як узагальнена мітка IPSO_LABEL ("IPSO"). Для рядків, що походили з test.csv (де source_category була відсутня), значення label залишалося невизначеним (NaN на цьому етапі).

5) Об'єднано (pd.merge, how='inner') DataFrame повідомлень (df_messages) з DataFrame джерел (df_sources), використовуючи source_id як ключ. Це дозволило додати до кожного повідомлення відповідну бінарну мітку (або NaN, якщо вона не була визначена).

6) Фінальна фільтрація: З об'єднаного DataFrame було видалено всі рядки, де значення в колонці label залишилося невизначеним (NaN). Це гарантувало, що для навчання будуть використовуватися лише ті повідомлення, для яких вдалося встановити бінарну мітку "SAFE_CONTENT" або "IPSO".

Текстові дані з колонки text пройшли етап стандартизації та очищення за допомогою функції clean_text. Ця функція послідовно виконувала наступні операції за допомогою регулярних виразів:

1) Приведення всього тексту до нижнього регістру.

2) Видалення URL-адрес.

3) Видалення згадок користувачів (токенів, що починаються з @) та хештегів (токенів, що починаються з #).

4) Видалення всіх цифр.

5) Видалення більшості символів пунктуації, за винятком дефісів та підкреслень, які можуть бути частиною слів.

6) Нормалізація пробілів (видалення зайвих та пробілів на початку/кінці рядка).

Наступним кроком була токенізація – розбиття очищеного тексту на окремі слова (токени). Для цього використовувався інструмент Tokenizer з бібліотеки tensorflow.keras.preprocessing.text. Його було налаштовано з параметром num_words=VOCAB_SIZE, встановивши максимальний розмір словника на 30000 найчастіше вживаних слів. Також було визначено спеціальний токен <OOV> (Out-Of-Vocabulary) для представлення всіх слів, що не увійшли до цього обмеженого словника. Токенізатор було навчено (fit_on_texts) на всьому корпусі підготовлених текстів, що дозволило створити відображення "слово -> індекс". Потім кожен текст було перетворено на послідовність відповідних цілочисельних індексів (texts_to_sequences).

Оскільки рекурентні нейронні мережі у реалізації Keras зазвичай потребують вхідних послідовностей однакової довжини, було застосовано паддінг за допомогою функції pad_sequences. Всі послідовності було приведено до фіксованої довжини MAX_LENGTH (200 токенів) шляхом додавання нулів наприкінці коротших послідовностей (padding='post') або обрізання довших послідовностей також з кінця (truncating='post'). Результатом цього етапу став двовимірний масив NumPy X_pad, де кожен рядок – це числове представлення одного повідомлення фіксованої довжини.

Перед фінальним розділенням даних на вибірки було проведено аналіз розподілу отриманих бінарних класів у колонці label за допомогою методу value_counts(). Це дозволило кількісно оцінити ступінь дисбалансу між класами "SAFE_CONTENT" та "IPSO", що є важливим фактором при виборі стратегії навчання та оцінки моделі.

Для перетворення текстових міток ("SAFE_CONTENT", "IPSO") у числовий формат, придатний для бінарної класифікації, використовувався LabelEncoder з бібліотеки sklearn.preprocessing. Енкодер було навчено на списку BINARY_LABELS, що гарантувало послідовне кодування: "SAFE_CONTENT" як 0 та "IPSO" як 1. Результатом застосування методу transform став одновимірний масив NumPy y, що містить цілочисельні мітки класів для кожного повідомлення.

Завершальним кроком підготовки стало розділення масивів X_pad (ознаки) та y (мітки) на навчальну та валідаційну вибірки за допомогою функції train_test_split. Було вибрано співвідношення 80% для навчання та 20% для валідації (test_size=0.2). Для забезпечення відтворюваності експериментів було зафіксовано стан генератора випадкових чисел (random_state=42). Ключовим моментом було використання параметра stratify=y, який гарантує, що пропорції класів "SAFE_CONTENT" та "IPSO" у навчальній та валідаційній вибірках будуть такими ж, як і у вихідному датасеті. Це особливо важливо при роботі з незбалансованими даними, оскільки дозволяє отримати більш надійну оцінку здатності моделі до узагальнення.

Після виконання всіх описаних кроків було отримано навчальні (X_train, y_train) та валідаційні (X_val, y_val) набори даних, а також навчені об'єкти tokenizer та label_encoder, повністю готові для використання на етапах побудови, навчання та тестування нейромережевої моделі.

2.2 Розробка моделі рекурентної нейронної мережі.

Запропонована архітектура нейронної мережі складається з наступних ключових компонентів, що послідовно обробляють вхідні дані:

1) Вхідні шари (Input Layers)

Модель розрахована на прийом двох типів даних, тому визначено два окремі входи:

a) text_input: Приймає послідовності цілочисельних індексів слів фіксованої довжини MAX_LENGTH (200), отримані після токенізації та паддінгу тексту.

b) reliability_input: Приймає одне числове значення – показник надійності джерела повідомлення (від 0 до 1).

2) Шар векторного представлення слів (Embedding Layer)

Безпосередньо до входу text_input підключено шар Embedding. Його функція – перетворити кожен дискретний індекс слова на щільний вектор дійсних чисел фіксованої розмірності EMBEDDING_DIM (128). Цей шар містить матрицю ваг розміром (VOCAB_SIZE, EMBEDDING_DIM), де VOCAB_SIZE (30000) – це розмір словника, що використовується. Ваги цього шару ініціалізуються випадково і навчаються під час тренування моделі, що дозволяє адаптувати векторні представлення слів для специфіки задачі виявлення ІПСО.

3) Рекурентний шар (LSTM Layer)

Послідовність векторів слів, отримана з шару Embedding, подається на вхід шару LSTM. Цей шар, маючи внутрішню пам'ять (стан комірки) та вентильні механізми, здатен обробляти послідовності, враховуючи довгострокові залежності та контекст. Кількість рекурентних юнітів (LSTM_UNITS) встановлено на 64. Для боротьби з перенавчанням застосовано механізми dropout та recurrent_dropout зі

значенням DROPOUT_RATE (0.4). Шар LSTM повертає лише останній прихований стан, який узагальнює інформацію про всю оброблену текстову послідовність.

4) Об'єднання даних (Concatenate Layer)

Вихідний вектор з шару LSTM об'єднується (конкатенується) з вектором, що містить показник надійності джерела (з входу reliability_input). Це дозволяє моделі одночасно враховувати інформацію, видобуту з тексту, та інформацію про репутацію джерела.

5) Повнозв'язні шари (Dense Layers) / Класифікатор

Об'єднаний вектор ознак пропускається через два послідовні повнозв'язні шари (Dense) з функцією активації ReLU. Перший шар має 128 нейронів, другий – 64. Ці шари виконують нелінійні перетворення над отриманими ознаками, готуючи їх до фінальної класифікації. Для стабілізації навчання та додаткової регуляризації після кожного Dense шару (перед Dropout) застосовано шар BatchNormalization, а також Dropout з коефіцієнтом DROPOUT_RATE (0.4).

6) Вихідний шар (Output Layer)

Оскільки задача полягає у бінарній класифікації (IPSO або SAFE_CONTENT), останній повнозв'язний шар має один нейрон (Dense(1, ...)) з функцією активації sigmoid. Функція sigmoid видає на вихід одне число в діапазоні від 0 до 1, яке інтерпретується як ймовірність належності вхідного повідомлення до класу "IPSO" (клас з міткою 1).

Для навчання моделі бінарної класифікації було обрано відповідні компоненти:

– Функція втрат: Використано binary_crossentropy. Це стандартна функція втрат для задач бінарної класифікації, яка вимірює розбіжність між передбаченою моделлю ймовірністю та істинною міткою (0 або 1).

– Оптимізатор: Застосовано алгоритм Adam (tf.keras.optimizers.Adam) зі стандартними параметрами. Adam є ефективним адаптивним оптимізатором для навчання глибоких нейронних мереж.

– Метрики: Для оцінки якості під час навчання використовувалася метрика accuracy (загальна точність).

Використання LSTM дозволяє ефективно моделювати семантичні та контекстуальні залежності в текстових послідовностях, що є важливим для виявлення ознак ІПСО. Інтеграція показника надійності джерела як окремого входу дозволяє моделі використовувати додаткову інформацію для класифікації. Шар Embedding забезпечує гнучке представлення слів, адаптоване до задачі. Послідовність повнозв'язних шарів з BatchNormalization та Dropout формує класифікаційну частину моделі, підвищуючи її стабільність та запобігаючи перенавчанню. Вихідний шар з одним нейроном та активацією sigmoid разом з функцією втрат binary_crossentropy є стандартним та ефективним рішенням для задач бінарної класифікації. Обрані значення гіперпараметрів (VOCAB_SIZE, MAX_LENGTH, EMBEDDING_DIM, LSTM_UNITS, DROPOUT_RATE) є типовими для подібних задач НЛП і забезпечують баланс між складністю моделі та її здатністю до навчання і узагальнення.

2.3 Реалізація механізму управління надійністю джерела.

Важливим аспектом розробленої системи є врахування репутації джерела інформації як фактора, що впливає на класифікацію повідомлень. Було реалізовано механізм, який не лише передає поточний показник надійності джерела на вхід нейронної мережі, але й динамічно оновлює цей показник на основі результатів класифікації, а також запобігає багаторазовому впливу одного й того ж повідомлення на репутацію джерела [5].

Для зберігання даних про надійність джерел та відстеження оброблених повідомлень використовувалася файлова СУБД SQLite через стандартний модуль sqlite3 мови Python. У функції init_db було забезпечено створення двох таблиць (якщо вони не існують):

1) Таблиця sources: Призначена для зберігання поточної надійності кожного унікального джерела.

a) source_id TEXT PRIMARY KEY: Унікальний ідентифікатор джерела (з Telegram)

b) reliability REAL NOT NULL: Поточне значення надійності (від 0.1 до 1.0)

c) last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP: Час останнього оновлення запису.

Для цієї таблиці також створено індекс idx_source_id для прискорення пошуку.

2) Таблиця processed_messages: Призначена для запобігання повторній обробці одного й того ж повідомлення від одного джерела та повторному впливу на його надійність.

a) message_hash TEXT PRIMARY KEY: Унікальний хеш, що ідентифікує пару (джерело + контент повідомлення).

b) processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP: Час обробки повідомлення.

Перед класифікацією кожного повідомлення (окрім прямих повідомлень боту) необхідно отримати актуальний показник надійності його джерела. Це реалізовано у функції get_source_reliability(conn, source_id):

– Якщо source_id відсутній (пряме повідомлення), повертається максимальна надійність INITIAL_RELIABILITY (1.0).

– Якщо source_id надано, виконується пошук запису з цим ID у таблиці sources.

– Якщо джерело знайдено, повертається його поточне значення reliability.

– Якщо джерело зустрічається вперше, воно додається до таблиці sources з початковою надійністю INITIAL_RELIABILITY (1.0), і це значення повертається.

Після отримання результату класифікації від нейронної мережі (тобто визначення, чи є повідомлення ІПСО - is_ipso), показник надійності джерела оновлюється за допомогою функції update_source_reliability(conn, source_id, is_ipso).

Модифікований алгоритм оновлення надійності джерела, реалізований у функції update_source_reliability, тепер враховує як негативні, так і позитивні

сигнали класифікації, ігноруючи при цьому прямі повідомлення боту (для яких source_id відсутній). При обробці повідомлення з відомим source_id, функція спочатку зчитує актуальне значення reliability з бази даних. Потім, залежно від результату класифікації is_ipso, розраховується нове значення надійності. Якщо було виявлено ІПСО (is_ipso = True), поточна надійність зменшується на величину RELIABILITY_DECREMENT (0.02), при цьому не опускаючись нижче встановленого мінімального порогу MIN_RELIABILITY (0.1). У випадку, якщо контент визнано безпечним (is_ipso = False), надійність джерела, навпаки, збільшується на RELIABILITY_INCREMENT (0.01), але не перевищуючи максимального значення INITIAL_RELIABILITY (1.0). Тільки якщо розраховане нове значення надійності відрізняється від поточного, виконується запит UPDATE до бази даних для збереження нового показника та оновлення часу останньої зміни.

Такий підхід дозволяє не лише штрафувати джерела за поширення ІПСО, але й поступово відновлювати їхню репутацію, якщо вони починають публікувати безпечний контент. Крок збільшення надійності (RELIABILITY_INCREMENT) навмисно зроблено меншим за крок зменшення (RELIABILITY_DECREMENT), щоб відновлення репутації відбувалося повільніше, ніж її втрата.

Щоб уникнути ситуації, коли одне й те саме повідомлення, переслане кілька разів з одного джерела, багаторазово впливає на його надійність, було реалізовано механізм перевірки дублікатів у функції predict_single_message:

1) Хешування: Перед викликом update_source_reliability для кожного повідомлення, що має source_id, розраховується унікальний ідентифікатор – хеш SHA-256 від конкатенації source_id та очищеного тексту повідомлення (cleaned_text).

2) Перевірка в БД: Виконується запит до таблиці processed_messages для перевірки, чи існує вже запис з таким хешем.

3) Умовне оновлення надійності: Якщо хеш знайдено, це означає, що ідентичне повідомлення від цього джерела вже оброблялося раніше. У цьому випадку виклик функції update_source_reliability пропускається, і надійність джерела не змінюється. Повертається поточне значення надійності.

4) Запис хешу: Якщо хеш не знайдено, то після виклику update_source_reliability (тобто після того, як надійність була потенційно оновлена), розрахований хеш записується у таблицю processed_messages. Це гарантує, що при наступних надходженнях цього ж повідомлення від цього ж джерела воно буде розпізнано як дублікат.

Показник надійності, отриманий функцією get_source_reliability, перед кожним викликом моделі передається на відповідний вхідний шар reliability_input, дозволяючи нейронній мережі враховувати репутацію джерела під час класифікації тексту.

Реалізований комплексний механізм дозволяє не лише оцінювати контент, але й динамічно управляти репутацією джерел на основі їхньої поведінки, враховуючи як негативні (поширення ІПСО), так і позитивні (публікація безпечного контенту) сигнали, та ігноруючи дублікати.

2.4 Процес навчання та налаштування моделі

Навчання нейронної мережі є ітеративним процесом оптимізації її внутрішніх параметрів (ваг) з метою мінімізації помилки на навчальних даних. Для розробленої бінарної LSTM моделі цей процес було організовано з використанням фреймворку Keras/TensorFlow, застосовуючи специфічні підходи для роботи з незбалансованими даними та контролю навчання [16].

Навчання здійснювалося за допомогою методу model.fit(), який отримував підготовлені навчальні (X_train, y_train) та валідаційні (X_val, y_val) вибірки. Ключові аспекти організації процесу:

1) Вхідні дані

Модель приймала два входи: послідовності індексів слів (X_train/X_val) та відповідні їм вектори надійності. Оскільки динамічна надійність не може використовуватися під час самого навчання, для всіх прикладів у навчальній та валідаційній вибірках на вхід reliability_input подавалося фіксоване значення початкової надійності INITIAL_RELIABILITY (1.0).

2) Цільові мітки

Навчання проводилося на бінарних мітках y_train (масив значень 0 та 1), а якість оцінювалася за відповідними мітками y_val.

3) Епохи та батчі

Максимальна кількість епох (EPOCHS) була встановлена на 15, а розмір батчу (BATCH_SIZE) – на 64. Це означає, що модель проходила по всьому навчальному набору даних до 15 разів, обробляючи дані групами по 64 приклади за одну ітерацію оновлення ваг.

Враховуючи виявлений значний дисбаланс між класами "SAFE_CONTENT" (0) та "IPSO" (1) у навчальних даних, було застосовано техніку зважування класів (class weighting) [7]. Перед початком навчання було розраховано ваги для кожного класу за допомогою функції compute_class_weight з бібліотеки sklearn.utils, використовуючи режим 'balanced'. Цей режим автоматично призначає ваги, обернено пропорційні частоті класів у навчальній вибірці y_train. Таким чином, рідкісніший клас "SAFE_CONTENT" отримував значно вищу вагу, ніж домінуючий клас "IPSO". Розрахований словник ваг (class_weights_dict) передавався безпосередньо у метод model.fit(..., class_weight=class_weights_dict). Це змушувало модель під час розрахунку функції втрат (binary_crossentropy) надавати більшої "важливості" помилкам на прикладах менш численного класу, стимулюючи її краще розрізняти безпечний контент [8].

Для контролю процесу навчання, запобігання перенавчанню та збереження найкращої версії моделі використовувалися стандартні колбеки Keras:

– EarlyStopping - цей колбек відстежував значення функції втрат на валідаційній вибірці (monitor='val_loss'). Якщо це значення не зменшувалося протягом трьох послідовних епох (patience=3), навчання автоматично припинялося. Параметр restore_best_weights=True гарантував, що після зупинки ваги моделі повертались до стану, який відповідав мінімальному значенню val_loss, досягнутому за весь час навчання.

– ModelCheckpoint - цей колбек також відстежував val_loss і зберігав тільки ваги моделі (save_weights_only=True) у файл MODEL_WEIGHTS_FILE (...

.weights.h5) лише тоді, коли досягалося нове мінімальне значення val_loss (save_best_only=True).

Процес навчання ініціювався викликом model.fit(). Завдяки колбекам, навчання завершувалося або по досягненню максимальної кількості епох, або раніше, якщо спрацьовувала рання зупинка. В результаті у файлі MODEL_WEIGHTS_FILE зберігалися ваги моделі, що продемонстрували найкращу узагальнюючу здатність на валідаційній вибірці. Об'єкт history, що повертався методом fit(), містив записи значень втрат та точності на кожній епосі, які використовувалися для візуалізації кривих навчання (див. розділ 3).

2.5 Розробка інтерфейсу для інтеграції моделі

Після успішного навчання та збереження найкращої версії моделі виникає необхідність надати зручний та стандартизований спосіб її використання для аналізу нових, раніше не бачених повідомлень. Це особливо важливо для інтеграції моделі в зовнішні системи, такі як Telegram-бот, розроблений у рамках суміжного проекту. З цією метою було розроблено спеціальну функцію-інтерфейс predict_single_message, яка інкапсулює всю логіку, необхідну для обробки одного вхідного повідомлення та отримання результату класифікації разом з оновленою надійністю джерела.

Основне призначення функції predict_single_message – виступати єдиною точкою входу для взаємодії з навченою моделлю та пов'язаним з нею механізмом управління надійністю. Вона абстрагує складність внутрішніх процесів (передобробка, виклик моделі, робота з БД) від системи, яка її викликає (наприклад, від коду Telegram-бота).

Функція приймає наступні вхідні параметри:

– model: Навчений об'єкт моделі Keras.

– tokenizer: Навчений об'єкт Tokenizer Keras, необхідний для перетворення тексту на послідовність індексів.

– db_conn: Активне з'єднання з базою даних SQLite, де зберігається інформація про надійність джерел.

– text: Вхідний текст повідомлення, що підлягає аналізу (рядок).

– source_id: Ідентифікатор джерела повідомлення (рядок) або None, якщо повідомлення надійшло безпосередньо від користувача (не є пересланим).

– max_length: Максимальна довжина послідовності, що використовувалась під час навчання (для коректного паддінгу).

– threshold: Поріг (за замовчуванням 0.5), який використовується для перетворення вихідної ймовірності моделі на бінарне рішення (IPSO/не IPSO)

При виклику функція predict_single_message виконує наступну послідовність дій:

1) Перевірка та очищення тексту: Перевіряється наявність та коректність вхідного тексту, після чого він очищується за допомогою функції clean_text.

2) Отримання поточної надійності: Викликається get_source_reliability для отримання актуального показника надійності для вказаного source_id.

3) Перевірка на дублікат: Якщо source_id надано, розраховується хеш SHA-256 від комбінації source_id та cleaned_text. Виконується запит до таблиці processed_messages для перевірки, чи оброблялося вже ідентичне повідомлення від цього джерела. Встановлюється прапорець message_processed.

4) Токенізація та паддінг: Очищений текст перетворюється на послідовність індексів за допомогою tokenizer та доповнюється/обрізається до max_length за допомогою pad_sequences.

5) Підготовка вхідних даних для моделі: Формуються два вхідні масиви NumPy: padded_sequence для тексту та reliability_input для надійності.

6) Прогнозування моделлю: Підготовлені дані подаються на вхід моделі (model.predict()). Модель повертає одне числове значення – ймовірність того, що повідомлення належить до класу "IPSO" (клас 1).

7) Бінарна класифікація: Отримана ймовірність порівнюється з переданим порогом threshold. Якщо ймовірність більша або дорівнює порогу, результат класифікації is_ipso встановлюється як True, інакше – False.

8) Умовне оновлення надійності: Якщо повідомлення не було позначено як вже оброблене (message_processed == False) і воно надійшло від відомого джерела (source_id is not None), викликається функція update_source_reliability для коригування показника надійності в базі даних (зменшення для IPSO, збільшення для SAFE) та записується хеш обробленого повідомлення в таблицю processed_messagesно, розраховується хеш SHA-256 від комбінаціїsource_idtacleaned_text. Виконується запит до таблиціprocessed_messagesдля перевірки, чи оброблялося вже таке повідомлення від цього джерела. Встановлюється прапорецьmessage_processed`.

9) Підготовка даних для моделі: Текст перетворюється на послідовність індексів за допомогою tokenizer та доповнюється/обрізається (pad_sequences) до max_length. Поточна надійність перетворюється на масив NumPy.

10) Передбачення моделлю: Підготовлені дані (текст та надійність) подаються на вхід model.predict(). Модель повертає одне число – ймовірність того, що повідомлення належить до класу "IPSO".

11) Бінарна класифікація: Отримана ймовірність порівнюється з переданим порогом threshold. Якщо ймовірність більша або дорівнює порогу, результат класифікації is_ipso встановлюється як True, інакше False.

12) Умовне оновлення надійності: Викликається функція update_source_reliability з результатом is_ipso. Ця функція оновлює надійність у базі даних (зменшує для IPSO, збільшує для SAFE_CONTENT) лише у випадку, якщо повідомлення НЕ є дублікатом (message_processed == False) і НЕ є прямим (source_id is not None).

13) Запис хешу (якщо потрібно): Якщо повідомлення не було дублікатом і надійність була потенційно оновлена, розрахований раніше хеш записується в таблицю processed_messages, щоб позначити повідомлення як оброблене.

14) Повернення результату: Функція повертає кортеж з двох значень:

15) is_ipso: Булеве значення (True – виявлено ознаки ІПСО, False – безпечний контент).

16) final_reliability: Фінальне значення надійності джерела після всіх перевірок та можливого оновлення

Розробка функції predict_single_message як єдиного інтерфейсу для взаємодії з моделлю забезпечує слабку зв'язаність (loose coupling) між ядром системи (нейромережевою моделлю та логікою надійності) та зовнішніми компонентами (Telegram-ботом). Код бота не потребує знань про внутрішню архітектуру моделі, процес передобробки чи структуру бази даних надійності. Йому достатньо викликати одну функцію, передавши текст і ID джерела, та отримати готовий результат для відображення користувачеві або подальшої обробки. Це спрощує розробку, тестування та подальшу підтримку як самої моделі, так і системи, в яку вона інтегрована.

Таким чином, функція predict_single_message виступає ключовим елементом, що забезпечує практичне застосування розробленої та навченої нейронної мережі в реальних сценаріях використання, таких як аналіз повідомлень у Telegram-боті.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ МОДЕЛІ

У цьому розділі представлено опис середовища, в якому проводилось навчання та тестування розробленої моделі рекурентної нейронної мережі для бінарної класифікації ІПСО. Визначено ключові метрики, що використовувались для оцінки якості її роботи, проаналізовано процес навчання та наведено результати експериментальної оцінки ефективності моделі на валідаційній вибірці.

3.1 Середовище проведення експериментів

Для проведення експериментів з навчання та оцінки розробленої нейромережевої моделі було використано наступне апаратне та програмне забезпечення:

Апаратне забезпечення:

Центральний процесор (CPU): 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz.

Оперативна пам'ять (RAM): 16.0 GB.

Графічний процесор (GPU): NVIDIA GeForce RTX 3050 Laptop GPU (4 GB).

Накопичувач: 477 GB SSD NVMe SAMSUNG MZVLQ512HBLU-00B00.

Програмне забезпечення:

Операційна система: Windows 10 Home x64.

Мова програмування: Python версії 3.11.9

Основні бібліотеки та фреймворки (з версіями):

TensorFlow: версія 2.16.1 [9].

Pandas: версія 2.1.2 [16].

NumPy: версія 1.26.1 [17].

Scikit-learn: версія 1.5.0 [18].

SQLite3: версія 2.6.0 [19].

Matplotlib: версія 3.10.1 [20].

NLTK: версія 3.8.1 [21].

Вибір даного середовища був зумовлений доступністю інструментів та їхньою широкою підтримкою спільнотою для задач машинного навчання та обробки природної мови.

3.2 Метрики оцінки якості класифікації

Для всебічної оцінки ефективності розробленої моделі бінарної класифікації використовувався набір стандартних метрик:

1) Точність (Accuracy): Загальна частка правильно класифікованих прикладів (як "IPSO", так і "SAFE_CONTENT") відносно загальної кількості прикладів:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN). \quad (3.1)$$

Хоча ця метрика є інтуїтивно зрозумілою, вона може бути недостатньо інформативною при значному дисбалансі класів.

2) Точність для класу (Precision):

Для класу "IPSO" (позитивний клас): Частка повідомлень, які дійсно є ІПСО, серед усіх повідомлень, які модель класифікувала як ІПСО:

$$TP / (TP + FP). \quad (3.2)$$

Для класу "SAFE_CONTENT" (негативний клас): Частка повідомлень, які дійсно є безпечними, серед усіх повідомлень, які модель класифікувала як безпечні:

$$TN / (TN + FN), \quad (3.3)$$

де FN тут означає хибно-негативні для SAFE, тобто IPSO, позначені як SAFE).

Висока Precision для класу "IPSO" означає, що якщо модель каже "це ІПСО", то їй можна довіряти.

3) Повнота для класу (Recall / Sensitivity):

Для класу "IPSO": Частка реальних ІПСО-повідомлень, які модель змогла правильно ідентифікувати:

$$TP / (TP + FN). \quad (3.4)$$

Для класу "SAFE_CONTENT": Частка реальних безпечних повідомлень, які модель змогла правильно ідентифікувати:

$$TN / (TN + FP), \quad (3.5)$$

де FP тут означає хибно-позитивні для SAFE, тобто SAFE, позначені як IPSO).

Висока Recall для класу "IPSO" означає, що модель добре виявляє більшість ІПСО-повідомлень.

4) F1-міра (F1-score): Гармонійне середнє між Precision та Recall, що надає збалансовану оцінку, особливо корисну при дисбалансі класів. Розраховується для кожного класу окремо.

$$F1 = 2 * (Precision * Recall) / (Precision + Recall). \quad (3.6)$$

5) Матриця помилок (Confusion Matrix): Таблиця розміром 2x2, що візуалізує ефективність класифікатора. Вона показує кількість:

– Істинно позитивних (TP): ІПСО, правильно класифіковані як ІПСО.

– Істинно негативних (TN): Безпечні, правильно класифіковані як безпечні.

– Хибно позитивних (FP): Безпечні, помилково класифіковані як ІПСО (помилка I роду).

– Хибно негативних (FN): ІПСО, помилково класифіковані як безпечні (помилка II роду).

Аналіз матриці помилок дозволяє зрозуміти, якого типу помилок модель припускається частіше.

Ці метрики розраховувалися на валідаційній вибірці для оцінки здатності моделі до узагальнення на даних, яких вона не бачила під час навчання.

3.3 Аналіз процесу навчання моделі

Для оцінки ефективності процесу навчання розробленої бінарної LSTM моделі та виявлення потенційних проблем, таких як перенавчання або недонавчання, було проаналізовано динаміку основних метрик – функції втрат (loss) та точності (accuracy) – на навчальній (train) та валідаційній (val) вибірках протягом всіх епох навчання. Цей аналіз також дозволяє оцінити адекватність обраних гіперпараметрів та ефективність застосованих механізмів регуляризації (Dropout, BatchNormalization) та ранньої зупинки.

Для наочного представлення динаміки навчання було побудовано графіки залежності функції втрат та точності від номера епохи. Ці графіки відображають:

1) Втрати на навчанні (loss) та валідації (val_loss): Показують, наскільки передбачення моделі відрізняються від істинних міток на відповідних вибірках.

2) Точність на навчанні (accuracy) та валідації (val_accuracy): Показують частку правильно класифікованих прикладів.

Згідно з графіком втрат (рисунок 3.1, зліва), спостерігається очікувана поведінка:

– Втрати на навчальній вибірці (Втрати Навчання, синя лінія) демонструють стабільне та значне зниження протягом усіх 10 епох. Починаючи з високого значення (близько 0.69 на першій епосі), вони поступово зменшуються до дуже низького значення (близько 0.06 на десятій епосі). Це свідчить про те, що модель ефективно навчається на тренувальних даних, мінімізуючи помилку.

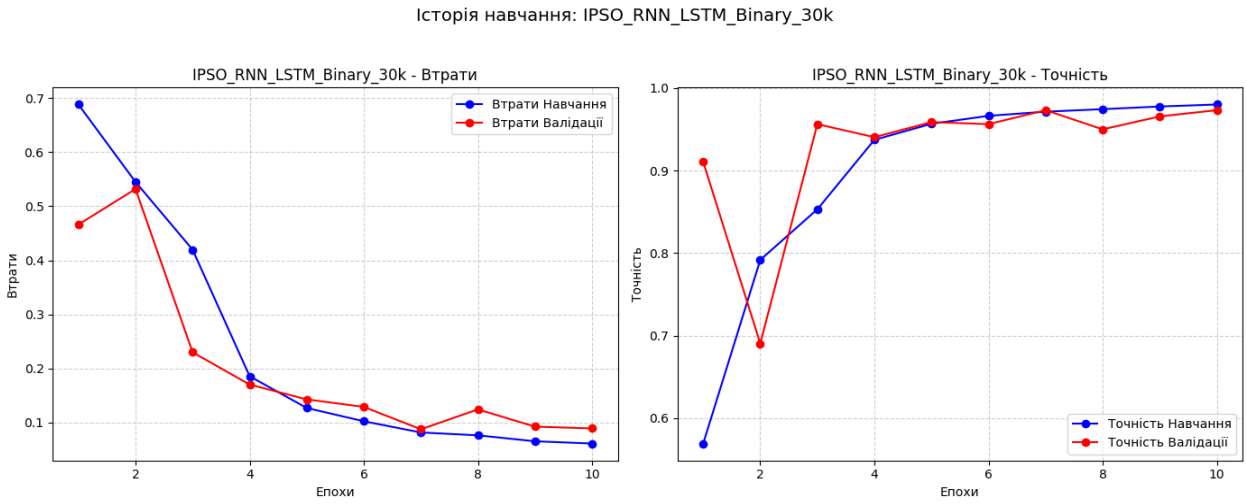

Рисунок 3.1 - Історія навчання моделі IPSO_RNN_LSTM_Binary_30k: Втрати (зліва) та Точність (справа)

– Втрати на валідаційній вибірці (Втрати Валідації, червона лінія) також демонструють зниження, особливо помітне на перших 4-5 епохах. Починаючи з приблизно 0.47 на першій епосі, вони різко падають до близько 0.18 на четвертій епосі, після чого досягають свого мінімуму приблизно 0.09 на шостій епосі. На наступних епохах (7-10) спостерігається незначне коливання та легка тенденція до зростання (до 0.09-0.12), що може бути ранньою ознакою початку перенавчання. Той факт, що валідаційні втрати не продовжують стрімко падати разом із навчальними після 6-ї епохи, вказує на те, що модель починає підлаштовуватися під специфіку навчальних даних, і її здатність до узагальнення на нових даних перестає покращуватися такими ж темпами.

Графік точності (рисунок 1, справа) показує наступну динаміку:

– Точність на навчальній вибірці (Точність Навчання, синя лінія) демонструє швидке зростання з приблизно 0.57 на першій епосі до майже 0.98 на десятій епосі. Це корелює зі зниженням навчальних втрат і показує, що модель добре "запам'ятовує" навчальні приклади.

– Точність на валідаційній вибірці (Точність Валідації, червона лінія) має більш складну поведінку. Після різкого зростання з 0.91 на першій епосі до приблизно 0.96 на третій епосі, спостерігається деяке зниження до 0.94-0.95 на

наступних епохах (7-8), після чого точність знову дещо зростає до 0.96-0.97 на 9-10 епохах. Варто відзначити, що валідаційна точність починається з досить високого рівня (0.91), що може бути пов'язано з ефективністю зважування класів або специфікою даних.

– Розрив між навчальною та валідаційною точністю на останніх епохах (навчальна ~0.98, валідаційна ~0.97) є невеликим, що є позитивним знаком і свідчить про те, що перенавчання не є надто сильним.

Для запобігання перенавчанню та оптимізації часу навчання використовувався колбек EarlyStopping з параметром patience=3, який відстежував значення val_loss.

Згідно з протоколом навчання (вивід якого наведено нижче), процес було зупинено на 10 епосі з 15 запланованих:

```
Epoch 10: val_loss did not improve from 0.08597
←[1m3208/3208←[0m    ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m382s←[0m 119ms/step - accuracy: 0.9760 - loss: 0.0697 - val_accuracy: 0.9722 - val_loss: 0.0901
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 7.
```

Це свідчить про те, що значення val_loss не покращувалося протягом трьох послідовних епох після досягнення свого мінімуму. Завдяки параметру restore_best_weights=True, ваги моделі були автоматично повернуті до стану, що відповідав найкращому (мінімальному) значенню val_loss. Саме ця версія моделі, збережена також колбеком ModelCheckpoint, використовувалася для подальшої оцінки та тестування.

Аналіз кривих навчання свідчить про те, що модель успішно навчалася: втрати на навчальній вибірці значно зменшились, а точність зросла до високих значень. Втрати та точність на валідаційній вибірці також показують хорошу динаміку, досягаючи оптимальних значень приблизно на 6-й епосі для втрат та високих значень (близько 0.96-0.97) для точності. Незначне розходження між навчальними та валідаційними кривими на пізніших епохах вказує на початок перенавчання, що підтверджує доцільність використання механізму ранньої

зупинки або вибору моделі на основі найкращих показників на валідаційній вибірці. Досягнуті показники точності на валідації (близько 96-97%) є обнадійливими для подальшої оцінки моделі.

3.4 Оцінка якості бінарної класифікації на тестовій вибірці

Після завершення процесу навчання та вибору найкращої версії моделі (на основі мінімальних втрат на валідаційній вибірці або спрацювання ранньої зупинки), було проведено детальну оцінку її якості класифікації. Ця оцінка здійснювалася на валідаційній вибірці (X_val, y_val), яка не використовувалася безпосередньо для налаштування ваг моделі, що дозволяє отримати більш об'єктивне уявлення про її здатність до узагальнення на нових даних.

Для оцінки використовувалися метрики, описані в підрозділі 3.2, зокрема, матриця помилок (Confusion Matrix) та детальний звіт про класифікацію (Classification Report), що включає Precision, Recall та F1-score для кожного з двох класів: "SAFE_CONTENT" та "IPSO". Передбачення моделі (ймовірності) перетворювалися на бінарні мітки (0 або 1) з використанням стандартного порогу 0.5.

Матриця помилок візуалізує ефективність класифікатора, показуючи кількість істинно позитивних (TP), істинно негативних (TN), хибно позитивних (FP) та хибно негативних (FN) спрацювань.

Таблиця 3.1 - Матриця помилок для бінарної моделі на валідаційній вибірці

	Передбачено: IPSO	Передбачено: SAFE_CONTENT
Істинно: IPSO	48410 (TP)	1199 (FN)
Істинно: SAFE CONTENT	175 (FP)	1532 (TN)

З Таблиці 3.1 видно, що:

– Істинно позитивні (TP): Модель правильно класифікувала 48410 повідомлень як ІПСО.

– Істинно негативні (TN): Модель правильно класифікувала 1532 повідомлення як безпечний контент (SAFE_CONTENT).

– Хибно позитивні (FP): 175 безпечних повідомлень було помилково класифіковано як ІПСО (помилка I роду).

– Хибно негативні (FN): 1199 ІПСО-повідомлень було помилково класифіковано як безпечний контент (помилка II роду).

Кількість хибно негативних спрацювань (FN = 1199) є значно вищою за кількість хибно позитивних (FP = 175). Це вказує на те, що при стандартному порозі 0.5 модель має тенденцію частіше пропускати ІПСО (вважаючи їх безпечними), ніж помилково маркувати безпечний контент як ІПСО. Це може бути пов'язано із значним дисбалансом класів у вихідних даних, незважаючи на застосування зважування класів під час навчання.

Детальний звіт про класифікацію (Таблиця 3.2) надає значення Precision, Recall та F1-score для кожного класу.

Таблиця 3.2 - Звіт про класифікацію для бінарної моделі на валідаційній вибірці

Клас	Precision	Recall	F1-score	Support
IPSO	1.00	0.98	0.99	49609
SAFE_CONTENT	0.56	0.90	0.69	1707
accuracy			0.97	51316
macro avg	0.78	0.97	0.84	51316
weighted avg	0.98	0.97	0.98	51316

Загальна точність (accuracy) моделі на валідаційній вибірці становить 0.97 (97%), що є високим показником. Однак, аналіз метрик для окремих класів розкриває важливі нюанси. Для класу "IPSO":

– Precision = 1.00: Це означає, що практично всі повідомлення, які модель класифікувала як ІПСО, дійсно є ІПСО. Модель дуже надійно визначає ІПСО, коли вона робить таке передбачення.

– Recall = 0.98: Модель правильно ідентифікує 98% всіх реальних ІПСО-повідомлень у валідаційній вибірці. Це дуже хороший показник повноти виявлення ІПСО.

– F1-score = 0.99: Високий F1-score підтверджує відмінну якість класифікації для класу "IPSO".

Для класу "SAFE_CONTENT":

– Precision = 0.56: Це означає, що лише 56% повідомлень, які модель класифікувала як безпечні, дійсно є такими. Інші 44% (1 - 0.56) повідомлень, позначених як безпечні, насправді були ІПСО (це відповідає високому числу FN з матриці помилок).

– Recall = 0.90: Модель правильно ідентифікує 90% всіх реальних безпечних повідомлень. Це означає, що вона відносно рідко помилково маркує безпечний контент як ІПСО (що відповідає низькому числу FP).

– F1-score = 0.69: Цей показник є значно нижчим, ніж для класу "IPSO", переважно через низький Precision.

Отримана модель демонструє високу загальну точність (97%) на валідаційній вибірці. Вона надзвичайно ефективно розпізнає ІПСО-повідомлення (високі Precision, Recall та F1-score для класу "IPSO"). Однак, спостерігається значна проблема з точністю (Precision) для класу "SAFE_CONTENT": коли модель каже, що повідомлення безпечне, існує значна ймовірність (близько 44%), що це насправді ІПСО. Це підтверджується відносно великою кількістю хибно негативних спрацювань (FN). Такий результат, незважаючи на застосування зважування класів, все ще відображає вплив сильного дисбалансу у вихідних даних, де клас "IPSO" був значно більш представленим. Модель "навчилася" бути дуже обережною з маркуванням чогось як "SAFE_CONTENT", але при цьому все ще пропускає частину ІПСО. Для покращення розрізнення класу "SAFE_CONTENT" та зменшення кількості FN може знадобитися подальше налаштування порогу

класифікації, застосування більш складних технік балансування даних (наприклад, SMOTE для "SAFE_CONTENT" у поєднанні з андерсемплінгом "IPSO") або збір більшої кількості репрезентативних прикладів безпечного контенту.

3.5 Розробка інструменту для інтерактивного тестування моделі (test_interactive.py)

Для забезпечення можливості оперативного тестування навченої нейронної мережі та механізму управління надійністю джерела без необхідності повної інтеграції в кінцеву систему (Telegram-бот) або повторного запуску всього циклу навчання, було розроблено допоміжний скрипт test_interactive.py (див. Додаток Б). Цей скрипт призначений для роботи з моделлю в інтерактивному режимі через командний рядок.

Основною метою скрипта test_interactive.py є надання зручного інтерфейсу для завантаження попередньо навченої моделі та всіх необхідних компонентів (токенізатора, кодувальника міток), ініціалізації з'єднання з базою даних надійності джерел, введення користувачем текстових повідомлень та ідентифікаторів джерел через командний рядок, виклику основної функції класифікації predict_single_message (з модуля ipso_model.py) з введеними даними та відображення результатів аналізу, включаючи передбачення (IPSO/SAFE_CONTENT), ймовірність ІПСО та оновлений показник надійності джерела.

Скрипт test_interactive.py реалізовано на мові Python з використанням наступної логіки:

1) Імпорт необхідних компонентів

На початку скрипта імпортуються необхідні функції (init_db, get_source_reliability, update_source_reliability, clean_text, predict_single_message, build_ipso_rnn_model) та константи (шляхи до файлів, параметри моделі) з основного модуля ipso_model.py. Це забезпечує узгодженість та перевикористання коду.

2) Перевірка наявності файлів

Перед завантаженням ресурсів скрипт перевіряє наявність файлів з вагами моделі (MODEL_WEIGHTS_FILE), токенізатором (TOKENIZER_FILE) та кодувальником міток (LABEL_ENCODER_FILE). У разі відсутності будь-якого з файлів, виводиться повідомлення про необхідність попереднього навчання моделі.

3) Завантаження ресурсів

Якщо всі необхідні файли присутні, виконується послідовне завантаження:

– Ініціалізація з'єднання з базою даних ipso_database_binary.db за допомогою функції init_db.

– Завантаження збереженого токенізатора з файлу .json за допомогою tokenizer_from_json.

– Завантаження збереженого кодувальника міток з файлу .pkl за допомогою pickle.load.

– Побудова архітектури моделі за допомогою build_ipso_rnn_model (з параметрами, що відповідають навченій моделі) та завантаження в неї збережених ваг з файлу .weights.h5 за допомогою model.load_weights.

4) Інтерактивний цикл

Після успішного завантаження всіх компонентів скрипт переходить у нескінченний цикл, в якому:

– Користувачеві пропонується ввести текст повідомлення.

– Користувачеві пропонується ввести ID джерела (або залишити поле порожнім для симуляції прямого повідомлення).

– Введені дані передаються у функцію predict_single_message.

– Результати, повернуті функцією (факт класифікації як ІПСО та оновлена надійність джерела), виводяться у консоль.

– Цикл продовжується доти, доки користувач не введе команду "exit" (або не перерве виконання скрипта, наприклад, через Ctrl+C).

5) Завершення роботи: Після виходу з циклу скрипт коректно закриває з'єднання з базою даних.

Перевірка роботи нейронної мережі за допомогою скрипта test_interactive.py здійснювалася шляхом послідовного введення тестових текстових повідомлень, що репрезентують як безпечний контент, так і різні потенційні прояви ІПСО. Для кожного такого повідомлення користувач мав можливість вказати ідентифікатор джерела або залишити його порожнім для імітації прямого звернення. Після кожного введення скрипт виводив у консоль деталізований результат аналізу: передбачення моделі (IS_IPSO=True/False), розраховану ймовірність належності до класу ІПСО, а також актуальний показник надійності джерела, який динамічно оновлювався. Це дозволило наочно спостерігати за реакцією моделі, оцінювати адекватність її класифікації та коректність роботи механізму управління надійністю. Нижче наведено приклади таких інтерактивних сесій та результатів, отриманих для різних типів вхідних даних (рисунки 3.1-3.2).

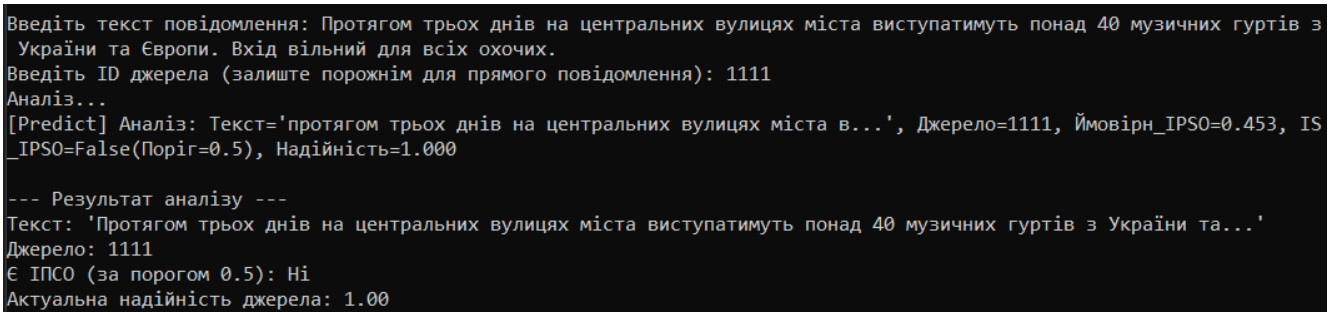

Рисунок 3.1 - Результат аналізу повідомлення, що класифікується як безпечний контент, за допомогою test_interactive.py.

Як видно з рисунку 3.1, повідомлення «Протягом трьох днів на центральних вулицях міста виступатимуть понад 40 музичних гуртів з України та Європи. Вхід вільний для всіх охочих» було коректно класифіковано як безпечне із ймовірністю ІПСО нижче порогового значення (0,453). Надійність джерела 'channel_safe' при цьому залишилась 1.00.

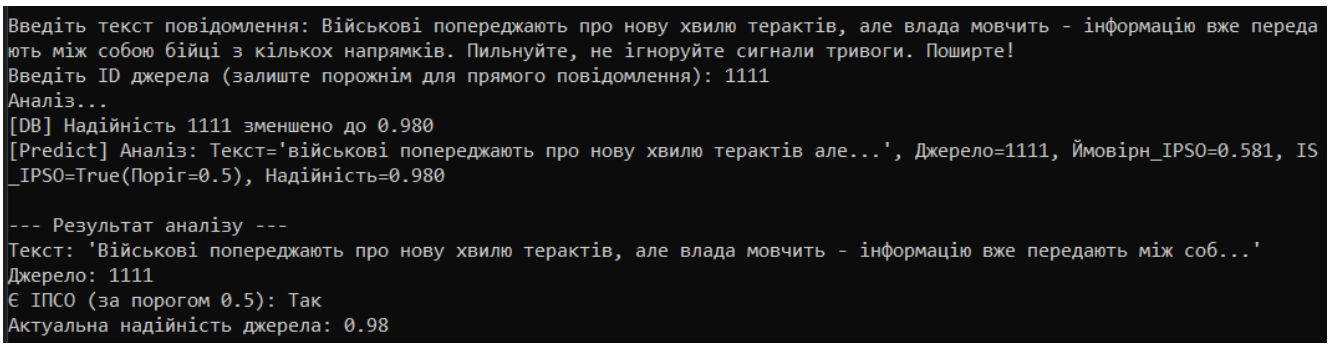

Рисунок 3.2 - Результат аналізу повідомлення з ознаками ІПСО від того ж джерела за допомогою test_interactive.py.

Повідомлення «Військові попереджають про нову хвилю терактів, але влада мовчить - інформацію вже передають між собою бійці з кількох напрямків. Пильнуйте, не ігноруйте сигнали тривоги. Поширте!» від того ж джерела «1111» було класифіковано як ІПСО із ймовірністю, більшою за порогове значення (0.581). Надійність джерела було знижено до 0,98.

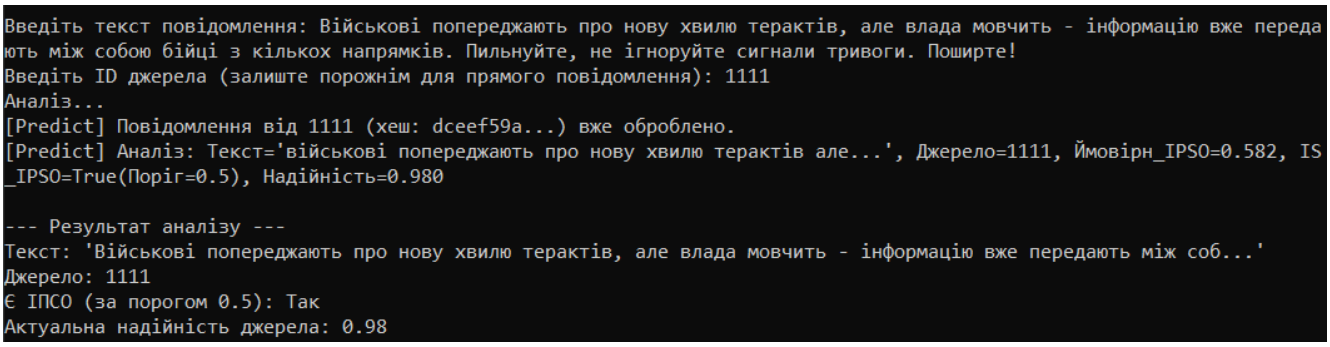

Рисунок 3.3 - Демонстрація ігнорування дублікату ІПСО-повідомлення за допомогою test_interactive.py.

При повторному введенні того ж повідомлення «Військові попереджають про нову хвилю терактів, але влада мовчить - інформацію вже передають між собою бійці з кількох напрямків. Пильнуйте, не ігноруйте сигнали тривоги. Поширте!» від джерела «1111», система коректно ідентифікувала його як дублікат. Повідомлення було класифіковано як ІПСО, але показник надійності джерела залишився незмінним (0,98), що підтверджує роботу механізму запобігання повторному впливу.

Розробка test_interactive.py надала низку суттєвих переваг для процесу дослідження та валідації моделі. Зокрема, цей скрипт забезпечив можливість швидкого тестування, дозволяючи оперативно перевіряти реакцію нейронної мережі на різноманітні текстові повідомлення та оцінювати коректність роботи механізму динамічного оновлення надійності джерела без необхідності розгортання та налаштування повноцінного Telegram-бота. Окрім того, він забезпечив ізоляцію модуля аналізу ІПСО, дозволяючи тестувати його функціонал незалежно від інших потенційних компонентів системи. Це, в свою чергу, значно спростило процес відладки ключової функції predict_single_message та логіки її взаємодії з базою даних. Нарешті, test_interactive.py може слугувати простим, але ефективним інструментом для демонстрації базових можливостей та принципів роботи розробленої системи. Таким чином, впровадження даного інтерактивного скрипта виявилося корисним та практичним кроком на етапах розробки та тестування нейромережевої моделі.

3.6 Аналіз впливу показника надійності джерела

Однією з ключових особливостей розробленої моделі є інтеграція динамічного показника надійності джерела як додаткової вхідної ознаки, поряд з текстовим контентом повідомлення. Передбачалося, що ця ознака дозволить моделі більш точно класифікувати повідомлення, враховуючи попередню "репутацію" джерела щодо поширення ІПСО.

Теоретично, інформація про надійність джерела може слугувати важливим контекстуальним фактором. Якщо джерело систематично поширює контент, що раніше класифікувався як ІПСО (і його надійність знижується), модель може з більшою ймовірністю віднести наступні повідомлення з цього джерела до категорії ІПСО, навіть якщо сам текст є неоднозначним. І навпаки, висока надійність джерела може слугувати сигналом на користь класифікації повідомлення як безпечного.

В архітектурі розробленої нейронної мережі показник надійності джерела подається на окремий вхідний шар (reliability_input). Після обробки текстової послідовності шаром LSTM, отриманий векторний контекст тексту конкатенується (об'єднується) з числовим значенням надійності. Цей об'єднаний вектор потім подається на вхід повнозв'язних шарів класифікатора. Таким чином, модель отримує можливість навчатися взаємозв'язкам між текстовими ознаками та показником надійності джерела, зважуючи їхній вплив на фінальне рішення.

Хоча під час навчання моделі вхід надійності був константою (1.0) для всіх прикладів, щоб не вносити залежність від ще не навченої моделі, вплив динамічно оновлюваної надійності можна спостерігати під час практичного використання моделі (наприклад, у тестових викликах або при інтеграції з ботом).

У тестових викликах, наведених у блоці if __name__ == "__main__": скрипту ipso_model.py (див. Додаток А) було продемонстровано роботу механізму оновлення надійності.

При першому надходженні повідомлення від нового джерела (рисунок 3.4), яке класифікувалося як ІПСО, надійність джерела знижувалася з початкових 1.0 до 0.98 (при RELIABILITY_DECREMENT = 0.02).

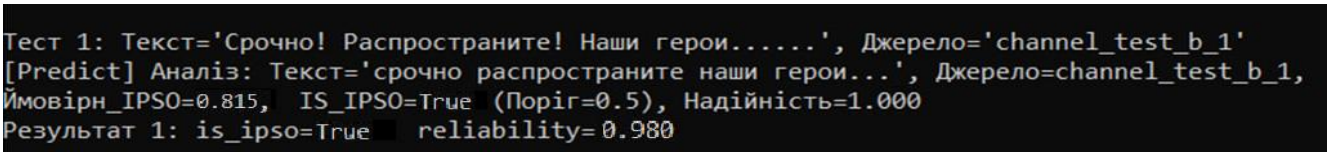

Рисунок 3.4 - Результат класифікації нейронної мережі першого тестового повідомлення від нового джерела "channel_test_b_1"

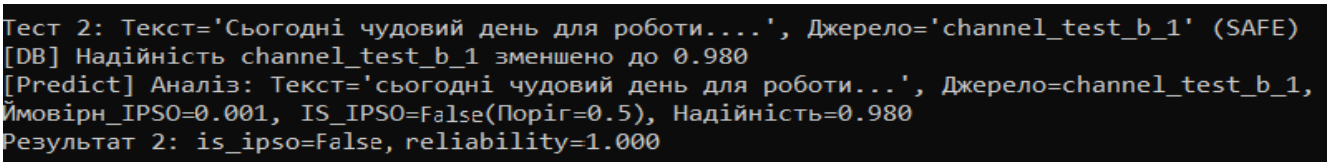

Рисунок 3.5 - Результат класифікації нейронної мережі другого тестового повідомлення від нового джерела "channel_test_b_1"

При наступному повідомленні від цього ж джерела, яке класифікувалося як безпечне (рисунок 3.5), надійність збільшувалася на RELIABILITY_INCREMENT (0.01).

Важливим аспектом реалізованого механізму управління надійністю є асиметричний підхід до її оновлення. Крок зменшення надійності (RELIABILITY_DECREMENT), який застосовується при виявленні ІПСО-повідомлення, встановлено на 0.02, тоді як крок збільшення надійності (RELIABILITY_INCREMENT) при отриманні безпечного контенту становить 0.01. Таке співвідношення обрано свідомо, виходячи з припущення, що довіру до джерела інформації складніше заслужити та відновити, ніж втратити. Таким чином, система більш чутливо реагує на поширення потенційно шкідливого контенту, вимагаючи від джерела більшої кількості послідовних безпечних публікацій для відновлення початкового рівня надійності.

Для практичної оцінки впливу показника надійності джерела на результат класифікації було проведено цілеспрямований експеримент. За допомогою зовнішнього інструменту для роботи з базами даних SQLite (DB Browser for SQLite) було створено два тестових джерела з різними рівнями надійності. Першому джерелу було присвоєно ідентифікатор «77777777» та штучно встановлено значення надійності на рівні 0.50. Для другого джерела з ідентифікатором «88888888» було залишено початкове значення надійності за замовчуванням – 1.00. Стан бази даних з цими тестовими джерелами перед експериментом наведено на рисунку 3.6.

Після цього на аналіз моделі було подано одне й те саме текстове повідомлення, яке за своїм змістом могло бути інтерпретоване неоднозначно (наприклад, текст, що знаходиться на межі між безпечним контентом та стриманою інформаційною операцією). Це повідомлення було послідовно "переслано" від обох тестових джерел.

Table: sources

	source_id	reliability	last_updated
	Filter	Filter	Filter
1	channel_test_b_1	0.98	2025-05-14 19:55:24
2	88888888	0.50	2025-05-16 18:29:55
3	77777777	1.0	2025-05-16 18:25:49

Рисунок 3.6 - Фрагмент бази даних ipso_database_binary.db з тестовими джерелами «77777777» (надійність 0.50) та «88888888» (надійність 1.0) перед експериментом

Результат класифікації повідомлення, поданого від джерела «77777777» зі зниженою надійністю (0.50), показано на рисунку 3.7.

```
Введіть текст повідомлення: Після вибуху на блокпосту поранених залишили без евакуації. Медикаментів бракує, зв'язок н
естабільний. Поширте, люди повинні знати правду.
Введіть ID джерела (залиште порожнім для прямого повідомлення): 77777777
Аналіз...
[Predict] Аналіз: Текст='після вибуху на блокпосту поранених залишили без е...', Джерело=77777777, Ймовірн_IPSO=0.480,
 IS_IPSO=False(Поріг=0.5), Надійність=1.000

--- Результат аналізу ---
Текст: 'Після вибуху на блокпосту поранених залишили без евакуації. Медикаментів бракує, зв'язок нестабільни...'
Джерело: 77777777
Є ІПСО (за порогом 0.5): Ні
Актуальна надійність джерела: 1.00
```

Рисунок 3.7 - Результат класифікації тестового повідомлення від джерела «77777777» (початкова надійність 0.50).

Як видно з рисунку, модель класифікувала повідомлення як SAFE_CONTENT з ймовірністю 0,480.

Потім те ж саме текстове повідомлення було подано на аналіз від джерела 88888888 з початковою максимальною надійністю (1.00). Результат наведено на рисунку 3.8.

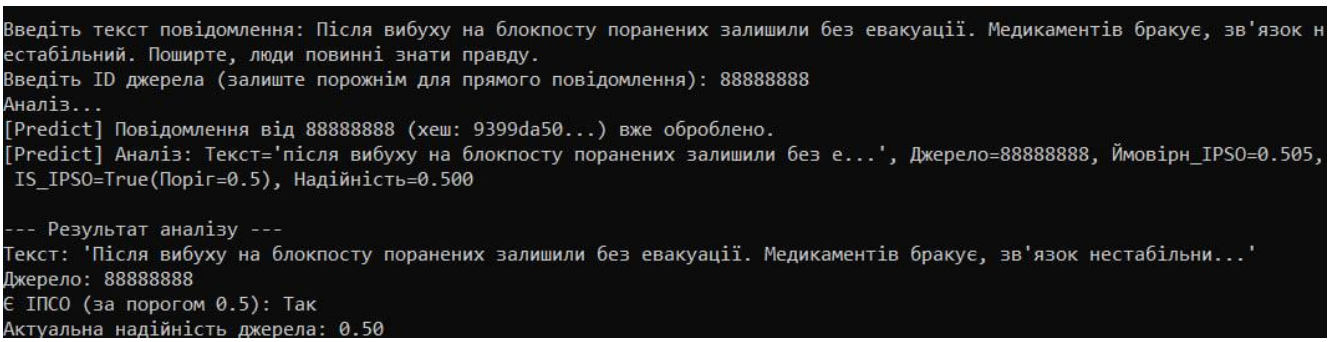

Рисунок 3.8 - Результат класифікації того ж тестового повідомлення від джерела 88888888 (початкова надійність 1.00).

У цьому випадку модель класифікувала повідомлення як IPSO з ймовірністю 0.505.

Порівнюючи результати, отримані на рисунку 3.7 та рисунку 3.8, можна зробити висновок, що показник надійності джерела мав помірний вплив на кінцеву ймовірність, видану моделлю. При нижчій надійності джерела (0.50) ймовірність класифікації як ІПСО була вищою (0.505.), ніж при високій надійності джерела (1.00), де ймовірність ІПСО склала 0.480. Показник надійності джерела мав вирішальний вплив на класифікацію тестового повідомлення, оскільки для джерела зі штучно зниженою надійністю тестове повідомлення було класифіковано як IPSO, у той час як для джерела із високою надійністю це повідомлення було розпізнано як SAFE_CONTENT. Цей експеримент якісно демонструє, що модель враховує переданий показник надійності, і він може модулювати її рішення, особливо для текстів, які не мають однозначно виражених ознак ІПСО або безпечного контенту.

Спостереження на тестових прикладах демонструють, що механізм динамічного оновлення надійності функціонує, а сама надійність як вхідна ознака потенційно може впливати на кінцеве рішення класифікатора, особливо для повідомлень з неоднозначним текстовим контентом. Подальші дослідження могли б бути спрямовані на більш глибокий кількісний аналіз цього впливу.

3.7 Аналіз помилок моделі: приклади хибних спрацювань

Незважаючи на досягнутий рівень загальної точності, будь-яка модель машинного навчання припускається помилок. Аналіз цих помилок є важливим для розуміння слабких сторін моделі, виявлення типів повідомлень, які їй важко класифікувати, та визначення напрямків для подальшого покращення. У контексті бінарної класифікації ІПСО основними типами помилок є хибно позитивні (False Positives, FP) та хибно негативні (False Negatives, FN) спрацювання.

Хибно позитивне спрацювання відбувається тоді, коли модель класифікує безпечне повідомлення (SAFE_CONTENT) як ІПСО (IPSO). Згідно з матрицею помилок (Таблиця 3.1), у валідаційній вибірці було виявлено 175 таких випадків.

Приклад 1 (Реальний випадок хибного спрацювання):

– Вхідний текст: "У Центрі культури й дозвілля презентували виставку фотографій, присвячену європейським містам-партнерам. Експозиція триватиме до кінця травня."

– Істинна мітка (припускається): SAFE_CONTENT.

– Передбачення моделі (рисунок 3.9): IPSO (Ймовірність ІПСО = 0.837).

```
Введіть текст повідомлення: У Центрі культури й дозвілля презентували виставку фотографій, присвячену європейським міс
там-партнерам. Експозиція триватиме до кінця травня.
Введіть ID джерела (залиште порожнім для прямого повідомлення): channel1
Аналіз...
[DB] Надійність channel1 зменшено до 0.980
[Predict] Аналіз: Текст='у центрі культури й дозвілля презентували виставку...', Джерело=channel1, Ймовірн_IPSO=0.837,
 IS_IPSO=True(Поріг=0.5), Надійність=0.980

--- Результат аналізу ---
Текст: 'У Центрі культури й дозвілля презентували виставку фотографій, присвячену європейським містам-партне...'
Джерело: channel1
Є ІПСО (за порогом 0.5): Так
Актуальна надійність джерела: 0.98
```

Рисунок 3.9 - Результат помилкової класифікації безпечного повідомлення як ІПСО.

Незважаючи на те, що повідомлення має цілком нейтральний та інформативний характер, пов'язаний з культурною подією, модель з високою ймовірністю (0.837) віднесла його до ІПСО. Можливими причинами такого хибного спрацювання можуть бути:

– Недостатня кількість подібних "культурно-новинних" прикладів у класі SAFE_CONTENT під час навчання.

Якщо навчальні дані для безпечного контенту були переважно дуже короткими, побутовими або не містили специфічної лексики, пов'язаної з офіційними подіями чи культурними заходами, модель могла не навчитися коректно їх ідентифікувати.

– Вплив "шуму" з навчальних даних ІПСО

Можливо, деякі ІПСО-повідомлення в навчальній вибірці мали схожу формальну структуру або використовували подібні ключові слова (наприклад, "презентація нової ідеології", "культурний фронт"), що призвело до того, що модель навчилася асоціювати такі патерни з ІПСО.

– Особливості роботи шару Embedding

Якщо векторні представлення для слів з цього повідомлення виявилися ближчими до векторів слів, характерних для ІПСО в навчальному наборі, це могло вплинути на рішення.

Цей приклад підкреслює важливість різноманітності та репрезентативності навчальних даних для обох класів. Для зменшення кількості подібних хибно позитивних спрацювань необхідно збагачувати навчальну вибірку для класу SAFE_CONTENT прикладами текстів різної тематики та стилістики, включаючи офіційно-діловий стиль, новини культури, науки тощо.

Хибно негативне спрацювання відбувається тоді, коли модель класифікує повідомлення, що містить ознаки ІПСО, як безпечне (SAFE_CONTENT). Це більш критичний тип помилки для системи виявлення ІПСО. Згідно з матрицею помилок (Таблиця 3.1), у валідаційній вибірці було 1199 таких випадків.

Приклад 2 (Реальний випадок хибного спрацювання):

Для ілюстрації хибно негативного спрацювання розглянемо реальний приклад роботи моделі (рисунок 3.8). На аналіз було подано наступне текстове повідомлення: "Після вибуху на блокпосту поранених залишили без евакуації. Медикаментів бракує, зв'язок нестабільний. Поширте, люди повинні знати правду." Це повідомлення містить ознаки, характерні для ІПСО, зокрема, нагнітання паніки,

повідомлення про критичну ситуацію без надання конкретних деталей або джерел, а також прямий заклик до поширення інформації. Його істинна мітка (припускається) – IPSO.

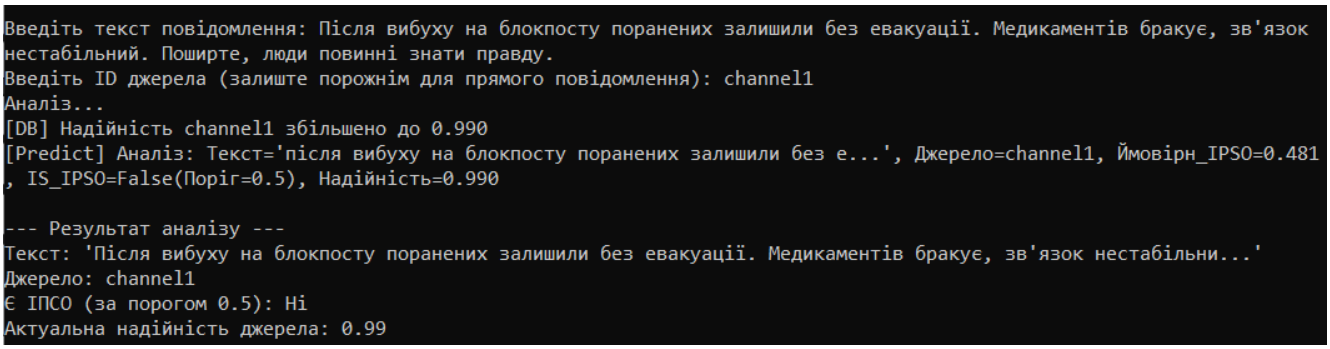

Рисунок 3.10 - Результат помилкової класифікації ІПСО-повідомлення як безпечного.

Як видно з протоколу роботи (Рис. 3.10), модель класифікувала це потенційно шкідливе повідомлення як безпечне (IS_IPSO=False). Ймовірність того, що це ІПСО, була оцінена лише в 0.481, що нижче за встановлений поріг 0.5.

Таке хибно негативне спрацювання може бути зумовлене кількома факторами:

– Відсутність яскраво виражених агресивних маркерів

Хоча повідомлення містить тривожні елементи та заклик "Поширте", воно може не містити тієї ж концентрації агресивної лексики або прямих образ, на які модель могла бути "натренована" для ідентифікації ІПСО.

– Схожість з реальними новинами про надзвичайні ситуації

Модель могла сплутати це повідомлення з легітимними новинами про інциденти, де також можуть описуватися негативні події та проблеми. Якщо навчальна вибірка SAFE_CONTENT містила багато новинних зведень (навіть з негативним забарвленням, але правдивих), модель могла навчитися вважати такий стиль менш підозрілим.

– Недостатня вага для певних патернів

Можливо, комбінація таких елементів, як "вибух", "поранені", "без евакуації", "медикаментів бракує", "зв'язок нестабільний", "поширте", "знати правду", не отримала достатньої ваги під час навчання як ознака ІПСО.

– Вплив надійності джерела (якщо вона була високою)

Якщо джерело "channel1" до цього моменту мало високий показник надійності, це могло незначно "зменшити" фінальну ймовірність ІПСО, навіть якщо текстові ознаки були досить сильними. (Хоча на скріншоті видно, що надійність до цього моменту була 0.980, що вже свідчить про попередні ІПСО).

– Неоптимальний поріг

Можливо, стандартний поріг 0.5 є занадто високим для виявлення подібних, менш очевидних ІПСО, і нижчий поріг міг би допомогти їх ідентифікувати, хоча це й збільшило б кількість хибно позитивних спрацювань.

Аналіз хибних спрацювань показує, що модель, хоча й досягла високої загальної точності, все ще має труднощі з розпізнаванням. Для покращення здатності моделі виявляти такі випадки необхідно збагачувати навчальну вибірку для класу IPSO різноманітними прикладами завуальованих маніпуляцій та панічних повідомлень.

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно розв'язано актуальну науково-технічну задачу автоматичного виявлення інформаційно-психологічних операцій в текстових даних інтернет-простору. Метою роботи, що полягала у розробці, реалізації та експериментальному дослідженні ефективності рекурентної нейронної мережі для бінарної класифікації ІПСО з урахуванням динамічної надійності джерела, було досягнуто.

У ході дослідження було проведено детальний аналіз предметної області, включаючи особливості ІПСО та існуючі підходи до їх виявлення, що обґрунтувало вибір рекурентних нейронних мереж, зокрема архітектури LSTM, для вирішення поставленої задачі. Було розроблено та реалізовано комплексну методику підготовки даних, яка включала консолідацію вихідних файлів, перетворення багатокласових міток на бінарні ("IPSO" / "SAFE_CONTENT"), очищення тексту, токенізацію зі словником на 30000 слів та векторизацію. На основі бібліотек Keras/TensorFlow було спроектовано та реалізовано модель РНМ на базі LSTM, що інтегрує текстові дані та показник надійності джерела. Ключовою особливістю розробки став механізм динамічного управління надійністю джерела, що зберігається в базі даних SQLite, який передбачає зниження надійності при виявленні ІПСО, підвищення для безпечного контенту та ігнорування дублікатів повідомлень.

Процес навчання розробленої моделі проводився на підготовленому наборі даних із застосуванням техніки зважування класів для боротьби з дисбалансом та механізмів ранньої зупинки. Експериментальна оцінка якості на валідаційній вибірці, що включала аналіз матриці помилок та звіту класифікації (Precision, Recall, F1-score), продемонструвала загальну точність моделі на рівні 97%. Модель показала високу ефективність у виявленні ІПСО-повідомлень, хоча й мала нижчу точність для класу безпечного контенту, що частково пояснюється вихідним дисбалансом даних. Для забезпечення взаємодії з зовнішніми системами було розроблено програмний інтерфейс у вигляді функції predict_single_message.

Таким чином, в результаті виконання кваліфікаційної роботи було запропоновано та реалізовано архітектуру РНМ для бінарної класифікації ІПСО, розроблено модифікований алгоритм управління надійністю джерела, експериментально підтверджено працездатність підходу та створено функціональний прототип системи. Практична цінність роботи полягає у можливості використання розробленої моделі як ядра для інструментів моніторингу інформаційного простору, що підтверджується її потенційною інтеграцією в Telegram-бот, розроблений у рамках суміжного проекту.

Незважаючи на досягнуті результати, існують напрямки для подальших досліджень, зокрема, більш глибока робота з дисбалансом класів, тонке налаштування порогу класифікації, експерименти з більш складними архітектурами нейронних мереж та розширення системи для аналізу мультимодального контенту. Загалом, виконана робота демонструє ефективність застосування рекурентних нейронних мереж для задачі виявлення ІПСО та закладає основу для створення практично значущих інструментів протидії інформаційним загрозам.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hochreiter S., Schmidhuber J. Long short-term memory [Електронний ресурс] / Neural Computation. – 1997. – Vol. 9, No. 8. – P. 1735–1780. – Режим доступу: https://doi.org/10.1162/neco.1997.9.8.1735 (дата звернення: 10.11.24).

2. Zhang Y., Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification [Електронний ресурс] / Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers). – 2017. – P. 253–263. – Режим доступу: https://aclanthology.org/I17-1026/ (дата звернення: 18.01.25).

3. Przybyla P. Capturing the style of fake news [Електронний ресурс] / Proceedings of the AAAI Conference on Artificial Intelligence. – 2020. – Vol. 34, No. 01. – P. 490–497. – Режим доступу: https://doi.org/10.1609/aaai.v34i01.5397 (дата звернення: 16.01.24).

4. Shu K., Sliva A., Wang S., Tang J., Liu H. Fake news detection on social media: A data mining perspective [Електронний ресурс] / SIGKDD Explorations. – 2017. – Vol. 19, No. 1. – P. 22–36. – Режим доступу: https://doi.org/10.1145/3137597.3137600 (дата звернення: 18.01.25).

5. Kim J., Tabibian B., Oh A., Schölkopf B., Gomez-Rodriguez M. Leveraging the crowd to detect and reduce the spread of fake news and misinformation [Електронний ресурс] / Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. – 2018. – P. 324–332. – Режим доступу: https://doi.org/10.1145/3159652.3159731 (дата звернення: 18.01.25).

6. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation [Електронний ресурс] / Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2014. – P. 1532–1543. – Режим доступу: https://aclanthology.org/D14-1162 (дата звернення: 25.02.25).

7. Johnson J. M., Khoshgoftaar T. M. Survey on deep learning with class imbalance [Електронний ресурс] / Journal of Big Data. – 2019. – Vol. 6, No. 1. – Article

27. – Режим доступу: https://doi.org/10.1186/s40537-019-0192-5 (дата звернення: 25.02.25.

8. Krawczyk B. Learning from imbalanced data: open challenges and future directions [Електронний ресурс] / Progress in Artificial Intelligence. – 2016. – Vol. 5, No. 4. – P. 221–232. – Режим доступу: https://link.springer.com/article/10.1007/s13748-016-0094-0 (дата звернення: 16.04.25).

9. Abadi M., Barham P., Chen J. et al. TensorFlow: A system for large-scale machine learning [Електронний ресурс] / Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). – 2016. – P. 265–278. – Режим доступу: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf (дата звернення: 25.02.25).

10. Chollet F. et al. Keras Documentation [Електронний ресурс]. – Режим доступу: https://keras.io (дата звернення: 25.02.25).

11. Mohammad S. M., Kiritchenko S., Zhu X. The Future of Natural Language Processing: A Survey of Recent Advances and Emerging Trends [Електронний ресурс] / SSRN Electronic Journal. – 2023. – Режим доступу: https://doi.org/10.2139/ssrn.4539657 (дата звернення: 16.04.25).

12. Азарова А.О., Пугач В.С. Інформаційна технологія виявлення фейкової інформації у соцмережах [Електронний ресурс] / Вінницький національний технічний університет. – 2020. – Режим доступу: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/43985/138732.pdf?sequence=2&isAllowed=y (дата звернення: 18.01.25).

13. Барковська О., Воропаєва К., Руських О. Обґрунтування вибору нейромережного лінгвістичного класифікатора [Електронний ресурс] / Сучасний стан наукових досліджень та технологій в промисловості. – 2023. – Том 1, № 1. – С. 5–14. – Режим доступу: https://itssi-journal.com/index.php/ittsi/article/view/415 (дата звернення: 25.02.25).

14. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] / arXiv. – 2013. –

arXiv:1301.3781. – Режим доступу: https://arxiv.org/abs/1301.3781 (дата звернення: 16.04.25).

15. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information [Електронний ресурс] / Transactions of the Association for Computational Linguistics. – 2017. – Vol. 5. – P. 135–146. – Режим доступу: https://aclanthology.org/Q17-1010 (дата звернення: 25.02.25).

16. Chollet F. Deep Learning with Python [Електронний ресурс] / Manning Publications. – 2018. – Режим доступу: https://www.manning.com/books/deep-learning-with-python (дата звернення: 25.02.25).

17. Pandas 2.1.2 Documentation [Електронний ресурс] / The Pandas Development Team. – 2023. – Режим доступу: https://pandas.pydata.org/docs/ (дата звернення: 25.02.25).

18. NumPy Documentation [Електронний ресурс] / The NumPy community. – 2023. – Режим доступу: https://numpy.org/doc/ (дата звернення: ДД.ММ.РРРР).

19. Scikit-learn 1.5.0 documentation [Електронний ресурс] / The Scikit-learn developers. – 2024. – Режим доступу: https://scikit-learn.org/stable/ (дата звернення: 25.02.25).

20. SQLite Documentation [Електронний ресурс] / D. Richard Hipp et al. – Режим доступу: https://www.sqlite.org/docs.html (дата звернення: 25.02.25).

21. Matplotlib 3.10.1 documentation [Електронний ресурс] / The Matplotlib development team. – 2023. – Режим доступу: https://matplotlib.org/stable/index.html (дата звернення: 25.02.25).

22. NLTK Documentation [Електронний ресурс] / NLTK Team. – Режим доступу: https://www.nltk.org/ (дата звернення: 25.02.25).

Додаток А. Лістинг програмного коду модуля аналізу ІПСО (ipso_model.py)

```
import os
import sqlite3
import hashlib
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Embedding, LSTM, Dense, Dropout, Concatenate,
BatchNormalization
)
from tensorflow.keras.preprocessing.text import Tokenizer,
tokenizer_from_json
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import re
import nltk
import pickle
import matplotlib.pyplot as plt
import sys
from sklearn.utils import class_weight
from sklearn.metrics import classification_report, confusion_matrix

# --- Конфігурація ---
DB_FILE = 'ipso_database_binary.db'
MESSAGES_PATH = 'data/messages/messages.jsonl'
TRAIN_CSV_PATH = 'data/train.csv'
TEST_CSV_PATH = 'data/test.csv'
MODEL_WEIGHTS_DIR = 'model_weights_binary'
MODEL_WEIGHTS_FILE = os.path.join(MODEL_WEIGHTS_DIR,
'ipso_rnn_binary_lstm_30k.weights.h5')
TOKENIZER_FILE = 'tokenizer_binary_30k.json'
LABEL_ENCODER_FILE = 'label_encoder_binary.pkl'

# Параметри надійності
INITIAL_RELIABILITY = 1.0
RELIABILITY_DECREMENT = 0.02
RELIABILITY_INCREMENT = 0.01
MIN_RELIABILITY = 0.1

# Параметри моделі
VOCAB_SIZE = 30000
MAX_LENGTH = 200
EMBEDDING_DIM = 128
LSTM_UNITS = 64
DROPOUT_RATE = 0.4
EPOCHS = 15
```

```
BATCH_SIZE = 64

# Бінарні класи
SAFE_CONTENT_LABEL = "SAFE_CONTENT"
IPSO_LABEL = "IPSO"
BINARY_LABELS = [SAFE_CONTENT_LABEL, IPSO_LABEL]

# --- Функції для роботи з БД---
def init_db(db_path):
    """Ініціалізує БД, створює таблиці sources та processed_messages."""
    db_dir = os.path.dirname(db_path)
    if db_dir and not os.path.exists(db_dir): os.makedirs(db_dir)
    conn = sqlite3.connect(db_path, check_same_thread=False); cursor = conn.cursor()
    try:
        # Таблиця джерел
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS sources (
                source_id TEXT PRIMARY KEY,
                reliability REAL NOT NULL,
                last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        ''')
        cursor.execute('CREATE INDEX IF NOT EXISTS idx_source_id ON sources(source_id)')

        # Таблиця для оброблених повідомлень
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS processed_messages (
                message_hash TEXT PRIMARY KEY,
                processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        ''')

        conn.commit(); print(f"БД '{db_path}' ініціалізована.")
    except sqlite3.Error as e: print(f"Помилка ініціалізації БД: {e}"); conn.close(); return None
    return conn

def get_source_reliability(conn, source_id):
    if source_id is None: return INITIAL_RELIABILITY
    if conn is None: return INITIAL_RELIABILITY
    try:
        cursor = conn.cursor(); cursor.execute('SELECT reliability FROM sources WHERE source_id = ?', (str(source_id),)); result = cursor.fetchone()
        if result: return result[0]
        else: cursor.execute('INSERT OR IGNORE INTO sources (source_id, reliability) VALUES (?, ?)', (str(source_id), INITIAL_RELIABILITY)); conn.commit(); # print(f"[DB] Додано/знайдено джерело {source_id}..."); return INITIAL_RELIABILITY
```

```
    except sqlite3.Error as e: print(f"[DB] Помилка (get): {e}");
return INITIAL_RELIABILITY
    return INITIAL_RELIABILITY # Повертаємо на випадок помилки

def update_source_reliability(conn, source_id, is_ipso):
    if source_id is None: return None
    if conn is None: return None
    try:
        cursor = conn.cursor(); cursor.execute('SELECT reliability
FROM sources WHERE source_id = ?', (str(source_id),)); result =
cursor.fetchone()
        current_reliability = result[0] if result else
INITIAL_RELIABILITY
        new_reliability = current_reliability
        if is_ipso: new_reliability = max(MIN_RELIABILITY,
current_reliability - RELIABILITY_DECREMENT); change_type = "зменшено"
        else: new_reliability = min(INITIAL_RELIABILITY,
current_reliability + RELIABILITY_INCREMENT); change_type =
"збільшено"
        if abs(new_reliability - current_reliability) > 1e-9:
            cursor.execute('UPDATE sources SET reliability = ?,
last_updated = CURRENT_TIMESTAMP WHERE source_id = ?',
(new_reliability, str(source_id))); conn.commit()
            print(f"[DB] Надійність {source_id} {change_type} до
{new_reliability:.3f}")
        return new_reliability
    except sqlite3.Error as e: print(f"[DB] Помилка (update): {e}");
return None


# --- Функції для обробки тексту---
def clean_text(text):
    if not isinstance(text, str): return ""
    text = text.lower()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text,
flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#\w+', '', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s\-_]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# --- Завантаження та підготовка даних ---
def load_and_preprocess_data(messages_path, train_path, test_path,
binary_labels_list, vocab_size_limit):
    print("Завантаження даних...")
    try:
        with open(messages_path, 'r', encoding='utf-8') as f:
messages_data = [json.loads(line) for line in f if line.strip()]
        print(f"Завантажено {len(messages_data)} повідомлень.")
    except Exception as e: print(f"ПОМИЛКА читання {messages_path}:
{e}"); return None, None, None, None, None, None
    df_messages = pd.DataFrame(messages_data)
```

```
    required_msg_cols = ['text', 'source_id', 'message_id',
'published_at']
    if not all(col in df_messages.columns for col in
required_msg_cols): print(f"ПОМИЛКА: відсутні колонки в
{messages_path}"); return None, None, None, None, None, None
    df_messages['published_at'] =
pd.to_datetime(df_messages['published_at'], unit='s',
errors='coerce'); df_messages = df_messages.dropna(subset=['text',
'source_id', 'message_id']); print(f"Повідомлень після dropna:
{len(df_messages)}")
    try:
        df_train_sources = pd.read_csv(train_path); df_test_sources =
pd.read_csv(test_path)
        print(f"Завантажено {len(df_train_sources)} трен. і
{len(df_test_sources)} тест. джерел.")
    except Exception as e: print(f"ПОМИЛКА читання CSV: {e}"); return
None, None, None, None, None, None
    if 'source_id' not in df_train_sources.columns or
'source_category' not in df_train_sources.columns: print(f"ПОМИЛКА: У
{train_path} відсутні колонки."); return None, None, None, None, None,
None
    if 'source_id' not in df_test_sources.columns: print(f"ПОМИЛКА: У
{test_path} відсутня 'source_id'."); return None, None, None, None,
None, None
    df_sources = pd.concat([df_train_sources, df_test_sources],
ignore_index=True).drop_duplicates(subset=['source_id'])
    df_sources['label'] = df_sources['source_category'].apply(lambda
x: SAFE_CONTENT_LABEL if x == SAFE_CONTENT_LABEL else (IPSO_LABEL if
pd.notna(x) else None))
    print("Об'єднання даних...")
    df = pd.merge(df_messages, df_sources[['source_id', 'label']],
on='source_id', how='inner')
    df = df.dropna(subset=['label'])
    print(f"Повідомлень після об'єднання/dropna(label): {len(df)}")
    if df.empty: print("ПОМИЛКА: Дані порожні."); return None, None,
None, None, None, None
    print("Очищення тексту...")
    df['cleaned_text'] = df['text'].apply(clean_text)
    df = df.dropna(subset=['cleaned_text'])
    df = df[df['cleaned_text'].str.len() > 0]
    print(f"Після очищення тексту: {len(df)}")
    if df.empty: print("ПОМИЛКА: Дані порожні після очищення.");
return None, None, None, None, None, None
    print("\nРозподіл бінарних класів у даних:");
print(df['label'].value_counts()); print("-" * 30)
    print("Кодування міток...")
    label_encoder = LabelEncoder();
label_encoder.fit(binary_labels_list)
    y = label_encoder.transform(df['label'])
    print(f"Мітки закодовано: {label_encoder.classes_} ->
{label_encoder.transform(label_encoder.classes_)}")
    print("Токенізація...")
```

```
    tokenizer = Tokenizer(num_words=vocab_size_limit,
oov_token="<OOV>")
    tokenizer.fit_on_texts(df['cleaned_text'])
    X_seq = tokenizer.texts_to_sequences(df['cleaned_text'])
    effective_vocab_size = tokenizer.num_words if tokenizer.num_words
else (len(tokenizer.word_index) + 1)
    print(f"Ефективний розмір словника: {effective_vocab_size}")
    print("Паддинг..."); X_pad = pad_sequences(X_seq,
maxlen=MAX_LENGTH, padding='post', truncating='post')
    print("Розділення даних...")
    X_train, X_val, y_train, y_val = train_test_split(X_pad, y,
test_size=0.2, random_state=42, stratify=y)
    print("-" * 30); print(f"Підготовка даних завершена:
Навч={len(X_train)}, Валід={len(X_val)}"); print("-" * 30)
    return X_train, X_val, y_train, y_val, tokenizer, label_encoder

# --- Побудова моделі ---
def build_ipso_rnn_model(vocab_size, embedding_dim, max_length,
lstm_units=LSTM_UNITS, dropout_rate=DROPOUT_RATE):
    text_input=Input(shape=(max_length,), name='text_input');
reliability_input=Input(shape=(1,), name='reliability_input')
    embedding_layer=Embedding(input_dim=vocab_size,
output_dim=embedding_dim, name='embedding')(text_input)
    lstm_layer=LSTM(lstm_units, dropout=dropout_rate,
recurrent_dropout=dropout_rate, name='lstm')(embedding_layer)
    concatenated=Concatenate(name='concatenate')([lstm_layer,
reliability_input])
    x=Dense(128, activation='relu', name='dense_1')(concatenated);
x=BatchNormalization()(x); x=Dropout(dropout_rate)(x)
    x=Dense(64, activation='relu', name='dense_2')(x);
x=BatchNormalization()(x); x=Dropout(dropout_rate)(x)
    output=Dense(1, activation='sigmoid', name='output')(x)
    model=Model(inputs=[text_input, reliability_input],
outputs=output, name='IPSO_RNN_LSTM_Binary_Model')
    optimizer=tf.keras.optimizers.Adam();
model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    print("Модель LSTM для бінарної класифікації побудована:")
    model.summary(line_length=100)
    return model

# --- Навчання моделі ---
def train_ipso_model(model, X_train, y_train, X_val, y_val,
epochs=EPOCHS, batch_size=BATCH_SIZE,
model_weights_file=MODEL_WEIGHTS_FILE, class_weights=None):
    print("\n--- Початок навчання моделі ---")
    train_reliability=np.full((X_train.shape[0], 1),
INITIAL_RELIABILITY, dtype='float32');
val_reliability=np.full((X_val.shape[0], 1), INITIAL_RELIABILITY,
dtype='float32')
    if not os.path.exists(MODEL_WEIGHTS_DIR): print(f"Створення
директорії: {MODEL_WEIGHTS_DIR}"); os.makedirs(MODEL_WEIGHTS_DIR)
```

```
    early_stopping=EarlyStopping(monitor='val_loss',      patience=3,
verbose=1, restore_best_weights=True)
    model_checkpoint=ModelCheckpoint(model_weights_file,
monitor='val_loss',   save_best_only=True,   save_weights_only=True,
verbose=1)
    print(f"Параметри: епохи={epochs}, батч={batch_size}")
    if   class_weights:   print(f"Використовуються   ваги   класів:
{class_weights}")
    else: print("Зважування класів не використовується.")
    print(f"Збереження ваг у: {model_weights_file}"); print("-" * 30)
    history   =   model.fit([X_train,   train_reliability],   y_train,
epochs=epochs,    batch_size=batch_size,    validation_data=([X_val,
val_reliability],         y_val),         callbacks=[early_stopping,
model_checkpoint], class_weight=class_weights, verbose=1)
    print("-" * 30); print(f"Навчання завершено.")
    if  os.path.exists(model_weights_file):  print(f"Найкращі  ваги
збережено: {model_weights_file}")
    else: print(f"УВАГА: Файл ваг {model_weights_file} не збережено.")
    return model, history

# --- Функція передбачення---
def predict_single_message(model, tokenizer, label_encoder, db_conn,
text, source_id, max_length=MAX_LENGTH, threshold=0.5):
    if  not  text  or  not  isinstance(text,  str):  print("[Predict]
Помилка:  Порожній  текст.");  r  =  get_source_reliability(db_conn,
source_id); return False, r
    cleaned_text = clean_text(text)
    if  not  cleaned_text:  print("[Predict]  Помилка:  Текст  порожній
після  очищення.");  r  =  get_source_reliability(db_conn,  source_id);
return False, r
    current_reliability = get_source_reliability(db_conn, source_id);
message_processed = False; message_hash = None
    if source_id and db_conn:
        try:
            hash_input     =     f"{str(source_id)}::{cleaned_text}";
message_hash = hashlib.sha256(hash_input.encode('utf-8')).hexdigest()
            cursor = db_conn.cursor(); cursor.execute("SELECT 1 FROM
processed_messages WHERE message_hash = ?", (message_hash,)); result
= cursor.fetchone()
            if result: message_processed = True; print(f"[Predict]
Повідомлення  від  {source_id}  (хеш:  {message_hash[:8]}...)  вже
оброблено.")
        except Exception as e_hash_check: print(f"[Predict] Помилка
перевірки хешу: {e_hash_check}")
    try:  sequence  =  tokenizer.texts_to_sequences([cleaned_text]);
padded_sequence    =    pad_sequences(sequence,    maxlen=max_length,
padding='post', truncating='post')
    except Exception as e_tok: print(f"[Predict] Помилка токенізації:
{e_tok}"); return False, current_reliability
    reliability_input       =       np.array([[current_reliability]],
dtype='float32'); is_ipso = False; confidence = 0.0
    try:
```

```
        if padded_sequence.shape != (1, max_length): raise
ValueError(f"Форма тексту: {padded_sequence.shape}")
        if reliability_input.shape != (1, 1): raise ValueError(f"Форма
надійності: {reliability_input.shape}")
        prediction_prob_ipso = model.predict([padded_sequence,
reliability_input], verbose=0)[0][0]
        confidence = prediction_prob_ipso; is_ipso =
prediction_prob_ipso >= threshold
    except Exception as e_pred: print(f"[Predict] Помилка
передбачення: {e_pred}"); return False, current_reliability
    final_reliability = current_reliability
    if not message_processed and source_id is not None:
        updated_reliability_val = update_source_reliability(db_conn,
source_id, is_ipso)
        final_reliability = updated_reliability_val if
updated_reliability_val is not None else current_reliability
        if message_hash and db_conn:
            try: cursor = db_conn.cursor(); cursor.execute("INSERT OR
IGNORE INTO processed_messages (message_hash) VALUES (?)",
(message_hash,)); db_conn.commit()
            except Exception as e_hash_insert: print(f"[Predict]
Помилка збереження хешу: {e_hash_insert}")
    elif message_processed: final_reliability = current_reliability
    print(f"[Predict] Аналіз: Текст='{cleaned_text[:50]}...',
Джерело={source_id}, Ймовірн_IPSO={confidence:.3f},
IS_IPSO={is_ipso}(Поріг={threshold}),
Надійність={final_reliability:.3f}")
    return is_ipso, final_reliability

# # --- Функція побудови графіків---
def plot_training_history(history, model_name="Model"):
    if history is None or not hasattr(history, 'history') or not
history.history: print("Неможливо побудувати графіки."); return
    history_dict=history.history; acc=history_dict.get('accuracy');
val_acc=history_dict.get('val_accuracy');
loss=history_dict.get('loss'); val_loss=history_dict.get('val_loss')
    required_keys = ['accuracy', 'val_accuracy', 'loss', 'val_loss'];
    if not all(key in history_dict for key in required_keys):
print(f"Відсутні метрики: {[k for k in required_keys if k not in
history_dict]}."); return
    if not acc or not val_acc or not loss or not val_loss:
print("Списки метрик порожні."); return
    epochs_range = range(1, len(acc)+1); plt.figure(figsize=(14,6))
    plt.subplot(1,2,1); plt.plot(epochs_range, loss, 'bo-',
label='Втрати Навчання'); plt.plot(epochs_range, val_loss, 'ro-',
label='Втрати Валідації'); plt.title(f'{model_name} - Втрати');
plt.xlabel('Епохи'); plt.ylabel('Втрати'); plt.legend();
plt.grid(True, linestyle='--', alpha=0.6)
    plt.subplot(1,2,2); plt.plot(epochs_range, acc, 'bo-',
label='Точність Навчання'); plt.plot(epochs_range, val_acc, 'ro-',
label='Точність Валідації'); plt.title(f'{model_name} - Точність');
plt.xlabel('Епохи'); plt.ylabel('Точність'); plt.legend();
plt.grid(True, linestyle='--', alpha=0.6)
```

```
    plt.suptitle(f'Історія навчання: {model_name}', fontsize=14); plt.tight_layout(rect=[0,0.03,1,0.95])
    try: plot_filename = f"{model_name.replace(' ','_')}_binary_30k_history.png"; plt.savefig(plot_filename); print(f"Графіки збережено: {plot_filename}") # <--- Оновлено ім'я файлу
    except Exception as e_plot: print(f"Помилка збереження графіків: {e_plot}")
    plt.show()

# --- Функція завантаження ресурсів---
def load_ai_resources(
        model_weights_path=MODEL_WEIGHTS_FILE,
        tokenizer_path=TOKENIZER_FILE,
        encoder_path=LABEL_ENCODER_FILE,
        db_path=DB_FILE):
    print("\n[AI Loader] Завантаження ресурсів ШІ...")
    loaded_model, loaded_tokenizer, loaded_encoder, db_conn = None, None, None, None
    try:
        required = {'Ваги': model_weights_path, 'Токенізатор': tokenizer_path, 'Енкодер': encoder_path}
        print("[AI Loader] Перевірка файлів..."); missing = [n for n, p in required.items() if not os.path.exists(p)]
        if missing: raise FileNotFoundError(f"Відсутні файли: {', '.join(missing)}")
        print("[AI Loader] Файли знайдено.")
        db_conn = init_db(db_path);
        if db_conn is None: raise ConnectionError(f"Не вдалося підключитися до БД: {db_path}")
        print(f"[AI Loader] Завантаження {tokenizer_path}...");
        with open(tokenizer_path, 'r', encoding='utf-8') as f: loaded_tokenizer = tokenizer_from_json(f.read())
        print(f"[AI Loader] Завантаження {encoder_path}...");
        with open(encoder_path, 'rb') as f: loaded_encoder = pickle.load(f)
        print("[AI Loader] Токенізатор та енкодер завантажено.")
        if list(loaded_encoder.classes_) != BINARY_LABELS: print("!!! ПОПЕРЕДЖЕННЯ: Класи в енкодері не бінарні!")
        build_vocab_size = VOCAB_SIZE # <--- Використовуємо константу
        print(f"[AI Loader] Побудова моделі (словник={build_vocab_size})...");
        loaded_model = build_ipso_rnn_model(build_vocab_size, EMBEDDING_DIM, MAX_LENGTH, LSTM_UNITS, DROPOUT_RATE) # <--- Використовуємо звичайний LSTM
        print(f"[AI Loader] Завантаження ваг {model_weights_path}...")
        loaded_model.load_weights(model_weights_path)
        print("[AI Loader] Прогрів моделі...");
        _ = loaded_model.predict([np.zeros((1, MAX_LENGTH)), np.zeros((1, 1))], verbose=0)
        print("[AI Loader] Модель завантажена та готова.")
```

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

# --- Основний блок ---
if __name__ == "__main__":
    db_connection = init_db(DB_FILE);
    if db_connection is None: print("Помилка БД. Вихід."); sys.exit(1)
    tokenizer, label_encoder, model, history = None, None, None, None
    load_successful = False;

    # Спроба завантажити компоненти
    if               os.path.exists(TOKENIZER_FILE)               and
os.path.exists(LABEL_ENCODER_FILE)                                and
os.path.exists(MODEL_WEIGHTS_FILE):
        print("Виявлено компоненти. Завантаження...");
        try:
            model, tokenizer, label_encoder, db_connection_loaded =
load_ai_resources(
                MODEL_WEIGHTS_FILE,                    TOKENIZER_FILE,
LABEL_ENCODER_FILE, DB_FILE
            )
            if   model   and   tokenizer   and   label_encoder   and
db_connection_loaded:
                db_connection = db_connection_loaded
                load_successful = True
                print("Ресурси успішно завантажено.")
                # # ---> Оцінка завантаженої моделі на валідації <---
                print("\nЗавантаження    валідаційних    даних    для
оцінки...")
                try:
                    X_val = np.load('X_val_binary.npy')
                    y_val = np.load('y_val_binary.npy')
                    val_reliability                                  =
np.load('val_reliability_binary.npy')
                    print(f"Валідаційні      дані      завантажено:
X({X_val.shape}), y({y_val.shape})")

                    print("\n---   Оцінка   завантаженої   моделі   на
валідаційній вибірці (поріг=0.5) ---")
                    y_pred_probs_val     =     model.predict([X_val,
val_reliability], batch_size=BATCH_SIZE, verbose=0)
                    if y_pred_probs_val.ndim > 1: y_pred_probs_val =
y_pred_probs_val.flatten()
                    y_pred_val_binary    =    (y_pred_probs_val    >=
0.5).astype(int)

                    print("Матриця помилок:")
                    cm = confusion_matrix(y_val, y_pred_val_binary)
```

```
                    print(pd.DataFrame(cm,
index=label_encoder.classes_, columns=label_encoder.classes_))
                    print("\nЗвіт класифікації:")
                    print(classification_report(y_val,
y_pred_val_binary,              target_names=label_encoder.classes_,
zero_division=0))
                except FileNotFoundError:
                    print("ПОМИЛКА: .npy файли валідаційних даних не
знайдено. Оцінка неможлива.")
                except Exception as e_eval_load:
                    print(f"Помилка  під  час  оцінки  завантаженої
моделі: {e_eval_load}")
                # # ---> КІНЕЦЬ ОЦІНКИ <---

            else:    print("Функція    load_ai_resources    повернула
помилку."); load_successful = False
        except  Exception  as  e_load_func:  print(f"Помилка  виклику
load_ai_resources: {e_load_func}"); load_successful = False
    else: print("Збережені компоненти не знайдені.")

    # Якщо не завантажили -> Навчання
    if not load_successful:
        print("\n--- Запуск обробки даних та навчання моделі ---")
        X_train, X_val, y_train, y_val, tokenizer, label_encoder =
load_and_preprocess_data(
            MESSAGES_PATH,       TRAIN_CSV_PATH,       TEST_CSV_PATH,
BINARY_LABELS, VOCAB_SIZE # Передаємо новий VOCAB_SIZE
        )
        if tokenizer is None or label_encoder is None or y_train is
None:
            print("\nПОМИЛКА обробки даних. Вихід.");
            if db_connection: db_connection.close(); sys.exit(1)
        else: print("\nДані для навчання підготовлено.")

        # Розрахунок ваг класів
        class_weights_dict = None
        try:
            print("\nРозрахунок ваг класів..."); y_integers = y_train
            unique_classes,    counts    =    np.unique(y_integers,
return_counts=True)
            print(f"Розподіл        класів        у        y_train:
{dict(zip(label_encoder.inverse_transform(unique_classes),
counts))}")
            if len(unique_classes) == 2:
                weights                                             =
class_weight.compute_class_weight(class_weight='balanced',
classes=unique_classes, y=y_integers)
                class_weights_dict     =     dict(zip(unique_classes,
weights)); print(f"Ваги розраховано: {class_weights_dict}")
            else: print("К-сть класів не 2, ваги не розраховано.")
        except  Exception  as  e_cw:  print(f"ПОМИЛКА  розрахунку  ваг:
{e_cw}."); class_weights_dict = None
```

```
        build_vocab_size = VOCAB_SIZE; print(f"\nРозмір словника для
моделі: {build_vocab_size}")
        try:
            model      =      build_ipso_rnn_model(build_vocab_size,
EMBEDDING_DIM, MAX_LENGTH, LSTM_UNITS, DROPOUT_RATE)
            model, history = train_ipso_model(model, X_train, y_train,
X_val, y_val,
                                              epochs=EPOCHS,
batch_size=BATCH_SIZE,

model_weights_file=MODEL_WEIGHTS_FILE,

class_weights=class_weights_dict)
            if model is None: raise ValueError("Навчання повернуло
None.")
            print("Нову модель навчено.")

            # Збереження компонентів
            print("\nЗбереження    токенізатора    та    кодувальника
міток...")
            try:
                with open(TOKENIZER_FILE, 'w', encoding='utf-8') as f:
f.write(tokenizer.to_json())
                print(f"Токенізатор збережено: {TOKENIZER_FILE}")
                with    open(LABEL_ENCODER_FILE,    'wb')    as    f:
pickle.dump(label_encoder, f)
                print(f"Кодувальник         міток         збережено:
{LABEL_ENCODER_FILE}")
                # ---> Збереження валідаційних даних <---
                print("\nЗбереження валідаційних даних для аналізу
порогу...")
                np.save('X_val_binary.npy', X_val)
                np.save('y_val_binary.npy', y_val)
                val_reliability_to_save  =  np.full((X_val.shape[0],
1), INITIAL_RELIABILITY, dtype='float32')
                np.save('val_reliability_binary.npy',
val_reliability_to_save)
                print("Валідаційні дані збережено у .npy файли.")
            except Exception as e_save: print(f"ПОМИЛКА збереження:
{e_save}")

            # # Побудова графіків
            if        history:        plot_training_history(history,
model_name="IPSO_RNN_LSTM_Binary_30k")

            # # ---> Оцінка моделі ПІСЛЯ НАВЧАННЯ <---
            if model and X_val is not None and y_val is not None:
                print("\n--- Оцінка моделі на валідаційній вибірці
(поріг=0.5) ---")
                try:
                    val_reliability  =  np.full((X_val.shape[0],  1),
INITIAL_RELIABILITY, dtype='float32')
```

```
                    y_pred_probs_val     =     model.predict([X_val,
val_reliability], batch_size=BATCH_SIZE, verbose=0)
                    if y_pred_probs_val.ndim > 1: y_pred_probs_val =
y_pred_probs_val.flatten()
                    y_pred_val_binary    =    (y_pred_probs_val    >=
0.5).astype(int)

                    print("Матриця помилок:")
                    cm = confusion_matrix(y_val, y_pred_val_binary)
                    print(pd.DataFrame(cm,
index=label_encoder.classes_, columns=label_encoder.classes_))
                    print("\nЗвіт класифікації:")
                    print(classification_report(y_val,
y_pred_val_binary,              target_names=label_encoder.classes_,
zero_division=0))
                except Exception as e_eval:
                    print(f"Помилка під час оцінки моделі: {e_eval}")
            # ---> КІНЕЦЬ ОЦІНКИ <---

        except Exception as e_train:
             print(f"\nКРИТИЧНА ПОМИЛКА навчання: {e_train}")
             if  db_connection:  db_connection.close();  print("\nБД
закрито."); sys.exit(1)

    # # Тестові виклики (використовуємо стандартний поріг 0.5)
    if model and tokenizer and label_encoder:
        print(f"\n--- Тестові виклики функції передбачення (поріг=0.5)
---")
        try:
            test_text_1="Срочно!   Распространите!   Наши   герои...";
test_source_id_1="channel_test_b_1"
            print(f"\nТест     1:     Текст='{test_text_1[:50]}...',
Джерело='{test_source_id_1}'")
            is_ipso1,                  reliability1                  =
predict_single_message(model,tokenizer,label_encoder,db_connection,t
est_text_1,test_source_id_1)
            print(f"Результат         1:         is_ipso={is_ipso1},
reliability={reliability1:.3f}")

            test_text_2="Сьогодні   чудовий   день   для   роботи.";
test_source_id_2 = test_source_id_1
            print(f"\nТест     2:     Текст='{test_text_2[:50]}...',
Джерело='{test_source_id_2}' (SAFE)")
            is_ipso2,                  reliability2                  =
predict_single_message(model,tokenizer,label_encoder,db_connection,t
est_text_2,test_source_id_2)
            print(f"Результат         2:         is_ipso={is_ipso2},
reliability={reliability2:.3f}")

            test_text_3="Срочно!   Распространите!   Наши   герои...";
test_source_id_3 = test_source_id_1
            print(f"\nТест     3:     Текст='{test_text_3[:50]}...',
Джерело='{test_source_id_3}' (Дублікат)")
```

```
            is_ipso3,                   reliability3                    =
predict_single_message(model,tokenizer,label_encoder,db_connection,t
est_text_3,test_source_id_3)
            print(f"Результат          3:          is_ipso={is_ipso3},
reliability={reliability3:.3f}")

            test_text_4="Всі на вила! Терпіти більше не можна!";
test_source_id_4 = None
            print(f"\nТест      4:      Текст='{test_text_4[:50]}...',
Джерело=None (пряме)")
            is_ipso4,                   reliability4                    =
predict_single_message(model,tokenizer,label_encoder,db_connection,t
est_text_4,test_source_id_4)
            print(f"Результат          4:          is_ipso={is_ipso4},
reliability={reliability4:.3f}")

            test_text_5="Який гарний котик."; test_source_id_5 = None
            print(f"\nТест      5:      Текст='{test_text_5[:50]}...',
Джерело=None (пряме)")
            is_ipso5,                   reliability5                    =
predict_single_message(model,tokenizer,label_encoder,db_connection,t
est_text_5,test_source_id_5)
            print(f"Результат          5:          is_ipso={is_ipso5},
reliability={reliability5:.3f}")

        except Exception as e_pred: print(f"\nПомилка тестів:
{e_pred}")
    else:       print("\nПропуск       тестів:       компоненти       не
завантажено/навчено.")

    # Закриття БД
    if db_connection:
        try: db_connection.close(); print("\n3'єднання з БД закрито.")
        except sqlite3.Error as e_close: print(f"Помилка закриття БД:
{e_close}")

    print("\nРоботу скрипта завершено.")
```

Додаток Б. Лістинг програмного коду скрипта для інтерактивного тестування моделі (test_interactive.py)

```
import os
import sqlite3
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import tokenizer_from_json
import pickle # Для завантаження label_encoder
import sys # Для перевірки шляхів імпорту

# --- Імпорт компонентів з БІНАРНОЇ версії ipso_model.py ---
# Переконайтесь, що ipso_model.py містить фінальну бінарну версію
try:
    from ipso_model import (
        init_db, get_source_reliability, update_source_reliability,
clean_text,
        predict_single_message, build_ipso_rnn_model, # Функції
        # --- Константи шляхів для БІНАРНОЇ версії ---
        DB_FILE,                # 'ipso_database_binary.db'
        MODEL_WEIGHTS_FILE,                                          #
'model_weights_binary/ipso_rnn_binary.weights.h5'
        TOKENIZER_FILE,         # 'tokenizer_binary.json'
        LABEL_ENCODER_FILE,     # 'label_encoder_binary.pkl'
        # --- Інші константи ---
        MAX_LENGTH, BINARY_LABELS, # Використовуємо BINARY_LABELS
        VOCAB_SIZE, EMBEDDING_DIM, LSTM_UNITS, DROPOUT_RATE,
        MODEL_WEIGHTS_DIR,      # 'model_weights_binary'
        SAFE_CONTENT_LABEL, IPSO_LABEL # Мітки класів
    )
    print("Компоненти з ipso_model.py (бінарна версія) успішно
імпортовано.")
    print(f"  Очікуваний файл ваг: {MODEL_WEIGHTS_FILE}")
    print(f"  Очікуваний файл токенізатора: {TOKENIZER_FILE}")
    print(f"  Очікуваний файл енкодера: {LABEL_ENCODER_FILE}")
    print(f"  Очікуваний файл БД: {DB_FILE}")

except ImportError as e:
    print(f"Помилка імпорту з ipso_model.py: {e}")
    print("Переконайтеся, що файл ipso_model.py (бінарна версія)
знаходиться в тій самій папці.")
    sys.exit(1)
except AttributeError as ae:
     print(f"Помилка імпорту атрибута з ipso_model.py: {ae}")
     print("Можливо, ви забули визначити якусь константу (напр.,
BINARY_LABELS, TOKENIZER_FILE) у бінарній версії ipso_model.py?")
     sys.exit(1)

# --- Перевірка наявності необхідних файлів ---
```

```
print("\nПеревірка наявності необхідних файлів...")

required_files = [MODEL_WEIGHTS_FILE, TOKENIZER_FILE,
LABEL_ENCODER_FILE]
missing_files = [f for f in required_files if not os.path.exists(f)]

if missing_files:
    print("\nПОМИЛКА: Не знайдено наступні необхідні файли:")
    for f in missing_files:
        print(f"  - {f}")
    print("\nБудь ласка, запустіть скрипт ipso_model.py (бінарну
версію) для навчання")
    print("та збереження компонентів.")
    sys.exit(1)
else:
    print("Усі необхідні файли знайдено.")


# --- Завантаження компонентів ---
print("\nЗавантаження моделі та компонентів...")

db_connection_test = None
model_test = None
tokenizer_test = None
label_encoder_test = None # LabelEncoder все ще потрібен для
predict_single_message

try:
    # 1. Ініціалізація БД
    print(f"Ініціалізація БД з файлу: {DB_FILE}...")
    db_connection_test = init_db(DB_FILE)
    if db_connection_test is None:
        raise ConnectionError(f"Не вдалося підключитися до БД:
{DB_FILE}")

    # 2. Завантаження токенізатора
    print(f"Завантаження токенізатора з {TOKENIZER_FILE}...")
    with open(TOKENIZER_FILE, 'r', encoding='utf-8') as f:
        tokenizer_test = tokenizer_from_json(f.read())
    print("Токенізатор успішно завантажено.")
    saved_vocab_size = len(tokenizer_test.word_index) + 1
    print(f"Розмір словника з файлу токенізатора: {saved_vocab_size}")

    # 3. Завантаження кодувальника міток
    print(f"Завантаження кодувальника міток з
{LABEL_ENCODER_FILE}...")
    with open(LABEL_ENCODER_FILE, 'rb') as f:
        label_encoder_test = pickle.load(f)
    print("Кодувальник міток успішно завантажено.")
    # Перевірка відповідності класів
    if list(label_encoder_test.classes_) != BINARY_LABELS:
         print("\n!!! ПОПЕРЕДЖЕННЯ: Класи в завантаженому LabelEncoder
не співпадають з BINARY_LABELS!")
```

```
        print(f"                          Класи        у        файлі:
{list(label_encoder_test.classes_)}")
        print(f"   Очікувані класи: {BINARY_LABELS}\n")

    # 4. Завантаження моделі
    print(f"Завантаження ваг моделі з {MODEL_WEIGHTS_FILE}...")
    build_vocab_size = VOCAB_SIZE # Використовуємо константу
    print(f"Розмір словника для побудови моделі: {build_vocab_size}")

    # ---> ЗМІНА: Викликаємо build_ipso_rnn_model БЕЗ num_classes <--
-
    model_test = build_ipso_rnn_model(
        build_vocab_size,  EMBEDDING_DIM,  MAX_LENGTH,  LSTM_UNITS,
DROPOUT_RATE
    )
    # Завантажуємо ваги
    model_test.load_weights(MODEL_WEIGHTS_FILE)
    print("Модель успішно завантажена.")
    # model_test.summary()

except FileNotFoundError as fnf_error:
     print(f"\nКритична помилка: Файл не знайдено: {fnf_error}")
     if db_connection_test: db_connection_test.close()
     sys.exit(1)
except ConnectionError as conn_error:
    print(f"\nКритична помилка підключення до БД: {conn_error}")
    sys.exit(1)
except Exception as e:
    print(f"\nКритична помилка під час завантаження компонентів: {e}")
    if db_connection_test: db_connection_test.close()
    sys.exit(1)


# --- Інтерактивний ввід ---
print("\n--- Інтерактивне тестування моделі (бінарна класифікація) --
-")
print("Введіть 'exit' або натисніть Ctrl+C для завершення.")

# Встановлюємо поріг для класифікації (можна змінити на оптимальний,
якщо ви його визначили)
CLASSIFICATION_THRESHOLD = 0.5
print(f"(Використовується       поріг       для       класифікації:
{CLASSIFICATION_THRESHOLD})")

while True:
    try:
        user_text = input("Введіть текст повідомлення: ")
        if user_text.lower() == 'exit':
            break
        if not user_text:
            print("Текст не може бути порожнім.")
            continue
```

```
        user_source_id_str = input("Введіть ID джерела (залиште
порожнім для прямого повідомлення): ")
        user_source_id = None
        if user_source_id_str.strip():
             user_source_id = user_source_id_str.strip()

        print("Аналіз...")
        # ---> ЗМІНА: Викликаємо predict_single_message з правильними
аргументами і отримуємо 2 значення <---
        is_ipso_user, reliability_user = predict_single_message(
            model_test,
            tokenizer_test,
            label_encoder_test, # Все ще передаємо, хоча він менш
критичний тут
            db_connection_test,
            user_text,
            user_source_id,
            MAX_LENGTH,
            threshold=CLASSIFICATION_THRESHOLD # Передаємо поріг
        )

        print("\n--- Результат аналізу ---")
        print(f"Текст: '{user_text[:100]}...'")
        print(f"Джерело: {user_source_id if user_source_id else 'Пряме
повідомлення'}")
        # ---> ЗМІНА: Виводимо тільки is_ipso <---
        print(f"Є ІПСО (за порогом {CLASSIFICATION_THRESHOLD}): {'Так'
if is_ipso_user else 'Ні'}")
        # Категорію більше не виводимо

        if user_source_id:
             # Отримуємо актуальне значення після можливого оновлення
для відображення
             final_reliability_display                                =
get_source_reliability(db_connection_test, user_source_id)
             print(f"Актуальна          надійність          джерела:
{final_reliability_display:.2f}")
        print("-" * 25)

    except KeyboardInterrupt:
        print("\nЗавершення роботи користувачем.")
        break
    except Exception as e:
        print(f"\nПід час обробки запиту сталася помилка: {e}")
        print("Спробуйте ще раз або введіть 'exit'.")

print("\nЗавершення інтерактивного тестування.")

# --- Закриття з'єднання з БД ---
if db_connection_test:
    try:
        db_connection_test.close()
        print("З'єднання з БД закрито.").
```