**Міністерство освіти і науки України**

**Луцький національний технічний університет**
(повне найменування закладу вищої освіти)

**Факультет комп'ютерних та інформаційних технологій**
(повне найменування факультету)

**Кафедра комп'ютерної інженерії та безпеки**
(повне найменування кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

# РОЗРОБКА ТА ОПТИМІЗАЦІЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИЯВЛЕННЯ ФЕЙКОВИХ НОВИН

# DEVELOPMENT AND OPTIMISATION OF A NEURAL NETWORK FOR FAKE NEWS DETECTION

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІм-21
Пасічніченко Дмитро Олександрович

(підпис)

Керівник:
к.т.н., доцент
Гордєєва Дар'я Валеріївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
«_____« грудня _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Гринюк Сергій Васильович

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т.ТЕРЛЕЦЬКИЙ
«_____« _____ 2025 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

*Пасічніченку Дмитру Олександровичу*
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Розробка та оптимізація нейронної мережі для виявлення фейкових новин*

Керівник роботи *к.т.н., доцент Гордєєва Дар'я Валеріївна*

затверджені наказом закладу вищої освіти від «17» червня 2025 року № 290/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 09.12.2025р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
*Вступ*
*Теоретичні основи аналізу текстів і виявлення фейкових новин*
*Проєктування та реалізація нейронної мережі*
*Оптимізація моделі та інтерпретація результатів*
*Висновки*

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Теоретичні основи аналізу текстів і виявлення фейкових новин* | *Гордєєва Д.В., доцент* | | |
| *Проєктування та реалізація нейронної мережі* | *Гордєєва Д.В., доцент* | | |
| *Оптимізація моделі та інтерпретація результатів* | *Гордєєва Д.В., доцент* | | |
| *Нормоконтроль* | *Багнюк Н.В., доцент* | | |
| *Гарант ОП* | *Гринюк С.В., доцент* | | |
| *Показник запозичень тексту* | ***%*** | | |
| *Академічна доброчесність* | *Міскевич О.І., ст.викладач* | | |

7. Дата видачі завдання 18.06.2025 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | *Огляд літератури із досліджуваної проблеми* | До 01.08.2025 р. | |
| 2. | *Теоретичні основи аналізу текстів і виявлення фейкових новин* | До 20.08.2025 р. | |
| 3. | *Проєктування та реалізація нейронної мережі* | До 25.09.2025 р. | |
| 4. | *Оптимізація моделі та інтерпретація результатів* | До 20.10.2025 р. | |
| 5. | *Висновки та пропозиції* | До 25.10.2025 р | |
| 6. | *Формування списку використаних джерел* | До 27.10.2025 р. | |
| 7. | *Формування додатків* | До 30.10.2025 р. | |
| 8. | *Оформлення ілюстративного матеріалу* | До 05.11.2025 р. | |
| 9. | *Представлення остаточного варіанту кваліфікаційної роботи керівникові* | До 11.11.2025 р. | |
| 10. | *Нормоконтроль* | До 29.11.2025 р. | |
| 11. | *Інструментальна перевірка на академічний плагіат* | До 02.12.2025 р. | |
| 12. | *Здача кваліфікаційної роботи та всіх супровідних документів на кафедру* | До 09.12.2025 р. | |

Здобувач вищої освіти _____ Пасічніченко Д.О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Гордєєва Д.В.
(підпис) (прізвище, ініціали)

# АНОТАЦІЯ

Пасічніченко Д. О. Розробка та оптимізація нейронної мережі для виявлення фейкових новин. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

Перший розділ присвячено огляду предметної області. Тут розглядаються основні поняття про фейкові новини, їх класифікація, джерела та вплив на суспільство. Також в цьому розділі здійснено огляд сучасних методів автоматичного виявлення дезінформації, зокрема архітектур глибокого навчання та трансформерних моделей.

В другому розділі здійснено проєктування та реалізацію нейромережевої системи. Проведено аналіз та обґрунтування вибору академічного датасету LIAR. Описано процес попередньої обробки даних, включаючи бінаризацію класів та дві різні стратегії очищення тексту. Також у цьому розділі визначено архітектури, реалізовано процес навчання та проведено тестування трьох моделей: CNN, Bi-LSTM та DistilBERT.

Третій розділ присвячено аналізу, оптимізації та практичному впровадженню найкращої моделі. Проведено порівняльний аналіз продуктивності, який довів перевагу трансформерних моделей над CNN та Bi-LSTM. Проведено практичну оптимізацію шляхом переходу на важчу архітектуру BERT-base з оптимізованими гіперпараметрами. На основі фінальної моделі розроблено програмний прототип у вигляді Telegram-бота.

Ключові слова: фейкові новини, глибоке навчання, обробка природної мови (NLP), BERT, DistilBERT, CNN, Bi-LSTM, LIAR dataset, Telegram-бот, класифікація тексту.

# ANNOTATION

Pasichnichenko D. Development and Optimization of a Neural Network for Fake News Detection. Manuscript.

Master's Thesis of the EP «Computer Engineering», specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The Master's thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter is devoted to a review of the subject domain. It examines the fundamental concepts of fake news, including its classification, sources, and societal impact. This chapter also provides an overview of modern methods for the automatic detection of disinformation, specifically focusing on deep learning architectures and transformer models.

The second chapter describes the design and implementation of the neural network system. An analysis and justification for selecting the academic LIAR dataset are provided. The data preprocessing pipeline is described, including class binarization and two distinct text cleaning strategies. This chapter also defines the architectures, implements the training process, and details the testing of the three models: CNN, Bi-LSTM, and DistilBERT.

The third chapter is devoted to the analysis, optimization, and practical implementation of the best-performing model. A comparative performance analysis was conducted, which proved the advantage of transformer models over CNN and Bi-LSTM. Practical optimization was performed by transitioning to the heavier BERT-base architecture using optimized hyperparameters. Based on the final model, a software prototype was developed in the form of a Telegram bot.

Keywords: fake news, deep learning, natural language processing (NLP), BERT, DistilBERT, CNN, Bi-LSTM, LIAR dataset, Telegram bot, text classification.

# ЗМІСТ

# ВСТУП

В умовах стрімкої цифровізації суспільства, інформація поширюється миттєво. Але разом із цим виникла величезна проблема – фейкові новини. Соціальні мережі, месенджери та різні сайти стали не просто місцем для спілкування, а й потужним інструментом для поширення неправди. Цю неправду використовують, щоб маніпулювати людьми, впливати на політику і навіть вести «гібридні війни». Через це люди все менше довіряють офіційним джерелам та ЗМІ, що є небезпечним для суспільства.

Перевіряти кожну новину вручну, як це роблять традиційні фактчекери, просто неможливо – новин з'являється надто багато і надто швидко. Саме тому актуальною задачею є створення автоматизованих «розумних» систем, які могли б самі знаходити фейки.

Проблема полягає в тому, що фейкові новини часто написані дуже хитро. Вони використовують складні речення, грають на емоціях (страх, гнів) або подають факти так, щоб ввести в оману. Звичайні комп'ютерні програми не можуть розпізнати такі маніпуляції. Для цього потрібні більш просунуті технології – методи глибокого навчання, а саме нейронні мережі. Це системи, які можна «навчити» розуміти контекст і помічати приховані ознаки брехні у тексті.

Мета цієї роботи – побудувати нейронну мережу, здатну автоматично знаходити фейкові новини в текстових даних.

Об'єкт дослідження – це сам процес автоматичного пошуку фейків у текстах новин.

Предметом дослідження є моделі та методи нейронної мережі, процес її «навчання» на текстових даних та методи оцінки її здатності знаходити фейки.

Для досягнення поставленої мети необхідно розв'язати наступні завдання:

– проаналізувати поняття фейкових новин, їх класифікацію, джерела походження та вплив на сучасне інформаційне суспільство;

– здійснити огляд сучасних методів автоматизованого виявлення дезінформації та визначити переваги й недоліки існуючих підходів;

– дослідити особливості використання нейронних мереж для аналізу текстів та обґрунтувати доцільність застосування методів глибокого навчання;

– визначити критерії та метрики для об’єктивного оцінювання якості моделей класифікації;

– здійснити огляд доступних наборів даних та обрати оптимальний датасет для навчання та тестування моделей;

– розробити та реалізувати алгоритми попередньої обробки текстових даних, необхідні для коректної роботи нейронних мереж;

– обґрунтувати вибір перспективних архітектур нейронних мереж для проведення порівняльного експерименту;

– програмно реалізувати та навчити обрані моделі у середовищі програмування з використанням відповідних бібліотек;

– провести валідацію та тестування розроблених нейронних мереж на незалежній вибірці даних для отримання первинних метрик якості;

– виконати порівняльний аналіз продуктивності моделей та визначити найбільш ефективну архітектуру;

– здійснити інтерпретацію отриманих результатів, проаналізувати причини помилок та поведінку моделей на різних класах даних;

– провести оптимізацію найкращої моделі для підвищення точності класифікації;

– розробити програмний прототип системи у вигляді Telegram-бота для практичної демонстрації роботи моделі в режимі реального часу;

– оцінити можливості впровадження системи, визначити обмеження дослідження та сформулювати рекомендації щодо подальшого розвитку.

Для розв’язання поставлених завдань у роботі використано комплекс загальнонаукових та спеціальних методів:

– аналіз і синтез – для детального вивчення теоретичних засад фейкових новин та поєднання різних підходів при проєктуванні власної системи;

– абстрагування та формалізація – для перетворення текстової інформації у структуровані числові вектори (токенізація, ембединги) та виділення ключових ознак;

– математичне моделювання – для побудови та налаштування архітектур глибоких нейронних мереж;

– експериментальний метод – для проведення навчання моделей, перевірки гіпотез щодо впливу параметрів та валідації результатів на тестовій вибірці;

– порівняльний аналіз – для зіставлення результатів ефективності різних архітектур та вибору найкращої моделі на основі емпіричних даних;

– узагальнення – для формулювання загальних висновків щодо ефективності архітектур на основі отриманих експериментальних даних;

– методи математичної статистики – для кількісного оцінювання якості класифікації (розрахунок метрик Accuracy, Precision, Recall, F1-Score);

– методи програмної інженерії – для проєктування архітектури системи, реалізації програмного прототипу та інтеграції навченої моделі у Telegram-бот.

Наукова новизна одержаних результатів полягає у підвищенні точності виявлення фейкових новин у коротких повідомленнях. Удосконалено метод налаштування моделі BERT шляхом оптимізації довжини вхідної послідовності (max_len=60), що покращило точність порівняно зі стандартними параметрами. Набув подальшого розвитку порівняльний аналіз архітектур, який довів перевагу трансформерних моделей у забезпеченні балансу між класами на відміну від традиційних підходів. Обґрунтовано диференційований підхід до попередньої обробки даних із застосуванням різних стратегій очищення залежно від типу нейронної мережі.

Практичне значення полягає в тому, що створена модель здатна аналізувати новини та давати оцінку їхньої правдивості. Її можна вбудувати у вебсайти або додатки, щоб допомагати користувачам відрізняти правдиву інформацію від дезінформації.

Апробацію результатів дослідження здійснено на II Міжнародній науково-практичній конференції «Progressive Approaches in Science and Engineering».За результатами конференції опубліковано тези доповіді у секції «Information Technology & Cybersecurity» збірника наукових праць [1].

# РОЗДІЛ 1
# ТЕОРЕТИЧНІ ОСНОВИ АНАЛІЗУ ТЕКСТІВ І ВИЯВЛЕННЯ ФЕЙКОВИХ НОВИН

## 1.1 Поняття фейкових новин: класифікація, джерела, вплив на суспільство

У сучасному інформаційному просторі, що характеризується стрімким поширенням цифрових технологій, поняття фейкових новин стало центральним у дискусіях про медіа та суспільство [2]. Воно описує феномен поширення умисно неправдивої або маніпулятивної інформації, стилізованої під справжні новинні повідомлення, з метою введення аудиторії в оману. Актуальність проблеми різко зросла через домінування соціальних мереж, які виступають не лише платформою для спілкування, але й потужним каналом розповсюдження контенту, часто без належної верифікації [3].

Фейкові новини визначаються як навмисно сфабрикована інформація, що імітує журналістський матеріал за формою, але не за змістом та редакційними стандартами. Ключовим критерієм, що відрізняє фейкові новини від ненавмисних помилок (місінформації), є намір завдати шкоди або отримати вигоду – фінансову, політичну чи репутаційну [4].

Для систематизації цього явища в науковій спільноті використовується кілька рівнів класифікації (табл. 1.1).

Таблиця 1.1 – Класифікація типів інформаційного розладу [5]

| Тип | Назва (англ.) | Визначення | Приклад |
|---|---|---|---|
| Дезінформація | Disinformation | Свідоме створення та поширення неправдивої інформації з метою завдати шкоди чи маніпулювати. | Політична кампанія запускає сайт-двійник новинного видання для публікації компромату на опонента. |
| Місінформація | Misinformation | Ненавмисне поширення недостовірної інформації. Людина, що ділиться нею, вважає її правдивою. | Користувач у соцмережі поширює статтю про ліки від хвороби, не знаючи, що її науково спростовано. |

Продовження таблиці 1.1

| Тип | Назва (англ.) | Визначення | Приклад |
|---|---|---|---|
| Малінформація | Malinformation | Поширення правдивої, але конфіденційної чи вирваної з контексту інформації, з метою завдати шкоди репутації. | Публікація приватного листування публічної особи для її дискредитації. |

Перший рівень базується на умисності поширення неправдивої інформації та відомий як модель «інформаційного розладу».

Другий рівень класифікації деталізує, яким саме чином відбувається маніпуляція з контентом [6].

До цих типів належать:

– повністю вигаданий контент. Створення новин, персонажів або цитат, які не мають жодного підґрунтя в реальності;

– напівправда. Змішування реальних фактів із вигаданими деталями, що ускладнює спростування;

– маніпулятивна інтерпретація. Подача правдивих фактів під таким кутом або в такому контексті, що призводить до хибних висновків;

– клікбейтні заголовки. Створення провокативних, сенсаційних заголовків, що не відповідають змісту статті, з метою залучення трафіку;

– сатира або пародія. Матеріали, створені з гумористичною метою (наприклад, видання The Onion), які можуть бути помилково сприйняті як реальні новини.

Третій рівень класифікації враховує формат подання фейкового контенту [7]:

– текстові повідомлення (статті, пости в соціальних мережах, твіти);

– візуальні фейки (оброблені у фоторедакторах зображення, меми, маніпулятивна інфографіка);

– відеофейки (зокрема, deepfake). Синтетичне відео або аудіо, створене за допомогою нейронних мереж для імітації реальних людей, що становить особливо серйозну загрозу;

– гібридні форми, що поєднують текст, фото та відео для підвищення рівня довіри.

Сучасна цифрова екосистема значно прискорює поширення дезінформації. Ключовими каналами та факторами є [8]:

– соціальні мережі та платформи (наприклад, Facebook, X, TikTok, Telegram). Їхні алгоритми, спрямовані на максимізацію залучення, можуть сприяти вірусному поширенню сенсаційного контенту;

– автоматизовані акаунти (соціальні боти), що використовуються для штучного посилення певних наративів та створення ілюзії масової підтримки;

– штучно створені сайти-двійники, які імітують вигляд та назви авторитетних ЗМІ, вводячи користувачів в оману;

– «інформаційні бульбашки» та «ехо-камери». Ефекти, за яких алгоритми персоналізації обмежують доступ до альтернативних думок, посилюючи існуючі переконання користувачів .

Психологічний вплив фейкових новин ґрунтується на експлуатації когнітивних упереджень, зокрема схильності до підтвердження, через яку люди схильні шукати та довіряти інформації, що підтверджує їхні погляди.

Наслідки поширення фейкових новин для суспільства є системними та глибокими, торкаючись політичної, соціальної, економічної сфер та громадського здоров’я [9]:

– політична поляризація. Дезінформація посилює розбіжності між різними соціальними групами, розпалює ворожнечу, ускладнює суспільний діалог та досягнення консенсусу, що може дестабілізувати політичні процеси, особливо під час виборів;

– ерозія довіри. Постійний потік суперечливої та неправдивої інформації знижує довіру громадян до ключових інститутів – уряду, традиційних ЗМІ, наукових установ та експертів. Це підриває основи функціонування демократичного суспільства;

– загрози національній безпеці. Іноземні держави та недержавні актори можуть використовувати дезінформацію як інструмент гібридної війни для

маніпулювання громадською думкою, дестабілізації внутрішньої ситуації в інших країнах та підриву міжнародних відносин;

– вплив на громадське здоров'я. Як показала пандемія COVID-19, поширення фейкових новин щодо хвороб, вакцин та методів лікування може мати катастрофічні наслідки, призводячи до ризикованої поведінки, відмови від ефективних медичних заходів та зростання недовіри до системи охорони здоров'я;

– економічні збитки. Дезінформація може завдавати шкоди бізнесу (наприклад, через поширення неправдивих чуток про продукцію компанії), впливати на фінансові ринки та спотворювати економічні рішення споживачів;

– соціальна напруга. Фейкові новини часто спрямовані на розпалювання ненависті та дискримінації щодо певних соціальних, етнічних чи релігійних груп, що може призводити до реальних конфліктів та насильства;

– зростання попиту на автоматизовані системи фактчекінгу та підвищення рівня медіаграмотності населення як ключових інструментів протидії дезінформації.

Таким чином, проблема виявлення фейкових новин є комплексним викликом. Неможливість ефективно протидіяти цьому явищу за допомогою ручної модерації зумовлює гостру потребу в розробці інтелектуальних автоматизованих систем. Це визначає актуальність дослідження методів на основі нейронних мереж, здатних аналізувати складні лінгвістичні патерни, що буде розглянуто в наступних підрозділах.

## 1.2 Сучасні методи автоматичного виявлення фейкових новин

Сучасне поширення фейкових новин у цифровому середовищі потребує автоматизованих методів їх виявлення. З появою великих обсягів текстових даних та різноманітних джерел інформації традиційні методи ручної перевірки та базові алгоритми класифікації виявляються неефективними. Науковці розробляють різні підходи, які можна умовно поділити на класичні методи машинного навчання, нейронні мережі та гібридні системи, що часто використовують трансформери та графові нейронні мережі [10].

Класичні методи машинного навчання, такі як класифікатор методу Наївного Баєсу, метод опорних векторів (англ. support vector machine (SVM)) та дерева рішень, були серед перших спроб автоматизації цього процесу. Метод Наївного Баєсу є простим та швидким, але його ефективність обмежується припущенням про незалежність ознак. SVM добре працює з високовимірними текстовими ознаками (наприклад, англ. term frequency inverse document frequency (TF-IDF)), будуючи оптимальну гіперплощину для розділення класів. Дерева рішень моделюють складні взаємозв'язки, але схильні до перенавчання. Головним недоліком усіх цих методів є необхідність ручного формування ознак (feature engineering) та їхня обмежена здатність враховувати семантичний контекст, сарказм чи маніпулятивне подання фактів [11].

Для подолання цих обмежень активно застосовуються нейронні мережі. Рекурентні нейронні мережі (англ. recurrent neural network (RNN)), зокрема їхні вдосконалені варіанти LSTM (англ. long short-term memory) та GRU (англ. gated recurrent units), здатні обробляти послідовності слів та враховувати попередній контекст. Використання шарів векторних представлень слів (embeddings) дозволяє відображати семантичні зв'язки між словами. LSTM ефективні для аналізу довших текстів завдяки здатності зберігати довготривалі залежності. Проте, класичні RNN/LSTM зазвичай аналізують текст односпрямовано і потребують значних ресурсів для навчання на великих корпусах [12].

Найсучаснішим етапом є використання трансформерних архітектур та гібридних підходів. Трансформери, такі як BERT (англ. Bidirectional Encoder Representations from Transformers), моделюють двосторонній контекст завдяки механізму уваги (attention), що дозволяє значно краще розуміти значення слів у реченні та виявляти тонкі мовні маніпуляції. Окрім аналізу самого контенту, дослідники почали враховувати соціальний контекст поширення новини, моделюючи його за допомогою графових нейронних мереж (англ. graph neural network (GNN)). Ці мережі аналізують структуру зв'язків між користувачами, які поширюють новину, виходячи з гіпотези, що фейки поширюються інакше, ніж правдиві новини. Гібридні моделі поєднують аналіз контенту (BERT) та аналіз графу поширення (GNN), часто використовуючи механізми спів-уваги (co-

attention) для інтеграції інформації, що демонструє найвищу ефективність на сьогодні [13].

Для ілюстрації еволюції підходів розглянуто три приклади використання різних методів виявлення фейкових новин, що представляють різні етапи розвитку.

Класичне машинне навчання на основі лінгвістичних ознак. Цей підхід представляє ранні спроби автоматизації виявлення обману, коли основна увага приділялася аналізу стилю тексту за допомогою традиційних алгоритмів. Яскравим прикладом цього напряму є піонерська робота [14], присвячена автоматичному розпізнаванню навмисно неправдивого тексту, що імітує новинний стиль. Дослідження проводилося ще до вибухового поширення терміну «фейкові новини», фокусуючись на загальній проблемі комп'ютерної лінгвістики – виявленні обману в тексті. Мотивацією було дослідити, чи існують вимірювані лінгвістичні ознаки, які відрізняють правдивий текст від сфабрикованого. Для цього дослідники створили спеціальний корпус даних: було взято реальні новини та попросили людей переписати їх, зберігаючи стиль, але роблячи їх неправдивими. Далі з текстів вилучалися різноманітні лінгвістичні ознаки (n-грами слів та символів, складність тексту, психолінгвістичні характеристики за словником LIWC). Як зазначається у [14], для класифікації використовувалися класичні алгоритми машинного навчання – метод Наївного Баєсу та SVM. Точна вартість невідома, оскільки це академічне дослідження. Ресурси включали час дослідників на збір та анотацію даних (залучення людей для переписування новин), розробку скриптів для вилучення ознак та експерименти з моделями. Фінансові витрати, ймовірно, були обмежені інфраструктурою університету. Система досягла точності (accuracy) близько 70 % у розрізненні правдивих та сфабрикованих текстів на власному специфічному датасеті [14]. Це був значний результат для свого часу, що довів принципову можливість автоматичного виявлення обману на основі стилю. Система існувала як дослідницький прототип. Публічно доступного сервісу створено не було. Робота [14] мала значний вплив, часто цитується як одна з перших у цій галузі, але сам підхід сьогодні вважається застарілим через обмежену точність та нездатність до узагальнення на різноманітніші та складніші фейки.

Рекурентні нейронні мережі для аналізу послідовностей. Цей етап відображає перехід до глибокого навчання, намагаючись врахувати порядок слів та контекст за допомогою RNN-архітектур. Дослідження, представлена приблизно у 2017 році [15], ілюструє застосування рекурентних нейронних мереж, зокрема LSTM, для аналізу послідовності слів у текстах новин з метою виявлення рівня їх правдивості. Дослідження проводилося на даних, зібраних з сайту перевірки фактів PolitiFact (ймовірно, пов'язаних з датасетом LIAR), і мало на меті перевірити, чи можуть моделі, що враховують порядок слів, покращити результати порівняно з підходами, що ігнорують послідовність. Система використовувала векторні представлення слів GloVe як вхідні дані. LSTM обробляла послідовність векторів слів у заяві, намагаючись вловити контекстуальні залежності, що можуть вказувати на неправдивість. Модель тестувалася як на багатокласовій (6 рівнів правдивості), так і на бінарній (правда/фейк) класифікації. Додатково експериментували з додаванням психолінгвістичних ознак (LIWC) [15]. Використовувалися загальнодоступні датасет на основі PolitiFact та попередньо навчені ембедінги GloVe. Навчання LSTM вимагало використання GPU, ймовірно, в межах університетської інфраструктури. Ефективність моделі була відносно скромною. F1-міра склала близько 20 % для 6-класової задачі та 56 % для бінарної класифікації [15]. Важливим висновком роботи стало те, що на цьому конкретному датасеті простіші моделі (TF-IDF з методом Наївного Баєса), особливо при додаванні ознак LIWC, показали порівняні або навіть кращі результати, ніж LSTM. Ця робота важлива тим, що продемонструвала складнощі застосування ранніх моделей глибокого навчання до цієї задачі та показала, що вони не завжди автоматично перевершують добре налаштовані класичні методи [15].

Гібридні системи з використанням GNN та трансформерів. Цей підхід представляє сучасний стан досліджень, інтегруючи аналіз контенту та соціального контексту за допомогою найновіших нейромережевих архітектур. Яскравим прикладом є система, розроблена у 2021 році в рамках магістерської роботи [16]. Мотивацією було створення моделі, яка б не тільки ефективно виявляла фейки, але й надавала пояснення своєму рішенню, аналізуючи одночасно контент новини та соціальний контекст її поширення (багатоконтекстний підхід). Це стало відповіддю

на недоліки попередніх моделей, які часто працювали як «чорні скриньки». Система використовує гібридну архітектуру. Для аналізу соціального контексту будується граф поширення новини. Характеристики користувачів (профіль, історія твітів) та структура графу аналізуються за допомогою графової нейронної мережі з увагою. Для аналізу контенту самої новини використовуються векторні представлення слів. Далі застосовується механізм спів-уваги, який одночасно навчається визначати важливість слів у тексті та вузлів (користувачів) у графі [16]. Фінальне рішення приймається нейронним класифікатором на основі агрегованих векторів уваги. Система навчалася на реальних даних з датасету FakeNewsNet (Politifact, Gossipcop), що містить новини та інформацію про їх поширення у Twitter [16]. Як і для більшості академічних проектів, точна фінансова вартість невідома. Розробка вимагала значних часових ресурсів магістра та його керівників, доступу до обчислювальних потужностей (ймовірно, університетських кластерів або хмарних сервісів) для навчання глибоких нейронних мереж на великих графах та текстах. Збір даних з Twitter API також потребував часу та дотримання політик платформи. Модель продемонструвала високу ефективність, перевершивши низку попередніх підходів на датасетах Gossipcop та Politifact. Наприклад, на Gossipcop точність досягла близько 96 % [16]. Вагомою перевагою системи стала можливість візуально відображати, хто з користувачів у мережі поширення та які саме слова в тексті новини найбільше вплинули на її класифікацію як фейкової. Інформація про вартість підтримки, надійність чи швидкість обробки запитів у реальному часі відсутня, оскільки система не розгорталася як комерційний продукт. Однак, сам гібридний підхід з використанням GNN, трансформерів та механізмів уваги є актуальним напрямом досліджень і продовжує розвиватися, що свідчить про довгострокову релевантність такої архітектури. Недоліком, відзначеним у роботі, є проблема катастрофічного забування при спробі адаптувати модель до нових доменів (напр., з політичних новин на новини про зірок), що потребує подальших досліджень у сфері неперервного навчання.

Для систематизації результатів аналізу та наочного зіставлення сильних і слабких сторін кожного підходу було сформовано порівняльну характеристику, наведену в таблиці 1.2 [14-16].

Таблиця 1.2 – Порівняння підходів автоматичного виявлення фейкових новин [14-16]

| Метод / Підхід | Основна ідея | Переваги | Недоліки |
|---|---|---|---|
| Класичне машинне навчання (SVM, Naive Bayes) | Статистичний аналіз лінгвістичних ознак тексту (n-грами, синтаксис) без глибокого розуміння семантики. | – простота реалізації та інтерпретації;<br>– низькі вимоги до обчислювальних ресурсів;<br>– ефективність на малих датасетах. | – низька точність (~70 %);<br>– нездатність вловлювати контекст;<br>– потребує складного ручного конструювання ознак. |
| Рекурентні нейронні мережі (RNN, LSTM) | Послідовна обробка тексту для врахування порядку слів та контекстних залежностей. | – Здатність моделювати послідовності слів;<br>– Автоматичне вилучення ознак (через Embeddings);<br>– Краще розуміння контексту, ніж у класичних методів. | – висока обчислювальна складність;<br>– проблема "зникаючого градієнта" на довгих текстах;<br>– обмежена ефективність на коротких заявах (LIAR) |
| Гібридні системи (GNN + Трансформери) | Інтеграція аналізу текстового контенту з аналізом графа соціальних зв’язків (хто і як поширює). | – найвища точність (SOTA, ~96 %);<br>– пояснюваність рішень (візуалізація зв’язків);<br>– врахування соціального контексту. | – залежність від метаданих (соц. графів), які часто недоступні;<br>– висока складність архітектури;<br>– проблема адаптації до нових доменів. |

Порівняння цих трьох аналогів чітко ілюструє прогрес у галузі виявлення фейкових новин. Від простих моделей на лінгвістичних ознаках (класичне машинне навчання), які хоч і довели принципову можливість автоматичного розпізнання обману за допомогою класичних методів, але страждали від нездатності враховувати семантичний контекст та мали обмежену точність на реальних даних. Застосування перших глибоких моделей типу LSTM мало на меті включити контекстну послідовність, однак їхня ефективність на складних наборах даних була варіативною і не завжди виправдовували очікування, показуючи скромну ефективність, іноді поступаючись простішим методам, а також потребуючи значних ресурсів. Складні гібридні системи демонструють найвищу точність і, що важливо, пропонують можливість інтерпретації результатів, хоча й

залишаються найбільш складними в реалізації та мають виклики щодо адаптації до нових доменів.

### 1.3 Використання нейронних мереж для аналізу та виявлення фейкових новин

Зі зростанням обсягів інформації у мережі Інтернет та соціальних медіа проблема автоматичного виявлення фейкових новин стала однією з ключових у сфері обробки природної мови (NLP). Традиційні алгоритми машинного навчання, попри свою ефективність у структурованих завданнях, виявилися недостатньо гнучкими для аналізу складних лінгвістичних і контекстуальних зв'язків, притаманних текстовим повідомленням. У зв'язку з цим у дослідженнях дедалі ширше застосовуються глибокі нейронні мережі, які здатні автоматично навчатися релевантних ознак із великих корпусів даних і виявляти приховані закономірності у тексті [10].

Для вирішення задачі виявлення фейкових новин застосовується широкий спектр архітектур глибокого навчання. Еволюція цих методів проходила від базових моделей для обробки послідовностей до складних трансформерних та графових мереж. Кожна архітектура має свої унікальні механізми роботи, переваги та обмеження. У таблиці 1.3 [17-20] представлено порівняльну характеристику розглянутих розглянутих нейронних мереж.

Таблиця 1.3 – Порівняння архітектур нейронних мереж для аналізу текстів [17-20]

| Архітектура | Механізм роботи | Переваги | Недоліки / Обмеження |
|---|---|---|---|
| CNN (Згорткові мережі) | Використання згорткових фільтрів для виявлення локальних патернів ($n$-грам) у тексті. | – Висока швидкість навчання;<br>– Ефективне виявлення ключових фраз;<br>– Низька обчислювальна вартість. | – Втрата інформації про порядок слів;<br>– Нездатність моделювати довгострокові залежності;<br>– Потребує фіксованої довжини входу. |

Продовження таблиці 1.3

| Архітектура | Механізм роботи | Переваги | Недоліки / Обмеження |
|---|---|---|---|
| RNN / LSTM (Рекурентні мережі) | Послідовна обробка слів із збереженням "пам'яті" про попередній контекст. | – врахування порядку слів;<br>– моделювання залежностей у реченні;<br>– обробка вхідних даних змінної довжини | – проблема зникаючого градієнта (у простих RNN);<br>– повільне послідовне навчання (неможливість розпаралелення);<br>– обмежена кількість пам'яті. |
| Трансформери (BERT, RoBERTa) | Механізм Self-Attention для одночасної оцінки взаємозв'язків між усіма словами. | – двонапрямлений контекст (розуміння «зліва направо» і навпаки);<br>– попереднє навчання на великих корпусах (Transfer Learning); | – дуже висока обчислювальна складність;<br>– потреба у потужних GPU для навчання;<br>– великий розмір моделей (сотні мільйонів параметрів). |
| GNN (Графові мережі) | Аналіз графа поширення новини та зв'язків між користувачами. | – врахування соціального контексту;<br>– висока точність на специфічних датасетах;<br>– можливість аналізувати поведінку ботів. | – залежність від наявності графа (не працює лише з текстом);<br>– складність збору та обробки даних;<br>– вразливість до «холодного старту» (нова новина без репостів). |

Аналіз даних, наведених у таблиці 1.3, дозволяє виділити ключові тенденції у розвитку методів детекції дезінформації. Хоча графові нейронні мережі (GNN) демонструють високу ефективність у сучасних досл'ідженнях завдяки врахуванню соціального контексту, їх практичне застосування часто обмежене необхідністю доступу до складних метаданих про соціальні зв'язки, які не завжди є у відкритому доступі. Серед методів, орієнтованих безпосередньо на аналіз текстового контенту, згорткові (CNN) та рекурентні (LSTM) мережі залишаються важливими базовими архітектурами. Однак вони мають суттєві обмеження: CNN фокусуються переважно на локальних ознаках, втрачаючи глобальний контекст, тоді як LSTM, попри здатність працювати з послідовностями, мають труднощі з обробкою складних семантичних зв'язків у довгих реченнях та не підтримують ефективного

розпаралелення обчислень. Натомість трансформерні архітектури (зокрема BERT) виглядають найбільш перспективними. Завдяки механізму Self-Attention , вони здатні моделювати залежності між усіма словами в реченні одночасно, забезпечуючи глибше розуміння контексту та семантичних нюансів, що є критично важливим для виявлення маніпулятивного контенту. Крім того, можливість використання трансферного навчання (Fine-Tuning попередньо навчених моделей) дозволяє досягти високої точності навіть на відносно невеликих наборах даних. Саме тому у даній роботі подальша реалізація системи базуватиметься на використанні нейронних мереж типу BERT, що дозволяють досягти оптимального балансу між точністю, швидкістю та гнучкістю.

### 1.4 Методи оцінювання якості моделей

Після побудови моделі для виявлення фейкових новин одним із ключових етапів є оцінювання її ефективності. Від правильності вибору метрик залежить адекватність інтерпретації результатів і можливість порівняння різних моделей між собою. Для задач класифікації, до яких належить і виявлення фейкових новин, використовуються як базові, так і розширені методи оцінювання.

Базовим інструментом для аналізу роботи класифікатора є матриця змішування (рис. 1.1). Вона дозволяє візуалізувати результати передбачень моделі, зіставляючи їх із реальними мітками класів.

| | Predicted Positive | Predicted Negative |
|---|---|---|
| Actual Positive | True Positive (TP) | False Negative (FN) |
| Actual Negative | False Positive (FP) | True Negative (TN) |

Рисунок 1.1 – Матриця змішування [21]

Матриця змішування є інструментом, який дає змогу візуалізувати результати класифікації. Вона складається з чотирьох комірок [21,22] :

– true positive (TP). Кількість фейкових новин, які модель правильно визначила як фейкові;

– true negative (TN). Кількість правдивих новин, які модель правильно класифікувала;

– false positive (FP). Кількість правдивих новин, які були хибно визначені як фейкові;

– false negative (FN). Кількість фейкових новин, які модель не виявила.

На основі цих показників можна обчислити всі основні метрики: точність (accuracy), повнота (recall), прецизійність (precision) та F1-міра (F1-score). Матриця змішування також дає змогу виявити систематичні помилки моделі, наприклад, якщо вона часто плутає певні типи новин.

Точність(accuracy). Відношення кількості правильно класифікованих прикладів до загальної кількості спостережень. Вона розраховується за формулою (1.1) [21]:

$$Accuracy = \frac{TP+T}{TP+TN+FP+F}. \quad (1.1)$$

Вона показує загальну ефективність моделі, однак може бути недостатньо інформативною при наявності нерівномірного розподілу класів (наприклад, коли справжніх новин значно більше, ніж фейкових).

Прецизійність (precision) (формула (1.2):

$$Precision = \frac{TP}{TP+FP}. \quad (1.2)$$

Відображає частку правильно визначених фейкових новин серед усіх новин, які модель класифікувала як фейкові [21]. Високе значення цієї метрики свідчить про малу кількість хибно-позитивних результатів.

Повнота (recall) (формула (1.3):

$$Recall = \frac{TP}{TP+FN}. \quad (1.3)$$

Показує частку правильно розпізнаних фейкових новин серед усіх фейкових новин у вибірці [21]. Максимізація повноти є критичною, коли ціна пропуску фейкової новини є високою і необхідно виявити якомога більше дезінформації.

F1-міра (f1-score) (формула (1.4):

$$F1_{score} = 2 * \frac{Precision * Recall}{Precision + Recall}. \quad (1.4)$$

Є гармонійним середнім між працездатністю та повнотою [21]. Вона забезпечує збалансовану оцінку, коли важливо одночасно мінімізувати як хибно-позитивні, так і хибно-негативні результати.

У задачах виявлення фейкових новин f1-міра часто вважається основною метрикою, оскільки вона дозволяє враховувати як якість виявлення фейків, так і кількість пропущених випадків.

На основі цих показників можна обчислити всі основні метрики. Матриця змішування також дає змогу виявити систематичні помилки моделі, наприклад, якщо вона часто плутає певні типи новин.

Для оцінювання якості бінарної класифікації при різних порогових значеннях ймовірності застосовується ROC-крива (англ. Receiver Operating Characteristic). Вона показує залежність між True Positive Rate (чутливістю) та False Positive Rate. Показником якості виступає площа під цією кривою – AUC (англ. Area Under the Curve). Візуалізацію цих показників зображено на рисунку 1.2.

Чим ближче значення AUC до 1, тим краща здатність моделі розрізняти фейкові та правдиві новини. Значення AUC < 0,5 означає, що модель працює гірше за випадкове вгадування [23].

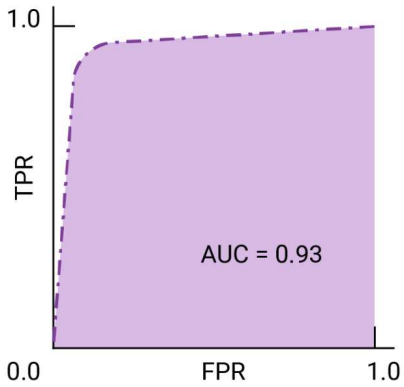


Рисунок 1.2 – Приклад ROC-кривої та візуалізація метрики AUC [23]

Щоб уникнути перенавчання і перевірити стабільність моделі, часто застосовують k-fold cross-validation. Вона передбачає розбиття вибірки на k частин, де кожна частина почергово виступає як тестова, а решта – як тренувальна. Це дозволяє оцінити, наскільки результати моделі узагальнюються на нові дані. У контексті виявлення фейкових новин перехресна перевірка дає можливість переконатися, що модель не переорієнтована на конкретний стиль чи джерело текстів, а дійсно вчиться розпізнавати закономірності, властиві фейковому контенту. Якщо важливо не пропустити жодну фейкову новину, доцільно максимізувати Recall. Якщо пріоритетом є уникнення помилкових звинувачень правдивих новин, акцент робиться на Precision. Для збалансованої оцінки ефективності застосовується F1-міра або AUC [24].

Таким чином, оцінювання якості моделі є невід’ємним етапом процесу виявлення фейкових новин. Коректно обрані метрики дозволяють не лише об’єктивно визначити ефективність побудованої нейронної мережі, але й порівняти її з альтернативними підходами [25].

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ НЕЙРОННОЇ МЕРЕЖІ

## 2.1 Вибір і підготовка датасету

На першому етапі розробки системи виявлення фейкових новин необ хідно обрати відповідний набір даних, який забезпечить достатню якість нав чання нейронної мережі та дозволить отримати достовірні результати.

Для аналізу були розглянуті кілька публічно доступних датасетів, що широко використовуються у дослідженнях з цієї тематики: LIAR, ISOT Fake News Dataset, GossipCop & PolitiFact (FakeNewsNet) та Fake News Dataset із платформи Kaggle.

ISOT Fake News Dataset [26] та Fake News Dataset (Kaggle) [27] містять тисячі повних новинних статей, розділених на бінарні категорії «fake» та «real». Їхньою перевагою є наявність повних текстів, що, на перший погляд, є зручним для побудови глибоких моделей. Проте головним теоретичним ризиком цих датасетів є висока ймовірність наявності системних артефактів (data artifacts). Оскільки дані для класів «fake» та «real» часто збираються з кардинально різних джерел (наприклад, офіційні інформаційні агентства для «real» та неперевірені блоги для «fake»), існує значний ризик, що нейронна мережа навчиться класифікувати не правдивість новини, а поверхневі стилістичні особливості джерела. Це може призвести до нереалістично високих показників точності під час тестування, але така модель буде нездатною до узагальнення на даних з реального світу.

FakeNewsNet (GossipCop & PolitiFact) [28] орієнтований на виявлення фейків у соціальних медіа і містить цінні соціальні метадані (репости, коментарі). Однак для цього проєкту він є надлишковим, оскільки мета дослідження сфокусована на аналізі виключно текстового контенту без залучення складних соціальних графів.

У цьому контексті, LIAR Dataset [29] виділяється як найбільш придатний для цілей даного дослідження. Це один із найвідоміших академічних наборів даних, що містить близько 12 800 коротких тверджень, зібраних з вебсайту PolitiFact. Кожен запис супроводжується оцінкою достовірності та метаданими (автор, контекст).

Хоча цей датасет має певні особливості (обмежена довжина текстів та шість класів), вони є перевагами для побудови надійної моделі. На відміну від ISOT, усі

твердження в LIAR походять зі схожого домену (політичні заяви) і мають схожий стиль. Це виключає ризик навчання на поверхневих патернах джерела і змушує нейронну мережу фокусуватися на змісті та тонких мовних маніпуляціях усередині тексту. Обмежена довжина текстів (короткі вислови) робить завдання складнішим, оскільки модель не може покладатися на довгий контекст, а має виявляти ознаки фейку в одному-двох реченнях.

Порівняльний аналіз розглянутих датасетів за ключовими критеріями наведено в таблиці 2.1 [26-29].

Таблиця 2.1 – Порівняння датасетів за критеріями [26-29]

| Датасет | Повнота Тексту | Актуальність Даних | Наявність Метаданих | Класифікаційна Складність |
|---|---|---|---|---|
| LIAR | Низька (фрагменти) | Середня | Висока (автор, джерело) | Висока (6 незбаланс. класів) |
| ISOT Fake News | Висока (статті) | Середня | Низька (лише текст) | Низька (бінарний, збаланс.) |
| FakeNewsNet (GossipCop/PolitiFact) | Залежить (різна) | Висока | Дуже висока (соц. дані) | Низька (бінарний) |
| Fake News Dataset (Kaggle) | Висока (статті) | Середня | Середня (тема, дата) | Низька (бінарний, збаланс.) |

Таблиця 2.1 надає структуроване порівняння розглянутих датасетів за чотирма ключовими критеріями, що впливають на їх придатність для розробки та тестування моделей виявлення фейкових новин:

– повнота тексту оцінює, чи містить датасет повні статті чи лише короткі фрагменти. Висока повнота є бажаною для моделей, що аналізують контекст.

– актуальність даних показує, наскільки нещодавно були зібрані дані, що важливо через швидку зміну стилів та тем фейків.

– наявність метаданих. Оцінює обсяг додаткової інформації (автор, дата, тема, соціальні дані), що може бути корисною, але ускладнює підготовку даних для контент-орієнтованих моделей.

– класифікаційна складність відображає складність завдання (кількість класів, їх збалансованість). Низька складність (бінарна, збалансована) є перевагою для розробки.

Аналізуючи таблицю 2.1, для цілей даної роботи було обрано LIAR Dataset. Хоча ISOT та Kaggle пропонують повні тексти та просту бінарну класифікацію, вони несуть високий ризик навчання на хибних патернах, що унеможливлює створення моделі, здатної до узагальнення.

LIAR, незважаючи на свою початкову класифікаційну складність, є академічно чесним та надійним датасетом. Він змушує моделі аналізувати зміст, а не стиль, що дозволяє отримати реалістичну, а не завищену, оцінку їхньої продуктивності. Це буде продемонстровано у практичних розділах даної роботи.

Оскільки оригінальний датасет має шість класів достовірності, що є надлишковим для базової задачі детекції, була проведена бінаризація. Для навчання та тестування були відібрані лише найбільш чіткі категорії:

– клас 1 (правда): «true» та «mostly-true»;

– клас 0 (фейк): «false» та «pants-on-fire».

Твердження з проміжними оцінками («half-true», «barely-true») були виключені з вибірки, щоб забезпечити чіткість бінарної класифікації для навчання моделей.

Датасет LIAR складається з трьох файлів: «train.tsv», «test.tsv» та «valid.tsv». Для наочного представлення структури даних розглянемо приклад одного запису (таблиця 2.2).

Ця таблиця демонструє типовий вигляд даних, з якими буде працювати модель. Кожна новина представлена заголовком, повним текстом, тематикою, датою та міткою правдивості.

Таблиця 2.2 – Приклад представлення даних у ISOT Fake News Dataset

| Поле (Назва колонки) | Приклад значення |
|---|---|
| label | half-true |
| statement | Says the Annies List political group supports third-trimester abortions on demand. |
| subject | abortion |
| speaker | dwayne-bohac |
| context | Other |

Порівняння показало, що LIAR Dataset є найпридатнішим для вирішення завдання виявлення фейкових новин. Він забезпечує необхідну семантичну

складність для навчання нейронних мереж, зокрема архітектур типу LSTM, CNN і BERT, і дозволяє отримати репрезентативні та реалістичні результати без ризику перенавчання на артефактах джерела.

### 2.2 Попередня обробка текстових даних

Попередня обробка текстових даних є одним із ключових етапів у побудові системи автоматичного виявлення фейкових новин. Якість цього етапу прямо впливає на точність та стабільність роботи нейронних мереж. Використаний у даній роботі LIAR Dataset має специфічну структуру, що відрізняється від датасетів повних статей. Він складається з коротких політичних заяв, а не повних текстів, і містить шість класів достовірності.

Ключовими завданнями препроцесингу є перетворення шести класів на два (1 – правда, 0 – фейк), видалення «шуму» (HTML, URL) та багатоетапна обробка тексту, яка включає очищення та фінальне перетворення у числовий формат.

Оскільки в даній роботі порівнюються архітектури різного типу (CNN/LSTM та BERT), підхід до очищення тексту для них має бути різним.

Моделі CNN та LSTM навчаються «з нуля». Їхній шар «Embedding» не має жодних попередніх знань про мову. Для цих моделей «шум», такий як стоп-слова (the, is) або різні форми одного слова (studies, studying), ускладнює навчання та збільшує розмір словника. Лістинг 2.1 демонструє процес агресивної попередньої обробки, призначений для підготовки текстових даних до аналізу моделями CNN та LSTM [30].

Лістинг 2.1 – Попередня обробка тестових даних для моделей CNN,LSTM

```
import re
import contractions
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def clean_text_aggressive(text):
```

```
    text = str(text)
    text = contractions.fix(text)
    text = text.lower()
    text = re.sub(r’https?://\S+|www\.\S+’, ‘‘, text)
    text = re.sub(r’<.*?>‘, ‘‘, text)
    text = re.sub(r’[^a-zA-Z\s]’, ‘‘, text)

    words = text.split()
    cleaned_words = []
    for word in words:
        if word not in stop_words:
            cleaned_words.append(lemmatizer.lemmatize(word))

    text = ‘ ‘.join(cleaned_words)
    text = re.sub(r’\s+’, ‘ ‘, text).strip()
    return text
```

кінець лістингу 2.1

На початку скрипта відбувається ініціаліація необхідних інструментів. Спочатку імпортуються ключові бібліотеки: «re» (Regular Expressions) для пошуку та заміни текстових патернів; «contractions» для нормалізації англійських скорочень; а також «stopwords» та «WordNetLemmatizer» з бібліотеки «NLTK».

Одразу після цього створюються два важливі об’єкти, які будуть використовуватися функцією очищення. Рядок «stop_words = set(stopwords. words(‘english’))» завантажує стандартний список англійських стоп-слів і перетворює його на множину (set) для миттєвої перевірки наявності. Аналогічно, «lemmatizer = WordNetLemmatizer()» створює екземпляр лематизатора один раз. Таке винесення ініціалізації поза тіло функції є важливою оптимізацією, що запобігає повторному завантаженню цих ресурсів при кожному виклику функції.

Основна логіка реалізована у функції «clean_text_aggressive(text)». Приймаючи текстовий рядок, вона насамперед гарантує, що вхідні дані мають тип str. Першим кроком обробки є нормалізація скорочень за допомогою «contracti ons.fix()», яка перетворює «don’t» на «do not». Одразу після цього текст приводиться до нижнього регістру («text.lower()»).

Наступним етапом є очищення від «шуму» за допомогою регулярних виразів. Послідовно викликаються три команди «re.sub»: перша видаляє будь-які

URL-адреси, друга – залишки HTML-тегів, а третя – всі не алфавітні символи, залишаючи лише літери та пробіли.

Після очищення, суцільний рядок тексту токенізується (розбивається на список слів) за допомогою методу «text.split()». Далі скрипт запускає цикл «for», який ітерує по кожному слову в цьому списку. Усередині циклу кожне слово проходить дворівневу фільтрацію: спочатку оператор «if word not in stop_words» відсіює всі стоп-слова. Якщо слово пройшло цю перевірку (тобто є семантично значущим), воно передається до «lemmatizer.lemmatize(word)», який зводить його до базової словникової форми. Тільки лематизовані слова, що не є стоп-словами, додаються до нового списку «cleaned_words».

На завершення, цей список «cleaned_words» реконструюється назад у текстовий рядок за допомогою «‘ ‘.join()». Фінальний виклик «re.sub(r’\s+’, ‘ ‘, text).strip()» прибирає будь-які зайві пробіли, що могли утворитися в процесі видалення слів, і повертає готовий до векторизації текст.

На відміну від CNN та LSTM, моделі-трансформери (як BERT) є попередньо навченими на мільярдах речень «живого» тексту. Вони розуміють контекст, який надають стоп-слова, пунктуація та навіть великі літери.

Агресивне очищення (лематизація, видалення стоп-слів) погіршило б результати BERT, оскільки позбавило б його звичного контексту. Тому для BERT була застосована лише базова обробка(лістинг 2.2).

Лістинг 2.2 – Попередня обробка тестових даних для BERT(DistilBERT)

```
import re

def clean_text_simple(text):
    text = str(text).lower()
    text = re.sub(r’https?://\S+|www\.\S+’, ‘ ‘, text)
    text = re.sub(r’<.*?>‘, ‘ ‘, text)
    text = re.sub(r’\s+’, ‘ ‘, text).strip()
    return text
```

кінець лістингу 2.2

На початку скрипта імпортується бібліотека «re», яка є стандартним модулем Python для роботи з регулярними виразами (Regular Expressions), необхідними для пошуку та заміни текстових патернів.

Сама функція «clean_text_simple» приймає один вхідний аргумент «text». Першим кроком усередині функції є гарантування, що вхідні дані оброблятимуться як рядок (str(text)), після чого весь текст негайно приводиться до нижнього регістру (.lower()). Це стандартний крок нормалізації, який запобігає тому, щоб модель сприймала, наприклад, «News» та «news» як різні слова.

Далі починається етап очищення від «шуму». Використовуючи функцію «re.sub», скрипт спочатку знаходить усі URL-адреси (за патерном r'https?://\S+|www\.\S+') і замінює їх на один пробіл. Це запобігає небажаному злиттю слів, які могли бути до і після посилання. Наступний рядок аналогічно видаляє HTML-теги, знаходячи будь-які послідовності в кутових дужках (r'<.*?>') і також замінюючи їх на пробіл.

На фінальному етапі виконується нормалізація пробілів. Спочатку «re.sub(r'\s+', ' ', text)» знаходить будь-які послідовності пробільних символів (включно з подвійними пробілами або новими рядками) і замінює їх на один пробіл. Одразу після цього метод «.strip()» видаляє будь-які залишкові пробіли на самому початку або в кінці рядка. Функція повертає очищений та нормалізований текстовий рядок, готовий до наступного етапу – токенізації BERT.

Після того, як були визначені дві різні стратегії очищення тексту, далі описано процес завантаження та початкового перетворення даних. Оскільки датасет LIAR має шість класів достовірності, першим кроком є створення функції «binarize_liar_label» (лістинг 2.3). Вона перетворює шість класів на два цільових (1 - правда, 0 - фейк) та відфільтровує неоднозначні проміжні значення.

Лістинг 2.3 – Спрощення класів LIAR

```
def binarize_liar_label(label):

    if label in ['true', 'mostly-true']:
        return 1

    if label in ['false', 'pants-on-fire']:
```

```
        return 0
    return -1
```

кінець лістингу 2.3

Функція приймає один аргумент «label» і виконує перевірку. Якщо мітка належить до категорій високої достовірності (‘true’ або ‘mostly-true’), функція повертає ціле число 1, що кодує «правду». Якщо ж мітка належить до категорій низької достовірності (‘false’ або ‘pants-on-fire’), вона повертає 0, що кодує «фейк». Для всіх інших, неоднозначних міток (half-true та barely-true), функція повертає -1. Це службове значення дозволить легко ідентифікувати та видалити ці проміжні записи з набору даних на наступному етапі завантаження.

Після перетворення класів, наступним кроком є створення основної функції «load_liar_partition» (лістинг 2.4). Це ядро процесу завантаження, яке відповідатиме за зчитування «.tsv» файлів, призначення імен стовпцям та застосування функції «binarize_liar_label» для фільтрації та підготовки даних.

Лістинг 2.4 – Обробка файлів LIAR

```
import pandas as pd
def load_liar_partition(filepath):
    try:
        df = pd.read_csv(
            filepath, sep='\t', header=None,
            names=[«id», «label», «statement», «subject», «speaker»,
«job_title»,
                   «state_info», «party_affiliation»,
«barely_true_counts»,
                   «false_counts», «half_true_counts»,
«mostly_true_counts»,
                   «pants_on_fire_counts», «context»]
        )
    except FileNotFoundError:
        print(f»ПОМИЛКА: Файл ‘{filepath}’ не знайдено.»)
        return None, None

    df[‘label_binary’] = df[‘label’].apply(binarize_liar_label)
    df_clean = df[df[‘label_binary’] != -1].copy()
    return df_clean[‘statement’].values, df_clean[‘label_binary’].values
```

кінець лістингу 2.4

Цей лістинг описує функцію «load_liar_partition», яка є основним інструментом для завантаження та початкової обробки даних датасету LIAR. Функція приймає один аргумент – шлях до файлу (filepath). Усередині функції спочатку реалізовано блок «try-except» для коректної обробки помилки «FileNotFoundError» (якщо файл не знайдено), що підвищує надійність скрипта. Основна операція завантаження виконується за допомогою функції «pd.read_csv» з бібліотеки «pandas». Оскільки LIAR є файлом формату TSV (де значення розділені табуляцією), явно вказано параметр «sep='\t'». Також, оскільки файли не містять рядка заголовків, вказано «header=None» і вручну надано список імен для всіх 14 стовпців (таких як «id», «label», «statement» тощо). Одразу після завантаження даних, скрипт створює новий стовпець «label_binary». Це досягається шляхом застосування (.apply) раніше визначеної функції «binarize_liar_label» (з лістингу 2.3) до кожного запису в оригінальному стовпці «label». Цей крок перетворює шість текстових класів на три числові: 1 (правда), 0 (фейк) або -1 (проміжне значення). Наступний рядок, «df_clean = ... «, виконує ключову фільтрацію. Він створює новий, чистий «DataFrame», копіюючи (.copy()) лише ті рядки, де значення «label_binary» не дорівнює -1. Таким чином, усі неоднозначні, проміжні класи (half-true, barely-true) ефективно видаляються з набору даних. Функція повертає (return) не весь DataFrame, а лише два масиви NumPy (.values), які необхідні для подальшого навчання: тексти заяв з колонки statement та відповідні їм бінарні мітки з колонки «label_binary».

Після того, як були визначені всі необхідні допоміжні функції (для бінаризації, завантаження та очищення), наступний крок (лістинг 2.5) їх практичне застосувати. Це етап, де дані фактично завантажуються в пам'ять і проходять відповідні процедури очищення.

Лістинг 2.5 – Фактичне завантаження даних

```
path_to_liar = «C:/Users/Nestors/Desktop/liar/»

X_train_texts, y_train = load_liar_partition(path_to_liar + «train.tsv»)
X_valid_texts, y_valid = load_liar_partition(path_to_liar + «valid.tsv»)
X_test_texts, y_test = load_liar_partition(path_to_liar + «test.tsv»)
```

кінець лістингу 2.5

Спочатку визначається змінна «path_to_liar», яка вказує на локальну директорію, де зберігаються файли датасету.

Далі, раніше визначена функція «load_liar_partition» (з лістингу 2.4) викликається тричі. Вона послідовно зчитує, бінаризує та фільтрує кожен із трьох файлів: «train.tsv», «valid.tsv» та «test.tsv».

В результаті виконання цього коду створюються шість змінних: три масиви містять не очищені тексти (X_train_texts, X_valid_texts, X_test_texts), а три відповідні масиви містять чисті бінарні мітки (y_train, y_valid, y_test). Ці дані є спільною вихідною точкою для подальшої, окремої обробки для різних моделей.

Після того, як не очищені дані завантажено в пам’ять, наступним кроком є їх очищення. Як було обґрунтовано раніше, різні архітектури моделей вимагають різного підходу до обробки, тому цей етап розділяється на дві паралельні стратегії.

Спочатку, для підготовки даних до моделей CNN та LSTM, застосовується агресивне очищення. Лістинг 2.6 демонструє, як раніше визначена функція «clean_text_aggressive» (з лістингу 2.1) застосовується до кожного набору текстів, створюючи оптимізований набір даних.

Лістинг 2.6 – Застосування агресивного очищення (для CNN та LSTM)

```
 X_train_cleaned_aggressive = [clean_text_aggressive(text) for text in
X_train_texts]
 X_valid_cleaned_aggressive = [clean_text_aggressive(text) for text in
X_valid_texts]
 X_test_cleaned_aggressive = [clean_text_aggressive(text) for text in
X_test_texts]
```

кінець лістингу 2.6

Цей код бере текстові дані, які були завантажені в пам’ять у лістингу 2.5 (масиви X_train_texts, X_valid_texts та X_test_texts), і послідовно обробляє їх. Використовуючи синтаксис list comprehension (генератор списку), скрипт ітерує по кожному окремому тексту в кожному з трьох наборів. До кожного тексту застосовується функція «clean_text_aggressive» (яка була описана в Лістингу 2.1). В результаті створюються три нові масиви (X_train_cleaned_aggressive, X_valid_cleaned_aggressive та X_test_cleaned_aggressive). Ці нові масиви містять

той самий текстовий контент, але вже у очищеному вигляді – без стоп-слів, пунктуації, та з проведеною лематизацією.

Після підготовки даних для моделей CNN та LSTM, необхідно виконати паралельну обробку даних для архітектури BERT. Ця модель потребує іншого, базового підходу до очищення, який зберігає лінгвістичну структуру тексту. Лістинг 2.7 показує, як ті ж самі дані (X_train_texts), що були завантажені в Лістингу 2.5, обробляються функцією clean_text_simple (з лістингу 2.2).

Лістинг 2.7 – Застосування базового очищення (для BERT)

```
 X_train_cleaned_simple = [clean_text_simple(text) for text in
X_train_texts]
 X_valid_cleaned_simple = [clean_text_simple(text) for text in
X_valid_texts]
 X_test_cleaned_simple = [clean_text_simple(text) for text in X_test_texts]
```

кінець лістингу 2.7

Тепер дані готові до фінального етапу попередньої обробки – токенізації. Цей процес перетворює текстові рядки на числовий формат, придатний для нейронної мережі. Як і у випадку з очищенням, стратегії токенізації для CNN/LSTM та BERT кардинально відрізняються. У лістингу 2.8, показано процес токенізації для моделей CNN та LSTM. Для цього використовується «Tokenizer» з бібліотеки «tensorflow.keras». Цей інструмент формує робочий словник моделі, обмежуючи його 10 000 найчастішими словами, які були знайдені виключно в агресивно очищених тренувальних текстах (X_train_cleaned_aggressive). Потім усі тексти перетворюються на послідовності числових індексів і, нарешті, вирівнюються до однакової довжини (max_len = 60) за допомогою паддінгу.

Лістинг 2.8 – Токенізація та Паддінг для CNN/LSTM

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_words = 10000
max_len = 60

tokenizer = Tokenizer(num_words=max_words, oov_token=«<OOV>«)
tokenizer.fit_on_texts(X_train_cleaned_aggressive)
```

```
 X_train_seq = tokenizer.texts_to_sequences(X_train_cleaned_aggressive)
 X_valid_seq = tokenizer.texts_to_sequences(X_valid_cleaned_aggressive)
 X_test_seq = tokenizer.texts_to_sequences(X_test_cleaned_aggressive)

 X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post',
truncating='post')
 X_valid_pad = pad_sequences(X_valid_seq, maxlen=max_len, padding='post',
truncating='post')
 X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post',
truncating='post')
```

кінець лістингу 2.8

На початку імпортуються необхідні інструменти з Keras. «Tokenizer» для перетворення слів у числа та «pad_sequences» для вирівнювання довжини. Далі визначаються два ключові гіперпараметри: «max_words» та «max_len».Значення «max_words = 10000» є обґрунтованим компромісом для розміру словника. Це стандартний вибір для завдань NLP, який дозволяє охопити переважну більшість семантично значущих слів, водночас відсікаючи рідкісні слова та опечатки. Це зменшує ризик перенавчання моделі та знижує обчислювальну складність. Значення «max_len = 60» було обрано на основі статистичного аналізу довжин текстів у тренувальному наборі LIAR. Оскільки датасет складається з коротких заяв, аналіз показує, що довжина в 60 токенів є оптимальною. Вона охоплює повний зміст переважної більшості заяв і водночас мінімізує кількість доданого паддінгу, що робить навчання більш ефективним. Після визначення цих гіперпараметрів, ініціалізується «Tokenizer» з токеном <OOV> для позначення невідомих слів. Далі «tokenizer.fit_on_texts(X_train_cleaned_aggressive) « формує частотний словник, аналізуючи виключно агресивно очищені тренувальні тексти. Це запобігає витоку даних з тестових та валідаційних наборів у процес навчання. Після того, як словник сформовано, метод «texts_to_sequences» застосовується до всіх трьох наборів даних (тренувального, валідаційного та тестового), перетворюючи кожен текст на послідовність цілих чисел. На завершальному етапі, функція «pad_sequences» вирівнює ці послідовності до фіксованої довжини «max_len = 60». Параметри «padding='post'» та «truncating='post'» вказують, що нулі додаватимуться в кінець коротших послідовностей, а довші будуть обрізані

також з кінця. В результаті виконання цього лістингу, ми отримуємо готові до навчання числові матриці: «X_train_pad», «X_valid_pad» та «X_test_pad».

На відміну від CNN та LSTM, токенізація даних для архітектури BERT є значно складнішою і вимагає іншого підходу. BERT використовує власний попередньо навчений токенізатор та має жорсткі вимоги до вхідного формату (зокрема, потрібні спеціальні керуючі токени та маски уваги). Для цього використовується «DistilBertTokenizer». Лістинг 2.9 демонструє, як цей інструмент застосовується до базово очищених текстів (X_train_cleaned_simple), виконуючи за один крок і токенізацію, і паддінг, і створення масок уваги.

Лістинг 2.9 – Токенізація та паддінг для BERT

```
 from transformers import DistilBertTokenizer

 model_name = 'distilbert-base-uncased'
 bert_tokenizer = DistilBertTokenizer.from_pretrained(model_name)
 max_len = 60

 def tokenize_data_for_bert(texts, tokenizer, max_len):
     # 'texts' - це вже список
     encodings = tokenizer.batch_encode_plus(
         texts,
         add_special_tokens=True, max_length=max_len,
         padding='max_length', truncation=True, return_tensors='tf'
     )
     return {'input_ids': encodings['input_ids'], 'attention_mask':
encodings['attention_mask']}

 X_train_dict = tokenize_data_for_bert(X_train_cleaned_simple,
bert_tokenizer, max_len)
 X_valid_dict = tokenize_data_for_bert(X_valid_cleaned_simple,
bert_tokenizer, max_len)
 X_test_dict = tokenize_data_for_bert(X_test_cleaned_simple,
bert_tokenizer, max_len)
```

кінець лістингу 2.9

Для токенізації BERT використовується «DistilBertTokenizer» з бібліотеки «Transformers». Цей інструмент є попередньо навченим і має фіксований словник WordPiece, тому його не потрібно навчати на наших даних. Основна логіка реалізована у функції «tokenize_data_for_bert», яка за один виклик «tokenizer.batch_encode_plus» виконує три критичні дії. Здійснюється токенізація

та додавання спеціальних токенів. Текст перетворюється на числовий індекс, при цьому автоматично додаються необхідні керуючі токени (наприклад, [CLS] на початку та [SEP] для розділення). Послідовності вирівнюються до довжини «max_len = 60» (параметр padding='max_length'), і довші тексти обрізаються (truncation=True). Створюються маски уваги (Attention Masks). Числовий масив, що створюється за допомогою «return_tensors='tf'», вказує моделі BERT, які токени є реальними словами (значення 1), а які – доповненням (значення 0). Маски уваги є критично важливим елементом, необхідним для коректної роботи механізму уваги в архітектурі трансформера. В результаті виконання цього лістингу, ми отримуємо три словники вхідних даних (X_train_dict, X_valid_dict, X_test_dict), де кожен містить два ключі (input_ids та attention_mask), готових для подачі на вхід моделі BERT.

### 2.3 Вибір архітектури моделей для класифікації новин

Вибір архітектури нейронної мережі є критичним етапом проєктування, оскільки він визначає здатність моделі ефективно вилучати семантичні та синтаксичні ознаки з текстових даних і формувати точну класифікаційну ознаку. З огляду на обсяг та складність підготовленого датасету, доцільно використовувати методи глибокого навчання (Deep Learning, DL).

Для забезпечення всебічного аналізу та вибору найбільш ефективного рішення в рамках цієї роботи буде розглянуто, реалізовано та порівняно три архітектури, які є провідними у сфері обробки природної мови (NLP). Всі архітектури працюють не з сирим текстом, а з його числовим представленням у вигляді векторів слів (ембедингів). Для CNN та LSTM будуть використані попередньо навчені ембединги або шар Embedding, що навчається разом з моделлю. Трансформери, такі як BERT, генерують власні ембединги під час обробки тексту. Такий підхід дозволить об'єктивно оцінити компроміс між обчислювальною складністю та точністю.

Згорткові мережі (Convolutional Neural Networks) будуть використані як ефективна базова модель (Baseline), оскільки вони вимагають менше

обчислювальних ресурсів порівняно з рекурентними моделями. Вони здатні виявляти локальні, позиційно-інваріантні ознаки в тексті. Використовується один шар 1D-згортки з фіксованим розміром ядра, яке проходить по векторизованому тексту, виявляючи важливі послідовності слів (п’ятиграми) або ключові фрази незалежно від їхнього розташування в новині. До переваг можна віднести високу швидкість навчання та інференсу, ефективність у виявленні сильних локальних лексичних маркерів, що є важливим для фейкового контенту (наприклад, clickbait-заголовки). Після шару згортки застосовується шар глобального максимального пулінгу (Global Max Pooling) для зменшення розмірності та виділення найбільш значущих ознак.

Рекурентні мережі, зокрема їхній вдосконалений варіант LSTM (Long Short-Term Memory), є класичним підходом для моделювання послідовностей і контекстних залежностей. Вони дозволяють аналізувати текст із урахуванням порядку слів. Модель обробляє послідовність слів покроково, використовуючи механізм внутрішньої пам’яті (комірки та вентилі) для збереження інформації про попередні слова. Для повноцінного розуміння контексту буде застосовано двонаправлену архітектуру. З переваг можна виділити здатність запам’ятовувати та моделювати довгострокові залежності між віддаленими словами, що важливо для аналізу структури повноцінних новинних статей (поле text з датасету). Буде розроблено двошарову модель Bi-LSTM (Bidirectional LSTM). Використання двонаправленого підходу дозволить обробляти послідовність як у прямому, так і у зворотному напрямках, забезпечуючи більш глибоке контекстуальне розуміння кожного слова, необхідне для розпізнавання сарказму чи прихованої маніпуляції.

Трансформерні архітектури, що ґрунтуються на механізмі уваги (Attention Mechanism), є найбільш сучасним і продуктивним підходом у NLP, що робить їх еталонною моделлю (SOTA). Модель використовує механізм Self-Attention для одночасного моделювання глобальних залежностей. Кожне слово в послідовності оцінюється з точки зору його важливості для всіх інших слів, що дозволяє генерувати якісні контекстно-залежні вбудовування. Найвища точність класифікації завдяки двонаправленому та глобальному контекстуальному аналізу, а також можливість використання трансферного навчання на попередньо навчених

моделях. Буде використовуватися адаптована модель на основі архітектури BERT (Bidirectional Encoder Representations from Transformers), або її оптимізований варіант (наприклад, DistilBERT), що дозволить використати знання, отримані моделлю під час попереднього навчання на великих корпусах, та донавчити її на цільовому датасеті новин.

Для забезпечення об'єктивного порівняння та ефективної реалізації, усі три архітектури (CNN, Bi-LSTM та DistilBERT) використовують єдину стандартизовану параметризацію. Фіксована довжина послідовності (max_len = 60) була обрана на основі аналізу датасету LIAR, оскільки вона охоплює переважну більшість коротких заяв. Розмір робочого словника для CNN та Bi-LSTM обмежено (max_words = 10000), що дозволяє охопити семантичне ядро даних, відсікаючи рідкісні слова, які можуть спричинити перенавчання. Всі моделі використовують вектори вбудовування розміром 128 вимірів (embedding_dim = 128), а класифікатор для всіх трьох архітектур є ідентичним (з шарами Dense(32) та Dropout(0.4)), що гарантує, що будь-яка різниця в результатах буде безпосередньо пов'язана з ефективністю базової архітектури (згорткової, рекурентної чи трансформерної).

### 2.4 Реалізація моделі в середовищі програмування

У цьому розділі описано практичне втілення обраних архітектур.Спочатку було реалізовано архітектури, які навчаються з нуля – CNN та Bi-LSTM, оскільки вони використовують спільну логіку токенізації та спільну стратегію навчання.

CNN є базовою, але ефективною, і складається з шару Embedding, за яким слідує шар 1D-згортки для виявлення локальних ознак у тексті. Архітектура моделі продемонстрована в лістингу 2.10.

Лістинг 2.10 – Визначення архітектури CNN

```
 from tensorflow.keras.models import Sequential
 from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,
Dense, Dropout
 max_words = 10000
```

```
max_len = 60
embedding_dim = 128

def create_cnn_model():
    model = Sequential()
    model.add(Embedding(max_words, embedding_dim, input_length=max_len))
    model.add(Conv1D(filters=128, kernel_size=5, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

кінець лістингу 2.10

Спочатку імпортуються необхідні класи: «Sequential» для послідовного побудування моделі та основні шари, такі як «Embedding», «Conv1D», «GlobalMaxPooling1D» та «Dense» з бібліотеки «tensorflow.keras.layers». Далі визначаються ключові гіперпараметри: «max_words» встановлює розмір робочого словника на 10000. «max_len» фіксує довжину вхідної послідовності на 60 токенів. «embedding_dim» задає розмір вектора вбудовування кожного слова на 128 вимірів. Сама функція «create_cnn_model()» починається зі створення об'єкта «model = Sequential()», що дозволяє будувати нейронну мережу шляхом послідовного додавання шарів.

Першим додається шар «Embedding». Він перетворює числовий індекс кожного слова на щільний вектор розміром 128 вимірів, і цей шар навчається з нуля під час тренування. Наступним є основний шар – «Conv1D». Він використовує 128 фільтрів з ядром розміром 5 (kernel_size=5), які проходять по тексту, виявляючи локальні ознаки або п'ятиграми. За ним слідує шар «GlobalMaxPooling1D()», який агрегує ознаки, вибираючи максимальне значення з кожного фільтра, що зменшує розмірність і фіксує найбільш сильні ознаки.

Далі починається фінальний класифікаційний блок. Перший шар «Dropout(0,4)» встановлює рівень 40 % вимкнення нейронів, що є механізмом регуляризації для запобігання перенавчанню. Потім додається повнозв'язний прихований шар «Dense(32, activation='relu')» з 32 нейронами. Після ще одного шару «Dropout(0,4)», фінальний вихідний шар «Dense(1, activation='sigmoid')»

використовує один нейрон і сигмоїдну функцію активації, щоб повернути ймовірність того, що заява належить до класу «правда» (1). Ймовірність, близька до 0, автоматично інтерпретується як клас «фейк» (0). Функція завершується поверненням «return model», тобто повністю визначеного об’єкта моделі.

Після реалізації архітектури CNN, наступним кроком є визначення Bi-LSTM – моделі, призначеної для аналізу послідовностей.

Bi-LSTM (лістинг 2.11) показує код для Bi-LSTM. Ця архітектура використовує двошарову двонаправлену структуру (Bidirectional) для кращого моделювання довгострокових залежностей та глибинного аналізу послідовності слів.

Лістинг 2.11 – Визначення архітектури Bi-LSTM

```
 from tensorflow.keras.models import Sequential
 from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense,
Dropout

 max_words = 10000
 max_len = 60
 embedding_dim = 128

 def create_bilstm_model():
     model = Sequential()
     model.add(Embedding(max_words, embedding_dim, input_length=max_len))

     model.add(Bidirectional(LSTM(64, return_sequences=True)))
     model.add(Bidirectional(LSTM(32, return_sequences=False)))
     model.add(Dropout(0.4))
     model.add(Dense(32, activation='relu'))
     model.add(Dropout(0.4))
     model.add(Dense(1, activation='sigmoid'))

     return model
```

кінець лістингу 2.11

На початку імпортуються необхідні класи: «Sequential» для послідовного побудування моделі та основні шари, включаючи «Embedding», «Bidirectional» (для двонаправленої обгортки), «LSTM» та «Dense» з бібліотеки «tensorflow.keras.layers». Далі визначаються ключові гіперпараметри, спільні з CNN: «max_words (10000)», «max_len (60)» та «embedding_dim (128)». Сама

функція «create_bilstm_model()» починається зі створення об’єкта «model = Sequential()». Першим додається шар «Embedding». Він перетворює числовий індекс кожного слова на щільний вектор розміром 128 вимірів. Цей шар навчається з нуля під час тренування, створюючи оптимальне представлення слів для LIAR.

Наступними додаються два послідовні шари Bi-LSTM. Перший шар (model .add(Bidirectional(LSTM(64, return_sequences=True)))) використовує 64 приховані одиниці LSTM і є двонаправленим, що дозволяє обробляти послідовність як у прямому, так і у зворотному напрямках. Параметр «return_sequences=True» змушує цей шар повертати повну послідовність (вектор для кожного токена), яка є необхідною вхідною інформацією для наступного рекурентного-шару. Другий шар (model.add(Bidirectional(LSTM(32,-return_sequences=False)))) приймає повну послідовність від попереднього шару і агрегує її в єдиний фінальний вектор (оскільки return_sequences=False). Цей фінальний вектор є повним контекстуальним представленням всього речення і передається далі для класифікації.

Далі йде фінальний класифікаційний блок, який є ідентичним для всіх архітектур. Він починається з шару «Dropout(0,4)», що встановлює 40 % вимкнення нейронів для регуляризації. Потім додається повнозв’язний прихований шар «Dense(32, activation=‘relu’)» з 32 нейронами. Після ще одного шару «Dropout(0,4), фінальний вихідний шар «Dense(1, activation=‘sigmoid’)» використовує сигмоїдну функцію активації, щоб повернути ймовірність того, що заява належить до класу «правда» (1). Ймовірність, близька до 0, автоматично інтерпретується як клас «фейк» (0). Функція завершується поверненням повністю визначеного об’єкта model.

Після реалізації архітектур CNN та Bi-LSTM, що навчаються з нуля, наступним кроком є реалізація моделі DistilBERT (лістинг 2.12). Ця модель реалізується за допомогою стратегії Fine-Tuning (донавчання), де до попередньо навченої базової моделі додаються класифікаційні шари (ідентичні шарам CNN/Bi-LSTM). Це дозволяє використати знання, отримані моделлю під час попереднього навчання на великих корпусах, та донавчити її на цільовому датасеті LIAR.

Лістинг 2.12 – Визначення архітектури DistilBERT

```
 from transformers import TFDistilBertModel
 from tensorflow.keras.layers import Input, Dense, Dropout,
GlobalAveragePooling1D
 from tensorflow.keras.models import Model

 model_name = 'distilbert-base-uncased'
 max_len = 60

 def create_bert_model():

     distilbert = TFDistilBertModel.from_pretrained(model_name,
use_safetensors=False)

     input_ids = Input(shape=(max_len,), dtype='int32', name='input_ids')
     attention_mask = Input(shape=(max_len,), dtype='int32',
name='attention_mask')

     embeddings = distilbert([input_ids, attention_mask])[0]

     out = GlobalAveragePooling1D()(embeddings)

     x = Dropout(0.4)(out)
     x = Dense(32, activation='relu')(x)
     output = Dropout(0.4)(x)
     output = Dense(1, activation='sigmoid', name='output')(output)

     model = Model(inputs=[input_ids, attention_mask], outputs=output)

     distilbert.trainable = True

     return model
```

кінець лістингу 2.12

Спочатку імпортуються необхідні класи з бібліотек «Transformers» та «tensorflow.keras.layers». На початку функції відбувається завантаження попередньо навченої базової моделі «TFDistilBertModel.from_pretrained(model_ name, use_safetensors=False)». Ця модель уже здатна обробляти мову. Оскільки трансформери вимагають складнішого входу, ніж CNN/LSTM, визначаються два окремі вхідні шари (Input): «input_ids» (числові індекси токенів) та «attention_mask» (маска уваги, що вказує, які токени є словами, а які – паддінгом).Вхідні дані передаються до завантаженої моделі «distilbert». Отриманий вихід (embeddings) далі агрегується за допомогою шару «GlobalAveragePooling1D()». Цей шар зводить

усі контекстно-залежні вектори, отримані з трансформера, в єдиний фінальний вектор представлення всього речення, який і передається класифікатору.

Далі йде фінальний класифікаційний блок, який є ідентичним для всіх трьох моделей (CNN та Bi-LSTM) і складається з двох повнозв'язних шарів (Dense(32)) з Dropout(0.4) для регуляризації. Фінальний вихідний шар «Dense(1, activation='sigmoid')» повертає ймовірність належності до класу «правда» (1). Ймовірність, близька до 0, автоматично інтерпретується як клас «фейк» (0). Рядок «distilbert.trainable = True» є критичним. Він вмикає Fine-Tuning, дозволяючи оновлювати всі параметри базової моделі DistilBERT разом із фінальним класифікаційним блоком під час навчання. Функція завершується поверненням повністю визначеного об'єкта «model».

Після успішного визначення архітектур трьох моделей, наступним етапом є їхня компіляції та навчання. Для забезпечення об'єктивного порівняння, моделі CNN та Bi-LSTM, які навчаються з нуля, використовують спільну процедуру налаштування Keras. Лістинг 2.13 демонструє налаштування гіперпараметрів, а також процедуру компіляції та навчання для архітектур CNN і Bi-LSTM.

Лістинг 2.13 – Код

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=7,
    restore_best_weights=True,
    verbose=1
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=4,
    min_lr=0.00001,
    verbose=1
)

callbacks = [early_stop, reduce_lr]

learning_rate_base = 0.001
```

```
 optimizer = Adam(learning_rate=learning_rate_base)
 loss_func = ‘binary_crossentropy’

 cnn_model = create_cnn_model()
 cnn_model.compile(optimizer=optimizer, loss=loss_func,
metrics=[‘accuracy’])

 lstm_model = create_bilstm_model()
 lstm_model.compile(optimizer=optimizer, loss=loss_func,
metrics=[‘accuracy’])
```

кінець лістингу 2.13

На початку коду відбувається імпорт класів «EarlyStopping» та «ReduceLROnPlateau» з бібліотеки «tensorflow.keras.callbacks». Ці механізми, відомі як Callbacks, призначені для контролю процесу навчання. Функція «Early Stopping (early_stop)» відстежує втрати на валідаційному наборі (val_loss) і припиняє тренування, якщо покращення не спостерігається протягом семи епох (patience=7), автоматично повертаючи найкращі ваги моделі. Паралельно, «ReduceLROnPlateau (reduce_lr)» динамічно зменшує швидкість навчання (на 80 % через factor=0,2), якщо валідаційна втрата не покращується протягом чотирьох епох, що допомагає стабілізувати сходимість. Обидва механізми об’єднуються у спільний список «callbacks».

Далі визначаються ключові гіперпараметри для навчання: встановлюється початкова швидкість навчання «learning_rate_base = 0,001», використовується оптимізатор «Adam» та функція втрат «binary_crossentropy» (бінарна крос-ентропія). На завершення, обидві моделі, «cnn_model» та «lstm_model» компілюються з ідентичними параметрами. Це гарантує, що відмінності в результатах, будуть пов’язані виключно з архітектурою, а не з налаштуваннями навчання.

Після того, як моделі CNN та Bi-LSTM були успішно скомпільовані та налаштовані механізмами контролю, розпочинається сам процес навчання. Процес навчання показано у лістингу 2.14.

Лістинг 2.14 – Навчання моделей CNN та Bi-LSTM

```
 epochs = 50
```

```
batch_size = 128

print(«Початок навчання CNN...»)
cnn_history = cnn_model.fit(
    X_train_pad,        # Використовує агресивно очищені послідовності
    y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_valid_pad, y_valid),
    callbacks=callbacks,
    verbose=1
)

print(«\nПочаток навчання Bi-LSTM...»)
lstm_history = lstm_model.fit(
    X_train_pad,        # Використовує ті ж самі дані та Callbacks
    y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_valid_pad, y_valid),
    callbacks=callbacks,
    verbose=1
)
```

кінець лістингу 2.14

Спочатку визначаються два ключові гіперпараметри для процесу «fit: epochs = 50» – максимальна кількість повних ітерацій навчання, та «batch_size = 128» – кількість прикладів, що обробляються за один раз. Далі відбувається безпосередній запуск навчання для обох моделей. Команди «cnn_model.fit» та «lstm_model.fit» запускають ітераційний процес оптимізації. Обидві моделі використовують ідентичні вхідні дані: «X_train_pad» (агресивно очищені та вирівняні послідовності) та «y_train» (бінарні мітки). Параметр «validation_data= (X_valid_pad, y_valid)» вказує моделі використовувати валідаційний набір для обчислення втрат після кожної епохи. Фінальний елемент, «callbacks=callbacks», передає в процес навчання механізми контролю (EarlyStopping та ReduceLROnPlateau), які гарантують, що навчання зупиниться, як тільки продуктивність перестане покращуватися, та що швидкість навчання буде динамічно регулюватися. В результаті виконання цього коду створюються два об’єкти «cnn_history» та «lstm_history», які містять повну історію навчання обох моделей.

Процедура реалізації для моделі DistilBERT (лістинг 2.15) є окремою, оскільки вона використовує інший формат вхідних даних та специфічні налаштування оптимізатора. На відміну від CNN та Bi-LSTM, для цієї архітектури застосовується стратегія Fine-Tuning, що вимагає вкрай низької швидкості навчання для збереження знань, отриманих моделлю під час попереднього тренування.

Лістинг 2.15 – Імпорт та присвоєння міток

```
 from tensorflow.keras.optimizers import Adam
 from tensorflow.keras.callbacks import EarlyStopping

 bert_model = create_bert_model()
 optimizer_bert = Adam(learning_rate=3e-5) # 0.00003
 loss_func = ‘binary_crossentropy’

 bert_model.compile(optimizer=optimizer_bert, loss=loss_func,
metrics=[‘accuracy’])

 epochs_bert = 20
 batch_size_bert = 32

 print(«Початок навчання BERT (Fine-Tuning)...»)
 bert_history = bert_model.fit(
     X_train_dict,
     y_train,
     epochs=epochs_bert,
     batch_size=batch_size_bert,
     validation_data=(X_valid_dict, y_valid),
     callbacks=[early_stop],
     verbose=1
 )
```

кінець лістингу 2.15

Спочатку створюється об’єкт моделі DistilBERT викликом функції «create_bert_model()». На етапі компіляції використовується оптимізатор Adam, але зі специфічно низькою швидкістю навчання. Ця низька швидкість є критичною для стратегії Fine-Tuning, оскільки вона запобігає руйнуванню знань, які модель отримала під час попереднього тренування на великих корпусах. Функція втрат встановлюється на «binary_crossentropy». Далі запускається процес донавчання (bert_model.fit). Параметри навчання тут відрізняються від Keras-моделей.

Використовується менша максимальна кількість епох (20) та менший розмір батчу (32). Менший «batch_size» є стандартом для трансформерів для зменшення навантаження на пам’ять. DistilBERT навчається на спеціальному форматі вхідних даних – словнику «X_train_dict» і використовує лише «EarlyStopping» для контролю перенавчання, оскільки трансформери зазвичай не потребують динамічного зменшення швидкості навчання. В результаті виконання цього коду створюється об’єкт «bert_history», що містить історію донавчання моделі DistilBERT.

### 2.5 Валідація та тестування нейронної мережі

Після успішного завершення етапу навчання, коли всі три моделі навчені, необхідно перейти до об’єктивної оцінки їхньої продуктивності. Основною метою є оцінити здатність кожної архітектури до узагальнення, тобто наскільки добре модель справляється з даними, які вона не бачила під час навчання. Тестування проводиться виключно на тестовому наборі (X_test), який був ізольований на етапі попередньої обробки.

Для всебічної оцінки продуктивності в умовах бінарної класифікації було обрано наступний набір ключових метрик, оскільки лише точність є недостатньо інформативною. Точність – базовий показник, що відображає загальну частку правильних передбачень.Матриця змішування надає деталізовану картину помилок, дозволяючи кількісно оцінити кількість хибнопозитивних (False Positi ves) та хибнонегативних (False Negatives) результатів. Це є критичним для розуміння надійності системи. Precision, recall, f1-score є найбільш значущими. F1-Score є особливо важливим, оскільки він являє собою гармонійне середнє між точністю та повнотою, забезпечуючи найкращий загальний показник ефективності моделі на збалансованих класах.

Процедура оцінки для моделей CNN та Bi-LSTM описана в лістингу 2.16.

Лістинг 2.16 – Оцінка ефективності моделей CNN та Bi-LSTM

```
 from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

 print(«--- Оцінка моделі CNN ---»)

 y_pred_prob_cnn = cnn_model.predict(X_test_pad)

 y_pred_cnn = (y_pred_prob_cnn > 0.5).astype(int)

 cnn_accuracy = accuracy_score(y_test, y_pred_cnn)
 cnn_report = classification_report(y_test, y_pred_cnn, digits=4)
 cnn_matrix = confusion_matrix(y_test, y_pred_cnn)

 print(f»Точність CNN (Accuracy): {cnn_accuracy:.4f}»)
 print(«Матриця змішування CNN:\n», cnn_matrix)
 print(«Звіт класифікації CNN:\n», cnn_report)


 print(«\n--- Оцінка моделі Bi-LSTM ---»)

 y_pred_prob_lstm = lstm_model.predict(X_test_pad)

 y_pred_lstm = (y_pred_prob_lstm > 0.5).astype(int)

 lstm_accuracy = accuracy_score(y_test, y_pred_lstm)
 lstm_report = classification_report(y_test, y_pred_lstm, digits=4)
 lstm_matrix = confusion_matrix(y_test, y_pred_lstm)

 print(f»Точність Bi-LSTM (Accuracy): {lstm_accuracy:.4f}»)
 print(«Матриця змішування Bi-LSTM:\n», lstm_matrix)
 print(«Звіт класифікації Bi-LSTM:\n», lstm_report)
```

кінець лістингу 2.16

На початку коду імпортуються необхідні метрики з бібліотеки «scikit-learn»: «accuracy_score, classification_report» та «confusion_matrix». Далі процес оцінки виконується ідентично для обох моделей, що є ключовим для об’єктивного порівняння. Спочатку для моделі CNN викликається функція «cnn_model. predict(X_test_pad)», яка отримує прогнози у вигляді ймовірностей належності. Ці ймовірності потім конвертуються в бінарний клас (y_pred_cnn) шляхом застосування порогу 0,5. Де ймовірності, більші за 0,5, стають 1 (правда), а менші – 0 (фейк). Після цього розраховуються і виводяться всі фінальні метрики. Загальна точність (cnn_accuracy), деталізована матриця змішування (cnn_matrix) та

повний звіт класифікації (cnn_report), який включає precision, recall та f1-score. Абсолютно та ж сама послідовність операцій виконується для моделі Bi-LSTM, що забезпечує отримання її фінальних результатів (lstm_accuracy, lstm_report, lstm_ matrix).

Обчислення метрик (лістинг 2.17) для моделі DistilBERT вимагає окремого лістингу, оскільки використовує інший формат вхідних даних.

Лістинг 2.17 – Оцінка ефективності моделі DistilBERT

```
 from sklearn.metrics import accuracy_score, classification_report, 
confusion_matrix

 print(«\n--- Оцінка моделі DistilBERT ---»)

 y_pred_prob_bert = bert_model.predict(X_test_dict)

 y_pred_bert = (y_pred_prob_bert > 0.5).astype(int)

 bert_accuracy = accuracy_score(y_test, y_pred_bert)
 bert_report = classification_report(y_test, y_pred_bert, digits=4)
 bert_matrix = confusion_matrix(y_test, y_pred_bert)

 print(f»Точність DistilBERT (Accuracy): {bert_accuracy:.4f}»)
 print(«Матриця змішування DistilBERT:\n», bert_matrix)
 print(«Звіт класифікації DistilBERT:\n», bert_report)
```

кінець лістингу 2.17

Цей лістинг демонструє фінальну процедуру оцінки моделі DistilBERT. На відміну від CNN та Bi-LSTM, ця модель приймає на вхід словник «X_test_dict», який містить два ключі: «input_ids» та «attention_mask».Спочатку викликається «bert_model.predict» для отримання прогнозів у вигляді ймовірностей. Ці ймовірності потім конвертуються в бінарні класи (y_pred_bert) з використанням порогу 0,5. На завершення, обчислюються всі фінальні метрики.

# РОЗДІЛ 3
# ОПТИМІЗАЦІЯ МОДЕЛІ ТА ІНТЕРПРЕТАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 3.1 Порівняння продуктивності використання різних моделей

На попередніх етапах було успішно реалізовано та навчено три моделі нейронних мереж (CNN, Bi-LSTM та DistilBERT) на датасеті LIAR. Цей підрозділ присвячений критичному аналізу та інтерпретації отриманих результатів, а також об'єктивному порівнянню ефективності архітектур.

Аналіз продуктивності розпочинається з фіксації ключових метрик, обчислених на тестовому наборі для кожної архітектури. Порівняння є об'єктивним, оскільки всі три моделі були протестовані на ідентичних даних та з використанням спільної стратегії попередньої обробки. Завдання виявлення фейкових новин є бінарною класифікацією, основна увага приділяється метрикам, які є стійкими до дисбалансу класів і забезпечують найкращий компроміс між Precision (точністю) та Recall (повнотою). Тому ключовим показником ефективності обрано F1-Score. Для наочності та фіксації результатів, отримані показники Точності (Accuracy), Precision, Recall та F1-Score зведені в таблицю 3.1.

Таблиця 3.1 – Порівняння ключових метрик продуктивності моделей

| Модель | Точність (Accuracy) | Precision | Recall | F1-Score |
|---|---|---|---|---|
| CNN | 65,62 % | 0,67 | 0,66 | 0,60 |
| Bi-LSTM | 66,52 % | 0,65 | 0,67 | 0,61 |
| DistilBERT | 68,62 % | 0,72 | 0,84 | 0,67 |

Таблиця 3.1 показує усереднені результати, однак для глибокого розуміння поведінки моделей необхідно проаналізувати їхні повні звіти класифікації.

Архітектура CNN (рисунок 3.1) показала найнижчу продуктивність. Загальна точність склала 65,62 %, а зважений F1-Score – 0,60. Аналіз матриці змішування [[46 203] [37 412]] показує асиметрію в її поведінці. З одного боку, модель продемонструвала високу ефективність у розпізнаванні класу «правда». Вона правильно ідентифікувала 412 правдивих новин і допустила лише 37 помилок, досягнувши високого Recall (0,92) для цього класу. З іншого боку, модель

виявилася майже нездатною ідентифікувати клас «фейк». Вона правильно розпізнала лише 46 фейків, тоді як 203 фейкові новини помилково класифікувала як правдиві. Модель CNN продемонструвала сильну упередженість до класу «правда».

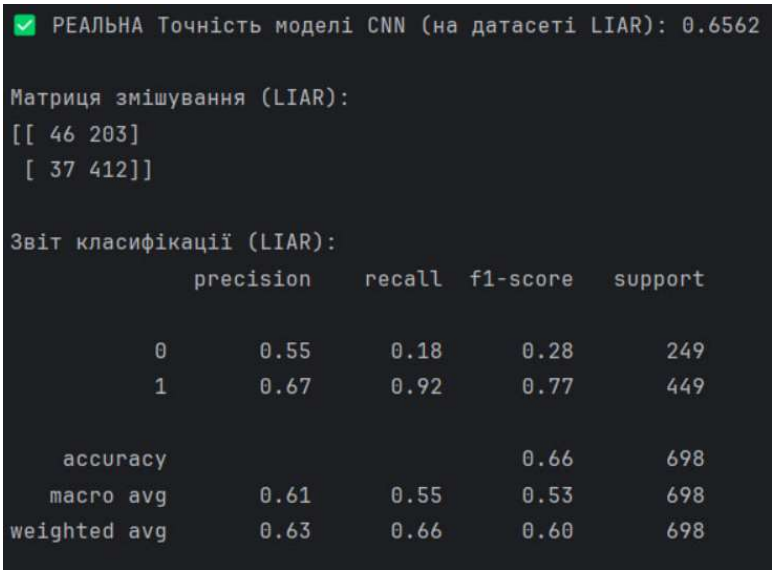


Рисунок 3.1 – Ключові метрики моделі CNN

Архітектура Bi-LSTM (рисунок 3.2) продемонструвала незначне покращення порівняно з CNN, досягнувши зваженого F1-Score – 0,61. Проте її звіту та матриці змішування [[48 201] [32 417]] показує, що вона страждає від тієї ж асиметрії, що й CNN. З одного боку, модель Bi-LSTM виявилася ще ефективнішою у розпізнаванні класу «правда». Вона правильно ідентифікувала 417 правдивих новин і допустила лише 32 помилки. Це забезпечило їй чудовий Recall (0,93) для цього класу. З іншого боку, як і CNN, вона майже повністю провалила ідентифікацію класу «фейк». Вона правильно розпізнала лише 48 фейків, тоді як 201 фейкову новину помилково класифікувала як правдиву. Хоча Bi-LSTM показала незначну перевагу над CNN, вона також продемонструвала сильну упередженість до класу «правда». Її низький F1-Score (0,61) є прямим наслідком нездатності розпізнати неправдиві заяви, що робить її однаково непридатною для надійної класифікації.

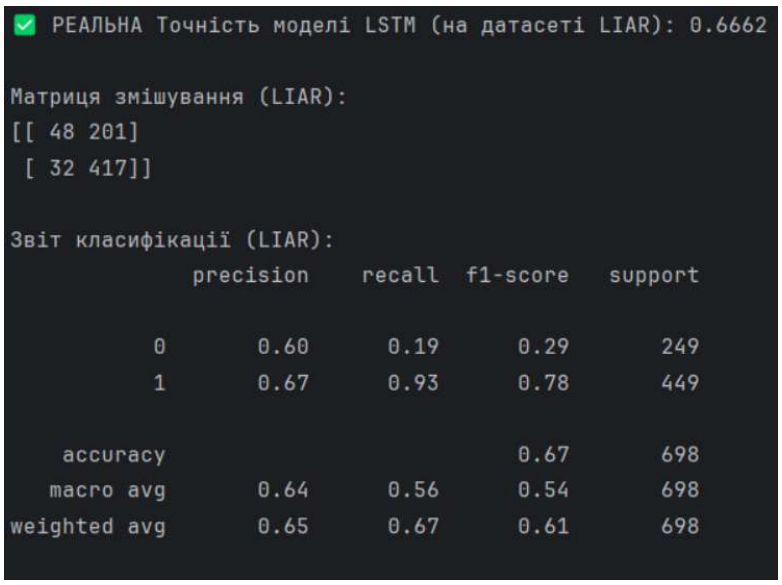


Рисунок 3.2 – Ключові метрики моделі BI-LSTM

Архітектура DistilBERT (рисунок 3.3) показала якісно вищий результат. Загальна точність (68,62 %) та F1-Score (0,67) є найвищими серед трьох моделей.

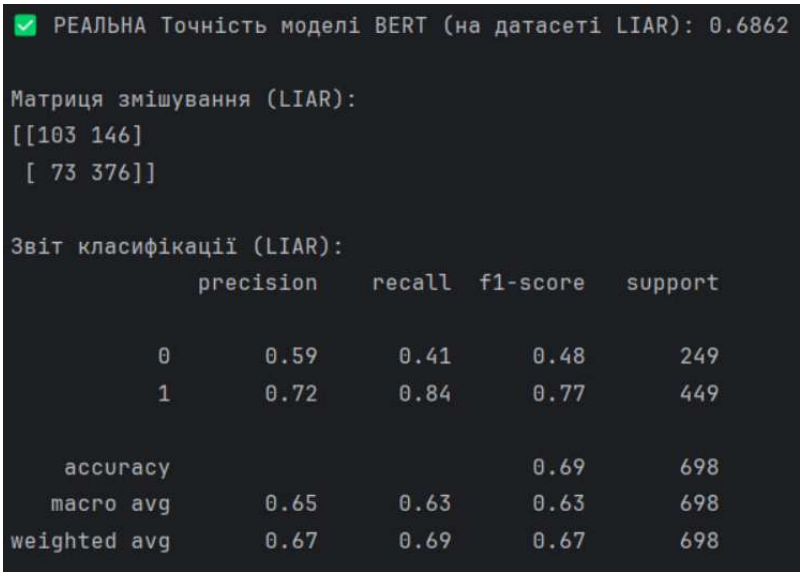


Рисунок 3.3 – Ключові метрики моделі DistilBERT

Аналіз її матриці змішування [[103 146] [73 376]] демонструє фундаментальну відмінність у її поведінці порівняно з CNN та Bi-LSTM. Модель змогла правильно ідентифікувати 103 фейки, що є більш ніж удвічі кращим показником, ніж у CNN (46) та Bi-LSTM (48). Хоча кількість хибнопозитивних помилок все ще значна, вона є набагато нижчою, ніж у CNN та Bi-LSTM. Водночас,

модель коректно знайшла 376 правдивих новин, допустивши 73 помилки. DistilBERT досяг кращого загального F1-Score, оскільки він пожертвував невеликою часткою точності у знаходженні класу «правда», але натомість кардинально покращив свою здатність знаходити клас «фейк». Це доводить, що трансформер здатний до більш глибокого семантичного аналізу та досягнення кращого балансу між двома класами, на відміну від CNN та Bi-LSTM, які просто схилялися до класифікації заяв як достовірних.

### 3.2 Інтерпретація отриманих результатів

У попередньому підрозділі було доведено, що архітектура DistilBERT має явну перевагу над традиційним моделями CNN та Bi-LSTM. Цей підрозділ присвячений глибокому аналізу та інтерпретації цих результатів, щоб пояснити, чому виникла така суттєва різниця в продуктивності. Основний фокус аналізу буде на тому, як кожна модель обробляла обидва класи, оскільки просте порівняння загальної точності приховує фундаментальні вади традиційних підходів.

Незважаючи на різницю в архітектурах моделі CNN та Bi-LSTM продемонстрували однакову ваду. Їхня нездатність розпізнавати недостовірні заяви є не випадковістю, а прямим наслідком їхньої архітектури та процесу навчання. Ці моделі навчалися «з нуля», і їхній шар «Embedding» формував уявлення про мову виключно на базі тренувальних прикладів LIAR. Архітектура CNN ефективна для пошуку локальних ключових слів (n-грам). Однак у датасеті LIAR достовірність часто визначається не ключовими словами, а тонким контекстом або взаємозв’язком слів, який CNN не може вловити. Хоча Bi-LSTM аналізує послідовність, її механізм «пам’яті» виявився недостатньо потужним, щоб впоратися зі складними семантичними нюансами коротких заяв. Не зумівши знайти надійні патерни для класу «фейк», обидві моделі обрали статистично безпечну стратегію вони розвинули сильну упередженість до класу «правда», який зустрічався частіше. Це пояснює, чому вони помилково класифікували понад 200 недостовірних заяв як достовірні.

Архітектура DistilBERT (рисунок 3.3) показала якісно інший результат. Причина її вищої ефективності (F1-Score 0,67) та кращого балансу полягає у двох фундаментальних перевагах:трансферному навчанні та механізмі уваги (Attention Mechanism). Також DistilBERT не навчався з нуля. Він був попередньо навчений і вже мав глибоке сприйняття граматики, семантики та контексту. Навчання на LIAR було лише процесом донавчання, де модель адаптувала свої вже існуючі знання до специфічного завдання. Крім того, на відміну від LSTM, механізм self-attention у трансформерах дозволяє моделі зважувати важливість кожного слова відносно всіх інших слів у реченні, незалежно від відстані. Це дозволяє їй вловлювати тонкі нюанси, які можуть змінити сенс твердження.

Саме завдяки цим перевагам DistilBERT зміг знайти реальні семантичні патерни для класу «фейк». Він правильно ідентифікував 103 недостовірні заяви – вдвічі більше, ніж CNN та Bi-LSTM. Цікаво, що DistilBERT досяг цього, за рахунок невеликого зниження повноти для класу «правда». Це доводить, що процес донавчання трансформера привів до більш збалансованого результату, а не просто до упередженості на користь одного класу.

Для об’єктивної оцінки отриманих результатів необхідно провести порівняльний аналіз з опублікованими показниками інших дослідників. У статті «A Comprehensive Benchmark for Fake News Detection» [31] автори провели детальне тестування «BERT-base-uncased» (110M параметрів) на тому ж датасеті. Перш ніж порівнювати фінальні результати, критично важливо порівняти гіперпараметри навчання (таблиця 3.2).

Таблиця 3.2 – Порівняння параметрів навчання різних моделей [31]

| Параметр | DistilBERT | BERT-base |
|---|---|---|
| Архітектура | DistilBERT (66M параметрів) | BERT-base (110M параметрів) |
| Max Length (max_len) | 60 | 512 |
| Learning Rate | 3e-5 | 2e-5 |
| Batch Size | 32 | 32 |

Як видно з таблиці, хоча швидкість навчання та розмір батчу є майже ідентичними, критична відмінність полягає у максимальній довжині послідовності

(max_len). Дослідники у [31] обрали стандартну довжину 512, тоді як у даній роботі, на основі аналізу даних, було обрано max_len=60. Використання «max_len=512» означає, що модель обробляє послідовність, яка складається з ~60 токенів заяви і ~452 нульових токенів. Вибір max_len=60 у даній роботі, ймовірно, і став ключовою причиною кращих результатів, оскільки механізм уваги фокусувався лише на релевантному тексті.

Ця різниця в методології безпосередньо відобразилася на фінальних показниках: модель BERT-base у [31] досягла Точності 63,0 % (F1-Score 62,8 %), тоді як модель DistilBERT у даній роботі досягла Точності 68,62 % (F1-Score 67,2 %). Це доводить, що оптимізація гіперпараметрів (зокрема max_len) на основі аналізу специфіки датасету є важливішою за просте збільшення розміру моделі.

Таким чином, отриманий результат 68,62 % є конкурентоспроможним та високим показником для даного датасету, а DistilBERT підтверджує свою перевагу як єдиний кандидат для подальшої оптимізації.

### 3.3 Оптимізація моделі

Порівняльний аналіз чітко визначив, що трансформерна архітектура є найкращою базовою архітектурою, а інтерпретація результатів в підрозділі 3.2 підтвердила, що її перевага полягає в глибокому семантичному аналізі.

Однак, незважаючи на те, що точність 68,62 % є конкурентоспроможною, вона не є достатньо високою для впровадження надійної системи в реальних умовах. Аналіз помилок також виявив слабкі місця, зокрема відносно низький recall для класу «фейк» та значну кількість хибнопозитивних помилок. Тому, перш ніж остаточно обрати цю модель для фінальної реалізації, цей підрозділ досліджує практичний метод оптимізації, спрямований на покращення ключових метрик. На відміну від підрозділу 2.4, де параметри були стандартизовані для справедливого порівняння, основна увага тут приділяється максимізації продуктивності обраної моделі-переможця.

Практична оптимізація полягає у перевірці того, чи була продуктивність DistilBERT обмежена його відносно невеликим розміром. Оскільки раніше було доведено, що гіперпараметр «max_len=60» є ефективним, було проведено порівняння. Архітектура для цього експерименту базується на тій, що описана в лістингу 2.12, але з трьома кардинальними змінами, показаними в лістингу 3.1. Ці зміни замінюють легку модель DistilBERT на її важчий аналог BERT-base, зберігаючи всі інші налаштування. Це дозволяє побачити, чи призводить збільшення кількості параметрів до кращого результату, коли налаштування ідентичні.

Лістинг 3.1 – Зміни для переходу на архітектуру BERT-base

```
from transformers import TFBertModel

model_name = 'bert-base-uncased'

bert = TFBertModel.from_pretrained(model_name, use_safetensors=False)
```

кінець лістингу 3.1

Результати цього експерименту виявилися успішними і продемонстрували, що важча архітектура при збереженні гіперпараметра «max_len=60» дійсно забезпечує кращу продуктивність, ніж базова модель DistilBERT (66M параметрів). На рисунку 3.4 наведено звіт класифікації та матрицю змішування для оптимізованої моделі.

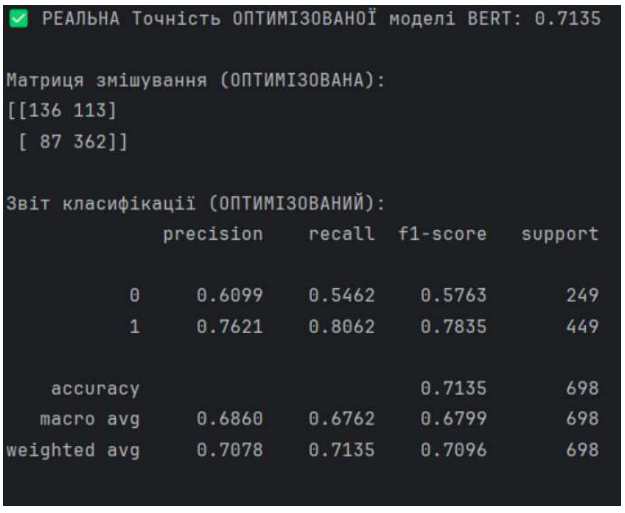


Рисунок 3.4 – Ключові метрики моделі Bert-base

Результати тестування оптимізованої моделі BERT-base підтвердили, що цей практичний експеримент виявився успішним. Модель досягла загальної точності 71,35 %. Для детального аналізу приросту продуктивності, зведемо ключові показники та гіперпараметри базової моделі DistilBERT та оптимізованої BERT-base в єдину порівняльну таблицю 3.3.

Таблиця 3.3 – Порівняння метрик базової та оптимізованої моделей

| Показник | Базова модель (DistilBERT) | Оптимізована модель (BERT-base) |
|---|---|---|
| Архітектура | DistilBERT | BERT-base |
| F1-Score(Avg) | 0,67 | 0,71 |
| F1-Score (клас фейк) | 0,48 | 0,58 |
| Точність | 68,62 % | 71,35 % |

Перехід на важчу архітектуру BERT-base дав приріст у всіх ключових метриках. Загальний зважений «F1-Score» зріс з 0,67 до 0,71, а загальна точність – з 68,62 % до 71,35 %. Аналіз матриць змішування дозволяє зрозуміти, чому оптимізована модель BERT-base виявилася ефективнішою. Щодо класу «фейк», BERT-base правильно ідентифікувала 136 таких заяв, що на 32 % більше, ніж DistilBERT. Крім того, вона зменшила кількість хибнопозитивних помилок, зі 146 до 113. Це найважливіше покращення, оскільки воно стосується найслабшого місця моделі. Водночас, аналізуючи клас «правда» є невеликий компромі, оскільки BERT-base збільшив кількість хибнонегативних помилок з 73 до 87, що призвело до незначного зниження Recall для цього класу. Цей аналіз доводить, що важча архітектура BERT-base досягла кращого загального балансу. Хоча ефективність у знаходженні достовірних заяв незначно знизилася, це було більш ніж компенсовано суттєвим покращенням здатності знаходити недостовірні заяви.

Виходячи з отриманих результатів, оптимізована модель BERT-base продемонструвала найкращі показники надійності та балансу в рамках даного дослідження. Отриманий приріст продуктивності, особливо в критично важливому аспекті ідентифікації фейків, повністю виправдовує використання більш ресурсоємної архітектури. Саме ця навчена та оптимізована модель буде інтегрована у Telegram-бот.

### 3.4 Розробка Telegram-бота для взаємодії з користувачами

Після вибору та оптимізації фінальної моделі BERT-base, логічним завершенням дослідження є її практична реалізація. Для того, щоб система була доступною для кінцевого користувача та могла надавати оцінки достовірності в режимі реального часу, необхідно було обрати платформу для взаємодії. Було розглянуто три основні архітектурні підходи: веб-додаток, розширення для браузера та чат-бот у месенджері.

Для оцінки оптимального рішення було визначено три ключові вимоги, які є вирішальними для вибору фінальної платформи: складність розробки, доступність для користувача та вимоги до підтримки. Вибір цих критеріїв не є випадковим. Складність розробки є ключовим фактором, оскільки фокус даної роботи зосереджений на розробці та оптимізації самої нейронної мережі, а не на створенні складного користувацького інтерфейсу. Тому перевага надається платформам, які мінімізують час на розробку фронтенду. Зручність для користувача – це другий вирішальний фактор. Система, яка вимагає складних дій, наприклад, встановлення окремого плагіну, може відлякувати багатьох людей. Платформа, яка не потребує інсталяції і доступна через вже існуючий додаток, є набагато ефективнішою, щоб охопити більше користувачів. Вимога підтримки та мобільна сумісність оцінюють життєздатність рішення в довгостроковій перспективі. Оскільки значна частина новин споживається з мобільних пристроїв, обрана платформа повинна нативно підтримувати мобільні ОС без необхідності розробки окремих адаптивних версій. Порівняльний аналіз трьох архітектурних підходів за цими критеріями наведено в таблиці 3.4 [32-34].

Таблиця 3.4 – Порівняння платформ для реалізації системи [32-34]

| Критерій | Веб-додаток | Розширення (Plugin) | Telegram-бот |
|---|---|---|---|
| Складність розробки | Висока (потрібен UI, хостинг) | Висока | Низька |
| Доступність для користувача | Середня | Низька (потрібна інсталяція) | Висока |
| Мобільна підтримка | Вимагає адаптивного дизайну | Майже відсутня | Повна |

Аналіз таблиці 3.4 показує, що для цілей даної роботи – швидко та надійно надати користувачам доступ до доказу концепції – telegram-бот є оптимальним рішенням. Він має нульовий поріг входу для користувача, миттєво працює на всіх пристроях та дозволяє повністю зосередитися на інтеграції моделі, а не на розробці інтерфейсу.

Обрана платформа є класичною системою «клієнт-сервер». Вона складається з трьох основних компонентів: інтерфейс, який приймає повідомлення від користувача; бекенд – скрипт, що використовує бібліотеку «pyTelegramBotAPI» (telebot) та містить всю логіку обробки; та модель (TensorFlow/Hugging Face), яка є завантаженою, навченою моделлю BERT-base та її токенізатором.

Процес реалізації вимагає двох ключових кроків: збереження навченої моделі з середовища навчання та її подальше завантаження у робочий скрипт бота.

Для того, щоб Telegram-бот міг використовувати модель, її необхідно спочатку зберегти на диск. Лістинг 3.2 демонструє, як модель BERT-base та її токенізатор зберігаються після навчання. Модель зберігається у форматі «TensorFlow SavedModel» (model.save()), який включає як архітектуру, так і ваги, а токенізатор зберігається за допомогою методу «save_pretrained()».

Лістинг 3.2 – Збереження навченої моделі та токенізатора

```
 model_save_path =
«C:/Users/Nestors/Desktop/bot/bert_base_model_savedmodel»
 tokenizer_save_path =
«C:/Users/Nestors/Desktop/bot/my_bert_base_tokenizer/»

 model.save(model_save_path)
 tokenizer.save_pretrained(tokenizer_save_path)
```

кінець лістингу 3.2

Для підготовки системи до прийому запитів, збережені файли моделі та токенізатора завантажуються в пам’ять при старті бота, як це показано в лістингу 3.3. Спочатку завантажується токенізатор «BertTokenizer.from_ pretrained()». Потім завантажується повна архітектура моделі «tf.keras.models.

load_model()», якій необхідно передати «custom_objects={«TFBertModel»: TFBertModel}», щоб Keras 3 коректно розпізнав кастомний шар трансформера.

Лістинг 3.3 – Завантаження моделі та токенізатора у скрипті бота

```
 import os
 import re
 import tensorflow as tf
 import telebot
 from transformers import BertTokenizer, TFBertModel
 from tensorflow.keras.layers import Input, Dense, Dropout,
GlobalAveragePooling1D
 from tensorflow.keras.models import Model
 os.environ[«TF_USE_LEGACY_KERAS»] = «1»
 model_path = «C:/Users/Nestors/Desktop/bot/bert_base_model_savedmodel»
 tokenizer_path = «C:/Users/Nestors/Desktop/bot/my_bert_base_tokenizer/»
 API_TOKEN = ‘API_TOKEH’
 max_len = 60
 print(«Завантаження токенізатора...»)
 tokenizer = BertTokenizer.from_pretrained(tokenizer_path)
 print(«Завантаження моделі... (Це може зайняти час)»)
 final_model = tf.keras.models.load_model(
     model_path,
     custom_objects={«TFBertModel»: TFBertModel}
 )
 print(«✅ Модель та токенізатор успішно завантажено.»)
```

кінець лістингу 3.3

Коли модель та токенізатор успішно завантажено в пам’ять, наступним кроком є визначення основного процесу обробки запиту. Це ключова частина логіки бота, яка перетворює початковий текстовий рядок від користувача у числовий формат, придатний для моделі BERT-base. Критично важливо, щоб цей процес прогнозування точно відтворював кроки попередньої обробки, використані під час навчання самої моделі. Це означає, що до тексту, отриманого від користувача, має бути застосована та сама функція базового очищення та ті самі правила токенізації. Будь-яка розбіжність у цих кроках призведе до того, що модель отримає дані у невідомому їй форматі, що унеможливить достовірний прогноз.

Реалізація цього процесу у скрипті бота продемонстрована в лістингу 3.4. Він включає необхідну функцію очищення, а також головну функцію «predict_text», яка об’єднує всі етапи (очищення, токенізація, прогноз) в єдину операцію.

Лістинг 3.4 – Реалізація функцій очищення та прогнозування у скрипті бота

```
 def clean_text_simple(text):
     text = str(text).lower()
     text = re.sub(r'https?://\S+|www\.\S+', ' ', text)
     text = re.sub(r'<.*?>', ' ', text)
     text = re.sub(r'\s+', ' ', text).strip()
     return text
 def predict_text(text_input):
     cleaned_input = clean_text_simple(text_input)
     input_enc = tokenizer.batch_encode_plus(
         [cleaned_input],
         add_special_tokens=True,
         max_length=max_len,
         padding='max_length',
         truncation=True,
         return_tensors='tf'
     )
     input_dict = {
         'input_ids': input_enc['input_ids'],
         'attention_mask': input_enc['attention_mask']
     }
     prediction_prob = final_model.predict(input_dict)[0][0]
     if prediction_prob > 0.5:
         return (f»✅ Достовірна заява (Впевненість:
{prediction_prob*100:.2f}%)»)
     else:
         return (f»❌ Недостовірна заява (Впевненість: {(1-
prediction_prob)*100:.2f}%)»)
```

кінець лістингу 3.4

Коли визначено основний процес прогнозування, фінальним кроком (лістинг 3.5) є інтеграція цієї функції з telegram bot api за допомогою бібліотеки «telebot». Цей код відповідає за запуск бота, обробку вхідних повідомлень та виклик функції «predict_text».

Лістинг 3.5 – Завантаження моделі та токенізатора у скрипті бота

```
 import telebot # Імпорт бібліотеки для роботи з API Telegram
 bot = telebot.TeleBot(API_TOKEN)
 @bot.message_handler(commands=['start', 'help'])
 def send_welcome(message):
     bot.reply_to(message, «Вітаю! Я бот для детекції фейкових новин,
навчений на моделі BERT-base.\n\n»
                           «Надішліть мені англомовне твердження, і я
спробую оцінити його достовірність (точність моєї моделі ~71%).»)
 @bot.message_handler(func=lambda message: True)
```

```
 def handle_text(message):
     try:
         user_text = message.text
         if not user_text.strip():
             bot.reply_to(message, «Будь ласка, надішліть текст для
аналізу.»)
             return
         print(f»Отримано запит: {user_text}»)
         response_text = predict_text(user_text)
         print(f»Відповідь: {response_text}»)
         bot.send_message(message.chat.id, response_text)
     except Exception as e:
         print(f»ПОМИЛКА обробки: {e}»)
         bot.reply_to(message, «Вибачте, сталася помилка під час обробки
вашого запиту.»)
 print(«Бот запущений і очікує на повідомлення...»)
 bot.polling()
```

кінець лістингу 3.5

Щоб перевірити, як бот працює на практиці, було проведено тестування. На рисунку 3.5 продемонстровано приклад реальної взаємодії користувача з ботом. Користувач надсилав твердження, а бот, використовуючи навчену модель, аналізував їх. Як видно з рисунка, бот успішно обробляє запити та повертає відповідь: або «Достовірна заява», або «Недостовірна заява», разом із відсотком впевненості.

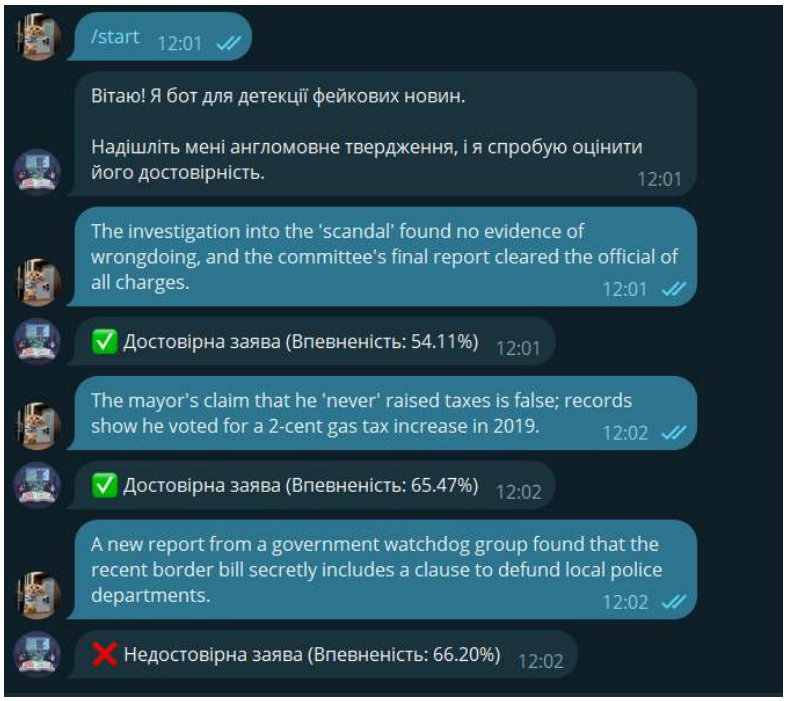


Рисунок 3.5 – Приклад взаємодії з Telegram-ботом

## 3.5 Можливості впровадження системи в реальні умови, обмеження дослідження та рекомендації щодо подальших досліджень

У попередніх підрозділах було розроблено, протестовано та оптимізовано транформернумодель, яка досягла найкращої точності 71,35 %, і цю модель було успішно інтегровано у Telegram-бот. Хоча ця система є успішним доказом концепції, її впровадження в реальні умови має суттєві обмеження.

Точність 71,35 %, хоч і є високою для складного датасету LIAR, все ж означає, що система буде помилятися приблизно в 4 з 10 випадках. Для системи, що має на меті боротьбу з дезінформацією, такий рівень помилок, особливо хибнопозитивних, є неприйнятним, оскільки він підриває довіру користувача. Головним обмеженням є вузька спеціалізація моделі. Вона була навчена виключно на політичних заявах. Це класичний приклад проблеми невідповідності тренувальних та реальних даних. Модель, навчена на вузькоспеціалізованому політичному лексиконі, нездатна узагальнити свої знання на новини з інших сфер. Крім того, реалізація працює лише з англійською мовою та аналізує лише текст. Це робить систему абсолютно беззахисною перед сучасними, більш небезпечними формами дезінформації, такими як маніпулятивні зображення, меми та «діпфейки».

Найбільшим обмеженням є відсутність реальної перевірки фактів. Модель аналізує лише те, як сама новина написана, але не перевіряє зміст по базі даних реальних фактів. Вона «не знає», що заява «Земля плоска» є неправдивою, а лише оцінює, чи схоже це речення на правдиве, спираючись на вивчені патерни.

На основі цих обмежень, можна сформулювати рекомендації для подальших досліджень. По-перше, для покращення точності слід реалізувати гібридну архітектуру, яка б використовувала метадані датасету LIAR. Дослідження, такі як FakeNewsNet [8] або роботи з моделювання соціального контексту [20], доводять, що інформація про спікера (speaker) та його партійну приналежність (party_affiliation) є потужними статистичними індикаторами упередженості, які могли б значно покращити точність.

По-друге, критичним обмеженням поточної системи є її мовна упередженість. Модель «bert-base-uncased» працює лише з англійською мовою. Для

розширення системи та підтримки української мови простої заміни архітектури на мультилінгвальну, як це запропоновано у [13], недостатньо. Поточна модель була донавчена на специфічних патернах англійського датасету LIAR. Мультилінгвальна модель, донавчена на цих же даних, так само не зможе коректно аналізувати український текст. Тому, для реальної підтримки української мови необхідна комплексна стратегія. Важливим є розробка та розмітка нового, великого та збалансованого датасету українських фейкових новин. Наразі в публічному доступі бракує якісних анотованих корпусів для української мови, подібних до LIAR. Цей новий датасет має охоплювати різні сфери, щоб уникнути проблеми невідповідності тренувальних та реальних даних, виявленої раніше. Лише після створення такого датасету, мультилінгвальна модель або спеціалізована українська модель зможе бути на ньому ефективно донавчена для виявлення фейків в українському інформаційному просторі. Найважливішою стратегічною рекомендацією є поєднання поточної моделі з системою пошуку фактів. Як зазначається у роботах [4, 15], інтеграція зовнішніх знань є ключем до створення надійної системи. Це перетворило б бота з пасивного класифікатора стилю на активний інструмент верифікації, який міг би шукати підтвердження або спростування заяви на авторитетних веб-сайтах.

Аналіз та практичне тестування довели, що поточна модель BERT-base, хоч і є ефективною, по суті є пасивним класифікатором стилю. Вона аналізує, як написано твердження, але не здатна перевірити його фактичний зміст. Наприклад, модель може помилково класифікувати добре написану, але абсурдну заяву як достовірну, оскільки вона не має доступу до зовнішніх знань (про реальний світ. Як зазначається у дослідженнях [4] та [15], інтеграція зовнішніх знань є ключем до створення по-справжньому надійної системи детекції. Це перетворило б бота з пасивного класифікатора на активний інструмент верифікації. Такий гібридний підхід працював би у два етапи:

– етап 1 (семантичний аналіз). Модель BERT-base надає початкову оцінку наскільки правдоподібно звучить заява;

– етап 2 (фактчекінг). Одночасно система виділяє з тексту ключові дані і шукає підтвердження або спростування цих фактів на авторитетних, попередньо

визначених веб-сайтах. Фінальне рішення бот приймав би на основі синтезу обох результатів.

Нарешті, для досягнення максимальної точності, майбутні дослідження могли б зосередитися на гібридних ансамблях, які поєднують переваги різних архітектур. У цьому дослідженні було виявлено, що кожна модель має унікальні сильні та слабкі сторони. CNN та Bi-LSTM продемонстрували сильну упередженість, але високу повноту для класу «правда». BERT-base досяг кращого семантичного балансу і показав значно вищу здатність розпізнавати клас «фейк». Гібридна модель, могла б об’єднати ці переваги. Такий підхід використовує виходи з кількох моделей як ознаки для фінального, простішого класифікатора. Цей мета-класифікатор, по суті, вчиться, якій з моделей довіряти в тій чи іншій ситуації. Наприклад, він може навчитися, що коли CNN виявляє сильний локальний патерн, а BERT  підтверджує контекст, прогноз є надійним.

Як зазначається у дослідженнях [2] та [5], використання таких ансамблевих методів, що поєднують CNN, LSTM та інші класифікатори, часто дає кращий сукупний результат, ніж будь-яка окрема модель, оскільки різні архітектури компенсують індивідуальні недоліки одна одної.

# ВИСНОВКИ

У магістерській роботі було проведено оптимізацію та розробку системи для автоматичного виявлення фейкових новин на основі нейромережевих моделей. Робота складається з трьох розділів, в кожному з яких висвітлено певні аспекти проєктування та реалізації моделі глибокого навчання.

У першому розділі були розглянуті теоретичні основи фейкових новин, їх класифікація та сучасні методи автоматичного виявлення. У другому розділі було описано процес вибору та підготовки даних, а також визначено архітектури трьох нейронних мереж (CNN, Bi-LSTM, BERT) та етапи їх реалізації. У третьому розділі проведено порівняльний аналіз продуктивності, оптимізацію найкращої моделі та її практичну реалізацію у вигляді Telegram-бота.

За результатами вирішення поставлених завдань сформульовано наступні висновки:

– проаналізовано теоретичні основи феномену фейкових новин, їх класифікацію та джерела походження. Встановлено, що дезінформація має системний характер і деструктивний вплив на суспільство, що зумовлює необхідність автоматизації процесів фактчекінгу;

– здійснено огляд сучасних методів виявлення дезінформації. Визначено, що класичні алгоритми машинного навчання поступаються методам глибокого навчання у здатності аналізувати семантичний контекст, що стало підставою для фокусування дослідження на нейромережевих підходах;

– досліджено особливості використання нейронних мереж для NLP-задач та обґрунтовано доцільність застосування глибокого навчання (Deep Learning). Доведено, що архітектури, здатні враховувати послідовність слів (RNN) та механізм уваги (Transformers), є найбільш перспективними для семантичного аналізу коротких текстових повідомлень;

– визначено критерії та метрики для оцінювання якості моделей. Для об’єктивного аналізу в умовах бінарної класифікації обрано комплекс показників: Accuracy, Precision, Recall та F1-Score, причому останній визначено як ключовий для оцінки балансу між класами;

– здійснено огляд доступних наборів даних та обрано академічний датасет LIAR як оптимальний для навчання. На відміну від спрощених аналогів (як-от ISOT), LIAR не містить явних стилістичних артефактів, що дозволило уникнути перенавчання та забезпечити чесну перевірку здатності моделей до узагальнення;

– розроблено та реалізовано алгоритми попередньої обробки даних, що включали бінаризацію класів та дві диференційовані стратегії очищення тексту: «агресивну» (з лематизацією) для моделей, що навчаються з нуля, та «базову» –для збереження контексту в трансформерних моделях;

– обґрунтовано вибір трьох архітектур для порівняльного експерименту: згорткової (CNN) для виявлення локальних патернів, двонаправленої рекурентної (Bi-LSTM) для аналізу послідовностей та трансформерної (BERT) для глибокого контекстуального аналізу;

– програмно реалізовано та навчено обрані моделі у середовищі Python з використанням бібліотек TensorFlow та Transformers, забезпечивши ідентичні умови для проведення експериментів;

– проведено валідацію та тестування моделей на незалежній вибірці. Отримано первинні метрики, які показали, що традиційні архітектури (CNN, Bi-LSTM) демонструють точність у межах 65-66 %, тоді як трансформерні моделі показують вищий потенціал;

– виконано порівняльний аналіз продуктивності, який виявив суттєву перевагу трансформерної архітектури (DistilBERT, F1-Score 0,67) над CNN (0,60) та Bi-LSTM (0,61);

– здійснено інтерпретацію результатів та аналіз помилок. Встановлено, що моделі CNN та Bi-LSTM не здатні ефективно розпізнавати клас «Фейк» (Recall < 0,20) через сильну упередженість до позитивного класу. Натомість механізм уваги (Attention) дозволив трансформеру досягти значно кращого балансу та вдвічі підвищити ефективність виявлення недостовірних заяв;

– проведено практичну оптимізацію найкращої моделі шляхом переходу на важчу архітектуру BERT-base (110 млн параметрів) зі збереженням оптимізованих гіперпараметрів (max_len=60). Це дозволило підвищити фінальну точність до

71,35 % (F1-Score 0,71), що перевищує базові показники аналогічних моделей в інших дослідженнях;

– розроблено програмний прототип системи у вигляді Telegram-бота, в який успішно інтегровано оптимізовану модель BERT-base. Бот працює в режимі реального часу, виконує повний цикл обробки тексту та надає оцінку достовірності, що демонструє практичну придатність розробленого рішення;

– оцінено можливості впровадження та визначено обмеження системи (вузький домен політичних новин, відсутність верифікації фактів). На основі цього сформульовано рекомендації щодо подальшого розвитку: інтеграція метаданих, створення українського корпусу даних та поєднання моделі з системами пошуку зовнішніх знань.

Таким чином, у магістерській роботі не лише теоретично обґрунтовано та експериментально підтверджено переваги трансформерних архітектур над традиційними методами, а й доведено їхню практичну придатність через створення робочого прототипу. Розроблена система є масштабованою та гнучкою, що дозволяє в майбутньому інтегрувати нові мовні моделі та модулі перевірки фактів, перетворюючи її на повноцінну платформу для боротьби з фейковими новинами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пасічніченко Д. О. Аналіз архітектур нейронних мереж для виявлення фейкових новин на основі датасету LIAR. Progressive Approaches in Science and Engineering: Collection of Scientific Papers with Proceedings of the 2nd International Scientific and Practical Conference (November 26-28, 2025, Copenhagen, Denmark). Copenhagen : International Scientific Unity, 2025. Issue № 47. С. 268-270.

2. Fake news detection using machine learning ensemble methods / I. Ahmad et al. Complexity. 2020. Vol. 2020. P. 8885861.

3. Fake News and Aggregated Credibility: Conceptualizing a Co-Creative Medium for Evaluation of Sources Online / M. Faraon et al. International Journal of Ambient Computing and Intelligence (IJACI). 2020. Vol. 11, no. 4. P. 93-117.

4. Compare to The Knowledge: Graph Neural Fake News Detection with External Knowledge / L. Hu et al. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics. 2021. P. 754-763.

5. Fake news detection using machine learning ensemble methods / I. Ahmad et al. Complexity. 2020. Vol. 2020. P. 8885861.

6. Aphiwongsophon S., Chongstitvatana P. A survey of fake news detection using natural language processing. Proceedings of the 2021 18th International Joint Conference on Computer Science and Software Engineering (JCSSE). 2021. P. 204-209.

7. McIntosh L. D., White W., Hudson Vitale C. Unveiling Deception: Establishing a Taxonomic Framework for Disinformation within Scientific Discourse. arXiv. 2024. URL: https://arxiv.org/pdf/2311.11344 (дата звернення: 10.09.2025).

8. FakeNewsNet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media / K. Shu et al. Big Data. 2020. Vol. 8, iss. 3. P. 171-188.

9. De Beer D., Matthee M. A Conceptual Fake News Typology: A Systematic Literature Review. Proceedings of the 23rd Annual International Conference on Digital Government Research (dgo ‘22). New York : Association for Computing Machinery, 2022. P. 1-10.

10. Іващенко В. В. Застосування методів машинного навчання для виявлення та класифікації фейкових новин. Проблеми програмування. 2023. № 2. С. 58-67.

11. Савчук Р. О., Ковальчук Л. П. Порівняльний аналіз традиційних і нейромережевих підходів до виявлення дезінформації. Системи обробки інформації. 2023. № 1 (182). С. 132-140.

12. Мельник Р. П., Коваль Д. С. Моделі глибокого навчання на основі LSTM та GRU для класифікації фейкових новин. Науковий вісник Ужгородського національного університету. Серія: Фізика. 2022. Вип. 52. С. 52-58.

13. Карпенко А. І., Саченко О. Л. Використання трансформерних моделей (BERT, RoBERTa) для підвищення точності виявлення фейкових новин. Вісник Хмельницького національного університету. Технічні науки. 2024. № 1 (320). С. 150-158.

14. Mihalcea R., Strapparava C. The Lie Detector: Explorations in the Automatic Recognition of Deceptive Language. Proceedings of the ACL-IJCNLP 2009 Conference Short Papers. Singapore, 2009. URL: https://aclanthology.org/P09-2078.pdf (дата звернення: 15.09.2025).

15. Truth of Varying Shades: Analyzing Language in Fake News and Political Fact-Checking / H. Rashkin et al. Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017). Copenhagen, Denmark, 2017. URL: https://aclanthology.org/D17-1317.pdf (дата звернення: 16.09.2025).

16. Kapadia P. An Explainable Approach to Multi-contextual Fake News Detection: Master Thesis. Eindhoven : Eindhoven University of Technology, 2021. 50 p. URL: https://pure.tue.nl/ws/portalfiles/portal/199440931/Kapadia_P.pdf (дата звернення: 17.09.2025).

17. Meel P., Vishwakarma D. K. Fake news, rumor, information pollution in social media and web: A contemporary survey of state-of-the-arts, challenges and opportunities. Expert Systems with Applications. 2020. Vol. 153. P. 112986.

18. A survey of fake news: Fundamental theories, detection methods, and opportunities / B. Rath et al. ACM Computing Surveys. 2023. Vol. 56, iss. 9. P. 1-36.

19. Kaliyar R. K., Goswami A., Narang P. FakeBERT: Fake news detection in social media with a BERT-based deep learning approach. Multimedia Tools and Applications. 2021. Vol. 80, No. 8. P. 11765-11788.

20. Степанюк І. О., Костюк П. С. Моделювання соціального контексту в задачі виявлення фейкових новин за допомогою графових нейронних мереж. Науково-технічний журнал «Системи та технології». 2023. № 2 (46). С. 58-65.

21. Understanding the Confusion Matrix in Machine Learning. GeeksforGeeks. URL: https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/ (дата звернення: 29.09.2025).

22. Overview of the 8th author profiling task at PAN 2020: Profiling fake news spreaders on Twitter / F. Rangel et al. CEUR Workshop Proceedings. 2020. Vol. 2696. P. 1-18.

23. Класифікація: ROC й AUC. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=uk (дата звернення: 29.09.2025).

24. Fake Detect: A Deep Learning Ensemble Model for Fake News Detection / N. Aslam et al. Complexity. 2021. Vol. 2021. P. 5557784. URL: https://doi.org/10.1155/2021/5557784 (дата звернення: 29.09.2025).

25. Колісник О. В. Оцінка узагальнювальної здатності моделей машинного навчання за допомогою перехресної перевірки. Інформаційні системи та технології. 2024. № 1 (40). С. 45-53.

26. Yetim E. Fake News Detection Datasets. Kaggle. URL: https://www.kaggle.com/datasets/emineyetm/fake-news-detection-datasets/data (дата звернення: 25.09.2025).

27. Goldani M. H., Momtazi S., Safabakhsh R. Detecting fake news with deep learning methods: A systematic survey. Journal of Web Science. 2021. Vol. 7, P. 1-27.

28. El-Shafie A., Diab F. M. E. Z. K., Abd-El-Hafiz M. A. Fake News Detection: A Machine Learning Approach. 2022 International Conference on Computer Engineering & Systems (ICCES). Cairo, Egypt, 2022. P. 1-6.

29. LIAR Dataset: A Benchmark Dataset for Fake News Detection. Kaggle. URL: https://www.kaggle.com/datasets/doanquanvietnamca/liar-dataset (дата звернення: 25.09.2025).

30. Loukas S. Text Classification Using Naive Bayes: Theory & A Working Example. Towards Data Science. 2023. URL: https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a (дата звернення: 29.09.2025).

31. A Comprehensive Benchmark for Fake News Detection / R. K. Al-Snaiwi et al. Iraqi Journal for Computer Science and Mathematics. 2022. Vol. 3, No. 1. P. 1-15.

32. Dang T. Why Use React? To Reclaim Developers' Joy in Complex Web Development. Orient Software. URL: https://www.orientsoftware.com/blog/why-use-react/ (дата звернення: 30.10.2025).

33. Agarwal S. Helping or Hindering? How Browser Extensions Undermine Web Security. 2022. URL: https://swag.cispa.saarland/papers/agarwal2022helping.pdf (дата звернення: 02.11.2025).

34. Arderiu A. Python Telegram Bot Development Made Easy. Sirvelia. URL: https://sirvelia.com/en/telegram-bot-python/ (дата звернення: 05.11.2025).