МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____Євген ДАВИДЕНКО

«___» _____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

АНАЛІЗ МАНІПУЛЯТИВНОСТІ НОВИН ІЗ ВИКОРИСТАННЯМ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмногозабезпечення»

Здобувач _____ **Вадим ЮХНЕНКО**

«__» _____2025 р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«__» _____2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інженерії
програмного забезпечення
_____Євген ДАВИДЕНКО
«___»_____2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Юхненку Вадиму Сергійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи
Аналіз маніпулятивності новин із використанням алгоритмів машинного навчання

Затверджена наказом ЧНУ ім. Петра Могили від «02» липня 2025 р. №182

2. Строк представлення кваліфікаційної роботи «22» грудня 2025 р.

3. Очікуваний результат роботи та початкові дані якщо такі потрібні
Очікуваним результатом є Телеграм-бот для аналізу новинного контенту на основі алгоритмів машинного навчання, який забезпечує класифікацію технік маніпуляції і виявлення відповідних фрагментів тексту в україномовних і російськомовних матеріалах.

4. Перелік питань, що підлягають розробці:

- дослідження предметної області й аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проєктування програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмного забезпечення;
- здійснення тестування та оптимізації роботи програмного забезпечення;

Перелік графічних матеріалів

Презентація

Завдання до спеціальної частини

5. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Дата видачі завдання «02» липня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема кваліфікаційної роботи**:**

Аналіз маніпулятивності новин із використанням алгоритмів машинного навчання

Аналіз маніпулятивності новин
із використанням алгоритмів машинного навчання

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання кваліфікаційної магістерської роботи (КРМ)	01.07.2025 р.	02.07.2025 р.	Виконано
2.	Огляд літератури за темою роботи	15.07.2025 р.	24.07.2025 р.	Виконано
3.	Складання календарного плану КРМ	24.07.2025 р.	26.07.2025 р.	Виконано
4.	Аналіз предметної області	26.07.2025 р.	01.08.2025 р.	Виконано
5.	Розробка проєктних рішень	01.08.2025 р.	07.08.2025 р.	Виконано
6.	Моделювання і конструювання програмного забезпечення (ПЗ)	07.08.2025 р.	15.08.2025 р.	Виконано
7.	Кодування, тестування розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	15.08.2025 р.	10.11.2025 р.	Виконано
8	Апробація	12.11.2025	12.11.2025	Виконано
9.	Оформлення КРМ і презентації	13.11.2025	18.11.2025	Виконано
10.	Відгук керівника КРМ	18.11.2025	18.11.2025	Виконано
11.	Попередній захист	24.11.2025	24.11.2025	Виконано
12.	Завершення оформлення КРМ та презентації	25.11.2025	30.11.2025	Виконано
13.	Рецензування	14.12.2025	16.12.2025	Виконано
14.	Захист кваліфікаційної роботи	22.12.2025	22.12.2025	Виконано

Здобувач _____ **Вадим ЮХНЕНКО**

_____ «__» _____2025 р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«__» _____2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра

«Аналіз маніпулятивності новин

із використанням алгоритмів машинного навчання»

Здобувач 608 гр.: Юхненко Вадим Сергійович

Керівник: канд. техн. наук, доцент Давиденко Євген Олександрович

Робота присвячена аналізу маніпулятивності новин із використанням алгоритмів машинного навчання для виявлення дезінформації в україномовних і російськомовних текстах. Вибір теми зумовлений зростаючою проблемою поширення маніпулятивного контенту в інформаційному просторі, зокрема в соціальних мережах і месенджерах. У контексті гібридної війни, спробах впливати на суспільну думку й ескалації соціальної напруги наявність комплексного рішення для аналізу новинних матеріалів є конче необхідним. Використання алгоритмів машинного навчання забезпечить класифікацію новин за типами маніпуляцій, виявлення маніпулятивних фрагментів методом span-detection й інтеграцію результатів у Телеграм-бот, що значно спростить процес оцінки якості текстів. Рішення стане у нагоді адміністраторам каналів, підвищить медіаграмотність користувачів і зробить боротьбу з дезінформацією більш ефективною.

Об’єкт: процес аналізу україномовного і російськомовного текстового контенту на наявність маніпулятивних технік за допомогою алгоритмів машинного навчання для підвищення ефективності оцінювання текстів новин перед подальшою публікацією.

Предмет: алгоритми машинного навчання для класифікації новин за типами маніпулятивних технік (включно з визначенням неманіпулятивних), а також для визначення відповідних фрагментів тексту методом span-detection із подальшою інтеграцією результатів у Телеграм-бот.

Мета: аналіз тексту новин на предмет використання маніпулятивних технік із залученням алгоритмів машинного навчання, що передбачає класифікацію новин за типами маніпуляцій і визначення відповідних фрагментів (span-

detection) із інтеграцією в Телеграм-бот.

Кваліфікаційна робота магістра складається зі вступу, чотирьох розділів, висновків і переліку джерел посилання.

У вступі визначається актуальність теми, формується мета дослідження, проводиться короткий огляд поставленого завдання, предмета й об’єкта дослідження.

У першому розділі наведено порівняння існуючих альтернативних застосунків, проводиться аналіз їх сильних і слабких сторін, що допоможе у складанні специфікації вимог до розроблюваного програмного забезпечення.

У другому розділі розглядаються теоретичні й методологічні основи розпізнавання маніпулятивних технік у багатомовних текстах Telegram. Оглянуто сучасні підходи до навчання моделей – від традиційних до трансформерних архітектур. Обґрунтовується вибір оптимальних методів для багатоміткової класифікації і span-виділення з акцентом на стратегії адаптації, такі як pre-training, fine-tuning і few-shot learning.

У третьому розділі описано підготовку даних із очищенням, нормалізацією, анотацією і стратифікованим розбиттям для врахування дисбалансу класів у багатомовному контенту. Висвітлено процес навчання моделей із використанням накопичення градієнтів, планування швидкості навчання, ранньої зупинки та кастомних функцій для класифікації маніпуляцій і детекції спанів.

У четвертому розділі розглянуто ефективність моделей span-detection і class-detection, їх узагальнювальну здатність на тестових даних, а також оцінено роботу інтегрованої системи в Телеграм-боті. Визначено основні обмеження і потенційні напрями удосконалення.

КРМ викладена на 81 сторінці (без додатків), вона містить 4 розділи, 38 ілюстрацій, 5 таблиць, 34 джерел в переліку посилань, 2 додатки.

Ключові слова: *аналіз маніпулятивності новин, алгоритми машинного навчання, класифікація маніпулятивних технік, span-detection, Telegram-бот, багатомовний контент, дезінформація.*

ABSTRACT

of the Master's Thesis

"Analysis of news manipulativity using machine learning algorithms"

Student: Yukhnenko Vadym

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor

Davydenko Yevhen Oleksandrovych

The work is dedicated to the analysis of news manipulation using machine learning algorithms to detect disinformation in Ukrainian- and Russian-language texts. The choice of topic is motivated by the growing problem of the spread of manipulative content in the information space, particularly on social networks and messaging platforms. In the context of hybrid warfare, attempts to influence public opinion, and escalating social tensions, the presence of a comprehensive solution for analyzing news materials is critically important. The use of machine learning algorithms enables the classification of news by types of manipulations, detection of manipulative fragments using span-detection, and integration of results into a Telegram bot, significantly simplifying the process of assessing text quality. The solution will be useful for channel administrators, enhance media literacy among users, and make the fight against disinformation more effective.

Object: the process of analyzing Ukrainian- and Russian-language textual content for manipulative techniques using machine learning algorithms to improve the efficiency of news text evaluation before further publication.

Subject: machine learning algorithms for classifying news by types of manipulative techniques (including identifying non-manipulative content), as well as for detecting corresponding text fragments using span-detection with subsequent integration of results into a Telegram bot.

Objective: analysis of news texts for the use of manipulative techniques using machine learning algorithms, including the classification of news by types of manipulations and identification of relevant fragments (span-detection) with integration into a Telegram bot.

The master's qualification work consists of an introduction, four chapters,

conclusions, and a list of references.

The introduction outlines the relevance of the topic, defines the research objective, and provides a brief overview of the task, subject, and object of the study.

The first chapter presents a comparison of existing alternative applications, analyzing their strengths and weaknesses, which helps in drafting the specification requirements for the developed software.

The second chapter examines the theoretical and methodological foundations for recognizing manipulative techniques in multilingual Telegram texts. Modern approaches to model training are reviewed, ranging from traditional to transformer-based architectures. The choice of optimal methods for multi-label classification and span detection is justified, with emphasis on adaptation strategies such as pre-training, fine-tuning, and few-shot learning.

The third chapter describes data preparation, including cleaning, normalization, annotation, and stratified splitting to account for class imbalance in multilingual content. The process of model training is detailed, including gradient accumulation, learning rate scheduling, early stopping, and custom functions for manipulation classification and span detection.

The fourth chapter considers the effectiveness of span-detection and class-detection models, their generalization ability on test data, and evaluates the performance of the integrated system in the Telegram bot. Key limitations and potential directions for improvement are identified.

The conclusions analyze the work and the results obtained.

The master's thesis is presented on 81 pages (without appendices), it contains 4 chapters, 38 figures, 5 tables, 34 references, 2 appendices.

Keywords: analysis of news manipulativity, machine learning algorithms, classification of manipulative techniques, span-detection, Telegram bot, multilingual content, disinformation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення
ПК – персональний комп’ютер
UX – user experience
UI – user interface
API – application programming interface
AI – artificial intelligence
WAP – wireless application protocol
SVC – support vector classifier
CNB – complement naïve bayes
LR – linear regression
RF – random forest
GB – gradient boost
DL – deep learning
CNN – convolutional neural network
RNN – recurrent neural network
LSTM – long short-terrm memory
BiLTSM – bidirectional LSTM
GRU – gated recurrent unit
ReLU – rectified linear unit
GELU – gaussian error linear unit
BIO – begin, inside, outside
GPU, – graphic processing unit
CPU – central processing unit
EWC – elastic weight consolidation
NLP – natural language processing
WAP – wireless application protocol
NER – named entity recognition

ВСТУП

Актуальність теми кваліфікаційної магістерської роботи зумовлена зростаючою проблемою поширення маніпулятивного контенту в інформаційному просторі, зокрема в соціальних мережах і месенджерах. У контексті гібридної війни, дезінформація і маніпулятивні техніки, такі як вибіркова правда, узагальнення, апеляція до страху чи кліше без суті, використовуються для впливу на суспільну думку, підриву довіри до державних інституцій і ескалації соціальної напруги. Особливо актуальним це є для україномовного і російськомовного контенту, оскільки обидві мови активно використовуються в інформаційних кампаніях, спрямованих на українську аудиторію. Розробка Телеграм-бота, який аналізує текст новин на наявність маніпулятивних технік, дозволить адміністраторам каналів оцінювати якість текстів новин перед публікацією, а звичайним користувачам – впевнитись у достовірності опублікованої новини. Це сприятиме підвищенню якості контенту, зниженню впливу дезінформації й підвищенню медіаграмотності не тільки адміністраторів, а також аудиторії.

Об’єкт: процес аналізу україномовного і російськомовного текстового контенту на наявність маніпулятивних технік за допомогою алгоритмів машинного навчання для підвищення ефективності оцінювання текстів новин перед подальшою публікацією.

Предмет: алгоритми машинного навчання для класифікації новин за типами маніпулятивних технік (включно з визначенням неманіпулятивних), а також для визначення відповідних фрагментів тексту методом span-detection із подальшою інтеграцією результатів у Телеграм-бот.

Мета: аналіз тексту новин на предмет використання маніпулятивних технік із залученням алгоритмів машинного навчання, що передбачає класифікацію новин за типами маніпуляцій і визначення відповідних фрагментів (span-detection) із інтеграцією в Телеграм-бот.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

– провести аналіз предметної області, зокрема технік маніпуляцій та особливостей їх виявлення в україномовних і російськомовних текстах.

– дослідити наявні набори даних й інструменти для аналізу маніпулятивного контенту.

– реалізувати модель машинного навчання для бінарної і багатокласової класифікації маніпулятивного контенту.

– реалізувати модель для виявлення спанів маніпулятивних фрагментів у тексті.

– розробити Телеграм-бот на основі бібліотеки python-telegram-bot, який інтегрує моделі машинного навчання, аналізує новини і виводить результати оцінки маніпулятивності для адміністраторів.

– провести тестування розробленого рішення на реальних даних із Телеграм-каналів й оцінити його ефективність.

Практичне значення: розроблений Телеграм-бот дозволить адміністраторам Телеграм-каналів аналізувати новини на наявність маніпулятивного контенту, що спростить прийняття рішень щодо їх публікації. Це сприятиме підвищенню якості контенту, зниженню поширення дезінформації та підвищенню медіаграмотності. У свою чергу звичайні користувачі зможуть підвищити власний рівень медіаграмотності, щоб не піддаватися на поширені маніпулятивні практики. Рішення також буде корисним для журналістів, аналітиків і громадських організацій, які працюють із україномовним та російськомовним контентом.

Апробація результатів КМР відбувалася під час XXVIII Всеукраїнської науково-практичної конференції «Могилянські читання – 2025», Миколаїв, 10-14 листопада, 2025 р. (Додаток А)

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність розробки застосунку для пошуку маніпуляцій

Тема кваліфікаційної роботи магістра у сфері розробки Телеграм-бота для аналізу маніпулятивного контенту є актуальною, оскільки відзначається постійним зростанням попиту на такі інструменти в різних сегментах інформаційного простору. Із кожним роком збільшується кількість новинних повідомлень, у яких використовуються маніпулятивні техніки, такі як вибіркова правда, перебільшення, апеляція до страху, використання кліше й інші. У сучасних умовах гібридної війни такі методи застосовуються систематично, щоб впливати на громадську думку, підривати довіру до державних інституцій і створювати атмосферу соціальної напруги. Попит на інструменти аналізу маніпулятивності новин зумовлений не лише стрімким розвитком інформаційних технологій у сучасному світі, а також загальною тенденцією до цифровізації в медіа. Новинний простір зараз характеризується зростанням популярності Телеграм-каналів, де адміністратори можуть свідомо вводити в оману власну аудиторію для особистих або чужих інтересів.

Зростання конкуренції в медіапросторі й обсягів інформації для обробки робить актуальною розробку інструментів для аналізу текстів новин. Традиційні методи ручної перевірки є повільними, трудомісткими й суб’єктивними, тому потребують заміни на сучасні технологічні рішення. Застосування алгоритмів машинного навчання дає змогу створити систему, яка швидко обробляє великі масиви текстів, об’єктивно визначає наявність маніпуляцій і допомагає користувачам робити зважені висновки після прочитанння новини.

Розробка Телеграм-бота як інструменту для аналізу маніпулятивності новин є не лише технічною, а й соціально значущою задачею. Вона сприяє розвитку медіаграмотності серед населення, і дозволяє користувачам змогу помічати патерни впливу на суспільну думку, що сприятиме формуванню критичного мислення. Не слід забувати про можливість інтеграції подібних рішень у популярні

месенджери дозволяє користувачам отримувати результати аналізу безпосередньо у звичному середовищі спілкування, що підвищує ефективність використання таких технологій і стимулює їхнє поширення.

Сучасні технології розробки програмного забезпечення дозволяють створювати складні системи на базі штучного інтелекту, що оптимізують процеси аналізу новин через автоматизовані моделі. У сучасному цифровому середовищі підхід із використанням двох моделей машинного навчання – для бінарної, багатокласової класифікації типів маніпулятивних технік і для виявлення конкретних фрагментів маніпулятивного тексту – не є дуже поширеним. Тим не менш вищезазначений підхід має усі шанси, щоб зарекомендувати себе як ефективний інструмент для розбиття складних задач на менші, автономні компоненти. Використання Artificial Intelligence (AI) підходу у розробці Телеграм-бота стає ключовим чинником у забезпеченні ефективної роботи інструменту із великими обсягами україномовних і російськомовних текстів. Розбиття задач на незалежні моделі дозволяє спростити їх розширення і підтримку, що важливо у забезпеченні стабільної і швидкої роботи в умовах обмеженої популярності таких інструментів.

У сфері розподілених систем, особливо в області аналізу контенту в месенджерах, завдання підтримки узгодженості даних під час керування станом різних моделей є першочерговим. Ця складність виникає через необхідність гарантувати, що всі діючі компоненти мають уніфіковане уявлення про дані, незважаючи на притаманну затримку і ймовірність збоїв у зв’язку. Масштабованість і розширення системи виявлення маніпуляцій залежать від надійної стратегії, яка допоможе застосунку впоратися з динамікою публікації новин, де маніпулятивні техніки постійно виявляються і класифікуються в режимі реального часу. Із вищезазначеними задачами може впоратися архітектура на базі AI-моделей. Необхідність координувати дії забезпечується без створення тісного зв’язку, що досягається використанням черги повідомлень у Телеграм-боті.

Виявлення маніпулятивного контенту в Телеграм-каналах, завдяки прогресу

в AI-технологіях, інтеграції моделей машинного навчання і підключення до месенджера може зробити вагомий внесок у покращення процесів перевірки новин. На жаль, подібні інструменти поки що не набули широкої популярності й рідко зустрічаються в Telegram-ботах. Зручність, доступність, прозорість, охоплення україномовної і російськомовної аудиторії розширять можливості як для адміністраторів каналів, так і для звичайних користувачів. У результаті індустрія боротьби з дезінформацією може стати більш прозорою, ефективною і доступною для ширшого кола осіб, попри поточну обмежену поширеність таких рішень.

Таким чином актуальність теми визначається не лише проблемою дезінформації, а також потребою в інструментах, що здатні оперативно і об’єктивно аналізувати великі масиви новинного контенту. Запропонований підхід забезпечує поєднання наукових методів аналізу тексту, практичної реалізації у вигляді програмного рішення і користувацької доступності через популярний месенджер Telegram.

1.2 Опис предметного середовища

Об’єкт дослідження – процес аналізу україномовного і російськомовного текстового контенту на наявність маніпулятивних технік за допомогою алгоритмів машинного навчання для підвищення ефективності оцінювання текстів новин перед подальшою публікацією.

Предмет дослідження – алгоритми машинного навчання для класифікації новин за типами маніпулятивних технік (включно з визначенням неманіпулятивних), а також для визначення відповідних фрагментів тексту методом span-detection із подальшою інтеграцією результатів у Телеграм-бот.

Процес аналізу україномовного і російськомовного текстового контенту на наявність маніпулятивних технік за допомогою алгоритмів машинного навчання виступає об’єктом дослідження. Ефективна оцінка включає в себе виявлення провокативних рядків у текстах, класифікацію за типами маніпулятивних технік з урахуванням випадків відсутності маніпулцій. Також процес передбачає

визначення конкретних фрагментів маніпуляцій методом span-detection із подальшою інтеграцією результатів у Телеграм-бот для зручності аналізу тексту.

Структурні і функціональні характеристики об’єкта дослідження формуються його внутрішніми компонентами і їх призначенням. Структурні характеристики об’єкта дослідження:

1) Телеграм-бот: включає в себе основну серверну частину, написану на мові програмування Python, яка забезпечує обробку вхідних повідомлень користувачів, взаємодію з моделями машинного навчання і формування відповідей у зручному форматі. Бот виступає інтерфейсом між користувачем і системою аналізу маніпулятивного контенту, забезпечуючи доступність інструменту безпосередньо в месенджері;

2) моделі машинного навчання: складаються з двох окремих компонентів – моделі класифікації текстів за типами маніпулятивних технік і моделі span-detection, яка визначає конкретні фрагменти маніпулятивного тексту. Обидва компоненти реалізовані з використанням бібліотек torch і transformers. Навчання моделей відбувається на попередньо підготовленому датасеті із застосуванням розподілу на навчальну і тестову вибірки;

3) інфраструктура для навчання і тестування: включає скрипти й допоміжні модулі, які автоматизують процеси підготовки даних, токенізації, навчання, валідації й оцінки моделей. Для реалізації використано широкий набір бібліотек, зокрема numpy, pandas, sklearn, tqdm, matplotlib, nltk, що дозволяють проводити аналіз результатів, обчислювати метрики і візуалізувати ефективність роботи моделей.

Функціональні характеристики об’єкта дослідження визначаються завданнями і функціями, які потрібно реалізувати для досягнення цілей дослідження. Функціональні характеристики об’єкта дослідження:

1) аналіз предметної області і методології розробки: дослідження, вибір методів і підходів, що найбільш ефективно застосовуються для створення систем

виявлення маніпулятивного контенту із використанням алгоритмів машинного навчання;

2) аналіз рішень-аналогів: вивчення існуючих сервісів для перевірки новин, визначення їхніх сильних і слабких сторін із метою подальшого використання отриманих висновків у власному проєкті;

3) аналіз технологій машинного навчання і бібліотек: вибір оптимальних фреймворків, інструментів, моделей для класифікації текстів і виконання span-detection;

4) визначення необхідного функціоналу Телеграм-бота: встановлення переліку можливостей, які повинна забезпечити система, зокрема класифікація тексту новини за видами використаних маніпуляцій, виділення маніпулятивних фрагментів і надання користувачеві результатів у зручному форматі.

Розробка Телеграм-бота для аналізу новинного контенту має на меті підвищити рівень медіаграмотності користувачів і забезпечити швидкий доступ до результатів перевірки достовірності інформації. Ця задача потребує ретельного аналізу доступних технологій, правильного вибору моделей і бібліотек, а також урахування особливостей україномовних і російськомовних текстів.

1.3 Огляд існуючих аналогів

Вивчення ринку існуючих альтернатив є необхідним етапом, оскільки воно дає можливість не лише побачити загальну картину, а й більш детально окреслити сильні й слабкі сторони кожного рішення. Такий підхід забезпечує ґрунтовне розуміння того, які саме функціональні можливості варто врахувати під час подальшої розробки власного програмного продукту. Огляд і порівняння застосунків-аналогів стають важливими кроками, що значно спрощують процес формування специфікації вимог до майбутнього програмного забезпечення (ПЗ). Із цією метою для проведення аналізу було відібрано низку найбільш популярних і поширених програмних рішень у сегменті виявлення маніпуляцій: Fallacy Detector (табл. 1.1), Gaslight Check (табл. 1.2), Fallacy Finder (табл. 1.3).

1.3.1 Fallacy Detector

Fallacy Detector – це онлайн-інструмент, призначений для виявлення логічних хиб у текстах. Сервіс аналізує введений користувачем матеріал і визначає, чи містить він маніпулятивні прийоми, софізми або помилки аргументації. Його основна мета полягає у підвищенні критичного мислення, допомозі користувачам краще розуміти структуру і якість аргументів. Інструмент може бути корисним як для студентів і викладачів, так і для журналістів, які працюють із великими обсягами текстової інформації [1].

Таблиця 1.1 – Опис Fallacy Detector

Назва	**Fallacy Detector**
Розробник	Logical Fallacies Project
Архітектура	Client-server
Мови реалізації	Python – backend JavaScript, HTML, CSS – frontend
Функції	1) аналіз введеного тексту на наявність логічних хиб і маніпулятивних прийомів; 2) виявлення широкого спектру софізмів і помилок аргументації; 3) надання пояснення щодо виявлених логічних помилок із прикладами; 4) класифікація знайдених помилок за категоріями і підрахунок їх кількості в тексті для узагальненої оцінки рівня маніпулятивності; 5) візуалізація результатів аналізу у зручному форматі (список виявлених хиб, короткі описи, іноді підсвічування у самому тексті).
Переваги	1) доступність онлайн без необхідності встановлення програмного забезпечення; 2) аналіз тексту до 10000 символів і PDF документів для користувачів із платною підпискою; 3) простий і інтуїтивно зрозумілий інтерфейс, що робить сервіс зручним у використанні для різних категорій користувачів.
Недоліки	1) працює лише з текстами англійською мовою; 2) не підтримує великий обсяг тексту для аналізу (до 1000 символів у безкоштовній версії). 3) погана реалізація авторизації користувача (після входу до облікового запису оновлення сторінки призводить до автоматичного виходу з облікового запису)
Вебсайт	https://finder.logicalfallacies.org/detector/

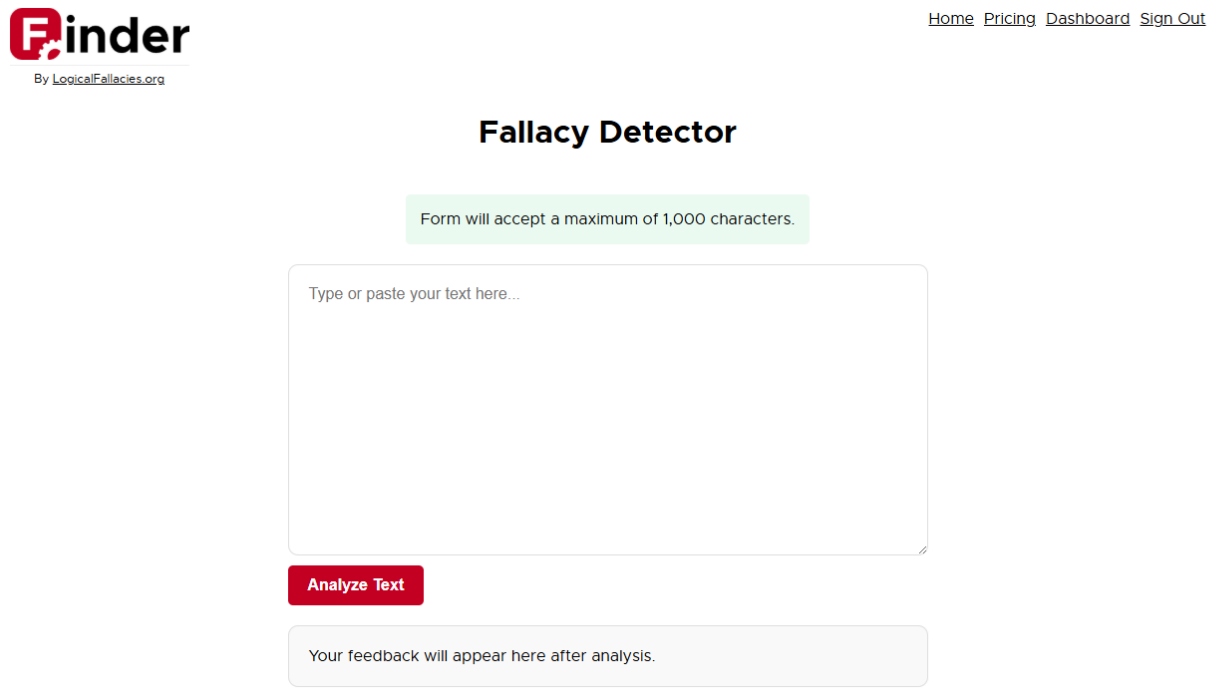


Рисунок 1.1 – Дизайн інтерфейсу застосунку «Fallacy Detector»

Графічний інтерфейс користувача створено з акцентом на мінімалізм і простоту використання. Основні пункти меню розташовані на верхніх панелях навігації, забезпечуючи легкий доступ до ключових функцій застосунку:

– логотип Finder (сторінка з описом логічних хиб);

– home (домашня сторінка);

– pricing (сторінка з описом можливостей при платній або безкоштовній підписках);

– dashboard (опис інструментів при преміум підписці)

– sign out (вихід з облікового запису).

Хоча Fallacy Detector пропонує дієвий інструмент для виявлення широкого спектру маніпуляцій із простим інтерфейсом і можливістю зворотнього зв’язку, його недоліки також слід враховувати. Серед них відсутність розширеної навігації чи додаткових функцій для кастомізації чату, а також серйозні проблеми з авторизацією користувачів. Такі недоліки можуть ускладнити UX (user experience) для швидких перевірок тексту без входу в систему.

1.3.2 Gaslight Check

Gaslighting Check – це онлайн-сервіс, розроблений для виявлення у текстах проявів газлайтингу й маніпулятивних висловлювань. Платформа аналізує введені користувачем повідомлення і визначає, чи містять вони ознаки психологічного тиску, перекручування фактів або нав'язування сумнівів у власному сприйнятті. Сервіс надає зрозумілий висновок із зазначенням рівня ризику маніпуляцій і класифікацією використаних технік [2].

Таблиця 1.2 – Опис Gaslight Check

Назва	**Gaslight Check**
Розробник	GaslightingCheck.com
Архітектура	Client Server
Мова реалізації	Python – backend JavaScript, HTML, CSS – frontend
Функції	1) аналіз текстів на прояви маніпуляцій у новинах і особистому спілкуванні; 2) можливість аналізу тексту, зображень і промов в аудіо або відео форматах; 3) AI Coach, що надає персоналізовані рекомендації стосовно розуміння маніпулятивних патернів; 4) оцінка рівня ризику маніпуляції для кожного повідомлення; 5) візуалізація результатів у зручному форматі (список проблемних фраз, підсвічування тексту).
Переваги	1) доступність онлайн без необхідності встановлення програмного забезпечення; 2) можливість аналізувати маніпуляції не лише у текстовому, а також у відео й аудіоформатах; 3) сучасний, інтуїтивно зрозумілий інтерфейс; 4) можливість працювати з різними мовами, що використовують зокрема кирилицю, латиницю.
Недоліки	1) занадто обмежений доступ до функціоналу для користувачів, що не мають платної підписки; 2) у деяких випадках хибно визначає вид маніпулятивну техніку або зовсім не знаходить її.
Вебсайт	https://www.gaslightingcheck.com/dashboard/check

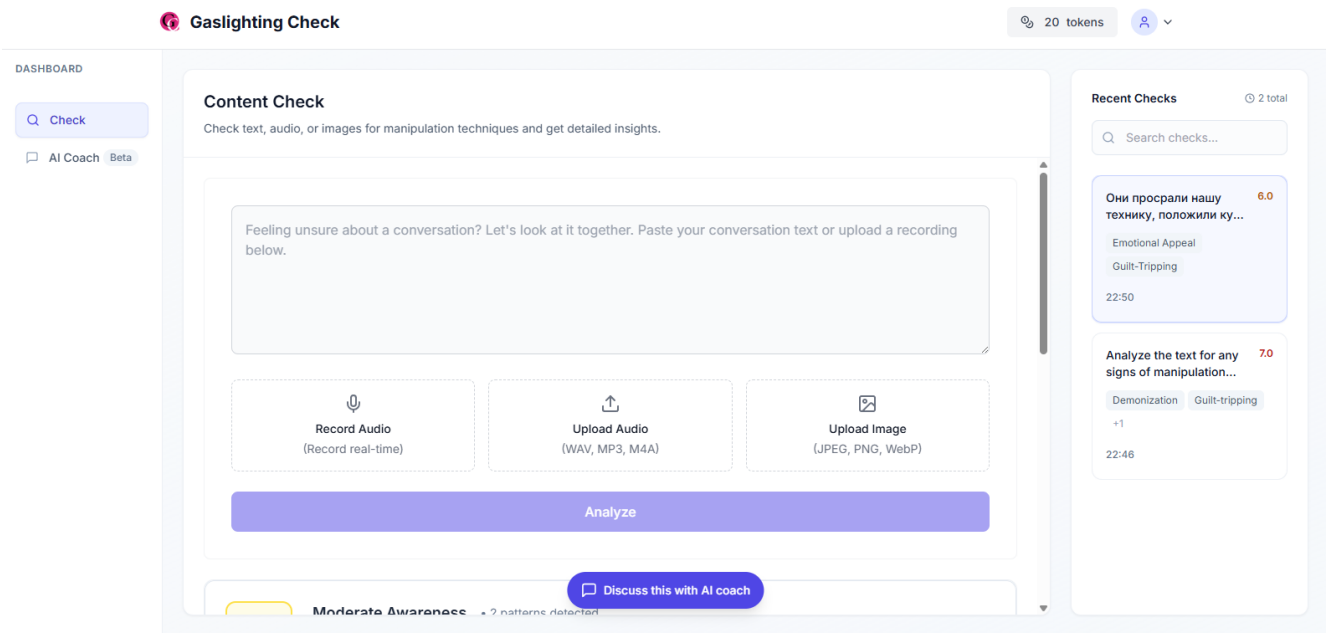


Рисунок 1.2 – Вигляд інтерфейсу застосунку «Gaslight Check»

Інтерфейс користувача є зручним і простим, проте його не можна налаштовувати. Також слід зазначити, що Gaslight Check працює з різними розширеннями файлів і підтримує аналіз маніпуляцій представлених різними мовами.

Крім того, його можливості разом з додатковим ботом AI Coach дозволяють ретельніше аналізувати не лише маніпулятивні техніки у сфері новин, а також в особистому спілкуванні. Наприклад, бот може надати вам персоналізовані рекомендації щодо стратегії спілкування з людиною, яка застосовувала до вас ті чи інші прийоми, щоб ввести вас уману або для власних інтересів. На жаль, доступна лише бета-версія, що значно знижує ефективність інтерпритації раніше проведеного аналізу на предмет маніпуляцій.

Хоча онлайн-сервіс Gaslight Check пропонує широкий спектр функцій для виявлення маніпулятивних висловлювань у текстах, зображеннях, аудіо- і відеофайлах, проте його інтерфейс, попри інтуїтивну простоту, страждає від відсутності можливостей налаштування. Крім того, обмежений доступ до повного функціоналу для неплатних користувачів не дає можливості у повній мірі оцінити якість застосунку й робить процес використання менш ефективним.

1.3.3 Fallacy Finder

Fallacy Finder – це онлайн-інструмент для аналізу текстів, що спеціалізується на виявленні логічних хиб і маніпулятивних аргументів у повідомленнях. Сервіс дозволяє користувачам вставити текст або завантажити уривок для перевірки, після чого система автоматично ідентифікує поширені софізми, помилки аргументації і ненадійні риторичні прийоми. Загалом застосунок призначений для аналізу тексту із метою виявити помилки, що допоможе користувачу зміцнити свої аргументи й уникнути поширених помилок у міркуваннях [3].

Таблиця 1.3 – Опис Fallacy Finder

Назва	**Fallacy Finder**
Розробник	Word.Studio
Архітектура	Client server
Мови реалізації	Python – backend JavaScript, HTML, CSS – frontend
Функції	1) можливість аналізу тексту із використанням класифікації маніпуляційних технік і виділенням провокативних рядків; 2) можливість скопіювати результати аналізу; 3) проведення аналізу контенту на різних мовах, що включає тексти на кирилиці, латиниці й ієрогліфи; 4) можливість використання сервісу без реєстрації шляхом вставлення тексту у потрібне поле. 5) надання змістовних пояснень щодо маніпулятивної частини тексту, зважаючи на контекст.
Переваги	1) детальний аналіз із поясненнями про використані маніпуляційні техніки і їх роль у впливі на думку читача; 2) відсутня потреба у реєстрації або авторизації в застосунку; 3) аналіз тексту за широким спектроми маніпулятивних технік.
Недоліки	1) немає можливості переглянути раніше проаналізовані повідомлення; 2) відсутній функціонал для кастомізації робочого середовища. 3) інколи помилково присвоює тексту техніки маніпуляції, що там не зустрічаються
Вебсайт	https://word.studio/tool/fallacy-finder/?utm_source=chatgpt.com

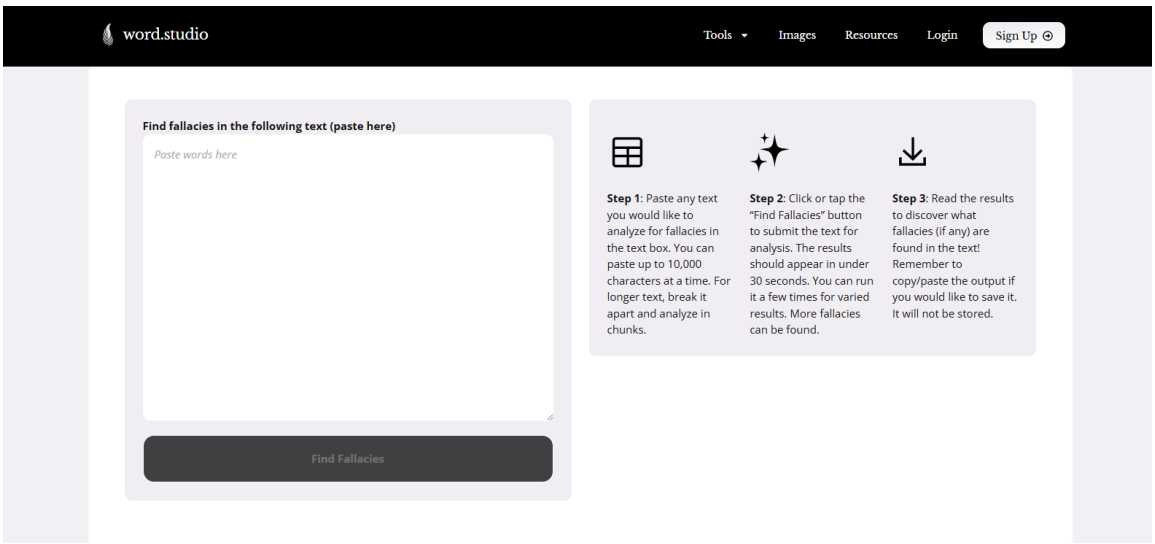


Рисунок 1.3 – Вигляд інтерфейсу застосунку «Fallacy Finder»

Інтерфейс Fallacy Finder вирізняється своєю інтуїтивною простотою, дозволяючи користувачам швидко аналізувати текст без необхідності реєстрації. Сервіс підтримує широкий спектр мов, включаючи кирилицю, латиницю і ієрогліфи, що робить його зручним для міжнародної аудиторії. У свою чергу детальні пояснення маніпулятивних технік додають цінності для користувачів, які прагнуть удосконалити свої аргументи.

Однак відсутність функцій кастомізації й можливості перегляду попередніх перевірок на наявність маніпуляцій може дещо ускладнити роботу з інструментом. Наприклад, користувачі, що потребують систематичного підходу або хочуть повернутися до раніше перевіреного тексту. Крім того, періодичні помилки у визначенні маніпулятивних технік нагадують про обмеження автоматизованого аналізу.

Загалом, попри свої сильні сторони, доступність і широкий спектр функцій, Fallacy Finder міг би бути ще ефективнішим при наявності гнучкості налаштувань й особистого кабінету з історією перевірок. Це суттєво підвищило б ефективність використання платформи, оскільки у її нинішньому вигляді сервіс залишається недостатньо зручним для користувачів, що працюють із великими використання і обсягами текстів, де потрібно порівнювати рівень маніпуляцій з різних джерел.

1.4 Аналіз розроблюваного застосунку

Інструменти для виявлення маніпуляцій у текстах сьогодні набувають дедалі більшої актуальності, адже інформаційний простір переповнений повідомленнями з прихованим впливом на сприйняття новин і політичних процесів.

Telegram-бот для пошуку маніпуляцій можна використовувати для швидкого аналізу повідомлень: достатньо скопіювати текст або переслати його з будь-якого каналу, щоб отримати результати з підсвіченими фрагментами і визначенням технік, що були застосовані. Додатково бот надає інформацію про різні види маніпуляцій і їх визначення, що робить його практичним інструментом для перевірки. У майбутньому функціонал можна розширити за рахунок інтеграції нових сценаріїв використання. Наприклад, персоналізовані поради для уникнення маніпулятивного впливу або можливість аналізу мультимедійного контенту.

Таблиця 1.4 – Опис системи що розробляється

Основні задачі	1) прийом і обробка текстових повідомлень від користувачів; 2) класифікація тексту за типами маніпуляцій; 3) підсвічування у тексті проблемних фрагментів; 4) інформування користувача про помилку у випадку введення тексту непідтримуваною мовою; 5) надання довідкової інформації про види маніпуляцій; 6) можливість пересилання повідомлень із каналів до боту для аналізу.
Користувачі системи	1. користувач, що авторизований у Telegram;
Сценарії роботи	1) користувач вставляє скопійований текст у чат із ботом і отримує результат аналізу; 2) користувач пересилає повідомлення з будь-якого каналу для автоматичного аналізу; 3) користувач вводить текст непідтримуваною мовою (наприклад, англійською) й отримує повідомлення про помилку; 4) користувач після аналізу тексту обирає функцію для отримання довідкової інформації про використані маніпулятивні прийоми .
Засоби апаратної та програмної реалізації	1. back-end: Python (AI-модель, NLP-бібліотеки), Telegram Bot API; 2. front-end: Telegram messenger (інтерфейс взаємодії з користувачем);

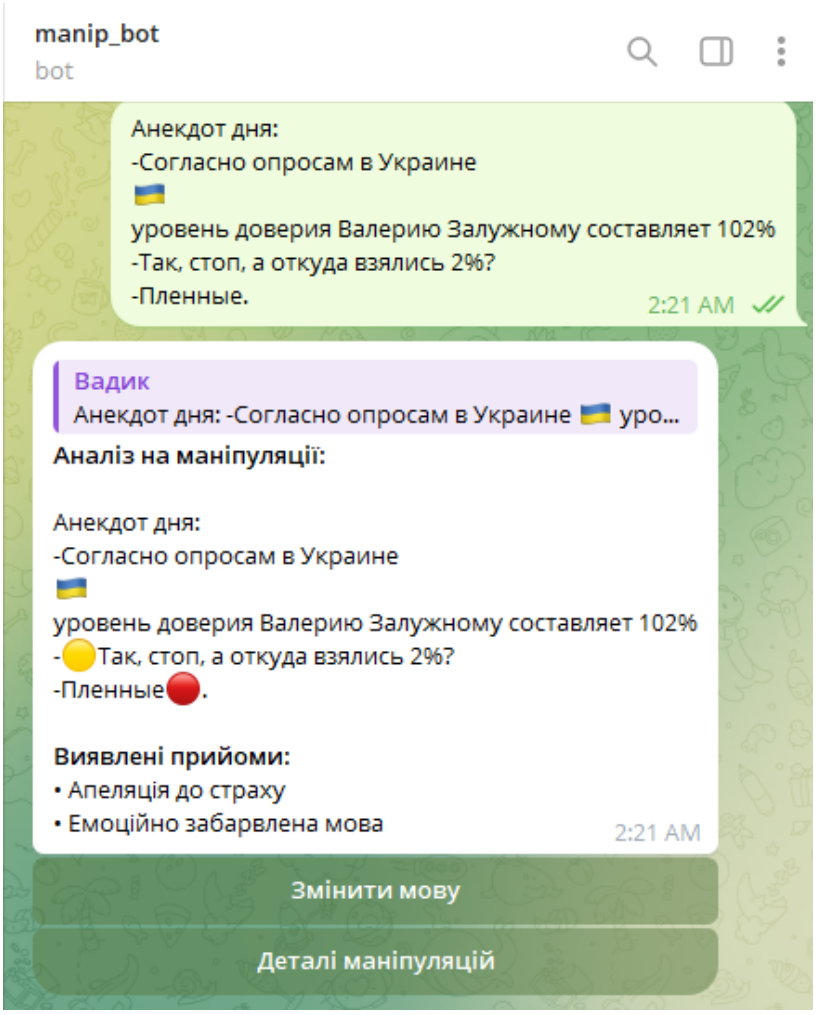


Рисунок 1.4 – Приклад роботи застосунку

Розроблюваний Telegram-бот має низку переваг у порівнянні з іншими рішеннями:

– зручність використання забезпечується завдяки інтеграції в популярному месенджері Telegram, який є одним із найпопулярніших ресурсів споживання новинного контенту в Україні. Користувачам треба лише скопіювати потрібний текст або переслати новину з будь-якого каналу без додаткових дій чи переходів на зовнішні сервіси;

– автоматична класифікація повідомлення й підсвічування маніпулятивних фрагментів, що робить процес аналізу швидким і зрозумілим;

– інформування про помилки у випадку введення непідтримуваних мов, що підвищує надійність і передбачуваність роботи;

Таким чином, бот забезпечує більш високу ефективність і зручність порівняно з альтернативними інструментами аналізу маніпулятивного контенту.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи магістра здійснено аналіз предметної області виявлення маніпулятивних технік у новинному контенті і проведено огляд наявних програмних рішень. Проаналізовано їхні функціональні можливості, архітектурні підходи, переваги й обмеження, що дозволило сформувати цілісне уявлення про сучасний стан засобів аналізу маніпулятивного контенту й обґрунтувати доцільність розробки власного Telegram-бота як зручного й доступного інструменту для користувачів.

Проведено детальне дослідження системи, що розробляється, шляхом визначення ключових функціональних завдань, основних сценаріїв взаємодії користувача з ботом і ресурсів, необхідних для її реалізації. Представлено концепцію структури Telegram-бота з описом вимог до функціоналу зважаючи на проведений аналіз застосунків аналогів і їх недоліків. Розглянуто перспективи подальшого вдосконалення рішення через розширення можливостей класифікації та додавання нових типів маніпуляцій.

За підсумками проведеного аналізу визначено основні напрями розвитку проєкту й сформульовано висновок щодо його актуальності й практичної цінності у сфері автоматизованого аналізу текстів. Розглянуті етапи проєктування і дослідження є важливими для побудови стратегії розвитку Telegram-бота й підкреслюють його переваги завдяки інтеграції в популярний месенджер та легкості у використанні.

2 ДОСЛІДЖЕННЯ, МОДЕЛЮВАННЯ І ТЕХНІЧНЕ ПРОЄКТУВАННЯ

2.1 Опис набору даних й ознак

Для навчання і оцінювання моделей у проєкті використано датасет, наданий у межах UNLP 2025 Shared Task on Detecting Social Media Manipulation [4]. Набір даних сформований командою Texty.org.ua й містить фрагменти текстів українською мовою, взяті з дописів у Telegram [5]. Датасет був створений на основі реальних повідомлень, що відображають інформаційний вплив і маніпулятивні практики в онлайн-контенті. Дані зібрані з метою аналізу маніпулятивних технік й ідентифікації фрагментів тексту, які містять маніпуляції.

Кожен приклад було анотовано за допомогою однієї або кількох із десяти маніпулятивних технік: straw_man, appeal_to_fear, fud, bandwagon, whataboutism, loaded_language, glittering_generalities, cherry_picking, euphoria і cliche. Анотацію виконували професійні журналісти, аналітики й медіаексперти, що в теорії повинно гарантувати високу якість розмітки. Водночас набір характеризується суттєвою диспропорцією між класами, що ускладнює побудову збалансованих моделей.

Для побудови моделей було використано файл train.parquet. Оцінювання результатів здійснювалось на офіційному тестовому наборі (test.csv) з використанням метрики Macro F1. Загальна структура файлів за підмножинами:

Таблиця 2.1 – Опис підмножин використаних на різних стадіях навчання

Підмножина	**Кількість прикладів**
Навчальна	3248
Валідаційна	574
Тестова	5735
Загальна кількість слів	805730
Унікальні слова	146410

Цільові змінні. Цільовими змінними в цьому дослідженні є поля manipulative для моделі класифікації й techniques для span-detection:

– маніпулятивність повідомлення (manipulative) – бінарна змінна, яка показує чи містить повідомлення ознаки маніпуляції. Значення true означає, що у тексті присутні маніпулятивні елементи, false – що повідомлення нейтральне. Це дозволяє

ставити завдання бінарної класифікації для моделей;

– маніпулятивні техніки (techniques) – список застосованих методів маніпуляції, таких як "loaded_language", "cherry_picking", "euphoria" й інші. Кожне повідомлення може містити кілька технік одночасно або зовсім не містити маніпуляцій;

У результаті аналізу користувач отримуватиме результат на предмет маніпуляцій беручи до уваги вищезазначені змінні.

Опис ознак. Датасет містить 6 основних ознак, які після обробки для категоріальних змінних (lang, manipulative). Загальна кількість записів становить 3822, кожен із яких відповідає одному повідомленню з Telegram. Ознаки можна розподілити на числові й категоріальні, а їхній опис наведено нижче:

– id – унікальний ідентифікатор кожного повідомлення. Використовується для однозначного посилання на конкретний текст у датасеті або при об’єднанні з іншими джерелами даних;

– content – текст повідомлення. Містить контент користувачів, який аналізується на наявність маніпуляцій. Довжина й складність тексту можуть суттєво варіюватися, що важливо враховувати при підготовці до Natural Language Processing (NLP) моделей (токенізація, векторизація, embeddings) [6];

– lang – мова повідомлення. Значення можуть бути «uk» для української або «ru» для російської. Ця ознака дозволяє проводити мультимовний аналіз і адаптувати алгоритми обробки тексту під конкретну мову, враховуючи морфологічні аспекти;

– manipulative – бінарна цільова змінна. Дозволяє однозначно класифікувати повідомлення як маніпулятивні або нейтральні;

– techniques – список технік маніпуляції, застосованих у повідомленні. Може містити один або декілька елементів. Кожна техніка описує специфічний прийом впливу на аудиторію. Для моделей машинного навчання техніки можуть бути закодовані у вигляді бінарних ознак для багатоміткової класифікації;

– trigger_words – список місць у тексті, де знаходяться слова або фрази, що є

тригерними. Позиції представлені у вигляді пар індексів [start, end], що дозволяє точно визначати місце впливу в тексті. Поділ інформації до вищезазначеного вигляду корисний для завдань span-detection.

Перед навчанням моделей тексти проходять ретельну багатоступеневу підготовку в межах єдиного гнучкого конвеєра обробки даних. Вхідні набори даних містили 3822 прикладів для тренування і 5735 прикладів для тестування у форматах parquet і csv. На початковому етапі оригінальний тренувальний набір був поділений на 85% тренувальних і 15% валідаційних підвибірок із використанням стратифікації за мітками маніпуляцій і фіксованого значення seed=42, що забезпечувало відтворюваність у межах обох завдань [7].

Також застосовувалася уніфікована процедура нормалізації тексту: заміна URL-адрес на спеціальний токен «[URL]», нормалізація пробілів, заповнення відсутніх значень, а також автоматичне визначення мови.

Таким чином, процес підготовки даних включає багатоступеневу нормалізацію тексту. Використання стратифікації при поділі тренувального набору на 85% тренувальних і 15% валідаційних прикладів забезпечило відтворюваність результатів, закладаючи міцну основу для ефективного навчання моделей і подолання викликів, пов’язаних із дисбалансом класів.

2.2 Огляд методологій і підходів для навчання моделей

Для отримання високих показників передбачень стосовно класифікації і знаходження span-ів потрібно проаналізувати підходи машинного навчання й особливості функціонування деяких моделей. Оцінка проводилась за макро-показниками:

– precision – частка правильно визначених позитивних прикладів серед усіх, які модель класифікувала як позитивні. Іншими словами, precision оцінює, наскільки передбачення є коректними;

– recall – відношення правильно знайдених позитивних прикладів до загальної кількості дійсно позитивних. Загалом характеризує здатність моделі

виявляти релевантні випадки;

– F1-score – гармонійне середнє між precision і recall, яке особливо корисне для незбалансованих датасетів;

Таким чином, наведені метрики дають змогу об’єктивно порівняти моделі і визначити, яка з них найкраще впорається із завданнями класифікації і виділення span-ів.

2.2.1 Аналіз Machine learning models

Machine learning (ML) models застосовуються як базовий підхід для задач класифікації технік маніпуляцій і визначення span-ів. Ці моделі добре підходять для створення репрезентативних базових рішень, швидких результатів і є відносно простими для інтерпретації. Водночас вони обмежені у захопленні складних контекстуальних залежностей у тексті, особливо якщо потрібно працювати з довгими або нелінійні зв’язками між словами. До ML models можна віднести Linear Support Vector Classification (SVC), Complement Naive Bayes (CNB), Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB). Усі вищезазначені моделі демонструють наступні результати в задачах класифікації (рис. 2.1) й span-detection [8].

Classifier	Precision	Recall	F1 Score
Technique Classification			
ML Models			
LinearSVC	0.3543	0.2878	0.3102
CNB	0.2680	0.2818	0.2553
LR	0.2807	0.5433	0.3291
RF	**0.5688**	0.1060	0.1309
GB	0.3926	0.1423	0.1846

Рисунок 2.1 – Результат застосування моделей в задачі класифікації

Linear SVC зазвичай добре працює з розрідженими матрицями TF-IDF і здатен захоплювати локальні кореляції між словами [9]. Під час тестування показав середню точність і низький recall – модель демонструє акуратність у визначенні позитивних прикладів. Загалом її передбачення переважно правильні, але вона

пропускає багато справжніх позитивних випадків, що знижує повноту класифікації. Така поведінка пов’язана з тим, що SVC оптимізує гіперплощину для максимального відділення класів, але не завжди враховує нерівномірний розподіл категорій і складні семантичні залежності між словами в тексті.

Complement Naive Bayes, навпаки, продемонстрував низькі значення усіх метрик, що підкреслює його обмежену здатність працювати з багатомітковими задачами при наявності дисбалансу класів. Алгоритм базується на простій ймовірнісній моделі, яка передбачає незалежність ознак, що не відповідає складній природі текстових даних, де слова взаємопов’язані й контекстуально впливають на значення один одного [10]. У результаті модель часто помиляється у багатокласових прикладах і не забезпечує адекватного покриття всіх технік маніпуляцій.

У той же час Random Forest показав дуже високий рівень точності, що говорить про малу кількість помилок при передбаченні позитивного класу. Не слід забувати про recall, який мав катастрофічно низькі значення. Із вищезазначеного випливає, що майже всі позитивні приклади залишаються непоміченими. Таку поведінку можна обґрунтувати архітектурою Random Forest побудованою на ансамблі рішень дерев [11]. Через цю особливість модель занадто сильно «концентрується» на тих ознаках, які найчастіше визначають правильний клас, ігноруючи менш очевидні сигнали. Вищезазначена обставина призводить до пропуску менш частих маніпулятивних технік, які знаходяться в датасеті.

Gradient Boosting працює подібно до Random Forest: модель намагається коригувати помилки попередніх дерев, що підвищує точність деяких передбачень [12]. Наявний набір даних характеризується обмеженим навчальним набором і великою кількістю класів, що вливає на низький показник F1 Score і відсутністю суттєвого покращення recall.

Таким чином, для задачі класифікації ML models можуть бути корисними для створення базового рішення, де необхідно відсіяти явні негативні або позитивні приклади, або провести попередній аналіз даних. Однак їхні обмеження полягають

у нездатності захоплювати складні семантичні залежності, довгі контексти, а також у проблемах із балансуванням precision і recall. Вищезазначені недоліки є критичними для багатоміткової класифікації з багатьма різними техніками маніпуляцій.

Span Identification			
ML Models			
LinearSVC	0.4020	0.3921	0.3970
LR	0.4169	0.3578	0.3851
MNB	0.4169	0.3578	0.3851
lightGBM	0.3599	0.4794	0.4112

Рисунок 2.2 – Результат застосування моделей в задачі span detection

Для задачі span-detection ситуація дещо відрізняється. У цьому випадку ML models працюють на рівні токенів. Основна задача полягає у пошуку початку і кінця span-ів, де наявні маніпулятивні фрагменти тексту. Linear SVC знову продемонстрував відносно збалансовані показники precision і recall, що робить його більш передбачуваним при роботі з окремими токенами. Logistic Regression і Multinomial Naive Bayes показали однакові результати: високу точність і трохи нижчий recall. Загалом вони добре ідентифікують токени, які точно належать до span-у, але деякі справжні токени залишаються невиявленими.

LightGBM відзначився найкращим F1 Score серед усіх MLmodels у виконанні span-detection. Основна його перевага – високий recall, який дозволяє знайти більшість маніпулятивних токенів, навіть якщо це супроводжується зниженням точності через хибнопозитивні передбачення. Особливість моделі відображає природну здатність градієнтного бустингу враховувати складні взаємозв’язки між ознаками, проте спостерігається менша стабільність у питаннях точності коротких або нечітких контекстів [13].

Загалом для span-detection ML models демонструють помірну ефективність: вони здатні захоплювати локальні контексти й деякі послідовності токенів, але не можуть повністю відтворити складні синтаксичні й семантичні залежності в тексті. Їх використання обмежується при детальному розпізнаванні маніпуляцій на рівні

контексту зважаючи на вищезазначені причини. Моделі більше підходять для швидкого прототипування і оцінки ефективності простих ознак, ніж для високоточних систем виявлення маніпуляцій, де важлива повнота й точність одночасно.

2.2.2 Аналіз Deep learning models

Для задач класифікації технік маніпуляцій і ідентифікації span-ів у тексті були застосовані різноманітні архітектури Deep Learning (DL), які дозволяють моделі захоплювати складні семантичні й контекстуальні залежності між словами. На відміну від традиційних моделей машинного навчання, DL моделі здатні ефективно інтегрувати як локальні патерни, так і довгострокові взаємозв'язки в тексті. Ця різниця критично важлива для багатоміткових задач і завдань послідовної розмітки.

Для класифікації технік маніпуляцій всі вхідні тексти представлялися у вигляді 300-вимірних BPEmb субсловних ембеддінгів, що дозволяє працювати навіть із рідковживаними словами, скороченнями й похідними формами [14].

DL Models			
CNN	0.2991	0.3287	0.2816
CNN+LSTM	0.3125	0.3388	0.3077
CNN+BiLSTM	0.3403	0.3443	0.3252
CNN+GRU	0.3649	0.3087	0.3179

Рисунок 2.3 – Результат застосування моделей в задачі класифікації

На практиці базова Convolutional Neural Network (CNN) модель досягла F1 Score = 0.2816, precision – 0.2991, recall – 0.3287. Результати свідчать про хороший пошук певних ознак маніпуляцій, проте здатність точно розрізняти всі техніки обмежена [15]. Основною причиною є те, що CNN добре захоплює локальні патерни, але не враховує довгострокові залежності між словами, що важливо для розпізнавання складних маніпулятивних структур [16].

Для вирішення цієї проблеми були розроблені гібридні CNN-RNN (Recurrent Neural Network) архітектури, де CNN виділяє локальні ознаки, а рекурентні шари – Long Short-Terrm Memory (LSTM), Bidirectional LSTM (BiLTSM) і Gated Recurrent Unit (GRU) – інтегрують контекст у межах усього тексту [17]. Наприклад, CNN+BiLSTM з двома шарами по 256 одиниць на напрямок (forward – обробка послідовності зліва направо, backward – у зворотному порядку, що дозволяє враховувати як попередній, так і наступний контекст) показала: F1 Score – 0.3252, precision – 0.3403, recall – 0.3443, що краще ніж у базової CNN [18]. Інша конфігурація, CNN+GRU, досягла найвищого F1 Score серед DL моделей для класифікації – 0.3179, із precision – 0.3649 й recall – 0.3087 [19]. Показники метрик підтверджують, що додавання рекурентних компонентів дозволяє моделі краще інтегрувати інформацію про порядок слів і контекст, що підвищує загальну ефективність багатоміткової класифікації.

DL Models			
CNN	0.2596	0.8715	0.4001
CNN+LSTM	0.2566	**0.9187**	0.4012
CNN+BiLSTM	0.2878	0.8126	0.4251
CNN+BiGRU	0.2949	0.8023	0.4313

Рисунок 2.4 – Результат застосування моделей в задачі span detection

Для задачі span-detection DL моделі демонструють іншу динаміку. Вхідні послідовності представлялися у вигляді 100-вимірних BPEmb ембеддінгів з 50.000 токенів, що fine-tune-илися під час навчання. Максимальна довжина послідовності становила 384 субслова, а CNN служив спільним фронтендом, початковим шаром для обробки ознак із тексту, для всіх архітектур span-detection. Він мав три паралельні Conv1D шари з ядрами 3, 5 та 7, кожний по 128 фільтрів, Rectified Linear Unit (ReLU) активацією і padding для збереження довжини. Dropout використовувався на початку й після конкатенації виходів шарів, щоб зменшити перенавчання.

У тестуванні базова CNN показала дуже високий recall – 0.8715, що вказує на знайдену більшість реальних span-ів. У той же час precision був лише 0.2596, що свідчить про значну кількість хибнопозитивних передбачень. Результати є довалі очікуваними, оскільки CNN добре виявляє локальні ознаки. У противагу вищезазначеному модель не враховує повний контекст послідовності, через що часто помилково відносить сусідні токени до span.

Додавання рекурентних шарів значно покращило баланс між precision і recall [20]. Таким чином CNN+BiGRU досягла F1 Score – 0.4313, precision – 0.2949 і recall – 0.8023. Модель більш точно виділяє span-и, зменшуючи хибнопозитивні передбачення, при цьому зберігаючи високу здатність знаходити реальні span-и. Подібна тенденція спостерігалася також у CNN+BiLSTM (F1 Score – 0.4251), що підтверджує ефективність використання двосторонніх рекурентних компонентів для інтеграції контексту при завданнях токен-левел розмітки.

Переваги Deep Learning архітектур очевидні: вони здатні працювати з субсловними ембеддінгами, ефективно інтегрувати локальні й глобальні ознаки, враховувати порядок слів і контекст, адаптуватися до багатоміткових завдань і послідовної розмітки [21]. Водночас існують певні обмеження: потреба у великих обсягах навчальних даних, висока обчислювальна складність, ризик перенавчання при невеликому наборіданих, а також складність інтерпретації передбачень.

Із аналізу результатів тестування можна зробити висновок, що в цілому, Deep Learning показує значний потенціал у завданнях інтеграції контексту і багатоміткової інформації в тексті порівняно з ML models.

2.2.3 Аналіз Transformer models

Для виконання задач класифікації технік маніпуляцій, ідентифікації span-ів у тексті були використані попередньо навчені Transformer-моделі, які демонструють високу ефективність у роботі з природною мовою завдяки механізму self-attention. Self-attention дозволяє моделі враховувати взаємозв’язки між кожним токеном у тексті і всіма іншими токенами, що дає можливість захоплювати як локальні, так і

глобальні контекстуальні залежності [22]. Така властивість особливо цінується в обох завданнях, оскільки класифікація і розмітка span-ів вимагають уваги до тонких семантичних деталей і взаємозв'язків у тексті. Часто прості моделі машинного або навіть глибокого навчання не можуть врахувати повністю вищезазначені аспекти.

Було обрано низку потужних багатомовних моделей з бібліотеки Hugging Face. Основні моделі включали mDeBERTa v3 Base, InfoXLM Large, XLM-RoBERTa Large, BERT Multilingual Base, а для класифікації українського тексту також застосовувалася спеціалізована Ukr-Roberta-Base. Для span-ів використовувався mT5 Base, який дозволяє моделювати завдання як послідовність генерації. Ця особливість допомагає точніше визначати початок і кінець span-ів.

Архітектурно всі моделі Transformer складаються з багатьох шарів, які дозволяють оцінювати важливість кожного слова відносно всього тексту [23]. Для адаптації до конкретних задач до них додавали спеціальні «голови» (heads): для класифікації – лінійні шари з Gaussian Error Linear Unit (GELU) активацією та multi-sample dropout для підвищення стійкості перед перенавчанням; для span detection – token classification heads із схемою BIO (Begin, Inside, Outside) для маркування початку, продовження і не span токенів.

Перед порівнянням моделей датасет проходив підготовку: видалялися URL, зайві пробіли, проводилась токенізація SentencePiece, усі послідовності вирівнювалися до 512 токенів. Для підвищення стійкості моделі застосовувалося випадкове видалення слів (rate 0.3), а дисбаланс класів компенсувався через Focal Loss або Binary Cross Entropy з вагами класів [24]. Також використовувалися label smoothing і Layerwise Learning Rate Decay, що знижувало ймовірність надмірної впевненості моделі у прогнозах і забезпечувало стабільне оновлення градієнтів [25].

Transformers			
mDeBERTa V3 Base	0.3453	0.5055	0.3901
InfoXLM Large	0.3855	0.5477	0.4451
XLM-RoBERTa-large	0.3917	**0.5667**	**0.4498**
BERT multilingual base	0.3710	0.3930	0.3772
Ukr-Roberta-Base	0.3687	0.4366	0.3660

Рисунок 2.5 – Результат застосування моделей в задачі класифікації

Результати тестування Transformer-моделей для класифікації технік маніпуляцій демонструють значне покращення у порівнянні з DL і ML моделями. XLM-RoBERTa-large показала F1 Score – 0.4498, precision – 0.3917 і recall – 0.5667, що є найкращим результатом серед усіх протестованих моделей. InfoXLM Large досягла F1 Score – 0.4451, precision – 0.3855 і recall – 0.5477. mDeBERTa V3 Base також продемонструвала добрий баланс між precision і recall (F1 Score – 0.3901), тоді як BERT Multilingual Base показала дещо нижчі значення (F1 – Score 0.3772). Ukr-Roberta-Base, спеціалізована під українську мову, дала F1 Score 0.3660, що вказує не на користь використання мовно-орієнтованих моделей. Стосовно вищезазначеного показника моделі можна зробити висновок, що обсяг і різноманітність даних важливі для досягнення найкращих показників.

Transformers			
infoXLM-large	0.5646	0.5510	0.5577
mDeBERTa-v3-base	**0.6367**	0.4644	0.5371
XLM-RoBERTa-large	0.5616	0.6500	**0.6026**
BERT-base-multilingual	0.5188	0.5697	0.5431
mt5-base	0.3930	0.6645	0.4939

Рисунок 2.6 – Результат застосування моделей в задачі span detection

Для задачі span-detection Transformer-моделі показали ще більш виражену перевагу у порівнянні з DL архітектурами. Наприклад, XLM-RoBERTa-large досягла F1 Score – 0.6026, precision – 0.5616 і recall – 0.6500, демонструючи чудовий баланс між точністю і покриттям span-ів. mDeBERTa V3 Base показала F1 Score – 0.5371 із високим precision – 0.6367, але нижчим recall – 0.4644. Результат свідчить про консервативну поведінку моделі: вона робить менше помилкових передбачень,

проте частково пропускає реальні span-и. InfoXLM Large досягла F1 Score – 0.5577, precision – 0.5646 і recall – 0.5510, демонструючи відмінну збалансованість. BERT Multilingual Base показала F1 Score – 0.5431, precision – 0.5188 і recall – 0.5697, що свідчить про помірну здатність моделі до узагальнення результатів і стабільну продуктивність без суттєвих перекосів між точністю і повнотою. mT5 Base має F1 Score – 0.4939 із високим recall – 0.6645, але низьким precision – 0.3930, що вказує на тенденцію до надмірного включення span-ів.

З аналізу результатів можна зробити такі висновки. Transformer-моделі значно перевершують традиційні ML і навіть класичні DL моделі. Особливо ця перевага помітна при span-detection, де досліджувані моделі ефективніше інтегрували контекст у межах всього тексту. Transformer-моделі дозволяють одночасно враховувати локальні патерни, глобальні залежності й семантичні нюанси, що важливо для завдань багатоміткової класифікації і токен-розмітки. Основними перевагами є висока точність, баланс між precision і recall, здатність працювати з багатомовними даними, стійкість до рідковживаних слів завдяки субсловним ембеддінгам. Недоліки – потреба у великих обсягах даних і ресурсах graphic processing unit (GPU), довге навчання, відносна складність інтерпретації результатів, а також чутливість до гіперпараметрів fine-tuning і pre-training стратегії [26].

Загалом Transformer-підходи демонструють найбільший потенціал для задач, де важливі контекстуальні деталі, багатомітковість і точне виділення span-ів. Можна підсумувати, що використання цих модей є доцільним в аналітичних системах розпізнавання маніпулятивних технік у тексті.

2.2.4 Вибір моделі для виконання машинного навчання

Із аналізу результатів трьох сімейств моделей – ML models, DL і Transformer-based – можна зробити комплексні висновки щодо їх придатності для завдань багатоміткової класифікації технік маніпуляції і точного span-detection у текстах українською і російською мовами. Оцінка проводилась за макро-показниками

precision, recall і F1-score. Вищезгадані метрики дозволяють збалансовано порівнювати моделі на різноманітних підзадачах, враховуючи специфіку багатомовних даних. Українська й російська мови мають спільні морфологічні риси, але відрізняються лексичними й синтаксичними нюансами, що ускладнює обробку.

Classifier	Precision	Recall	F1 Score
Technique Classification			
ML Models			
LinearSVC	0.3543	0.2878	0.3102
CNB	0.2680	0.2818	0.2553
LR	0.2807	0.5433	0.3291
RF	**0.5688**	0.1060	0.1309
GB	0.3926	0.1423	0.1846
DL Models			
CNN	0.2991	0.3287	0.2816
CNN+LSTM	0.3125	0.3388	0.3077
CNN+BiLSTM	0.3403	0.3443	0.3252
CNN+GRU	0.3649	0.3087	0.3179
Transformers			
mDeBERTa V3 Base	0.3453	0.5055	0.3901
InfoXLM Large	0.3855	0.5477	0.4451
XLM-RoBERTa-large	0.3917	**0.5667**	**0.4498**
BERT multilingual base	0.3710	0.3930	0.3772
Ukr-Roberta-Base	0.3687	0.4366	0.3660
Span Identification			
ML Models			
LinearSVC	0.4020	0.3921	0.3970
LR	0.4169	0.3578	0.3851
MNB	0.4169	0.3578	0.3851
lightGBM	0.3599	0.4794	0.4112
DL Models			
CNN	0.2596	0.8715	0.4001
CNN+LSTM	0.2566	**0.9187**	0.4012
CNN+BiLSTM	0.2878	0.8126	0.4251
CNN+BiGRU	0.2949	0.8023	0.4313
Transformers			
infoXLM-large	0.5646	0.5510	0.5577
mDeBERTa-v3-base	**0.6367**	0.4644	0.5371
XLM-RoBERTa-large	0.5616	0.6500	**0.6026**
BERT-base-multilingual	0.5188	0.5697	0.5431
mt5-base	0.3930	0.6645	0.4939

Рисунок 2.6 – Порівняння моделей різних сімейств

Традиційні моделі машинного навчання забезпечили базову точку відліку для порівняння, базуючись на простій векторизації тексту за допомогою TF-IDF або Bag-of-Words. Цей підхід робить їх чутливими до поверхневих патернів, але обмеженими в обробці глибоких семантичних зв'язків. Для задачі класифікації технік найбільш збалансований підхід продемонструвала Logistic Regression, яка досягає високого recall завдяки здатності ефективно моделювати лінійні залежності

між ознаками, дозволяючи виявляти широкий спектр маніпулятивних елементів. Серед недоліків слід виділити помірний precision через вразливість до шуму в текстах, сленгу, емодзі, абревіатурах, що ускладнює точну диференціацію. Random Forest вирізняється високою точністю за рахунок ансамблевого алгоритму, який агрегує рішення множини дерев для уникнення помилкових позитивів і фокусування на стабільних патернах. Тим не менш це призводить до критично низького recall, оскільки модель надмірно консервативна і часто пропускає тонкі маніпулятивні сигнали в довгих послідовностях тексту [27]. У задачі span-detection лідером серед ML-методів став LightGBM, чия перевага полягає в градієнтному бустингу. Він дозволяє швидко ітеративно покращувати виявлення токенів із високим recall завдяки фокусу на дрібних ознаках (granural features). У цьому випадку помірна точність пояснюється обмеженою здатністю точно окреслювати межі span-ів без врахування глобального контексту [28]. Logistic Regression і Multinomial Naive Bayes досягають кращого precision за рахунок спрощення моделі – перша через лінійну апроксимацію, а друга через припущення незалежності ознак. У результаті зменшується чутливість до кореляцій, проте при цьому спостерігається менший recall. Вищезазначене зауваження в більшій мірі стосується Naive Bayes, яка погано працює з умовними ймовірностями в багатомовних даних із рідковживаними термінами через обмежену статистичну базу. Результати демонструють, що класичні ML-підходи мають фундаментальні обмеження у балансуванні між точністю і відновленням на складних багатомовних даних. Серед недоліків слід виділити відсутність механізмів для захоплення контекстуальних і семантичних нюансів, що призводить до неповного охоплення маніпулятивних патернів.

Глибинні моделі продемонстрували змішані переваги. З одного боку вони дозволяють інтегрувати послідовні залежності через рекурентні шари й локальні патерни за допомогою згорткових шарів (convolutional layers). З іншого боку моделі обмежені фіксованою довжиною вхідних послідовностей і меншою адаптивністю до багатомовності без спеціального pre-training. Найбільш успішним виявився

CNN+BiLSTM, чия перевага полягає в двосторонньому врахуванні контексту (forward-backward), що дозволяє ефективно моделювати поступове наростання маніпулятивних наративів у текстах. Вищезгадана перевага дозволяє захоплювати як попередні, так і наступні сигнали для кращого розпізнавання складних патернів, на відміну від односторонніх підходів. Інші гібридні архітектури на основі CNN+GRU або CNN+LSTM також перевищували показники простого CNN. Рекурентні шари, такі як GRU для ефективної обробки довготривалих залежностей або LSTM з клітинковою пам'яттю для запобігання зникненню градієнтів додають здатність утримувати послідовні зв'язки в реченнях. Такий підхід особливо корисний для текстів із повторюваними мотивами – фейкові факти чи емоційні апеляції. У той же час звичайний CNN обмежується лише локальними згортками (convolutional) без динамічного контексту. У задачі span-detection найвищий баланс показників продемонстрував CNN+BiGRU, де високий recall пояснюється бінаправленим GRU. Він фокусується на гранулярному виявленні токенів через швидку обробку послідовностей і чутливість до локальних feature. Таким чином охоплюється широкий спектр потенційних маніпулятивних елементів. Також слід зазначити, що низький precision зумовлений обмеженою здатністю точно окреслювати межі span-ів. Цей недолік спричинений тим, що згорткові шари акцентують на поверхневих патернах, а не на глобальному семантичному контексті. У результаті відбувається надмірне розширення або звуження фрагментів. Загалом виконується ефективне визначення токенів у шумних багатомовних текстах, проте не зовсім точне визначення меж span-ів через відсутність глибоких механізмів самоуваги для інтеграції віддалених залежностей.

Найбільш виразно переваги демонстрували трансформерні моделі, які значно перевершили як традиційні ML, так і класичні DL-підходи. Причною цьому є архітектура, що одночасно враховує локальні патерни, глобальні залежності й семантичні нюанси тексту через динамічні механізми уваги. Завдяки вищезазначеному моделі здатні глибше розуміти контекст маніпулятивних наративів у шумних Telegram-повідомленнях. Для класифікації технік найкращим

виявилась XLM-RoBERTa-large, чия перевага полягає в оптимізованому pre-training на масивних багатомовних текстах з акцентом на динамічне маскування токенів (masked language modeling). Модель не тільки ефективно розпізнає тонкі семантичні зв'язки між словами, а також адаптується до крос-лінгвальних подібностей між українською і російською. У результаті XLM-RoBERTa-large забезпечує збалансоване розпізнавання як явних, так і прихованих маніпулятивних елементів [29]. Окрім цього вона вирізняється глибокими шарами self-attention, які динамічно зважують релевантність кожного токена. Як наслідок представлена особливість дозволяє перевершувати інші моделі в захопленні довготривалих залежностей у великих послідовностях тексту, де контекст будується поступово. Аналогічно для span-detection XLM-RoBERTa-large демонструє перевагу завдяки субсловним ембеддінгам (subword embeddings), які точно обробляють рідковживані терміни й морфологічні варіації в обох мовах. Завдяки цьому виявляє токени з маніпулятивним забарвленням і точно окреслює їхні межі через інтеграцію глобального контексту, що робить її ідеальною для гранулярного аналізу в реальному часі. Порівняно з іншими трансформеними моделями має меншу чутливість до шуму від емодзі чи сленгу. При співставленні з InfoXLM Large, яка ефективна в інформаційних текстах завдяки спеціалізованому pre-training на енциклопедичних даних, усе ж поступається в гнучкості для неформальних контекстів через меншу глибину шарів. Моделі mDeBERTa v3 base, мультимовний BERT також перевершують ML і DL підходи, але поступаються XLM-RoBERTa-large в точності семантичного моделювання. Причина полягає у менш оптимізованих механізмах уваги для багатомовних нюансів і довготривалих залежностей. Таким чином обмежується точність у визначенні складних маніпулятивних патернів із змішаним використанням декількох мов (code-switching). Високі результати трансформермених моделей загалом пояснюються попереднім тренуванням на багатомовних датасетах, що включають мільйони прикладів з подібних джерел. Не менш важливу роль відіграють глибокі механізми self-attention, які дозволяють моделі уловлювати тонкі контекстуальні сигнали –

сарказм, евфемізми й семантичні взаємозв'язки між токенами. Водночас серед недоліків XLM-RoBERTa-large варто відзначити її високу обчислювальну складність через велику кількість параметрів. У результаті навчання вимагатиме значних ресурсів GPU для fine-tuning і inference. Модель також може мати потенційну схильність до перенавчання (overfitting) на специфічних тренувальних патернах, якщо датасет недостатньо різноманітний. Вищезазначена обставина може знизити стабільність на еволюціонуючих лінгвістичних трендах.

Основні переваги трансформерних моделей включають високу точність, збалансованість між precision і recall, здатність працювати з багатомовними даними й ефективну обробку рідковживаних слів завдяки subword embeddings. До недоліків належать висока потреба у великих обсягах даних і ресурсах GPU, тривале навчання, відносна складність інтерпретації результатів, а також сильна чутливість до налаштувань гіперпараметрів під час fine-tuning і pre-training.

Отже, трансформерні моделі демонструють найвищий потенціал для задач, де критично важливі контекстуальні деталі, багатомітковість і точне виділення span-ів. Також це пояснюється їх архітектурою, яка дозволяє динамічно інтегрувати локальні й глобальні семантичні сигнали. Трансформерні моделі особливо ефективні для виявлення прихованих маніпулятивних патернів у неформальних, шумних текстах, де традиційні підходи гублять нюанси через обмежену обробку залежностей. Ці переваги особливо виражені в багатомовному середовищі української і російської, де моделі можуть знаходити крос-лінгвальні подібності – спільні морфологічні конструкції чи лексичні запозичення. Їх пошук виконується без потреби в окремому препроцесингу (preprocessing), забезпечуючи стійкість до варіацій сленгу. Серед усіх transformation models XLM-RoBERTa-large виявилася найбільш придатною для fine-tuning, завдяки своєму оптимізованому pre-training на масивних багатомовних об'ємах тексту з динамічним маскуванням. Вищезгадана перевага дозволяє моделі не тільки швидко адаптуватися до специфічних доменів маніпулятивних технік, а також зберігати глибоке розуміння семантики. У свою чергу досягається збалансоване співвідношення між точністю і повнотою

результатів. Високі показники забезпечуються завдяки ефективному зважуванню релевантних токенів і врахуванню довготривалих контекстуальних зв’язків. Такий підхід є оптимальним для обох підзадач – класифікації технік і виявлення span-ів.

2.3 Стратегії навчання для оптимізації трансформерних моделей у завданні розпізнавання маніпулятивних технік

На основі попереднього аналізу ефективності моделей було встановлено, що трансформерні архітектури мають найвищий потенціал для багатоміткової класифікації технік маніпуляції і span-detection у текстах українською, російською мовами. Наступоню не менш важливою задачею є вибір релевантої стратегії навчання. Деякі підходи фокусуються на використанні претренованих (pre-trained) моделей, тонких налаштуваннях (fine-tuning) і гібридних підходах, які адаптовані до специфіки набору даних – обмеженість анотованих даних у домені маніпулятивних текстів, багатомовність й потреба в контекстуальній глибині. Кожна стратегія має свої особливості, переваги і недоліки, тому слід приділити увагу правильному вибору зважаючи на доступні ресурси central processing unit (CPU), GPU.

Pre-training ґрунтується на моделюванні на масивних загальних корпусах з використанням завдань, таких як masked language modeling (MLM) або next sentence prediction (NSP), для захоплення семантичних та синтаксичних особливостей [30]. У контексті задачі перевагами цієї стратегії є фундаментальне розуміння багатомовних нюансів у російських і українських текстів, що дозволяє ефективно уловлювати тонкі маніпулятивні сигнали без залучення специфічних даних, а також створення міцної базової семантики для низькоресурсних мов з крос-лінгвальними подібностями, що сприяє узагальненості на небачених патернах у Telegram-текстах. Обмеженнями є значні вимоги до обсягів даних у вигляді мілярдів токенів і ресурсів, що включають значного часу обчислень на кластерах GPU. Також важливу роль відіграє недостатня адаптація до домену Telegram, де

шумні неформальні тексти виходять за межі типових корпусів попереднього навчання і потребують додаткового доменного або завданнєвого донавчання.

Fine-tuning передбачає адаптацію претренованої моделі на домен-специфічних анотованих даних з повним або частковим оновленням параметрів (наприклад, за допомогою оптимізації AdamW із learning rate $1e^{-5}$) [31]. У контексті задачі перевагами цієї стратегії є покращення F1-метрики на 20–30% для класифікації технік маніпуляції, швидкого виявлення span-ів, забезпечення моделі здатністю оперувати мітками маніпуляцій. Також варто згадати про баланс між precision і recall завдяки gradient clipping і dropout. що запобігає оверфіттингу на шумних даних, роблячи її придатною для середніх датасетів 5-10 тис. рядків. Обмеженнями є чутливість до гіперпараметрів і ризик catastrophic forgetting загальних знань, особливо на обмежених датасетах, де можливі упередження в багатомітковості.

Transfer Learning з домен-адаптацією (domain adaptation) охоплює перенесення знань із загального тексту до маніпуляцій через антагоністичне навчання (adversarial training) або продовжене претренування на незаанотованих даних [32]. У контексті задачі перевагами цієї стратегії є адаптація претренованої моделі до неформального стилю Telegram, покращення узагальненості на небачених маніпуляціях, а також ефективність для багатомовності, де адаптація на змішаних україно-російських корпусах зменшує помилки на 15–20% у виявленні span-ів, сприяючи стійкості до доменних розбіжностей. Обмеженнями слугують підвищена складність і потенційне розмивання семантики при значних розбіжностях між доменами, що збільшує обчислювальну навантаження.

Few-Shot Learning з Prompt Engineering базується на використанні претренованої моделі з мінімальними прикладами через інженерію промптів або in-context learning, без повного fine-tuning [33]. У контексті задачі перевагами цієї стратегії є ідеальна придатність для низькоресурсних сценаріїв (обмежена кількість анотацій маніпуляцій), швидке тестування на нових маніпулятивних техніках з recall понад 50% за рахунок семантичного розуміння, а також економія ресурсів

через акцент на креативних промптах для виділення span-ів у текстах. Обмеженнями є нижча точність і нестабільність на багатоміткових задачах, де промпт може «забути» кілька міток, що знижує надійність для повноцінної аналітики.

Continual Learning полягає в послідовному оновленні моделі на нових даних без забування попередніх знань за допомогою elastic weight consolidation (EWC) або replay buffers [34]. У контексті задачі перевагами цієї стратегії є доречність для еволюціонуючого маніпулятивного контенту Telegram, збереження продуктивності на попередніх техніках при адаптації до нових з мінімальною втратою F1. Також спостерігається користь для багатомовності, де модель «пам’ятає» лінгвістичні варіації, забезпечуючи довгострокову стійкість. Обмеженнями є потреба в буферах даних для повтору, що збільшує обсяг пам'яті й складність масштабування на великі моделі без регуляризації, що ускладнює початкове налаштування.

Найоптимальнішою стратегією для вирішення поставлених завдань є fine-tuning XLM-RoBERTa-large, яка забезпечує найкраще співвідношення точності, стабільності й ресурсних витрат. Саме fine-tuning дозволяє швидко адаптувати вже потужну багатомовну модель до специфіки Telegram-домену, використовуючи навіть відносно невеликі анотовані корпуси, при цьому гарантує приріст F1-метрики понад 0.50 для класифікації технік і визначення span-ів. На відміну від pre-training, що потребує колосальних ресурсів і часу, fine-tuning забезпечує оперативну спеціалізацію. У той час порівняно з domain adaptation уникає надмірної складності антагоністичних підходів, досягаючи схожого рівня узагальненості з меншими витратами. Few-shot learning залишається корисним інструментом для експрес-прототипування нових підзадач, але поступається fine-tuning у стабільності й багатомітковості, а continual learning доцільний переважно для динамічних сценаріїв, де контент постійно змінюється, що менш актуально для статичних датасетів.

2.4 Специфікації вимог до програмного забезпечення

Проєкт розробки програмного забезпечення має на меті створення інноваційного рішення для аналізу текстовиз повідомлень з метою виявлення маніпулятивних технік.

Призначення й межі проєкту:

1) призначення системи (застосунку), для якої розробляється програмне забезпечення: призначенням застосунку є аналіз новинних текстів і виявлення у них маніпулятивних прийомів;

2) погодження, що ухвалені в програмній документації: погоджено, що для створення ПЗ та його стабільної роботи будуть використовуватися фреймворки і бібліотеки – Python, aiogram, scikit-learn, Hugging Face Transformers;

3) межі проєкту ПЗ: крайня дата завершення роботи над ПЗ – 15.10.2025 р.

Загальний опис:

1) сфера застосування: може застосовуватися у сфері журналістики, медіаграмотності, освітніх курсів, досліджень у сфері інформаційної безпеки;

2) характеристики користувачів: основні характеристики користувачів: наявність персонального комп’ютеру (ПК), доступу до мережі Інтернет, акаунту в Telegram;

3) загальна структура й склад системи: основні частини для створення програмного забезпечення – модулі для обробки тексту за допомогою NLP, клієнтська фронтенд частина (Telegram user interface (UI));

4) загальні обмеження: обмеження для роботи з ПЗ – наявність ПК, підключення до мережі Інтернет, наявність акаунту в Telegram.

Функції системи (пошук лотів і відстеження ставок, які робили інші користувачі):

1) опис функції: функція дозволяє авторизованому користувачу відправити текст у бот, після чого система здійснює його аналіз і визначає наявність маніпулятивних прийомів;

2) вхідна і вихідна інформація: вхідна інформація – текстове повідомлення, яке надсилає користувач; вихідна інформація – структурований результат аналізу із зазначенням маніпулятивних технік у тексті.

3) функціональні вимоги:

а) обробка текстових повідомлень – функція дозволяє користувачам надсилати у бот довільні текстові повідомлення для аналізу:

– вхідні дані: текстове повідомлення користувача;

– вихідні дані: результат аналізу, що включає класифікацію маніпулятивних технік і підсвічування проблемних фрагментів;

б) повідомлення про помилки – функція інформує користувача у випадку введення тексту непідтримуваною мовою або при технічних збоях:

– вхідні дані: повідомлення у непідтримуваній мові або некоректний формат;

– вихідні дані: повідомлення про помилку з рекомендаціями;

в) довідкова інформація – функція дає можливість користувачу отримати опис різних видів маніпуляцій:

– вхідні дані: команда користувача;

– вихідні дані: список і пояснення використаних маніпулятивних технік.

Вимоги до інформаційного забезпечення:

1) джерелом вхідної інформації є текстові повідомлення, які надсилають або пересилають користувачі у Telegram-бот;

2) спеціальних вимог до нормативно-довідкової інформації немає;

3) інформація не зберігається у базі даних. Обробка відбувається в режимі реального часу за допомогою NLP-моделей, після чого результати аналізу повертаються користувачу.

Для розробки й роботи програмного забезпечення не передбачено значних технічних обмежень. Достатньо наявності серверного середовища з доступом до мережі Інтернет та підтримкою Python (для NLP-моделей) і Telegram Bot API.

Вимоги до програмного забезпечення:

1) архітектура програмної системи: система складається з Telegram-бота, двох NLP-моделей (класифікація типів маніпуляцій і виявлення спанів);

2) системне програмне забезпечення: для розробки використовується Python (основна мова реалізації NLP-моделей) та Telegram Bot API;

3) мережеве програмне забезпечення: для створення ПЗ використовується ОС Windows, редактор коду PyCharm і месенджер Telegram;

4) програмне забезпечення ведення інформаційної бази: не використовується, оскільки система не передбачає збереження історії запитів у БД;

5) мова і технологія розробки ПЗ: основна мова розробки – Python. Для NLP-моделей використовуються бібліотеки PyTorch або TensorFlow, для інтеграції – Telegram Bot API.

Вимоги до зовнішніх інтерфейсів:

1) інтерфейс користувача: взаємодія з ботом відбувається безпосередньо у середовищі Telegram. Інтерфейс реалізований у вигляді діалогу «користувач – бот»: користувач надсилає або пересилає повідомлення, бот виконує аналіз і повертає результати з підсвіченими маніпулятивними фрагментами й поясненнями. Додаткові команди (наприклад, довідка, інформація про типи маніпуляцій) доступні у вигляді текстових запитів чи кнопок Telegram;

2) апаратний інтерфейс: будь-який пристрій, що підтримує Telegram (смартфон, планшет, ПК), з доступом до мережі Інтернет;

3) програмний інтерфейс: основа роботи – Telegram Bot API для обміну повідомленнями. Для обробки тексту використовуються NLP-моделі на Python (PyTorch, TensorFlow);

4) комунікаційний протокол: застосунок базується на використанні мережних протоколів Wireless Application Protocol (WAP) – протокол бездротової передачі даних і TCP/IP.

Властивості програмного забезпечення:

1) доступність: бот доступний для всіх користувачів Telegram, які мають

доступ до Інтернету;

2) супроводжуваність: передбачена можливість оновлення NLP-моделей і розширення словників маніпуляційних технік;

3) переносимість: бот не залежить від платформи користувача, оскільки Telegram є кросплатформеним застосунком;

4) продуктивність: час обробки запиту залежить від швидкодії NLP-моделей і якості інтернет-з'єднання користувача;

5) надійність: система стабільно обробляє повідомлення у реальному часі. У випадку використання непідтримуваної мови бот повідомляє про помилку;

6) безпека: передача даних відбувається через зашифровані канали Telegram.

Впровадження цього програмного забезпечення дозволить забезпечити доступність, зручність і безпеку для користувачів. Проєкт надає можливість швидко отримати інформацію про маніпулятивність текстів без використання сторонніх ресурсів, інтегруючись безпосередньо у популярний месенджер Telegram.

Висновки до розділу 2

Другий розділ присвячено фундаментальним аспектам теоретичної і методологічної бази для розпізнавання маніпулятивних технік у багатомовних текстах з платформи Telegram, охоплюючи аналіз даних, огляд сучасних підходів до навчання моделей і стратегії їхньої оптимізації. Особливу увагу приділено характеристиці набору даних, включаючи структуру анотацій, специфіку обробки текстів українською і російською мовами, а також ключові ознаки, що відображають контекстуальні, семантичні особливості маніпулятивних елементів.

Розглянуто широкий спектр методологій навчання починаючи від традиційних моделей машинного навчання з їхнім фокусом на базових алгоритмах закінчуючи глибинними архітектурами, які інтегрують рекурентні й згорткові шари для послідовної обробки і до передових трансформерних моделей, здатних

знаходити глобальні залежності, нюанси в шумних неформальних текстах. Під час порівняння моделей показано, що класичні ML моделы можуть бути використані як базові орієнтири або інструменти попереднього аналізу, однак їхні можливості суттєво обмежені через нездатність враховувати складні контекстуальні та семантичні залежності. У свою чергу DL частково долають ці обмеження завдяки інтеграції локальних і послідовних ознак, проте все ще поступаються у здатності моделювати глобальний контекст у багатомовних текстах. На основі цього аналізу обґрунтовано вибір оптимальних трансформенних архітектур з акцентом на їхню придатність до завдань багатоміткової класифікації і точного виділення span-ів, враховуючи обмеженість ресурсів і специфіку домену. Детально висвітлено стратегії адаптації моделей, такі як претренування для нарощування базової семантики, fine-tuning для швидкої спеціалізації на анотованих даних, домен-адаптація для подолання розбіжностей між загальними й цільовими текстами, few-shot learning для низькоресурсних сценаріїв та постійне навчання для забезпечення стійкості до еволюціонуючого контенту.

Отримані у другому розділі теоретичні і практичні висновки формують методологічне підґрунтя для побудови системи розпізнавання маніпулятивних технік. Також обґрунтовано вибір моделі й стратегії навчання, які будуть реалізовані й оцінені у наступних розділах.

3 НАВЧАННЯ МОДЕЛЕЙ

У цьому розділі досліджується процес навчання моделей для задач span-детекції і класифікації маніпулятивних технік у текстових даних, що становить основу розробки застосунку для пошуку дезінформації. Зокрема, розглядається валідація даних, включаючи підготовчі етапи. До них належать очищення, нормалізація і анотація, які готують вхідні дані до ефективного навчання. Також описано процес безпосереднього навчання із використанням трансформерних моделей, зокрема XLM-RoBERTa-large, з акцентом на налаштування параметрів і адаптацію до особливостей багатомовних текстів для забезпечення високої точності розпізнавання маніпулятивних патернів.

3.1 Підготовка датасету для навчання моделі span-detection

Процес підготовки даних для моделі span-детекції, побудований на основі трансформерної архітектури XLM-RoBERTa-large, є фундаментальним етапом, який безпосередньо впливає на здатність моделі ефективно виявляти маніпулятивні сегменти в текстових даних. Цей процес охоплює комплексну послідовність дій, включаючи завантаження, очищення, нормалізацію, анотацію, вирівнювання і організацію даних у зручний для навчання формат. Кожен етап ретельно продуманий із урахуванням специфіки задачі, моделі й загальноприйнятих практик обробки даних у машинному навчанні.

Перший етап передбачає завантаження даних із файлу у форматі parquet (train.parquet) за допомогою функції load_data, яка використовує бібліотеку pandas для ефективного читання структурованих даних. Датасет із представленим розширенням обирається завдяки його компактності й підтримці стиснення, що особливо корисно при роботі з великими наборами даних, такими як корпус текстів із анотаціями. Початкове очищення фокусується на стовпці trigger_words, який містить анотації спанів у вигляді пар (start, end). Використання методу fillna() замінює відсутні значення порожніми рядками, а функція process_numpy_trigger_words конвертує numpy-масиви в списки кортежів,

відкидаючи некоректні дані з неправильними типами чи розмірами. Такий підхід є стандартним для попередньої обробки анотованих даних, оскільки він стандартизує формат спанів і усуває шумові артефакти, такі як порожні масиви чи невідповідності. Додаткова фільтрація валідних індексів (valid_indices) видаляє тексти, що є порожніми або не є рядками. Вищеописаний підхід відповідає загальноприйнятій практиці видалення «шумних» зразків перед подальшою обробкою (рис. 3.1).

```python
def load_data(file_path):
    """Load data from parquet file"""
    df = pd.read_parquet(file_path)
    return df

def process_numpy_trigger_words(trigger_array):
    """Convert numpy array of trigger words to list of tuples"""
    if not isinstance(trigger_array, np.ndarray) or trigger_array.ndim == 0 or trigger_array.size == 0:
        return []
    result = []
    if trigger_array.ndim == 2 and trigger_array.shape[1] == 2:
        for arr in trigger_array:
            if np.issubdtype(arr.dtype, np.number) and len(arr) == 2:
                try:
                    start, end = int(arr[0]), int(arr[1])
                    if start < end:
                        result.append((start, end))
                except (ValueError, TypeError):
                    continue
    elif trigger_array.ndim == 1 and trigger_array.dtype == 'object':
        for item in trigger_array:
            if isinstance(item, (list, tuple, np.ndarray)) and len(item) == 2:
                try:
                    start, end = int(item[0]), int(item[1])
                    if start < end:
                        result.append((start, end))
                except (ValueError, TypeError):
                    continue
    return result

def main():
    print("Loading training data...")
    train_df = pd.read_parquet("/kaggle/input/manip-dataset/data/span_detection/train.parquet")
    print("Processing trigger words...")
    train_df['trigger_words'] = train_df['trigger_words'].fillna('').apply(lambda x: x if isinstance(x, np.ndarray) else np.array([]))
    train_df['trigger_words_processed'] = train_df['trigger_words'].apply(process_numpy_trigger_words)
    texts = train_df['content'].tolist()
    spans = train_df['trigger_words_processed'].tolist()
    print(f"Total raw examples: {len(texts)}")
    valid_indices = [i for i, txt in enumerate(texts) if isinstance(txt, str) and len(txt.strip()) > 0]
    texts = [texts[i] for i in valid_indices]
    spans = [spans[i] for i in valid_indices]
    print(f"Using {len(texts)} non-empty text examples for training/validation.")
```

Рисунок 3.1 – Завантаження, очищення і фільтрація набору даних

Наступним кроком є розбиття даних на тренувальну і валідаційну вибірки у співвідношенні 85/15 за допомогою функції train_test_split із фіксованим параметром random_state=SEED=42. Використання фіксованого сиду є стандартною практикою в машинному навчанні, яка забезпечує відтворюваність результатів і дозволяє порівнювати експерименти в різних умовах. Співвідношення 85/15 вибрано як компроміс між розміром тренувального набору, необхідного для стабільного навчання складних моделей, таких як XLM-RoBERTa-large, і

формування валідаційного набору, достатнього для надійної оцінки продуктивності. Цей підхід відповідає загальноприйнятим стратегіям розподілу даних, особливо в задачах із обмеженим обсягом анотованих даних, де важливо зберегти репрезентативність обох підмножин. Відсутність стратифікації обґрунтована тим, що задача span-детекції не передбачає балансування класів на рівні зразків, а фокусується на розподілі спанів, що обробляється на етапі втрат (див. Додаток Б).

Токенізація текстів здійснюється за допомогою AutoTokenizer із моделі XLM-RoBERTa-large через функцію align_tokens_and_spans, що є ключовим етапом для адаптації символьних даних до формату, придатного для трансформерів. Параметр return_offsets_mapping=True генерує відображення між символами оригінального тексту й токенами, що є необхідним для точного вирівнювання спанів. Обрізання текстів до максимальної довжини (max_length=MAX_LEN=512) застосовується для уникнення перевантаження пам’яті моделі й забезпечення однакової довжини вхідних послідовностей, що є стандартною практикою для трансформерних архітектур. Параметр add_special_tokens=True додає спеціальні токени ([CLS] і [SEP]), які є частиною специфікації XLM-RoBERTa і важливі для коректної обробки контексту моделлю. Відкладення паддінгу (padding=False) на етап батчування дозволяє оптимізувати використання пам’яті на цьому етапі (див. Додаток Б).

Вирівнювання спанів із токенами базується на BIO-схемі, яка є широко прийнятою для задач сегментації, таких як Named Entity Recognition (NER), адаптованою для маніпулятивних спанів. Мітки присвоюються наступним чином:

– 0 для токенів поза спанами (Outside);

– 1 для початку спана (Beginning);

– 2 для внутрішньої частини спана (Inside).

Спани сортуються за початковими позиціями (sorted_spans) для послідовної обробки, а перекриття між токеном і спаном визначається через умову методу max(start, span_start) < min(end, span_end). Якщо токен перетинається зі спаном,

його мітка визначається як 1 або 2 залежно від його позиції відносно попередніх токенів і спанів, з урахуванням логіки початку й продовження спана. Мітка 100 застосовується до padding або спеціальних токенів, що є звичайною практикою в PyTorch для ігнорування цих позицій під час обчислення втрат. Параметр SPAN_MERGE_DISTANCE=1 дозволяє об'єднувати близькі спани, якщо відстань між їхніми кінцем і початком менша за 1 символ, що коригує можливі помилки анотації, покращуючи коректність вирівнювання і відображаючи гнучкий підхід до обробки неоднозначних меж (див. Додаток Б).

Створення датасету реалізується через клас ManipulationSpanDataset, який ініціалізується з токенізованими текстами й мітками. Кожен зразок включає input_ids, attention_mask, labels і offset_mapping, що зберігаються в списку encodings. Використання бібліотеки tqdm для відстеження прогресу токенізації дозволяє моніторити тривалість обробки великих наборів даних, що є зручним інструментом для відладки й оцінки продуктивності. Метод __getitem__ конвертує дані в тензори типу torch.long, що є стандартним для трансформерних моделей, забезпечуючи сумісність із PyTorch. Функція collate_fn у DataLoader динамічно паддить батчі до максимальної довжини в межах поточного батчу, використовуючи tokenizer_pad_token_id для input_ids і нульові маски для attention_mask. Мітки padding позицій встановлюються як 100, що відповідає ігноруванню цих токенів у втраті WeightedFocalLoss, що є стандартною практикою для уникнення спотворення градієнтів. Параметр batch_size=2 обрано як компроміс між обсягом пам'яті GPU і швидкістю навчання, тоді як num_workers=2 активує паралельне завантаження даних, що значно прискорює підготовку і вважається загальноприйнятим підходом для оптимізації продуктивності в задачах із великими наборами даних (рис 3.2).

```python
def collate_fn(batch):
    max_len = max([len(item["input_ids"]) for item in batch])
    tokenizer_pad_token_id = AutoTokenizer.from_pretrained(MODEL_NAME).pad_token_id
    input_ids_padded = []
    attention_mask_padded = []
    labels_padded = []
    offset_mappings = []
    texts = []
    for item in batch:
        padding_len = max_len - len(item["input_ids"])
        input_ids = torch.cat([item["input_ids"], torch.tensor([tokenizer_pad_token_id] * padding_len, dtype=torch.long)])
        attention_mask = torch.cat([item["attention_mask"], torch.zeros(padding_len, dtype=torch.long)])
        labels = torch.cat([item["labels"], torch.tensor([-100] * padding_len, dtype=torch.long)])
        input_ids_padded.append(input_ids)
        attention_mask_padded.append(attention_mask)
        labels_padded.append(labels)
        offset_mappings.append(item["offset_mapping"] + [(0, 0)] * padding_len)
        texts.append(item["text"])
    return {
        "input_ids": torch.stack(input_ids_padded),
        "attention_mask": torch.stack(attention_mask_padded),
        "labels": torch.stack(labels_padded),
        "offset_mappings": offset_mappings,
        "texts": texts,
    }

def main():
    print("Loading tokenizer...")
    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
    print("Creating datasets (this may take a while)...")
    train_dataset = ManipulationSpanDataset(train_texts, train_spans, tokenizer, max_len=MAX_LEN)
    val_dataset = ManipulationSpanDataset(val_texts, val_spans, tokenizer, max_len=MAX_LEN)
    print("Creating dataloaders...")
    train_dataloader = DataLoader(
        train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        collate_fn=collate_fn,
        num_workers=2
    )
    val_dataloader = DataLoader(
        val_dataset,
        batch_size=BATCH_SIZE * 2,
        shuffle=False,
        collate_fn=collate_fn,
        num_workers=2
    )
```

Рисунок 3.2 – Створення датасету й батчування

Оскільки дані можуть мати значний дисбаланс між класами (наприклад, більшість токенів позначені як Outside, тоді як Beginning і Inside є менш поширеними), застосовується кастомна функція втрат WeightedFocalLoss. Параметр alpha=[0.1, 0.45, 0.45] визначає ваги для класів (0 для Outside, 0.45 для Beginning, 0.45 для Inside), що дозволяє компенсувати дисбаланс, надаючи більшу вагу менш представленим класам. Значення gamma=2.0 фокусує модель на важких для класифікації прикладах, що є популярним підходом у задачах із шумом або складними межами. Використання цього принципу зменшує внесок добре класифікованих зразків і підкреслює помилки. Параметр ignore_index= -100 ігнорує паддінгові позиції під час обчислення втрат, що є стандартною практикою в PyTorch для уникнення спотворення градієнтів від нерелевантних токенів (рис 3.3).

```python
class WeightedFocalLoss(nn.Module):
    def __init__(self, alpha=[0.1, 0.45, 0.45], gamma=2.0, ignore_index=-100):
        super(WeightedFocalLoss, self).__init__()
        self.alpha = torch.tensor(alpha).float()
        self.gamma = gamma
        self.ignore_index = ignore_index
        self.log_softmax = nn.LogSoftmax(dim=-1)
    def forward(self, inputs, targets):
        mask = targets != self.ignore_index
        valid_inputs = inputs[mask]
        valid_targets = targets[mask]
        if valid_targets.numel() == 0:
            return torch.tensor(0.0, device=inputs.device, requires_grad=True)
        if self.alpha.device != inputs.device:
            self.alpha = self.alpha.to(inputs.device)
        log_probs = self.log_softmax(valid_inputs)
        gathered_log_probs = log_probs.gather(1, valid_targets.unsqueeze(1)).squeeze(1)
        probs = torch.exp(gathered_log_probs)
        alpha_t = self.alpha[valid_targets]
        focal_loss = alpha_t * torch.pow(1 - probs, self.gamma) * (-gathered_log_probs)
        return focal_loss.mean()
```

Рисунок 3.3 – Обробка дисбалансу класів та підготовка до втрат

Представлена підготовка даних гарантує, що модель ефективно навчатиметься розпізнавати як рідкісні початки й середини спанів, так і часті позначення поза спанами, адаптуючись до специфіки маніпулятивних текстів.

3.2 Підготовка датасету для навчання моделі class-detection

Перший етап розпочинається з завантаження даних і початкового очищення, що включає функцію clean_text, яка замінює URL-адреси на плейсхолдер [URL], усуває надлишкові пробіли й нормалізує текст, що є стандартною практикою для видалення шумових елементів, які можуть завадити моделі. Якщо стовпець techniques присутній, він парситься з рядків у списки за допомогою ast.literal_eval, а потім розкладається на бінарні мітки для кожної з 10 маніпулятивних технік, де 1 вказує на наявність техніки, а 0 – на її відсутність. Додатковий стовпець manipulative позначає тексти з принаймні однією технікою, що полегшує стратифікацію. Виявлення мови через detect_language базується на наявності специфічних українських літер, що дозволяє адаптувати модель до багатомовного контексту, хоча цей крок є спрощеним і може бути розширений у майбутньому для точнішого розпізнавання (рис. 3.4).

```python
def analyze_dataset(df):
    logger.info("Analyzing dataset...")
    print("Analyzing dataset...")
    technique_counts = {technique: df[technique].sum() for technique in TECHNIQUES}
    total_instances = len(df)
    logger.info(f"Total instances: {total_instances}")
    print(f"Total instances: {total_instances}")
    logger.info("Class distribution:")
    print("Class distribution:")
    for technique, count in technique_counts.items():
        percentage = (count / total_instances) * 100
        logger.info(f"{technique}: {count} instances ({percentage:.2f}%)")
        print(f"{technique}: {count} instances ({percentage:.2f}%)")
    cooccurrence = pd.DataFrame(0, index=TECHNIQUES, columns=TECHNIQUES)
    for _, row in df.iterrows():
        for i, tech1 in enumerate(TECHNIQUES):
            if row[tech1] == 1:
                for tech2 in TECHNIQUES:
                    if row[tech2] == 1:
                        cooccurrence.loc[tech1, tech2] += 1
    logger.info("Co-occurrence of techniques:")
    print("Co-occurrence of techniques:")
    for i, tech in enumerate(TECHNIQUES):
        co_techs = [(other_tech, cooccurrence.loc[tech, other_tech])
                    for other_tech in TECHNIQUES if other_tech != tech]
        co_techs = sorted(co_techs, key=lambda x: x[1], reverse=True)[:3]
        logger.info(f"{tech} frequently co-occurs with: {co_techs}")
        print(f"{tech} frequently co-occurs with: {co_techs}")
    df['text_length'] = df['content'].apply(lambda x: len(str(x).split()))
    avg_length = df['text_length'].mean()
    median_length = df['text_length'].median()
    max_length = df['text_length'].max()
    logger.info(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length}, Max={max_length}")
    print(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length}, Max={max_length}")
    over_limit = (df['text_length'] > CONFIG['max_length']).sum()
    logger.info(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit} ({over_limit/len(df)*100:.2f}%)")
    print(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit} ({over_limit/len(df)*100:.2f}%)")
    class_weights = {}
    for technique in TECHNIQUES:
        pos_samples = df[technique].sum()
        if pos_samples > 0:
            weight = total_instances / (2 * pos_samples)
            class_weights[technique] = min(weight, 10.0)
        else:
            class_weights[technique] = 1.0
    return class_weights
```

Рисунок 3.4 – Очищення і виявлення мови текстів

Наступний етап включає детальний аналіз даних через функцію analyze_dataset, який є важливим для розуміння розподілу класів і потенційного дисбалансу. Підрахунок кількості зразків для кожної техніки (technique_counts) й обчислення їхньої частки від загальної кількості зразків (total_instances) дозволяє оцінити частотність кожної маніпулятивної техніки. Аналіз співпадінь (cooccurrence) виявляє, які техніки часто з’являються разом, що може вказувати на кореляції та впливати на стратегію навчання. Обчислення статистики довжини текстів (середнє, медіана, максимум) і перевірка перевищення максимальної довжини (CONFIG['max_length']=512) допомагають оцінити придатність обраного параметра обрізання. Вагу класів обчислюють як відношення загальної кількості зразків до подвійної кількості позитивних зразків кожної техніки, із обмеженням максимального значення 10.0, що є стандартним методом для компенсації дисбалансу в мультилейблових задачах, забезпечуючи більшу вагу рідкісним класам (рис. 3.5).

```python
def analyze_dataset(df):
    logger.info("Analyzing dataset...")
    print("Analyzing dataset...")
    technique_counts = {technique: df[technique].sum() for technique in TECHNIQUES}
    total_instances = len(df)
    logger.info(f"Total instances: {total_instances}")
    print(f"Total instances: {total_instances}")
    logger.info("Class distribution:")
    print("Class distribution:")
    for technique, count in technique_counts.items():
        percentage = (count / total_instances) * 100
        logger.info(f"{technique}: {count} instances ({percentage:.2f}%)")
        print(f"{technique}: {count} instances ({percentage:.2f}%)")
    cooccurrence = pd.DataFrame(0, index=TECHNIQUES, columns=TECHNIQUES)
    for _, row in df.iterrows():
        for i, tech1 in enumerate(TECHNIQUES):
            if row[tech1] == 1:
                for tech2 in TECHNIQUES:
                    if row[tech2] == 1:
                        cooccurrence.loc[tech1, tech2] += 1
    logger.info("Co-occurrence of techniques:")
    print("Co-occurrence of techniques:")
    for i, tech in enumerate(TECHNIQUES):
        co_techs = [(other_tech, cooccurrence.loc[tech, other_tech])
                    for other_tech in TECHNIQUES if other_tech != tech]
        co_techs = sorted(co_techs, key=lambda x: x[1], reverse=True)[:3]
        logger.info(f"{tech} frequently co-occurs with: {co_techs}")
        print(f"{tech} frequently co-occurs with: {co_techs}")
    df['text_length'] = df['content'].apply(lambda x: len(str(x).split()))
    avg_length = df['text_length'].mean()
    median_length = df['text_length'].median()
    max_length = df['text_length'].max()
    logger.info(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length}, Max={max_length}")
    print(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length}, Max={max_length}")
    over_limit = (df['text_length'] > CONFIG['max_length']).sum()
    logger.info(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit} ({over_limit/len(df)*100:.2f}%)")
    print(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit} ({over_limit/len(df)*100:.2f}%)")
    class_weights = {}
    for technique in TECHNIQUES:
        pos_samples = df[technique].sum()
        if pos_samples > 0:
            weight = total_instances / (2 * pos_samples)
            class_weights[technique] = min(weight, 10.0)
        else:
            class_weights[technique] = 1.0
    return class_weights
```

Рисунок 3.5 – Аналіз набору даних і обчислення ваг

Розбиття даних на тренувальну і валідаційну вибірки виконується через функцію prepare_dataloaders із використанням train_test_split у співвідношенні 80/20. Використання стратифікації за стовпцем manipulative (наявність принаймні однієї техніки) забезпечує збалансований розподіл позитивних і негативних зразків, що є критичним для мультилейблової класифікації з можливим дисбалансом. Фіксований параметр random_state=SEED=42 гарантує відтворюваність, що є стандартною практикою для порівняння результатів. Цей підхід дозволяє зберегти репрезентативність обох наборів, що особливо важливо для задач із неоднорідним розподілом класів (див. Додаток Б).

Токенізація виконується через AutoTokenizer із моделі XLM-RoBERTa-large у класі ManipulationDataset. Параметри max_length=512, padding='max_length' і truncation=True забезпечують обрізання текстів до фіксованої довжини й додавання паддінгу, що є стандартним для трансформерів для уніфікації вхідних даних. Аугментація даних через data_augmentation із ймовірністю 0.5 і видаленням 20% слів (якщо текст довший за 5 слів) активується з імовірністю augment_ratio=0.2

лише для тренувального набору. Цей метод є простою стратегією для розширення даних, підвищуючи стійкість моделі до варіацій, хоча він не змінює мітки, що відповідає мультилейбловій природі задачі (рис. 3.6).

```python
def data_augmentation(text, techniques):
    if random.random() < 0.5:
        words = text.split()
        if len(words) > 5:
            indices_to_delete = random.sample(range(len(words)), int(len(words) * 0.2))
            words = [word for i, word in enumerate(words) if i not in indices_to_delete]
            text = ' '.join(words)
    return text, techniques

class ManipulationDataset(Dataset):
    def __init__(self, texts, targets=None, tokenizer=None, max_length=512, augment=False):
        self.texts = texts
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.augment = augment
    def __len__(self):
        return len(self.texts)
    def __getitem__(self, idx):
        text = str(self.texts[idx])
        text = clean_text(text)
        if self.augment and self.targets is not None and random.random() < CONFIG["augment_ratio"]:
            text, _ = data_augmentation(text, self.targets[idx])
        encoding = self.tokenizer(
            text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        item = {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten()
        }
        if self.targets is not None:
            item['targets'] = torch.tensor(self.targets[idx], dtype=torch.float)
        return item
```

Рисунок 3.6 – Токенізація й аугментація текстів

Виконання представлених етапів обробки датасету забезпечує якісну підготовку даних, адаптовану до мультилейблової класифікації технік маніпуляції в багатомовному контексті.

3.3 Навчання моделі класифікації технік маніпуляцій

Процес навчання моделі класифікації, заснованої на трансформерній архітектурі XLM-RoBERTa-large, є комплексним етапом, спрямованим на оптимізацію параметрів моделі для ефективного розпізнавання десяти маніпулятивних технік у текстах. Цей процес включає ініціалізацію моделі, налаштування оптимізатора й планувальника, тренування з накопиченням градієнтів, оцінку на валідаційному наборі, застосування ранньої зупинки й збереження найкращої моделі. Навчання адаптовано до мультилейблової природи задачі, де кожен текст може містити кілька технік одночасно, і враховує специфіку багатомовних даних (українська й російська).

Процес навчання розпочинається з ініціалізації моделі ManipulationClassifier, яка базується на XLM-RoBERTa-large з додаванням власного класифікатора для 10 технік. Модель переноситься на пристрій, що є стандартною практикою для прискорення обчислень. Оптимізатор AdamW налаштовується з групуванням параметрів: ваги з weight_decay=0.01 застосовуються до всіх параметрів, окрім bias і LayerNorm.weight, де weight_decay=0.0, що запобігає надмірному регуляризаційному ефекту на нормалізовані шари. Початкове значення швидкості навчання learning_rate=1.8e-5 є типовим для трансформерів, а планувальник косинусний або лінійний із warmup_ratio=0.1 забезпечує поступове збільшення швидкості на початку, що стабілізує навчання. Загальна кількість кроків обчислюється з урахуванням накопичення градієнтів (gradient_accumulation_steps=4), що дозволяє ефективно працювати з більшими батчами на обмежених ресурсах GPU (див. Додаток Б).

Тренування на одній епосі виконується через функцію train_epoch, де модель переводиться в режим тренування (model.train()). Вибір функції втрат залежить від конфігурації: за замовчуванням використовується BCEWithLogitsLoss із вагами класів (pos_weight_tensor), обчисленими на основі дисбалансу, якщо use_weighted_loss=True. Альтернативно, FocalLoss із gamma=2.0 застосовується для акценту на важких прикладах, якщо use_focal_loss=True. Лейбл-смутінг (label_smoothing=0.05) пом’якшує цільові значення, що є стандартною практикою для підвищення узагальнення. Накопичення градієнтів (gradient_accumulation_steps=4) розподіляє обчислення втрат на кілька кроків, дозволяючи ефективно працювати з меншими батчами (batch_size=8). Обрізання градієнтів (gradient_clipping=1.0) запобігає вибуху градієнтів, а оновлення оптимізатора з планувальником виконується після кожного накопиченого кроку. Прогрес відображається через tqdm, що є зручним інстурментом для моніторингу (див. Додаток Б).

Оцінка моделі виконується через функцію evaluate у режимі model.eval() із відключенням обчислення градієнтів. Ймовірності прогнозів обчислюються через

torch.sigmoid, а пороги для класифікації можуть бути фіксованими (0.5) або оптимальними, визначеними функцією find_optimal_threshold. Оптимальні пороги шукаються шляхом перебору значень від 0.15 до 0.85 із кроком 0.01, максимизуючи F1-score для кожної техніки, що є адаптивним підходом до мультилейблових задач із дисбалансом. Метрика macro F1 і F1 для кожної техніки обчислюються для оцінки загальної продуктивності, а результати зберігаються для подальшого аналізу (див. Додаток Б).

3.4 Навчання моделі span-detection

Процес навчання моделі span-детекції, заснованої на трансформерній архітектурі XLM-RoBERTa-large, спрямований на ідентифікацію маніпулятивних сегментів у текстах шляхом класифікації токенів за схемою BIO. Цей процес включає підготовку даних, ініціалізацію моделі й оптимізатора, тренування з накопиченням градієнтів, оцінку продуктивності на валідаційному наборі, ранню зупинку та прогнозування на тестових даних. Навчання адаптовано до специфіки багатомовних текстів (української і російської) і враховує дисбаланс класів та обмеження обчислювальних ресурсів.

Процес навчання розпочинається з підготовки даних, включаючи завантаження із файлу train.parquet, обробку спанів через process_numpy_trigger_words і розбиття на тренувальну (85%) та валідаційну (15%) вибірки з фіксованим random_state=SEED=42 для відтворюваності. Модель AutoModelForTokenClassification із XLM-RoBERTa-large ініціалізується з трьома класами (O, B, I) і переноситься на GPU, якщо доступно. Оптимізатор AdamW використовує Layer-wise Learning Rate Decay (LLRD) із коефіцієнтом LLRD_RATE=0.9, що знижує швидкість навчання для нижніх шарів моделі, адаптуючи їх повільніше, що є стандартною практикою для трансформерів. Планувальник із лінійним розігрівом (warmup_ratio=0.1) стабілізує початкові кроки, а weight_decay=0.01 регуляризує навчання (див. Додаток Б).

Тренування на одній епохі реалізується в функції train_model, де модель

переводиться в режим тренування. Використовується кастомна функція втрат WeightedFocalLoss, яка адаптована до задач із дисбалансом класів. Ігноровані індекси (ignore_index=-100) відповідають паддінговим токенам, що виключаються з обчислень втрат.

Втрата нормалізується на кількість кроків накопичення (accumulation_steps=4), що дозволяє ефективно працювати з малим розміром батча (batch_size=2), розподіляючи обчислення градієнтів на кілька ітерацій. Процес назаднього поширення (loss.backward()) виконується для кожного батча, а оновлення ваг оптимізатора відбувається лише після накопичення градієнтів на accumulation_steps кроків або наприкінці епохи. Обрізання градієнтів до значення 1.0 (torch.nn.utils.clip_grad_norm_) запобігає їх «вибуху», що є стандартною практикою для стабілізації навчання глибоких мереж. Після кожного оновлення оптимізатор скидається (optimizer.zero_grad()), а планувальник оновлює швидкість навчання (scheduler.step()) (див. Додаток Б).

Прогнози токенів обчислюються як індекси максимальних логітів (torch.argmax), а спани конвертуються у формат символів через функцію tokens_to_char_spans, яка враховує відображення токенів у символи (offset_mappings) і об’єднує близькі спани з відстанню до span_merge_distance=1. Справжні спани отримуються з початкового набору даних, якщо можливо, і порівнюються з прогнозами для обчислення F1-score за допомогою compute_span_f1, що базується на критерії перекриття. Прогрес відображається через tqdm із поточними значеннями втрати та швидкості навчання.

Оцінка моделі виконується в режимі model.eval() із відключенням обчислення градієнтів (torch.no_grad()), що економить обчислювальні ресурси. Процес аналогічний тренуванню: обчислюються логіти, втрата за допомогою WeightedFocalLoss, і прогнозуються спани через tokens_to_char_spans. Справжні спани з валідаційного набору порівнюються з прогнозами, і обчислюються метрики precision, recall та F1 через compute_span_f1, який базується на критерії перекриття спанів. Середні значення цих метрик розраховуються для всієї валідаційної

вибірки, що дозволяє об’єктивно оцінити продуктивність.

Механізм ранньої зупинки відстежує покращення середнього F1 на валідації. Якщо F1 зростає, зберігається копія стану моделі (best_model_state), а лічильник терпіння (patience=3) скидається. Якщо покращення відсутнє протягом трьох епох, тренування припиняється, що запобігає перенавчанню та економить ресурси. Наприкінці повертається модель із найкращим станом або, у крайньому випадку, із останнього кроку, якщо покращення не було.

Прогнозування спанів виконується через функцію predict_spans у режимі model.eval() із відключенням градієнтів. Створюється окремий InferenceDataset для тестування, де тексти токенізовані з поверненням offset_mapping для відображення токенів у символи. Даталоader із більшим batch_size=8 (у 4 рази більше, ніж для тренування) прискорює обчислення. Логіти прогнозуються, а спани конвертуються у символи через tokens_to_char_spans із об’єднанням близьких спанів за span_merge_distance=1. Результати зберігаються у список, який потім мапується до ідентифікаторів тестових зразків (див. Додаток Б).

Висновки до розділу 3

У результаті детального аналізу реалізації процесів підготовки даних і навчання моделей для класифікації технік маніпуляції і детекції спанів у текстах встановлено, що обидва етапи виконано з урахуванням складної мультилейблової природи задачі. Окремо підкреслено адаптацію методів до унікальних особливостей багатомовного контексту, зокрема української і російської мов. Під час підготовки даних забезпечено ретельне очищення текстових зразків від шумів, а також їхню нормалізацію, що сприяє підвищенню якості вхідних даних для подальшого аналізу. Анотація текстів із розкладанням міток на бінарні значення для кожної техніки чи спану, разом зі стратифікованим розбиттям на тренувальну і валідаційну вибірки, дозволяє зберегти репрезентативність розподілу класів, особливо враховуючи можливий дисбаланс між позитивними й негативними прикладами. Аналіз розподілу класів, обчислення статистики довжини текстів і

визначення ваг класів створюють міцну основу для адаптації моделей до рідкісних маніпулятивних технік і спанів, що суттєво підвищує їхню здатність до точного виявлення таких випадків. Токенізація з уніфікацією довжини, опціональна аугментація даних додають стійкості моделям до варіацій у текстах, забезпечуючи їхню гнучкість у різних сценаріях.

У процесі навчання моделей застосовуються сучасні методи машинного навчання, такі як накопичення градієнтів, що дозволяє ефективно працювати з обмеженими обчислювальними ресурсами, розподіляючи обчислення градієнтів на кілька кроків і оптимізуючи використання GPU. Планування швидкості навчання за допомогою лінійного чи косинусного розкладу з етапом розігріву сприяє стабілізації процесу на початкових етапах, тоді як механізм ранньої зупинки, заснований на моніторингу метрики F1, запобігає перенавчанню і забезпечують своєчасне завершення тренування при досягненні оптимальної продуктивності. Використання кастомних функцій втрат із налаштованими вагами для класів дозволяє ефективно компенсувати дисбаланс між класами, фокусуючись на важких для класифікації прикладах, що є особливо важливим для задач сегментації та детекції спанів. Адаптивні пороги, обчислені для кожної техніки чи спану, а також метрики, засновані на критерії перекриття спанів, забезпечують гнучкість у оцінці результатів і їхньої відповідності реальним умовам

Загалом, зазначені підходи створюють надійну основу для побудови високоякісних моделей, здатних до ефективного виявлення маніпулятивного вмісту в текстах. Проте зазначається, що потенціал для покращення залишається: розширення методів аугментації даних, таких як синонімізація чи парафразування, може підвищити стійкість моделей до різноманітних лінгвістичних конструкцій. Оптимізація гіперпараметрів може бути корисною для подальшого підвищення точності. Таким чином результати роботи демонструють значний прогрес у вирішенні поставлених завдань, водночас відкриваючи можливості для подальшого розвитку та адаптації до більш складних сценаріїв.

4 ОЦІНКА Й ТЕСТУВАННЯ РОЗРОБЛЮВАНОГО РІШЕННЯ

Ефективність будь-якої системи аналізу тексту, зокрема в задачах виявлення маніпулятивних технік, безпосередньо залежить від коректності побудови експериментів, умов навчання моделей, набору даних і повноті процедури їх тестування. Оскільки розроблене рішення інтегрується в Телеграм-бот і буде взаємодіяти з реальними користувачами, особливою вимогою є забезпечення стабільності й узагальнюваності на даних, які не використовувалися на етапі тренування.

У рамках цієї роботи процес навчання, валідації та тестування моделей здійснювався на Kaggle, що надало можливість використовувати виділене середовище з двома GPU для пришвидшеного експериментування та оптимізації параметрів [21]. Платформа виступила не лише як ресурс із обчислювальними можливостями, а й як зручне інтегроване середовище для проведення машинних експериментів із можливістю швидкого відтворення результатів. Завдяки підтримці популярних бібліотек і готових інструментів для роботи з наборами даних середовище дозволило зосередитися саме на моделюванні, а не на налаштуванні інфраструктури.

Рисунок 4.1 – Логотип сервісу Kaggle

У цьому розділі розглядаються підходи до оцінювання якості розроблюваного рішення, описуються метрики, обрані для різних типів задач (бінарної, багатокласової класифікації і послідовного маркування), а також подається аналіз отриманих результатів на кожному етапі експериментів. Особливу увагу приділено оцінці узагальнювальної здатності моделі на тестових підмножинах і виявленню основних помилок класифікації й визначення маніпулятивних фрагментів.

Окремо розглядається тестування інтегрованої системи в середовищі Телеграм-бота, що включає перевірку коректності обробки запитів, швидкості відповіді й стійкості роботи моделі з нестандартними реальними даними, отриманими з Телеграм-каналів. Таке комплексне оцінювання дозволяє встановити практичну придатність розробленого інструмента та визначити шляхи його покращення.

4.1 Аналіз й оцінка результатів навчання span-detection моделі

У процесі тренування span-модель проходила кілька повноцінних епох і на кожній із них спостерігався свій характерний прогрес. На початку першої епохи модель фактично ще «не розуміла», як виглядають межі маніпулятивних фрагментів. Вона робила багато хибних передбачень, часто помилялася з межами span-ів і плутала початки з продовженнями фрагментів. Через це початковий F1-score був досить низьким, що типово для таких задач. Протягом першої епохи після десятків batch-ітерацій train-loss почав поступово зменшуватися: фокальна функція втрат допомагала моделі більше концентруватися на складних випадках. До завершення першої епохи модель уже впевненіше вирізняла короткі фрагменти й краще відокремлювала їх від звичайного тексту. (рис. 4.2).

```
Setting up AdamW optimizer with LLRD (Rate: 0.9)...
Applying LLRD with rate 0.9 over 24 layers.
Setting up learning rate scheduler...
Total optimization steps: 3248, Warmup steps: 324
Starting training...

--- Epoch 1/8 ---
Epoch 1 Train Loss: 0.0512 | Train F1: 0.4361
Epoch 1 Val Loss: 0.0335 | Val Precision: 0.4824 | Val Recall: 0.9511 | Val F1: 0.5250
✨ New best model saved with F1: 0.5250!
```

Рисунок 4.2 – Результат навчання за першою епохою

У другій епосі ваги нижніх шарів трансформера стали рухатися більш впевнено завдяки layer-wise learning rate decay (LLRD), адже нижні шари трансформера зберігали базові багатомовні знання, а верхні поступово пристосовувалися до задачі. На цьому етапі вона все краще відокремлювала маніпулятивні фрагменти від звичайного тексту. Межі span-ів почали вирівнюватися з реальними, зменшилася кількість зайвих B-тегів, а I-теги почали об’єднуватися у більш природні неперервні послідовності. До кінця другої епохи точність суттєво зросла, а recall перестав суттєво зменшуватися, бо модель стала менш обережною і навчилася виявляти довші складні фрагменти. Валідаційний F1 почав підніматися й став першим показником, що сигналізував про реальний прогрес і формуванням у моделі паттернів маніпулятивних структур (рис. 4.3).

```
--- Epoch 2/8 ---
Epoch 2 Train Loss: 0.0351 | Train F1: 0.5176
Epoch 2 Val Loss: 0.0322 | Val Precision: 0.5370 | Val Recall: 0.9504 | Val F1: 0.5762
✨ New best model saved with F1: 0.5762!
```

Рисунок 4.3 – Результат навчання за першою епохою

Під час третьої епохи оптимізатор рухався вже дрібними кроками – scheduler зменшив швидкість навчання, тому зміни відбувалися м’яко й поступово. Однак модель почала краще розуміти контекст: стала помічати тонкі стилістичні ознаки маніпулятивності й коректно відокремлювати їх від нейтральних речень. У валідації стало помітним, що помилок стало менше не лише на токен-рівні, а й на рівні цілих span-ів. Загалом модель рідше пропускала межі або розривала їх без

потреби. Наприкінці епохи F1 досяг свого найкращого значення, і стало зрозуміло, що модель увійшла в зону стабільного навчання (рис. 4.4).

```
--- Epoch 3/8 ---
Epoch 3 Train Loss: 0.0298 | Train F1: 0.5691
Epoch 3 Val Loss: 0.0342 | Val Precision: 0.6178 | Val Recall: 0.9205 | Val F1: 0.6259
✨ New best model saved with F1: 0.6259!
```

Рисунок 4.4 – Результат навчання за третьою епохою

Четверта епоха вже не принесла значного приросту. Валідаційний F1 майже не змінився, а подекуди навіть трохи коливався. Train-loss продовжували повільно спадати, але модель робила це без видимого покращення якості. Було відчутно, що вона наближається до плато, коли подальше навчання вже не давало практичних покращень. Саме тут рання зупинка стала показовою — вона спрацювала після кількох епох без покращення. Саме на цьому етапі спрацював механізм ранньої зупинки: тренування завершилося, а система повернулася до параметрів, отриманих у тій точці, де валідаційний F1 був максимальним (рис. 4.5).

```
--- Epoch 4/8 ---
Epoch 4 Train Loss: 0.0260 | Train F1: 0.6190
Epoch 4 Val Loss: 0.0382 | Val Precision: 0.6595 | Val Recall: 0.9009 | Val F1: 0.6483
✨ New best model saved with F1: 0.6483!
```

Рисунок 4.5 – Результат навчання за четвертою епохою

Починаючи з п’ятої епохи модель увійшла в стабільну фазу навчання: train-loss продовжував повільно зменшуватися, однак це вже не давало відчутного приросту валідаційної якості. У шостій епосі стало помітно, що модель фактично відтворює ті самі патерни, які сформувала раніше, а невеликі коливання валідційного F1 перебували в межах статистичної похибки. Сьома епоха лише підтвердила цю тенденцію: система впевнено працювала на оптимальному рівні, не демонструючи ознак перенавчання, не покращуючи результати. Восьма епоха остаточно засвідчило про те, що подальше навчання не приводить до реальних змін у якості передбачень. У свою чергу активувався механізм ранньої зупинки, фіксуючи найкращий стан моделі на попередніх етапах (рис. 4.6).

```
--- Epoch 5/8 ---
Epoch 5 Train Loss: 0.0218 | Train F1: 0.6638
Epoch 5 Val Loss: 0.0472 | Val Precision: 0.6791 | Val Recall: 0.8852 | Val F1: 0.6543
✨ New best model saved with F1: 0.6543!

--- Epoch 6/8 ---
Epoch 6 Train Loss: 0.0183 | Train F1: 0.6985
Epoch 6 Val Loss: 0.0530 | Val Precision: 0.6527 | Val Recall: 0.8969 | Val F1: 0.6425
No F1 improvement (0.6425 vs best 0.6543). Counter: 1/3

--- Epoch 7/8 ---
Epoch 7 Train Loss: 0.0161 | Train F1: 0.7229
Epoch 7 Val Loss: 0.0566 | Val Precision: 0.6595 | Val Recall: 0.8904 | Val F1: 0.6430
No F1 improvement (0.6430 vs best 0.6543). Counter: 2/3

--- Epoch 8/8 ---
Epoch 8 Train Loss: 0.0142 | Train F1: 0.7388
Epoch 8 Val Loss: 0.0680 | Val Precision: 0.7027 | Val Recall: 0.8673 | Val F1: 0.6643
✨ New best model saved with F1: 0.6643!
Loading best model state with F1: 0.6643
Saving the fine-tuned model...
```

Рисунок 4.6 – Результат навчання за четвертою епохою

Після завершення тренування було проведено порівняння початкового набору даних, на якому навчалася модель, з датасетом, сформованим після навчання.Результати представлені у вигляді показників метрик якості – precision, recall, F1 (рис 4.7).

```
Precision: 0.5558104415360277
Recall: 0.6589397762924567
F1: 0.6029973942486231

Process finished with exit code 0
```

Рисунок 4.7 – Результат навчання за четвертою епохою

Показники свідчать про помірну точність моделі в виявленні маніпулятивних фрагментів, з акцентом на recall, здатність не пропускати релевантні спани, що свідчить про пріоритет мінімізації пропуску маніпуляцій навіть ціною появи певної кількості хибних спрацьовувань. Така поведінка значною мірою зумовлена використанням фокальної функції втрат, яка змушувала алгоритм агресивно реагувати на найменші ознаки цільового класу. Детальний розгляд розбіжностей у межах span-ів виявив тенденцію до так званого «розмиття кордонів», коли модель

коректно ідентифікує саме наявність маніпуляції, але не ідеально фіксує її початок і кінець, помилково включаючи до виділеного фрагмента сусідні нейтральні токени – розділові знаки чи сполучники. За результатами порівняння можно стверджувати, що модель успішно засвоїла структуру маніпулятивних висловлювань, досягла свого оптимуму в межах наявної архітектури й даних. Виявлені розбіжності між файлами підкреслюють необхідність подальшого розширення навчальної вибірки і впровадження етапу пост-обробки для фільтрації хибних спрацьовувань.

У результаті тренування по епохам продемонструвало поступовий рух від хаотичних і надто обережних передбачень до впевненого, збалансованого визначення span-ів із чіткими межами, що відповідають реальним маніпулятивним фрагментам. Валідаційний F1 зростав упродовж ключових етапів навчання й досяг максимального значення в точці, де модель стабільно відтворювала структуру маніпулятивних фрагментів на валідаційних даних. Водночас результати виявили й певні обмеження: після третьої і четвертої епох модель практично вийшла на плато, а подальше зменшення train-loss не сприяло поліпшенню узагальнюючої здатності. Це свідчить про чутливість моделі до розміру й різноманітності датасету. Попри загалом високі показники, система періодично плутала межі довгих фрагментів або некоректно позначала слабовиражені стилістичні маніпуляції. Стабілізація метрик у пізніх епохах і активація ранньої зупинки свідчать про те, що модель досягла оптимуму в рамках доступних даних. Слід зазначити, що потенціал для покращення залишається – насамперед через необхідність ширшої і більш збалансованої навчальної вибірки.

4.2 Аналіз й оцінка результатів навчання class-detection моделі

У першій епосі модель продемонструвала типові стартові характеристики для багатокласового мультилейблового завдання з нерівномірним розподілом класів. Значення macro-F1 становило 0.3103. Найвищі результати були отримані для класів із чітко вираженими лексико-семантичними індикаторами, зокрема

loaded_language (0.71) та glittering_generalities (0.41). Розраховані оптимальні пороги вказали на істотну різницю у впевненості моделі між класами. Деякі категорії (bandwagon, whataboutism) демонстрували низький F1 через слабку вираженість формальних ознак у даних. Під кінець епохи стало помітно зміщення оптимальних порогів (optimal thresholds) демонструючи спробу точніше відокремлювати справді релевантні сигнали маніпуляцій від шуму (рис. 4.7).

```
Epoch 1/10
Loading widget...
Training loss: 1.0161
Loading widget...
Validation Macro F1: 0.3103
Updated optimal thresholds based on best F1: {'straw_man': 0.5600000000000004, 'appeal_to_fear':
0.4700000000000003, 'fud': 0.5700000000000004, 'bandwagon': 0.43000000000000027, 'whataboutism':
0.5100000000000003, 'loaded_language': 0.5200000000000004, 'glittering_generalities': 0.65000000
00000005, 'euphoria': 0.4400000000000003, 'cherry_picking': 0.5700000000000004, 'cliche': 0.5000
000000000003}
straw_man: 0.1667
appeal_to_fear: 0.1720
fud: 0.3610
bandwagon: 0.1304
whataboutism: 0.1347
loaded_language: 0.7109
glittering_generalities: 0.4145
euphoria: 0.2927
cherry_picking: 0.4032
cliche: 0.3168
Saved best model!
```

Рисунок 4.7 – Результат навчання за першою епохою

У другій епосі macro-F1 підвищився до 0.3924. Покращення спостерігалося передусім у класах з помірно стабільними контекстними патернами, як-от glittering_generalities (0.64) і cherry_picking (0.44). Результати вказують на початок формування узгоджених ознак на рівні локальних контекстів і риторичних структур. Значення для loaded_language залишалось стабільно високим, що підтверджує наявність надійних ключових маркерів у корпусі. Цікаво, що поведінка моделі стала більш послідовною: вона менше покладалася на випадкові збіги та частіше відтворювала ті самі патерни у схожих контекстах. У цей момент стало очевидно, що вона вчиться не просто реагувати на ключові слова, а починає формувати більш узагальнене уявлення про логічні й риторичні конструкції (рис. 4.8).

```
 Epoch 2/10
Loading widget...
 Training loss: 0.8743
Loading widget...
 Validation Macro F1: 0.3924
 straw_man: 0.2683
 appeal_to_fear: 0.2570
 fud: 0.4541
 bandwagon: 0.1743
 whataboutism: 0.2297
 loaded_language: 0.7121
 glittering_generalities: 0.6426
 euphoria: 0.4262
 cherry_picking: 0.4370
 cliche: 0.3230
 Saved best model!
```

Рисунок 4.8 – Результат навчання за другою епохою

Третя епоха супроводжувалась невеликим зниженням macro-F1 до 0.3741. Найпомітнішим відхиленням став клас cliche, де F1 впав до нуля, що свідчить про нестачу розрізнювальних ознак або недостатню кількість репрезентативних прикладів. Модель почала більше зважати на складніші закономірності, але ще не могла їх стабільно застосовувати. Також слід зазначити, що класи fud і euphoria зберегли стабільні показники. Динаміка валідаційних метрик у цій епосі може відповідати фазі локальної нестабільності під час узагальнення нових структурних ознак (рис. 4.9).

```
 Epoch 3/10
Loading widget...
 Training loss: 0.8106
Loading widget...
 Validation Macro F1: 0.3741
 straw_man: 0.2740
 appeal_to_fear: 0.3407
 fud: 0.4615
 bandwagon: 0.2222
 whataboutism: 0.2675
 loaded_language: 0.6687
 glittering_generalities: 0.6182
 euphoria: 0.5191
 cherry_picking: 0.3690
 cliche: 0.0000
 No improvement for 1 epochs.
```

Рисунок 4.9 – Результат навчання за четвертою епохою

У четвертій епосі значення macro-F1 підвищилось до 0.4368 – на той момент це був найкращий показник. Значно покращились результати для glittering_generalities (0.67), loaded_language (0.75) і cherry_picking (0.49). Класи з менш явними стилістичними індикаторами (whataboutism, bandwagon) залишилися

на низькому рівні, що корелює з їх слабко структурованими патернами у даних. Загалом у четвертій епосі можна прослідкувати новий рівень збалансованості – модель суттєво знизила схильність до домінування окремих класів і демонструє значно збалансованішу поведінку, підтримуючи порівнянну продуктивність по всьому набору класів (рис 4.10).

```
Epoch 4/10
Loading widget...
Training loss: 0.7532
Loading widget...
Validation Macro F1: 0.4368
straw_man: 0.2887
appeal_to_fear: 0.4156
fud: 0.4739
bandwagon: 0.1250
whataboutism: 0.2317
loaded_language: 0.7589
glittering_generalities: 0.6725
euphoria: 0.5303
cherry_picking: 0.4967
cliche: 0.3750
Saved best model!
```

Рисунок 4.10 – Результат навчання за четвертою епохою

У п’ятій епосі модель досягла 0.4428 macro-F1. Найбільший приріст спостерігався у класі fud (0.53). Значно зміцніли позиції glittering_generalities (0.6492) і loaded_language (0.7406), які стабільно утримують лідерство серед усіх класів. Ці показники підтверджують високу чутливість моделі до емоційно забарвленої або ідеологічно навантаженої лексики. Покращення в інших класах було помірним, загальна поведінка моделі стала більш стабільною, що підтверджує поступове формування стійких ознак на рівні токенів і фраз (рис 4.11).

```
Epoch 5/10
Loading widget...
Training loss: 0.6944
Loading widget...
Validation Macro F1: 0.4428
straw_man: 0.2800
appeal_to_fear: 0.3164
fud: 0.5315
bandwagon: 0.2677
whataboutism: 0.2946
loaded_language: 0.7406
glittering_generalities: 0.6492
euphoria: 0.4771
cherry_picking: 0.4700
cliche: 0.4013
Saved best model!
```

Рисунок 4.11 – Результат навчання за п'ятої епохою

Шоста й сьома епохи характеризуються незначною варіацією метрики macro-F1 у межах 0.433-0.434. Зниження результатів у класах cliche, bandwagon свідчить про чутливість моделі до рідкісних конструкцій і можливу залежність від дисбалансу класів. Водночас класи типу loaded_language зберегли високі й стабільні показники, що підтверджує їх структурну відокремленість (рис 4.12).

```
Epoch 6/10
Loading widget...
 Training loss: 0.6437
Loading widget...
 Validation Macro F1: 0.4334
 straw_man: 0.3000
 appeal_to_fear: 0.3179
 fud: 0.4663
 bandwagon: 0.2370
 whataboutism: 0.3333
 loaded_language: 0.7324
 glittering_generalities: 0.6421
 euphoria: 0.5179
 cherry_picking: 0.4724
 cliche: 0.3143
 No improvement for 1 epochs.
```

```
Epoch 7/10
Loading widget...
 Training loss: 0.6019
Loading widget...
 Validation Macro F1: 0.4341
 straw_man: 0.2637
 appeal_to_fear: 0.3696
 fud: 0.4903
 bandwagon: 0.2131
 whataboutism: 0.2804
 loaded_language: 0.7507
 glittering_generalities: 0.6537
 euphoria: 0.5327
 cherry_picking: 0.4579
 cliche: 0.3288
 No improvement for 2 epochs.
```

Рисунок 4.12 – Результат навчання за шостою і сьомою епохами

У восьмій епосі був досягнутий найкращий результат – 0.4469 macro-F1. Покращення відбулося у класах euphoria (0.5385), fud і cliche. Значення розподілились більш рівномірно, що вказує на покращення здатності моделі відокремлювати семантично близькі риторичні техніки. На цьому етапі модель досягла оптимального балансу між узагальненням і пристосуванням до тренувальних даних (рис 4.13).

```
Epoch 8/10
Loading widget...
 Training loss: 0.5746
Loading widget...
 Validation Macro F1: 0.4469
 straw_man: 0.2716
 appeal_to_fear: 0.3841
 fud: 0.4966
 bandwagon: 0.2703
 whataboutism: 0.2791
 loaded_language: 0.7455
 glittering_generalities: 0.6476
 euphoria: 0.5385
 cherry_picking: 0.4721
 cliche: 0.3636
 Saved best model!
```

Рисунок 4.13 – Результат навчання за шостою і сьомою епохами

У дев’ятій та десятій епосах покращення не спостерігалося: macro-F1 залишався в межах статистичної похибки (0.439-0.442). Зниження позитивної динаміки при збереженні падіння тренувальних втрат може свідчити про початок перенавчання. Найбільш стабільним класом залишався loaded_language, тоді як bandwagon і whataboutism зберігали низькі значення через слабко формалізовану структуру.

```
 Epoch 9/10
Loading widget...
 Training loss: 0.5549
Loading widget...
 Validation Macro F1: 0.4423
 straw_man: 0.2532
 appeal_to_fear: 0.3797
 fud: 0.4845
 bandwagon: 0.2667
 whataboutism: 0.2887
 loaded_language: 0.7415
 glittering_generalities: 0.6351
 euphoria: 0.5225
 cherry_picking: 0.4774
 cliche: 0.3740
 No improvement for 1 epochs.
```

```
 Epoch 10/10
Loading widget...
 Training loss: 0.5463
Loading widget...
 Validation Macro F1: 0.4390
 straw_man: 0.2564
 appeal_to_fear: 0.3467
 fud: 0.4898
 bandwagon: 0.2581
 whataboutism: 0.2917
 loaded_language: 0.7446
 glittering_generalities: 0.6377
 euphoria: 0.5249
 cherry_picking: 0.4655
 cliche: 0.3745
 No improvement for 2 epochs.
```

Рисунок 4.14 – Результат навчання за дев’ятою і десятою епохами

Графік «Class-wise F1 Scores» є найбільш інформативним для оцінки здатності моделі до диференціації, оскільки він ілюструє, наскільки нерівномірно відбувається прогрес навчання та узагальнення ознак для кожної окремої риторичної техніки. Розглянемо детальніше динаміку F1-метрики в розрізі класів, щоб ідентифікувати найбільш стійкі й найбільш проблемні для моделі категорії маніпуляцій (рис. 4.16).

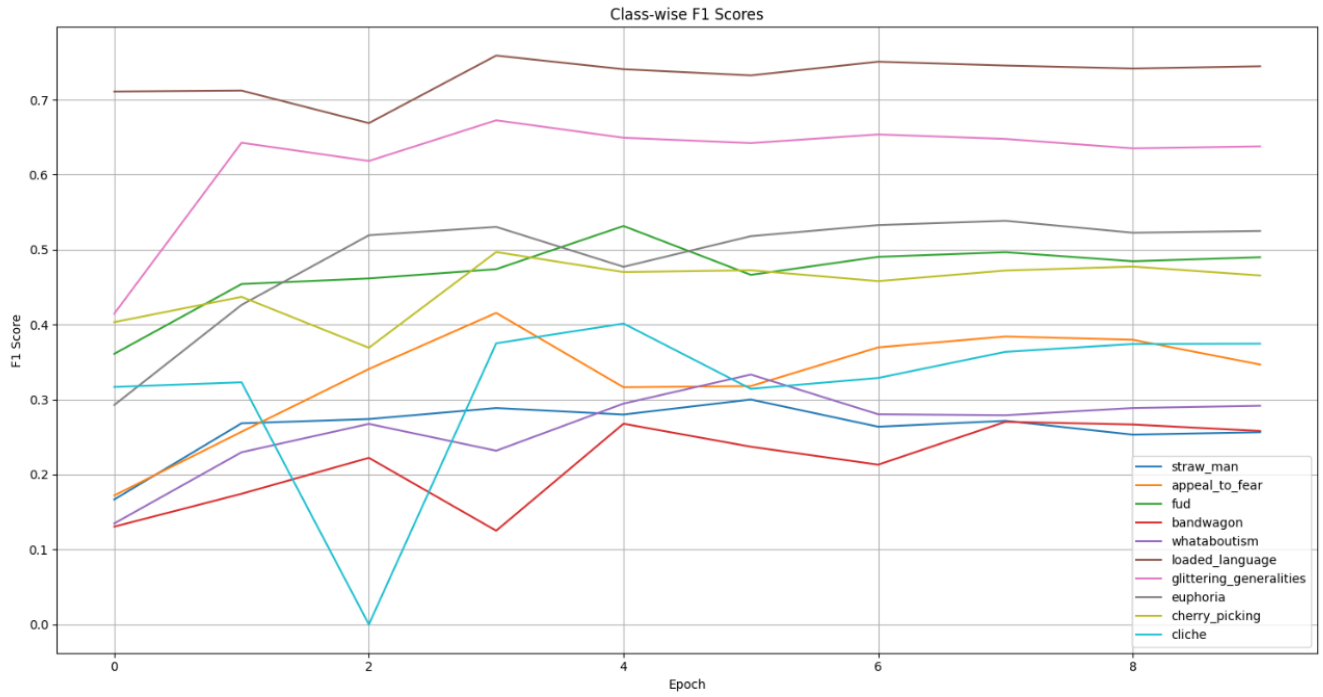


Рисунок 4.14 – Результуючий графік Class-wise F1 Scores

Очевидними лідерами за стабільністю і величиною F1-Score є класи «loaded_language» і «glittering_generalities», які постійно утримують значення F1 вище 0.65. Значення вказують на наявність у цих категоріях чітких, добре формалізованих лексико-семантичних маркерів, які модель легко ідентифікує. На противагу цьому класи, «bandwagon», «whataboutism» і «straw_man», протягом усього тренування залишаються у нижній частині графіка, ледве долаючи позначку 0.3. Їх низька продуктивність зумовлена високою контекстуальною залежністю, складністю формалізації патернів і меншою кількістю репрезентативних прикладів у навчальному датасеті, що не дає моделі змоги сформувати стійкі ознаки. Флуктуації спостерігаються у класі «cliche», який переживає різке падіння до нуля на третій епосі, але швидко відновлюється. Ця зміна свідчить про тимчасову втрату або перепризначення ваг під час фази узагальнення, але успішне їх відновлення в наступних ітераціях. Динаміка за класами підтверджує, що найбільшим стримуючим фактором для зростання загального Macro F1 є дисбаланс і неоднорідність класів, а не фундаментальна нездатність моделі до навчання.

Підсумковий найкращий результат Macro-F1 – 0.4469 на 8 епосі. Модель демонструє впевнене розпізнавання класів із чіткими лексико-семантичними патернами, але обмежену продуктивність для технік зі слабко вираженими або контекстно залежними ознаками. Основними стримуючими чинниками виступають дисбаланс класів і неоднорідність формальних індикаторів у ряді категорій.

4.3 Аналіз роботи моделей інтегрованих у Телеграм

Архітектура системи базується на послідовному використанні двох нейромережевих моделей. Для забезпечення високої швидкодії і оптимізації використання ресурсів обчислення за можливості переносяться на графічний процесор Compute Unified Device Architecture (CUDA) із використанням режиму половинної точності, а при ініціалізації системи виконується обов'язковий етап попередньої підготовки моделей для уникнення затримок під час перших запитів користувачів. При використанні Nvidia GeForce GTX 1650 4GB потрібно приблизно 2 хвилини для першого запуску бота.

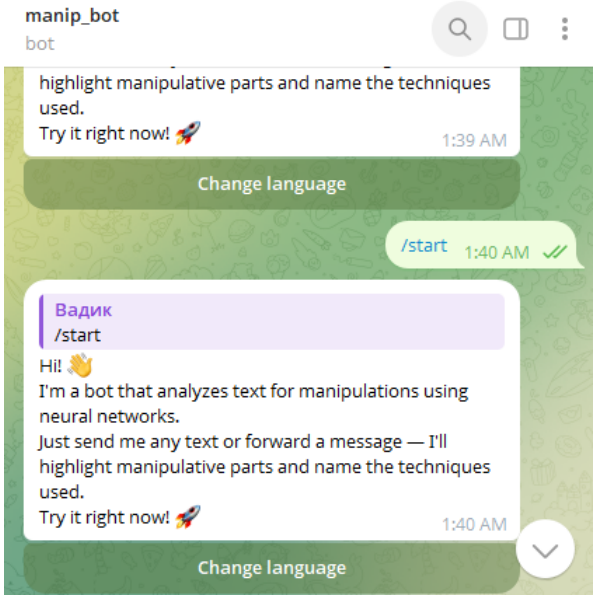


Рисунок 4.15 – Початок роботи боту

Процес обробки повідомлення розпочинається з токенізації вхідного тексту і роботи span-моделі, яка відповідає за локалізацію підозрілих фрагментів безпосередньо у реченні. Отримані прогнози проходять етап алгоритмічної

постобробки, під час якого окремі токени трансформуються у текстові інтервали. У свою чергу близькі за розташуванням фрагменти автоматично об'єднуються для покращення читабельності результату. Треба взяти до уваги, що span-detection модель показала кращі результати у знаходженні маніпуляцій ніж класифікаційна модель. Замість паралельної обробки тексту обома моделями буде відбуватися пошук маніпулятивних спанів і у випадку їх наявності здійснено класифікацію. Вона здійснюється за допомогою системи лінійних шарів, а функція активації визначає ймовірність наявності однієї з десяти маніпулятивних технік, порівнюючи отримані значення з індивідуальними порогами чутливості для кожного класу.

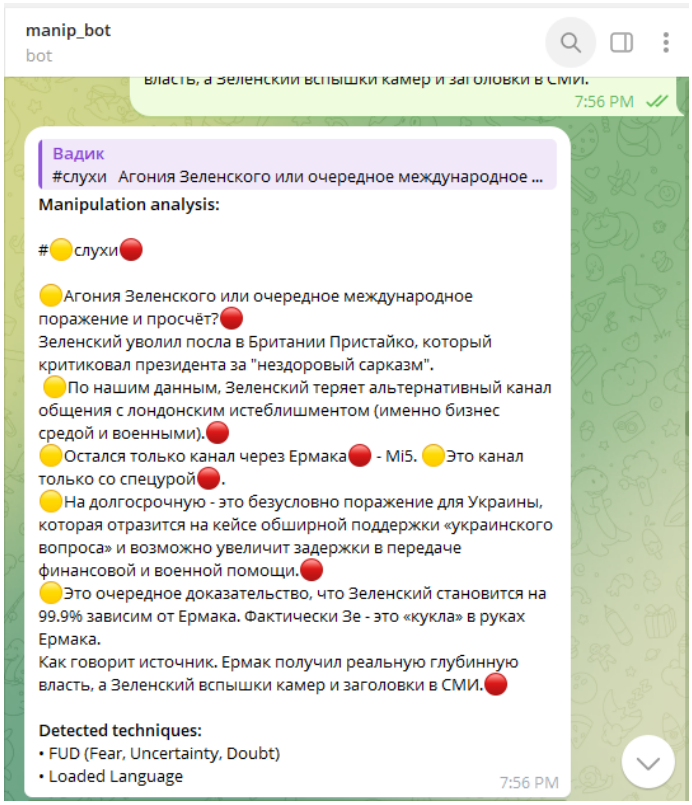


Рисунок 4.16 – Результат аналізу на наявність маніпуляцій

Додатково реалізовано постобробку результатів, яка надає пріоритет певним технікам у разі близьких ймовірностей, обмежує кількість виявлених технік трьома найбільш релевантними й передбачає резервний механізм для присвоєння базової техніки при наявності спанів без чіткої класифікації. Інтеграція з інтерфейсом бота забезпечує підтримку двох мовних режимів з автоматичним перемиканням, зберіганням останнього аналізу для подальшого доступу до детальних описів технік та візуальним виділенням маніпулятивних фрагментів за допомогою

спеціальних маркерів для кращої інтерпретації користувачем. Логування ймовірностей під час обробки сприяє моніторингу ефективності системи в реальному часі й допомагає при аналізі ефективності класифікаційної моделі. Потреба у цьому рішенні, як зазначалося раніше, виникає через доволі невеликого навчального набору даних.

```
Probabilities: straw_man: 0.4468, appeal_to_fear: 0.5571, fud: 0.4358, bandwagon: 0.4614, whataboutism: 0.4832, loaded_language: 0.5122, glittering_generalities:
Probabilities: straw_man: 0.4468, appeal_to_fear: 0.5571, fud: 0.4358, bandwagon: 0.4614, whataboutism: 0.4832, loaded_language: 0.5122, glittering_generalities:
```

Рисунок 4.17 – Результат логування аналізу повідомлення

Важливим елементом архітектури є розроблений механізм резервної класифікації (fallback mechanism), який забезпечує узгодженість роботи двох нейромереж. У ситуаціях, коли модель виділення сутностей (span-model) успішно ідентифікує підозрілі фрагменти тексту, але класифікатор не досягає необхідного порогу впевненості для жодної зі специфічних технік, система виконує вторинну перевірку на наявність ознак загальної маніпулятивної мови. Якщо відповідна ймовірність перевищує мінімально допустимий рівень, повідомлення маркується як маніпулятивне навіть без чіткої прив'язки до складної категорії, що мінімізує ризик пропуску загрози (False Negative).

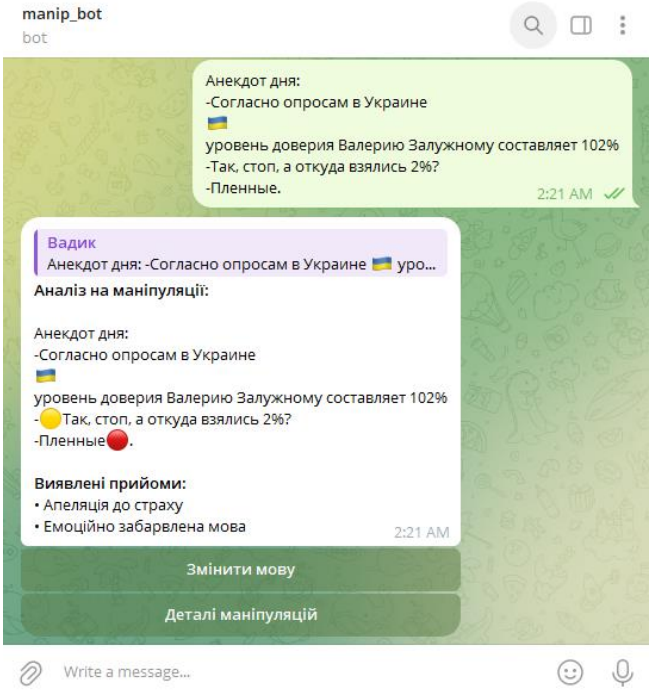


Рисунок 4.18 – Результат аналізу повідомлення на предмет маніпуляцій

Взаємодія з користувачем оптимізована шляхом використання тимчасового сховища станів у оперативній пам'яті сервера. Результати останнього успішного аналізу, включаючи список виявлених технік, прив'язуються до ідентифікатора користувача, що дозволяє реалізувати функцію отримання деталізованих пояснень через callback-запити без необхідності повторного прогону тексту через нейромережу. Такий підхід суттєво знижує обчислювальне навантаження на систему при активній взаємодії з ботом та забезпечує миттєву реакцію інтерфейсу при запиті довідкової інформації. Фінальна візуалізація результатів здійснюється з використанням спеціальних алгоритмів екранування символів розмітки Markdown, що гарантує коректне відображення виділених фрагментів незалежно від наявності службових символів у вихідному тексті.

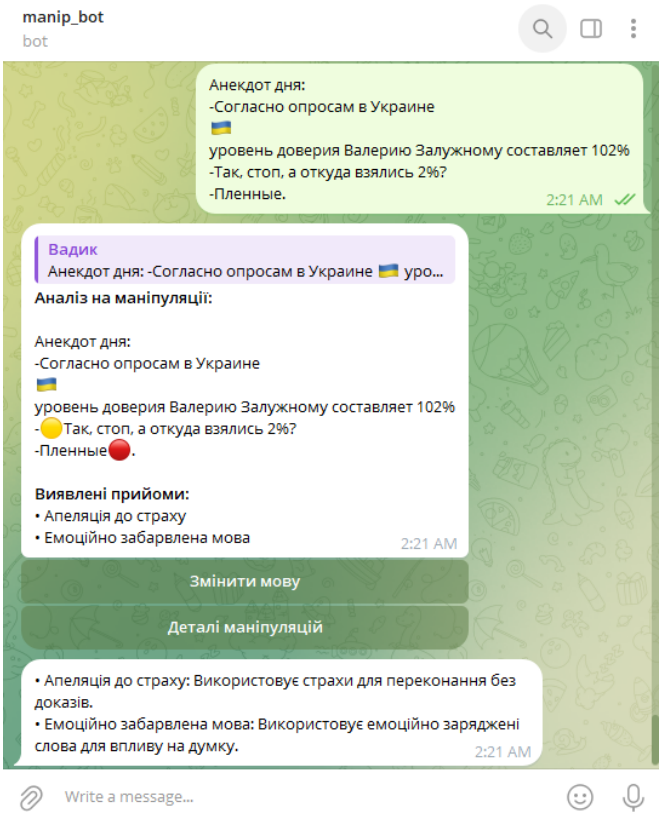


Рисунок 4.19 – Приклад використання callback-запитів

Вищезазначені рішення формують цілісну систему аналізу маніпулятивного контенту, придатну для використання в умовах з високими вимогами до швидкодії. Поєднання каскадної логіки, адаптивної постобробки результатів і механізмів

резервної перевірки дозволяють досягти змістовно-інтерпретованих результатів навіть за наявності помилкової класифікації. Такий підхід забезпечує баланс між точністю виявлення, обчислювальною ефективністю і зручністю взаємодії з користувачем, що є критично важливим для практичного впровадження системи автоматизованої перевірки текстів адаптованої до реального інформаційного середовища.

Висновки до розділу 4

Проведено оцінювання і тестування розробленого рішення для виявлення маніпулятивних технік у тексті із урахуванням вимог до узагальнюваності, стабільності й практичної придатності системи при використанні з реальними користувацькими даними. Проаналізовано середовище проведення навчання моделей на базі платформи Kaggle й обґрунтовано його використання для валідації і відтворюваності результатів моделей.

Проаналізовано динаміку навчання span-detection моделі, встановлено характер поступового підвищення якості виявлення маніпулятивних фрагментів, досягнення плато і ефективність застосування механізму ранньої зупинки. Визначено, що модель демонструє пріоритет високої повноти (recall), що мінімізує ризик пропуску маніпуляцій, однак супроводжується розмиттям меж окремих спанів. Описано основні обмеження цієї моделі, які зумовлені розміром і різноманітністю навчальної вибірки.

Проаналізовано результати навчання багатокласової мультилейблової моделі класифікації маніпулятивних технік, зокрема динаміку macro-F1 та поведінку моделі в розрізі окремих класів. Встановлено, що найвищу стабільність і точність демонструють класи з чіткими лексико-семантичними маркерами, тоді як техніки зі слабко формалізованими або контекстно залежними ознаками залишаються основним джерелом помилок. Було виявлено вплив дисбалансу класів на загальну якість моделі та визначено оптимальну точку навчання.

Було описано роботу інтегрованої системи в середовищі Телеграм-бота, включно з каскадною взаємодією двох моделей, алгоритмами постобробки, механізмом резервної класифікації та оптимізацією обчислювальних ресурсів. Було підтверджено, що запропонована архітектура забезпечує прийнятну швидкодію, узгодженість результатів і зручність інтерпретації для кінцевого користувача.

Додатково встановлено, що тестування в умовах, наближених до реального використання, дозволило виявити низку практичних аспектів, які не проявляються на етапі ізольованого навчання моделей. Зокрема, інтеграція span-detection і class-detection у каскадну архітектуру продемонструвала кращу стійкість до шумних і неструктурованих повідомлень, характерних для Telegram-контенту, а також зменшила кількість критичних помилок типу false negative. Реалізований механізм постобробки суттєво підвищили узгодженість результатів і їх інтерпретованість для кінцевого користувача. Практичне тестування Telegram-бота підтвердило здатність рішення працювати в режимі реального часу з прийнятною швидкодією, зберігаючи стабільність і зручність використання навіть за обмежених навчальних ресурсів.

Отримані результати в сукупності підтверджують працездатність і практичну доцільність розробленого рішення, а також окреслюють напрями подальшого вдосконалення, пов’язані з розширенням і балансуванням навчального корпусу та подальшою оптимізацією моделей.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано й спроєктовано застосунок для пошуку маніпулятивних технік у текстових даних, що сприяє підвищенню рівня довіри й безпеки в цифровому середовищі, зокрема для платформ соціальних мереж та комунікацій. Проведено аналіз предметної області, обґрунтувано актуальність розробки застосунку в умовах зростання дезінформації й маніпуляцій, де ключовими викликами є швидке виявлення прихованих патернів у багатомовних текстах. Було описано предметне середовище, включаючи архітектуру систем моніторингу контенту, взаємодію користувачів і специфіку обробки даних у реальному часі. При аналізі розроблюваного застосунку досліджено можливості інтеграції штучного інтелекту для гранулярного пошуку маніпуляцій. Сформовано специфікації вимог до програмного забезпечення, стандарти для функціональності інтерфейсу й продуктивної роботи застосунку.

Було розглянуто методології і підходів для навчання моделей охопив спектр від базових алгоритмів до передових архітектур, демонструючи еволюцію до контекстуально-орієнтованих методів для ефективного розпізнавання прихованих елементів. Проведено аналіз стратегії навчання для оптимізації трансформерних моделей, що забезпечило теоретичну основу для адаптації до специфіки пошуку маніпуляцій, враховуючи обмеженість ресурсів та динаміку текстових даних.

Розроблено концептуальну модель застосунку з вбудованими інструментами штучного інтелекту для автоматизованого пошуку маніпулятивних технік, що поєднує теоретичний аналіз з практичними рекомендаціями щодо впровадження. Отримані результати розв'язують ключові проблеми безпеки в цифрових комунікаціях, сприяючи формуванню стандартів для подібних систем і забезпечуючи стійкість до етичних і технічних викликів. Перспективи подальших досліджень охоплюють емпіричне тестування прототипу на розширених наборах даних, інтеграцію мультимодальних підходів (текст, зображення, аудіо) та адаптацію для інших платформ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Analog application: Fallacy Detector. URL: https://finder.logicalfallacies.org/detector/ (Last accessed: 04.10.2025).

2. Analog application: Gaslighting Check. URL: https://www.gaslightingcheck.com/dashboard/check (Last accessed: 04.10.2025).

3. Analog application: Fallacy Finder. URL: https://word.studio/tool/fallacy-finder/?utm_source=chatgpt.com (Last accessed: 04.10.2025).

4. UNLP 2025 Shared Task on Detecting Social Media Manipulation. URL: https://github.com/unlp-workshop/unlp-2025-shared-task/tree/main (Last accessed: 04.10.2025).

5. Видання Texty.org.ua. URL: https://texty.org.ua/ (Last accessed: 04.10.2025).

6. A Beginner’s Guide to Tokens, Vectors, and Embeddings in NLP. URL: https://medium.com/@saschametzger/what-are-tokens-vectors-and-embeddings-how-do-you-create-them-e2a3e698e037 (Last accessed: 04.10.2025).

7. Vaibhav T. Stratified sampling in Cohort-based data for Machine learning Model development. *International Scientific Journal of Engineering and Management*. 2025. № 1. P. 1–5. DOI: 10.55041/ISJEM03377.

8. Detecting Manipulation in Ukrainian Telegram: A Transformer-Based Approach to Technique Classification and Span Identification. URL: https://aclanthology.org/2025.unlp-1.20.pdf (Last accessed: 04.10.2025).

9. Multi-Class Text Classification with Scikit-Learn using TF-IDF model URL: https://medium.com/@rohit_batra/multi-class-text-classification-with-scikit-learn-using-tf-idf-model-161d395ce374 (Last accessed: 04.10.2025).

10. Complement Naive Bayes (CNB) Algorithm (math clearly explained). URL: https://medium.com/@tarunsaxena1000/complement-naive-bayes-cnb-algorithm-af7b5d2872bb

11. What is random forest? URL: https://www.ibm.com/think/topics/random-forest

12. Gradient Boosting vs. Random Forest: Which Ensemble Method Should You Use? URL: https://medium.com/@hassaanidrees7/gradient-boosting-vs-random-forest-which-ensemble-method-should-you-use-9f2ee294d9c6

13. Mastering LightGBM: Unravelling the Magic Behind Gradient Boosting. URL: http://data-ai.theodo.com/en/technical-blog/mastering-lightgbm-unravelling-the-magic-behind-gradient-boosting

14. Heinzerling B., Strube M. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), May. 2018.

15. How to implement CNN for NLP tasks like Sentence Classification. URL: https://medium.com/saarthi-ai/sentence-classification-using-convolutional-neural-networks-ddad72c7048c (Last accessed: 04.10.2025).

16. Pradeep K. R., Abhinav K. Convolutional Neural Network for Text: A Stepwise Working Guidance. *SSRN Electronic Journal*, Jan. 2021. P. 1–6. DOI: 10.2139/ssrn.3973041.

17. Recurrent Neural Networks - Combination of RNN and CNN. URL: https://collab.dvb.bayern/spaces/TUMlfdv/pages/69119924/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN

18. Liqun S., Yanchang L., Min T., Ming Y., Xueyuan B. CNN-BiLSTM Hybrid Neural Networks with Attention Mechanism for Well Log Prediction. Journal of Petroleum Science and Engineering, Oct. 2021. P. 108838. DOI: 10.1016/j.petrol.2021.108838.

19. Gated Recurrent Unit Networks. URL: https://www.geeksforgeeks.org/machine-learning/gated-recurrent-unit-networks/

20. Recurrent Neural Network. URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/recurrent-neural-network (Last accessed: 04.10.2025).

21. Deep learning architectures. URL: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ (Last accessed: 04.10.2025).

22. Marcial S. A., Karthik D., Yuning W., Ricardo V. Easy attention: A simple

self-attention mechanism for Transformers, Aug. 2023. P. 7–9. DOI: 10.48550/arXiv.2308.12874.

23. How Transformers Work: A Detailed Exploration of Transformer Architecture. URL: https://www.datacamp.com/tutorial/how-transformers-work

24. Focal Loss vs. Binary Cross Entropy Loss. URL: https://blog.dailydoseofds.com/p/focal-loss-vs-binary-cross-entropy (Last accessed: 04.10.2025).

25. Layer-Wise Learning Rate in PyTorch. URL: https://kozodoi.me/blog/20220329/discriminative-lr (Last accessed: 04.10.2025).

26. Fine tuning Vs Pre-training. URL: https://medium.com/@eordaxd/fine-tuning-vs-pre-training-651d05186faf (Last accessed: 04.10.2025).

27. Analysis of English Writing Text Features Based on Random Forest and Logistic Regression Classification Algorithm. URL: https://onlinelibrary.wiley.com/doi/10.1155/2022/6306025 (Last accessed: 04.10.2025).

28. «Unveiling the Power of Light GBM». URL: https://medium.com/@venkatyogesh003/unveiling-the-power-of-light-gbm-e3b46743a2b2 (Last accessed: 04.10.2025).

29. XLM-RoBERTa: The alternative for non-english NLP. URL: https://medium.com/deepset-ai/xlm-roberta-the-multilingual-alternative-for-non-english-nlp-cf0b889ccbbf (Last accessed: 04.10.2025).

30. What Does Pre-training a Neural Network Mean? URL: https://www.baeldung.com/cs/neural-network-pre-training

31. What is fine-tuning? URL: https://www.ibm.com/think/topics/fine-tuning

32. Transfer Learning vs Domain Adaptation. URL: https://www.baeldung.com/cs/transfer-learning-vs-domain-adaptation

33. What is few shot prompting? URL: https://www.ibm.com/think/topics/few-shot-prompting

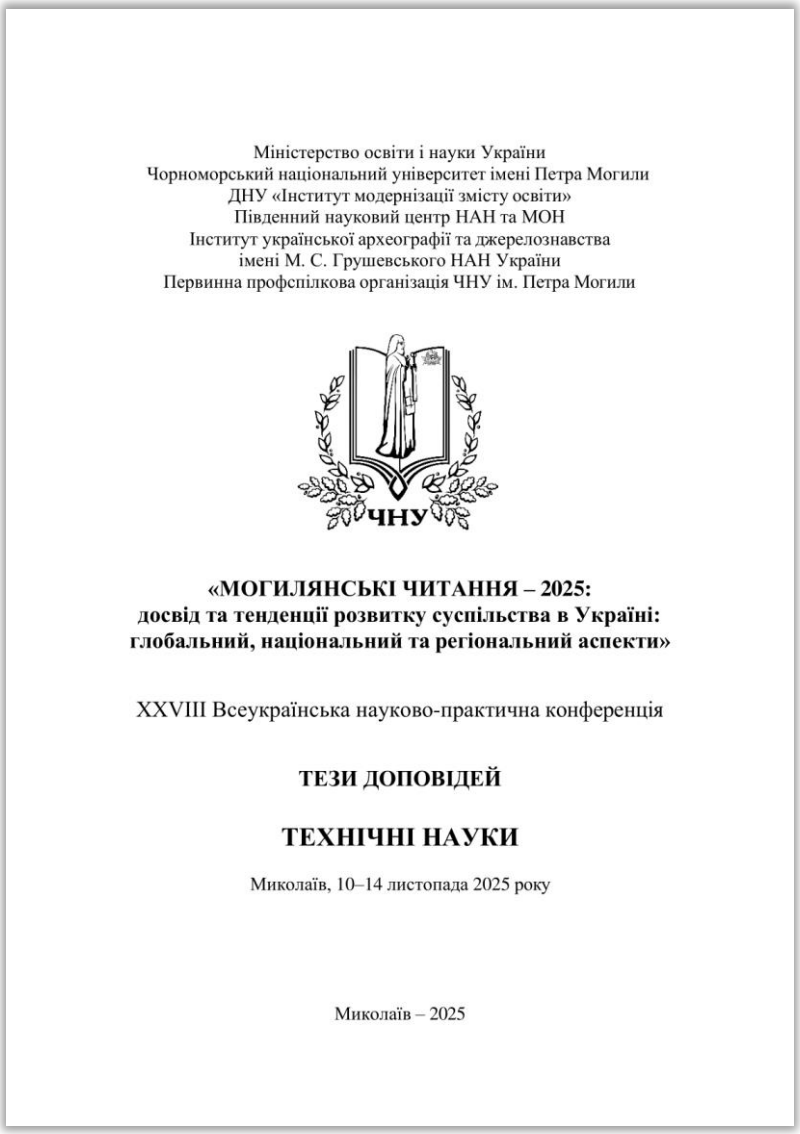
34. What is continual learning? URL: https://www.ibm.com/think/topics/continual-learning

ДОДАТОК А

Апробація кваліфікаційної магістерської роботи

Результати дослідження було представлено на конференції:

– Могилянські читання – 2025, : Аналіз маніпулятивності новин із використанням алгоритмів машинного навчання : XXVIII Всеукр. наук.-практ. конф. : тези доповідей : Комп'ютерні науки. Технічні науки, Миколаїв, 10-14 листоп. 2025 р. / ЧНУ ім. Петра Могили. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2025



Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ ім. Петра Могили

ЧНУ

**«МОГИЛЯНСЬКІ ЧИТАННЯ – 2025:
досвід та тенденції розвитку суспільства в Україні:
глобальний, національний та регіональний аспекти»**

XXVIII Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

ТЕХНІЧНІ НАУКИ

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Рисунок А.1 – Обкладинка збірника тез доповідей конференції Могилянські читання – 2025

ДОДАТОК Б

Лістинг коду навчання моделей

Код файлу *span_learning.py*:

```
import numpy as np
import pandas as pd
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from transformers import (
    AutoModelForTokenClassification,
    AutoTokenizer,
    get_linear_schedule_with_warmup,
    AutoConfig
)
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm
import warnings
warnings.filterwarnings('ignore')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

MODEL_NAME = "FacebookAI/xlm-roberta-large"
MAX_LEN = 512
BATCH_SIZE = 2
LEARNING_RATE = 2e-5
EPOCHS = 8
SEED = 42
TRAIN_VAL_SPLIT = 0.15
WEIGHT_DECAY = 0.01
ACCUMULATION_STEPS = 4
LLRD_RATE = 0.9
SPAN_MERGE_DISTANCE = 1
PATIENCE = 3

torch.manual_seed(SEED)
np.random.seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)

class WeightedFocalLoss(nn.Module):
    """
    Weighted Focal Loss implementation.
    Helps focusing on hard-to-classify examples and handles class imbalance.
    alpha: Weights for each class (e.g., [O_weight, B_weight, I_weight])
    gamma: Focusing parameter (>= 0). Higher gamma focuses more on hard examples.
    """
    def __init__(self, alpha=[0.1, 0.45, 0.45], gamma=2.0, ignore_index=-100):
        super(WeightedFocalLoss, self).__init__()
        self.alpha = torch.tensor(alpha).float()
        self.gamma = gamma
        self.ignore_index = ignore_index
        self.log_softmax = nn.LogSoftmax(dim=-1)

    def forward(self, inputs, targets):
```

```
        mask = targets != self.ignore_index
        valid_inputs = inputs[mask]
        valid_targets = targets[mask]

        if valid_targets.numel() == 0:
             return torch.tensor(0.0, device=inputs.device, requires_grad=True)

        if self.alpha.device != inputs.device:
            self.alpha = self.alpha.to(inputs.device)

        log_probs = self.log_softmax(valid_inputs)

        gathered_log_probs = log_probs.gather(1, valid_targets.unsqueeze(1)).squeeze(1)

        probs = torch.exp(gathered_log_probs)

        alpha_t = self.alpha[valid_targets]

        focal_loss = alpha_t * torch.pow(1 - probs, self.gamma) * (-gathered_log_probs)

        return focal_loss.mean()

def compute_span_f1(true_spans, pred_spans):
    """Compute span-level precision, recall, and F1 using overlap criterion"""
    true_spans = set(true_spans)
    pred_spans = set(pred_spans)

    if not true_spans and not pred_spans:
        return 1.0, 1.0, 1.0
    if not true_spans:
        return 0.0, 1.0, 0.0
    if not pred_spans:
        return 1.0, 0.0, 0.0

    tp = 0
    for p_span in pred_spans:
        for t_span in true_spans:
            if max(p_span[0], t_span[0]) < min(p_span[1], t_span[1]):
                tp += 1
                break

    precision = tp / len(pred_spans) if pred_spans else 0.0
    recall = tp / len(true_spans) if true_spans else 0.0
    tp_for_recall = 0
    for t_span in true_spans:
        for p_span in pred_spans:
             if max(p_span[0], t_span[0]) < min(p_span[1], t_span[1]):
                tp_for_recall += 1
                break

    recall = tp_for_recall / len(true_spans) if true_spans else 0.0

    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

    return precision, recall, f1

def load_data(file_path):
    """Load data from parquet file"""
    df = pd.read_parquet(file_path)
```

```
    return df

def process_numpy_trigger_words(trigger_array):
    """Convert numpy array of trigger words to list of tuples"""
    if not isinstance(trigger_array, np.ndarray) or trigger_array.ndim == 0 or
trigger_array.size == 0:
        return []

    result = []
    if trigger_array.ndim == 2 and trigger_array.shape[1] == 2:
        for arr in trigger_array:
            if np.issubdtype(arr.dtype, np.number) and len(arr) == 2:
                try:
                     start, end = int(arr[0]), int(arr[1])
                     if start < end :
                         result.append((start, end))
                except (ValueError, TypeError):
                    continue
    elif trigger_array.ndim == 1 and trigger_array.dtype == 'object':
         for item in trigger_array:
             if isinstance(item, (list, tuple, np.ndarray)) and len(item) == 2:
                 try:
                     start, end = int(item[0]), int(item[1])
                     if start < end:
                          result.append((start, end))
                 except (ValueError, TypeError):
                      continue
    return result

def align_tokens_and_spans(tokenizer, text, spans, max_length=MAX_LEN):
    """
    Map character-level spans to token-level spans using BIO tagging scheme.
    Returns token_ids and token-level labels (0=Outside, 1=Beginning, 2=Inside).
    Handles cases with no spans (all labels become 0).
    """
    encoded = tokenizer(
        text,
        return_offsets_mapping=True,
        add_special_tokens=True,
        truncation=True,
        max_length=max_length,
        padding=False
    )
    input_ids = encoded["input_ids"]
    offset_mapping = encoded["offset_mapping"]

    labels = [0] * len(input_ids)

    if spans:
        sorted_spans = sorted([s for s in spans if s[0] < s[1]], key=lambda x: x[0])

        span_idx = 0
        for i, (start, end) in enumerate(offset_mapping):
            if start == end == 0:
                labels[i] = -100
                continue

            token_label = 0

            while span_idx < len(sorted_spans) and sorted_spans[span_idx][1] <= start:
```

```
                span_idx += 1

            token_overlaps = False
            for k in range(span_idx, len(sorted_spans)):
                 span_start, span_end = sorted_spans[k]
                 if span_start >= end:
                     break

                 if max(start, span_start) < min(end, span_end):
                     token_overlaps = True
                     is_begin = False
                     if start >= span_start:
                        if start == span_start:
                            is_begin = True
                        if i > 0 and labels[i-1] == 0 :
                             prev_start, prev_end = offset_mapping[i-1]
                             if prev_end <= span_start:
                                 is_begin = True

                     char_tags = ['O'] * (end + 1)
                     try:
                         if span_start < len(char_tags): char_tags[span_start] = 'B'
                         for   char_i   in   range(span_start   +   1,   min(span_end,
len(char_tags))):
                             char_tags[char_i] = 'I'
                     except IndexError: pass

                     token_char_tags = char_tags[start:end]
                     if 'B' in token_char_tags:
                         token_label = 1
                     elif 'I' in token_char_tags:
                         token_label = 2
                     break

            labels[i] = token_label

            if token_label == 1 and i + 1 < len(labels):
                 next_start, next_end = offset_mapping[i+1]
                 if next_start == end:
                      for k in range(span_idx, len(sorted_spans)):
                          span_start, span_end = sorted_spans[k]
                          if span_start >= next_end: break
                          if max(next_start, span_start) < min(next_end, span_end):
                              if labels[i+1] == 0:
                                  labels[i+1] = 2
                              break

    return {
        "input_ids": input_ids,
        "attention_mask": encoded["attention_mask"],
        "labels": labels,
        "offset_mapping": offset_mapping,
        "text": text,

    }

class ManipulationSpanDataset(Dataset):
    def __init__(self, texts, spans, tokenizer, max_len=MAX_LEN):
        self.texts = texts
        self.spans = spans
```

```
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.encodings = []

        print(f"Tokenizing and aligning spans for {len(texts)} examples...")
        for text, span_list in tqdm(zip(texts, spans), total=len(texts), desc="Processing
dataset"):
             if not isinstance(span_list, list):
                 span_list = []
             processed = align_tokens_and_spans(tokenizer, text, span_list, max_len)
             self.encodings.append(processed)

    def __len__(self):
        return len(self.encodings)

    def __getitem__(self, idx):
        encoding = self.encodings[idx]
        item = {
             "input_ids": torch.tensor(encoding["input_ids"], dtype=torch.long),
             "attention_mask": torch.tensor(encoding["attention_mask"], dtype=torch.long),
             "labels": torch.tensor(encoding["labels"], dtype=torch.long)
        }
        item["offset_mapping"] = encoding["offset_mapping"]
        item["text"] = encoding["text"]
        return item

def collate_fn(batch):
    max_len = max([len(item["input_ids"]) for item in batch])
    tokenizer_pad_token_id = AutoTokenizer.from_pretrained(MODEL_NAME).pad_token_id

    input_ids_padded = []
    attention_mask_padded = []
    labels_padded = []
    offset_mappings = []
    texts = []

    for item in batch:
        padding_len = max_len - len(item["input_ids"])

        input_ids = torch.cat([item["input_ids"], torch.tensor([tokenizer_pad_token_id] *
padding_len, dtype=torch.long)])
        attention_mask   =   torch.cat([item["attention_mask"],   torch.zeros(padding_len,
dtype=torch.long)])
        labels   =   torch.cat([item["labels"],   torch.tensor([-100]   *   padding_len,
dtype=torch.long)])

        input_ids_padded.append(input_ids)
        attention_mask_padded.append(attention_mask)
        labels_padded.append(labels)
        offset_mappings.append(item["offset_mapping"] + [(0, 0)] * padding_len)
        texts.append(item["text"])


    return {
        "input_ids": torch.stack(input_ids_padded),
        "attention_mask": torch.stack(attention_mask_padded),
        "labels": torch.stack(labels_padded),
        "offset_mappings": offset_mappings,
        "texts": texts,
```

```
    }

def tokens_to_char_spans(tokenizer, text, token_preds, offset_mapping, merge_distance=1):
    """Convert BIO token-level predictions to character-level spans with merging"""
    char_preds = []
    current_span = None

    for i, (start, end) in enumerate(offset_mapping):
        if start == end == 0:
            continue
        if i >= len(token_preds):
             continue

        pred = token_preds[i]

        if pred == 1:
            if current_span is not None:
                char_preds.append(tuple(current_span))
            current_span = [start, end]
        elif pred == 2:
            if current_span is not None:
                if start >= current_span[0]:
                     current_span[1] = max(current_span[1], end)
                else:
                     char_preds.append(tuple(current_span))
                     current_span = [start, end]

            else:
                current_span = [start, end]
        elif pred == 0:
            if current_span is not None:
                char_preds.append(tuple(current_span))
            current_span = None

    if current_span is not None:
        char_preds.append(tuple(current_span))

    char_preds = [span for span in char_preds if span[0] < span[1]]
    if not char_preds: return []

    char_preds.sort(key=lambda x: x[0])

    if len(char_preds) > 1:
        merged_spans = [char_preds[0]]
        for span in char_preds[1:]:
            prev_span = merged_spans[-1]
            if span[0] - prev_span[1] <= merge_distance:
                merged_spans[-1] = (prev_span[0], max(prev_span[1], span[1]))
            else:
                merged_spans.append(span)
        char_preds = merged_spans

    return char_preds

def get_optimizer_grouped_parameters(
    model, learning_rate, weight_decay, layerwise_lr_decay_rate
):
    """
    Groups parameters for applying Layer-wise Learning Rate Decay (LLRD).
    Assigns different learning rates and weight decay to different parts of the model.
```

```
    """
    no_decay = ["bias", "LayerNorm.weight"]
    model_prefix = model.base_model_prefix

    encoder = getattr(model, model_prefix).encoder
    layers = encoder.layer

    num_layers = len(layers)
    print(f"Applying LLRD with rate {layerwise_lr_decay_rate} over {num_layers} layers.")

    optimizer_grouped_parameters = [
        {
            "params": [
                p for n, p in model.named_parameters() if "classifier" in n or "pooler" in
n
            ],
            "weight_decay": 0.0,
            "lr": learning_rate,
        },
    ]

    for i, layer in enumerate(layers):
        lr_scale = layerwise_lr_decay_rate ** (num_layers - 1 - i)
        layer_lr = learning_rate * lr_scale

        optimizer_grouped_parameters += [
            {
                "params": [
                    p for n, p in layer.named_parameters() if not any(nd in n for nd in
no_decay)
                ],
                "weight_decay": weight_decay,
                "lr": layer_lr,
            },
            {
                "params": [
                    p for n, p in layer.named_parameters() if any(nd in n for nd in no_decay)
                ],
                "weight_decay": 0.0,
                "lr": layer_lr,
            },
        ]

    embeddings = getattr(model, model_prefix).embeddings
    embeddings_lr_scale = layerwise_lr_decay_rate ** num_layers
    embeddings_lr = learning_rate * embeddings_lr_scale

    optimizer_grouped_parameters += [
        {
            "params": [
                p for n, p in embeddings.named_parameters() if not any(nd in n for nd in
no_decay)
            ],
            "weight_decay": weight_decay,
            "lr": embeddings_lr,
        },
        {
            "params": [
                p for n, p in embeddings.named_parameters() if any(nd in n for nd in
no_decay)
```

```
            ],
            "weight_decay": 0.0,
            "lr": embeddings_lr,
        },
    ]

    all_param_names = {n for n, p in model.named_parameters()}
    grouped_param_names = set()
    for group in optimizer_grouped_parameters:
        for param in group["params"]:
             for n, p in model.named_parameters():
                 if p is param:
                     grouped_param_names.add(n)
                     break
    assert all_param_names == grouped_param_names, "Not all parameters were assigned to optimizer groups!"

    return optimizer_grouped_parameters

def train_model(model, train_dataloader, val_dataloader, optimizer, scheduler, device, epochs, tokenizer, patience=PATIENCE, accumulation_steps=ACCUMULATION_STEPS, span_merge_distance=SPAN_MERGE_DISTANCE):
    best_val_f1 = 0.0
    best_model_state = None
    early_stop_counter = 0

    criterion = WeightedFocalLoss(alpha=[0.1, 0.45, 0.45], gamma=2.0, ignore_index=-100).to(device)

    num_train_steps = len(train_dataloader) // accumulation_steps * epochs

    global_step = 0
    for epoch in range(epochs):
        print(f"\n--- Epoch {epoch+1}/{epochs} ---")

        model.train()
        total_train_loss = 0
        all_train_preds_spans = []
        all_train_true_spans = []


        train_pbar = tqdm(train_dataloader, desc=f"Training Epoch {epoch+1}", leave=False)
        optimizer.zero_grad()

        for step, batch in enumerate(train_pbar):
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)
            offset_mappings = batch["offset_mappings"]
            texts = batch["texts"]

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            loss = criterion(logits.view(-1, model.config.num_labels), labels.view(-1))

            loss = loss / accumulation_steps
            total_train_loss += loss.item() * accumulation_steps

            loss.backward()
```

```
            if (step + 1) % accumulation_steps == 0 or (step + 1) == len(train_dataloader):
                torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
                optimizer.step()
                scheduler.step()
                optimizer.zero_grad()
                global_step += 1

            token_predictions = torch.argmax(logits, dim=2).cpu().numpy()

            for i in range(len(texts)):
                pred_spans = tokens_to_char_spans(
                    tokenizer,    texts[i],    token_predictions[i],    offset_mappings[i],
merge_distance=span_merge_distance
                )
                all_train_preds_spans.append(pred_spans)

                try:
                    original_item_index = train_dataloader.dataset.texts.index(texts[i])
                    true_spans = train_dataloader.dataset.spans[original_item_index]
                    all_train_true_spans.append(true_spans)
                except (ValueError, AttributeError):
                     print(f"Warning:    Could    not    find    true    spans    for    text:
{texts[i][:50]}...")
                     all_train_true_spans.append([])

            train_pbar.set_postfix({
                 "loss": f"{loss.item() * accumulation_steps:.4f}",
                 "lr": f"{scheduler.get_last_lr()[0]:.2e}"
                 })

        avg_train_loss = total_train_loss / len(train_dataloader)

        train_f1s = []
        for true, pred in zip(all_train_true_spans, all_train_preds_spans):
            _, _, f1 = compute_span_f1(true, pred)
            train_f1s.append(f1)
        avg_train_f1 = np.mean(train_f1s) if train_f1s else 0.0
        print(f"Epoch   {epoch+1}   Train   Loss:   {avg_train_loss:.4f}   |   Train   F1:
{avg_train_f1:.4f}")

        model.eval()
        total_val_loss = 0
        all_val_preds_spans = []
        all_val_true_spans = []

        val_pbar = tqdm(val_dataloader, desc=f"Validation Epoch {epoch+1}", leave=False)

        with torch.no_grad():
            for batch in val_pbar:
                input_ids = batch["input_ids"].to(device)
                attention_mask = batch["attention_mask"].to(device)
                labels = batch["labels"].to(device)
                offset_mappings = batch["offset_mappings"]
                texts = batch["texts"]

                outputs = model(input_ids=input_ids, attention_mask=attention_mask)
                logits = outputs.logits

                loss = criterion(logits.view(-1, model.config.num_labels), labels.view(-1))
```

```
                total_val_loss += loss.item()

                token_predictions = torch.argmax(logits, dim=2).cpu().numpy()
                for i in range(len(texts)):
                    pred_spans = tokens_to_char_spans(
                        tokenizer, texts[i], token_predictions[i], offset_mappings[i],
merge_distance=span_merge_distance
                    )
                    all_val_preds_spans.append(pred_spans)

                    try:
                        original_item_index = val_dataloader.dataset.texts.index(texts[i])
                        true_spans = val_dataloader.dataset.spans[original_item_index]
                        all_val_true_spans.append(true_spans)
                    except (ValueError, AttributeError):
                         print(f"Warning: Could not find true spans for validation text:
{texts[i][:50]}...")
                         all_val_true_spans.append([])

        avg_val_loss = total_val_loss / len(val_dataloader)
        val_f1s = []
        val_precisions = []
        val_recalls = []
        for true, pred in zip(all_val_true_spans, all_val_preds_spans):
            p, r, f1 = compute_span_f1(true, pred)
            val_precisions.append(p)
            val_recalls.append(r)
            val_f1s.append(f1)

        avg_val_precision = np.mean(val_precisions) if val_precisions else 0.0
        avg_val_recall = np.mean(val_recalls) if val_recalls else 0.0
        avg_val_f1 = np.mean(val_f1s) if val_f1s else 0.0

        print(f"Epoch {epoch+1} Val Loss: {avg_val_loss:.4f} | Val Precision:
{avg_val_precision:.4f} | Val Recall: {avg_val_recall:.4f} | Val F1: {avg_val_f1:.4f}")

        if avg_val_f1 > best_val_f1:
            best_val_f1 = avg_val_f1
            best_model_state = model.state_dict().copy()
            print(f"✨ New best model saved with F1: {best_val_f1:.4f}!")

            early_stop_counter = 0
        else:
            early_stop_counter += 1
            print(f"No F1 improvement ({avg_val_f1:.4f} vs best {best_val_f1:.4f}).
Counter: {early_stop_counter}/{patience}")
            if early_stop_counter >= patience:
                print(f"Early stopping triggered after {epoch+1} epochs.")
                break

    if best_model_state:
        print(f"Loading best model state with F1: {best_val_f1:.4f}")
        model.load_state_dict(best_model_state)
    else:
        print("Warning: No best model state found (e.g., validation F1 never improved).
Using model from last epoch.")

    return model
```

```
def predict_spans(model, tokenizer, texts, device, max_len=MAX_LEN, batch_size=16,
span_merge_distance=SPAN_MERGE_DISTANCE):
    model.eval()
    all_char_spans = []

    class InferenceDataset(Dataset):
        def __init__(self, texts, tokenizer, max_len):
            self.texts = texts
            self.tokenizer = tokenizer
            self.max_len = max_len

        def __len__(self):
            return len(self.texts)

        def __getitem__(self, idx):
            text = self.texts[idx]
            encoding = self.tokenizer(
                text,
                return_offsets_mapping=True,
                add_special_tokens=True,
                truncation=True,
                max_length=self.max_len,
                padding=False,
                return_tensors=None,
            )
            return {
                "text": text,
                "input_ids": encoding["input_ids"],
                "attention_mask": encoding["attention_mask"],
                "offset_mapping": encoding["offset_mapping"]
            }

    def inference_collate_fn(batch):
        texts = [item["text"] for item in batch]
        offset_mappings = [item["offset_mapping"] for item in batch]
        max_batch_len = max([len(item["input_ids"]) for item in batch])
        tokenizer_pad_token_id = tokenizer.pad_token_id

        input_ids_padded = []
        attention_mask_padded = []
        offset_mappings_padded = []

        for item in batch:
             padding_len = max_batch_len - len(item["input_ids"])
             input_ids = item["input_ids"] + [tokenizer_pad_token_id] * padding_len
             attention_mask = item["attention_mask"] + [0] * padding_len
             offset_mapping = item["offset_mapping"] + [(0, 0)] * padding_len

             input_ids_padded.append(torch.tensor(input_ids, dtype=torch.long))
             attention_mask_padded.append(torch.tensor(attention_mask, dtype=torch.long))
             offset_mappings_padded.append(offset_mapping)

        return {
            "texts": texts,
            "input_ids": torch.stack(input_ids_padded),
            "attention_mask": torch.stack(attention_mask_padded),
            "offset_mappings": offset_mappings_padded
        }

    inference_dataset = InferenceDataset(texts, tokenizer, max_len)
```

```
    inference_dataloader = DataLoader(inference_dataset, batch_size=batch_size,
shuffle=False, collate_fn=inference_collate_fn)

    print(f"Predicting spans for {len(texts)} texts with batch size {batch_size}...")
    with torch.no_grad():
        for batch in tqdm(inference_dataloader, desc="Prediction"):
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            offset_mappings_batch = batch["offset_mappings"]
            texts_batch = batch["texts"]

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            token_predictions = torch.argmax(logits, dim=2).cpu().numpy()

            for i in range(len(texts_batch)):
                valid_len = sum(attention_mask[i].cpu().numpy())
                current_offset_mapping = offset_mappings_batch[i]
                current_predictions = token_predictions[i][:len(current_offset_mapping)]

                char_spans = tokens_to_char_spans(
                    tokenizer, texts_batch[i], current_predictions,
current_offset_mapping, merge_distance=span_merge_distance
                )
                all_char_spans.append(char_spans)

    return all_char_spans

def format_spans_for_submission(spans_list):
    """Format spans list to match submission format '[(start1, end1), (start2, end2)]'"""
    if not spans_list:
        return "[]"
    return str([(int(s[0]), int(s[1])) for s in spans_list])

def main():
    print("Loading training data...")
    train_df = pd.read_parquet("/kaggle/input/manip-
dataset/data/span_detection/train.parquet")

    print("Processing trigger words...")
    train_df['trigger_words'] = train_df['trigger_words'].fillna('').apply(lambda x: x if
isinstance(x, np.ndarray) else np.array([]))
    train_df['trigger_words_processed'] =
train_df['trigger_words'].apply(process_numpy_trigger_words)

    texts = train_df['content'].tolist()
    spans = train_df['trigger_words_processed'].tolist()
    print(f"Total raw examples: {len(texts)}")

    valid_indices = [i for i, txt in enumerate(texts) if isinstance(txt, str) and
len(txt.strip()) > 0]
    texts = [texts[i] for i in valid_indices]
    spans = [spans[i] for i in valid_indices]
    print(f"Using {len(texts)} non-empty text examples for training/validation.")

    print(f"Splitting data into train/validation ({1-
TRAIN_VAL_SPLIT:.0%}/{TRAIN_VAL_SPLIT:.0%})...")

    train_texts, val_texts, train_spans, val_spans = train_test_split(
```

```
        texts, spans, test_size=TRAIN_VAL_SPLIT, random_state=SEED
    )
    print(f"Training examples: {len(train_texts)}")
    print(f"Validation examples: {len(val_texts)}")

    print("Loading tokenizer...")
    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

    print("Creating datasets (this may take a while)...")
    train_dataset = ManipulationSpanDataset(train_texts, train_spans, tokenizer,
max_len=MAX_LEN)
    val_dataset = ManipulationSpanDataset(val_texts, val_spans, tokenizer, max_len=MAX_LEN)

    print("Creating dataloaders...")
    train_dataloader = DataLoader(
        train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        collate_fn=collate_fn,
        num_workers=2
    )
    val_dataloader = DataLoader(
        val_dataset,
        batch_size=BATCH_SIZE * 2,
        shuffle=False,
        collate_fn=collate_fn,
        num_workers=2
    )

    print("Loading model...")
    config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=3)
    model = AutoModelForTokenClassification.from_pretrained(MODEL_NAME, config=config)
    model.to(device)


    print(f"Setting up AdamW optimizer with LLRD (Rate: {LLRD_RATE})...")
    optimizer_parameters = get_optimizer_grouped_parameters(
        model,
        learning_rate=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY,
        layerwise_lr_decay_rate=LLRD_RATE
    )
    optimizer = torch.optim.AdamW(optimizer_parameters, lr=LEARNING_RATE, eps=1e-8)

    print("Setting up learning rate scheduler...")
    num_update_steps_per_epoch = (len(train_dataloader) + ACCUMULATION_STEPS - 1) //
ACCUMULATION_STEPS
    total_steps = num_update_steps_per_epoch * EPOCHS
    num_warmup_steps = int(0.1 * total_steps)

    print(f"Total optimization steps: {total_steps}, Warmup steps: {num_warmup_steps}")
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=num_warmup_steps,
        num_training_steps=total_steps
    )

    print("Starting training...")
    model = train_model(
        model=model,
```

```
        train_dataloader=train_dataloader,
        val_dataloader=val_dataloader,
        optimizer=optimizer,
        scheduler=scheduler,
        device=device,
        epochs=EPOCHS,
        tokenizer=tokenizer,
        patience=PATIENCE,
        accumulation_steps=ACCUMULATION_STEPS,
        span_merge_distance=SPAN_MERGE_DISTANCE
    )

    print("Saving the fine-tuned model...")
    output_dir = "./manipulation_span_model_improved"
    model.save_pretrained(output_dir)
    tokenizer.save_pretrained(output_dir)
    print(f"Model saved to {output_dir}")

    print("Loading test data...")
    test_df = pd.read_csv("/kaggle/input/manip-dataset/data/span_detection/test.csv")
    test_texts = test_df['content'].tolist()
    test_ids = test_df['id'].tolist()

    valid_test_indices = [i for i, txt in enumerate(test_texts) if isinstance(txt, str) and
len(txt.strip()) > 0]
    test_texts_filtered = [test_texts[i] for i in valid_test_indices]
    test_ids_filtered = [test_ids[i] for i in valid_test_indices]
    print(f"Predicting on {len(test_texts_filtered)} non-empty test examples.")

    print("Making predictions on test data...")
    predictions = predict_spans(
        model=model,
        tokenizer=tokenizer,
        texts=test_texts_filtered,
        device=device,
        max_len=MAX_LEN,
        batch_size=BATCH_SIZE * 4,
        span_merge_distance=SPAN_MERGE_DISTANCE
    )

    prediction_map = {id_: spans for id_, spans in zip(test_ids_filtered, predictions)}

    print("Formatting predictions for submission...")
    submission_data = []
    for id_ in test_ids:
        spans = prediction_map.get(id_, [])
        submission_data.append({
            'id': id_,
            'trigger_words': format_spans_for_submission(spans)
        })

    submission_df = pd.DataFrame(submission_data)
    submission_df.to_csv("submission.csv", index=False)
    print("Submission file 'submission.csv' saved successfully!")

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

Код файлу *span_learning.py*:

```
import os
import numpy as np
import pandas as pd
import torch
import ast
import re
import random
from torch import nn
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import f1_score, classification_report
from transformers import (
    AutoTokenizer,
    AutoModel,
    AutoModelForSequenceClassification,
    #AdamW,
    get_linear_schedule_with_warmup,
    get_cosine_schedule_with_warmup,
    DataCollatorWithPadding
)
from torch.optim import AdamW
from tqdm.auto import tqdm
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import nltk
from nltk.corpus import stopwords
import logging
import gc

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
logger = logging.getLogger(__name__)

SEED = 42
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
np.random.seed(SEED)
random.seed(SEED)
os.environ["PYTHONHASHSEED"] = str(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

CONFIG = {
    "model_name": "xlm-roberta-large",
    "max_length": 512,
    "batch_size": 8,
    "learning_rate": 1.8e-5,
    "weight_decay": 0.01,
    "epochs": 10,
    "warmup_ratio": 0.1,
    "dropout_rate": 0.3,
    "device": torch.device("cuda" if torch.cuda.is_available() else "cpu"),
    "gradient_accumulation_steps": 4,
    "gradient_clipping": 1.0,
    "scheduler": "cosine",
    "patience": 4,
    "label_smoothing": 0.05,
    "use_focal_loss": False,
    "focal_gamma": 2.0,
    "use_weighted_loss": True,
```

```
    "use_data_augmentation": True,
    "augment_ratio": 0.2,
    "run_analysis": True,
}

TECHNIQUES = [
    'straw_man', 'appeal_to_fear', 'fud', 'bandwagon', 'whataboutism',
    'loaded_language', 'glittering_generalities', 'euphoria', 'cherry_picking', 'cliche'
]

class FocalLoss(nn.Module):
    def __init__(self, gamma=2.0, alpha=None, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.gamma = gamma
        self.alpha = alpha
        self.reduction = reduction

    def forward(self, inputs, targets):
        BCE_loss = nn.BCEWithLogitsLoss(reduction='none')(inputs, targets)
        pt = torch.exp(-BCE_loss)
        F_loss = (1-pt)**self.gamma * BCE_loss

        if self.alpha is not None:
            F_loss = self.alpha * F_loss

        if self.reduction == 'mean':
            return torch.mean(F_loss)
        elif self.reduction == 'sum':
            return torch.sum(F_loss)
        else:
            return F_loss

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = re.sub(r'https?://\S+|www\.\S+', '[URL]', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def data_augmentation(text, techniques):
    if random.random() < 0.5:
        words = text.split()
        if len(words) > 5:
            indices_to_delete = random.sample(range(len(words)), int(len(words) * 0.2))
            words = [word for i, word in enumerate(words) if i not in indices_to_delete]
            text = ' '.join(words)

    return text, techniques

def analyze_dataset(df):
    logger.info("Analyzing dataset...")
    print("Analyzing dataset...")

    technique_counts = {technique: df[technique].sum() for technique in TECHNIQUES}
    total_instances = len(df)

    logger.info(f"Total instances: {total_instances}")
    print(f"Total instances: {total_instances}")
    logger.info("Class distribution:")
    print("Class distribution:")
```

```
    for technique, count in technique_counts.items():
        percentage = (count / total_instances) * 100
        logger.info(f"{technique}: {count} instances ({percentage:.2f}%)")
        print(f"{technique}: {count} instances ({percentage:.2f}%)")

    cooccurrence = pd.DataFrame(0, index=TECHNIQUES, columns=TECHNIQUES)
    for _, row in df.iterrows():
        for i, tech1 in enumerate(TECHNIQUES):
            if row[tech1] == 1:
                for tech2 in TECHNIQUES:
                    if row[tech2] == 1:
                        cooccurrence.loc[tech1, tech2] += 1

    logger.info("Co-occurrence of techniques:")
    print("Co-occurrence of techniques:")
    for i, tech in enumerate(TECHNIQUES):
        co_techs = [(other_tech, cooccurrence.loc[tech, other_tech])
                    for other_tech in TECHNIQUES if other_tech != tech]
        co_techs = sorted(co_techs, key=lambda x: x[1], reverse=True)[:3]
        logger.info(f"{tech} frequently co-occurs with: {co_techs}")
        print(f"{tech} frequently co-occurs with: {co_techs}")

    df['text_length'] = df['content'].apply(lambda x: len(str(x).split()))

    avg_length = df['text_length'].mean()
    median_length = df['text_length'].median()
    max_length = df['text_length'].max()

    logger.info(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length},
Max={max_length}")
    print(f"Text length statistics: Avg={avg_length:.1f}, Median={median_length},
Max={max_length}")

    over_limit = (df['text_length'] > CONFIG['max_length']).sum()
    logger.info(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit}
({over_limit/len(df)*100:.2f}%)")
    print(f"Texts exceeding max_length ({CONFIG['max_length']}): {over_limit}
({over_limit/len(df)*100:.2f}%)")

    class_weights = {}
    for technique in TECHNIQUES:
        pos_samples = df[technique].sum()
        if pos_samples > 0:
            weight = total_instances / (2 * pos_samples)
            class_weights[technique] = min(weight, 10.0)
        else:
            class_weights[technique] = 1.0

    return class_weights

class ManipulationDataset(Dataset):
    def __init__(self, texts, targets=None, tokenizer=None, max_length=512, augment=False):
        self.texts = texts
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.augment = augment

    def __len__(self):
        return len(self.texts)
```

```
    def __getitem__(self, idx):
        text = str(self.texts[idx])
        text = clean_text(text)

        if self.augment and self.targets is not None and random.random() < CONFIG["augment_ratio"]:
            text, _ = data_augmentation(text, self.targets[idx])

        encoding = self.tokenizer(
            text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        item = {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten()
        }

        if self.targets is not None:
            item['targets'] = torch.tensor(self.targets[idx], dtype=torch.float)

        return item

class ManipulationClassifier(nn.Module):
    def __init__(self, model_name, num_labels):
        super(ManipulationClassifier, self).__init__()
        self.model = AutoModel.from_pretrained(model_name)

        self.dropouts = nn.ModuleList([
            nn.Dropout(CONFIG["dropout_rate"]) for _ in range(5)
        ])

        hidden_size = self.model.config.hidden_size
        self.pre_classifier = nn.Linear(hidden_size, hidden_size)
        self.activation = nn.GELU()
        self.classifier = nn.Linear(hidden_size, num_labels)

        nn.init.xavier_normal_(self.pre_classifier.weight)
        nn.init.xavier_normal_(self.classifier.weight)

    def forward(self, input_ids, attention_mask):
        outputs = self.model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )

        sequence_output = outputs[0]
        cls_output = sequence_output[:, 0, :]

        cls_output = self.pre_classifier(cls_output)
        cls_output = self.activation(cls_output)

        logits = torch.zeros(cls_output.size(0), len(TECHNIQUES)).to(cls_output.device)
        for dropout in self.dropouts:
            logits += self.classifier(dropout(cls_output))
```

```
        logits = logits / len(self.dropouts)

        return logits

def process_data(file_path):
    logger.info(f"Loading data from {file_path}")

    if file_path.endswith('.parquet'):
        df = pd.read_parquet(file_path)
    else:
        df = pd.read_csv(file_path)

    logger.info(f"Loaded {len(df)} rows")

    if 'techniques' in df.columns:
        df['techniques'] = df['techniques'].fillna("[]")

        if isinstance(df['techniques'].iloc[0], str):
            df['techniques'] = df['techniques'].apply(lambda x: ast.literal_eval(x) if
isinstance(x, str) else x)

        for technique in TECHNIQUES:
            df[technique] = df['techniques'].apply(lambda x: 1 if technique in x else 0)

    df['content'] = df['content'].apply(clean_text)

    if 'techniques' in df.columns:
        df['manipulative'] = df['techniques'].apply(lambda x: 1 if len(x) > 0 else 0)

    if 'language' not in df.columns and 'content' in df.columns:
        def detect_language(text):
            text = str(text).lower()
            ukr_chars = sum(text.count(c) for c in ['і', 'є', 'ї'])
            if ukr_chars > 0:
                return 'ukrainian'
            return 'russian'

        df['language'] = df['content'].apply(detect_language)

    return df

def prepare_dataloaders(df, tokenizer, class_weights):
    if 'manipulative' in df.columns:
        train_df, val_df = train_test_split(
            df, test_size=0.2, random_state=SEED,
            stratify=df['manipulative']
        )
    else:
        train_df, val_df = train_test_split(df, test_size=0.2, random_state=SEED)

    logger.info(f"Training set: {len(train_df)} samples")
    logger.info(f"Validation set: {len(val_df)} samples")

    train_dataset = ManipulationDataset(
        texts=train_df['content'].values,
        targets=train_df[TECHNIQUES].values,
        tokenizer=tokenizer,
        max_length=CONFIG['max_length'],
        augment=CONFIG['use_data_augmentation']
    )
```

```
    val_dataset = ManipulationDataset(
        texts=val_df['content'].values,
        targets=val_df[TECHNIQUES].values,
        tokenizer=tokenizer,
        max_length=CONFIG['max_length'],
        augment=False
    )

    train_loader = DataLoader(
        train_dataset,
        batch_size=CONFIG['batch_size'],
        shuffle=True,
        num_workers=2,
        pin_memory=True
    )

    val_loader = DataLoader(
        val_dataset,
        batch_size=CONFIG['batch_size'],
        shuffle=False,
        num_workers=2,
        pin_memory=True
    )

    return train_loader, val_loader, val_df, class_weights

def train_epoch(model, data_loader, optimizer, scheduler, device, class_weights=None):
    model.train()
    losses = []

    pos_weight_tensor = None
    if not CONFIG['use_focal_loss'] and CONFIG['use_weighted_loss'] and class_weights:
        weights = [class_weights.get(tech, 1.0) for tech in TECHNIQUES]
        pos_weight_tensor = torch.tensor(weights, dtype=torch.float).to(device)
        logger.debug(f"Using pos_weight for BCE: {pos_weight_tensor}")

    progress_bar = tqdm(data_loader, desc="Training")

    for step, batch in enumerate(progress_bar):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        targets = batch['targets'].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)

        if CONFIG['use_focal_loss']:
            criterion = FocalLoss(gamma=CONFIG['focal_gamma'])
            loss = criterion(outputs, targets)
        else:
            if pos_weight_tensor is not None:
                criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight_tensor)
            else:
                 criterion = nn.BCEWithLogitsLoss()

            if CONFIG['label_smoothing'] > 0:
                smoothed_targets = targets * (1 - CONFIG['label_smoothing']) +
(CONFIG['label_smoothing'] / 2)
                loss = criterion(outputs, smoothed_targets)
            else:
```

```
                loss = criterion(outputs, targets)

        loss = loss / CONFIG['gradient_accumulation_steps']

        loss.backward()

        if (step + 1) % CONFIG['gradient_accumulation_steps'] == 0 or (step + 1) ==
len(data_loader):
            torch.nn.utils.clip_grad_norm_(model.parameters(),
CONFIG['gradient_clipping'])

            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()

        losses.append(loss.item() * CONFIG['gradient_accumulation_steps'])

        progress_bar.set_postfix({'loss': np.mean(losses[-10:])})

    return np.mean(losses)

def find_optimal_threshold(y_true, y_pred_proba):
    thresholds = {}
    logger.info("Finding optimal thresholds...")

    for i, technique in enumerate(TECHNIQUES):
        best_f1 = 0
        best_threshold = 0.5

        true_labels = y_true[:, i]
        pred_probs = y_pred_proba[:, i]

        for threshold in np.arange(0.15, 0.85, 0.01):
            preds = (pred_probs >= threshold).astype(int)
            f1 = f1_score(true_labels, preds, zero_division=0)

            if f1 > best_f1:
                best_f1 = f1
                best_threshold = threshold

        thresholds[technique] = best_threshold

    return thresholds

def evaluate(model, data_loader, device, thresholds=None):
    model.eval()
    all_predictions_proba = []
    all_predictions = []
    all_targets = []

    with torch.no_grad():
        for batch in tqdm(data_loader, desc="Evaluating"):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            outputs_proba = torch.sigmoid(outputs).cpu().numpy()

            if thresholds is not None:
                preds = np.zeros_like(outputs_proba, dtype=int)
```

```
                for i, technique in enumerate(TECHNIQUES):
                    preds[:,       i]       =       (outputs_proba[:,       i]       >=
thresholds[technique]).astype(int)
            else:
                preds = (outputs_proba >= 0.5).astype(int)

            all_predictions_proba.extend(outputs_proba)
            all_predictions.extend(preds)

            if 'targets' in batch:
                targets = batch['targets'].cpu().numpy()
                all_targets.extend(targets)

    all_predictions_proba = np.array(all_predictions_proba)
    all_predictions = np.array(all_predictions)

    results = {}

    if len(all_targets) > 0:
        all_targets_array = np.array(all_targets)
        macro_f1 = f1_score(all_targets_array, all_predictions, average='macro')
        results["macro_f1"] = macro_f1

        class_f1_scores = f1_score(all_targets_array, all_predictions, average=None)
        class_metrics = {}
        for i, technique in enumerate(TECHNIQUES):
            class_metrics[technique] = class_f1_scores[i]

        results["class_metrics"] = class_metrics

        if thresholds is None:
            optimal_thresholds         =         find_optimal_threshold(all_targets_array,
all_predictions_proba)
            results["optimal_thresholds"] = optimal_thresholds

        results["targets"] = all_targets_array
    else:
        results["targets"] = None

    results["predictions"] = all_predictions
    results["predictions_proba"] = all_predictions_proba

    return results

def get_class_weights(df):
    class_weights = {}
    total_samples = len(df)

    for technique in TECHNIQUES:
        pos_samples = df[technique].sum()
        if pos_samples > 0:
            weight = total_samples / (2 * pos_samples)
            class_weights[technique] = min(weight, 10.0)
        else:
            class_weights[technique] = 1.0

    return class_weights

def train_model(train_path):
    logger.info(f"Using device: {CONFIG['device']}")
```

```
    print(f"Using device: {CONFIG['device']}")

    logger.info("Loading and processing data...")
    print("Loading and processing data...")
    df = process_data(train_path)

    class_weights = None
    if CONFIG['run_analysis'] or CONFIG['use_weighted_loss']:
        class_weights = analyze_dataset(df)
        if CONFIG['use_weighted_loss']:
             logger.info(f"Calculated class weights for weighted BCE: {class_weights}")
             print(f"Calculated class weights for weighted BCE: {class_weights}")

    logger.info(f"Loading tokenizer: {CONFIG['model_name']}")
    print(f"Loading tokenizer: {CONFIG['model_name']}")
    tokenizer = AutoTokenizer.from_pretrained(CONFIG['model_name'])

    train_loader, val_loader, val_df, _ = prepare_dataloaders(df, tokenizer, class_weights)

    logger.info(f"Initializing model: {CONFIG['model_name']}")
    print(f"Initializing model: {CONFIG['model_name']}")
    model = ManipulationClassifier(CONFIG['model_name'], len(TECHNIQUES))
    model.to(CONFIG['device'])

    no_decay = ['bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
        {
            'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in
no_decay) and p.requires_grad],
            'weight_decay': CONFIG['weight_decay']
        },
        {
            'params': [p for n, p in model.named_parameters() if any(nd in n for nd in
no_decay) and p.requires_grad],
            'weight_decay': 0.0
        }
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=CONFIG['learning_rate'])

    num_training_steps_per_epoch            =            len(train_loader)            //
CONFIG['gradient_accumulation_steps']
    if len(train_loader) % CONFIG['gradient_accumulation_steps'] != 0:
        num_training_steps_per_epoch += 1

    total_steps = num_training_steps_per_epoch * CONFIG['epochs']
    warmup_steps = int(total_steps * CONFIG['warmup_ratio'])

    logger.info(f"Total optimization steps: {total_steps}, Warmup steps: {warmup_steps}")
    print(f"Total optimization steps: {total_steps}, Warmup steps: {warmup_steps}")

    if CONFIG['scheduler'] == 'cosine':
        scheduler = get_cosine_schedule_with_warmup(
            optimizer,
            num_warmup_steps=warmup_steps,
            num_training_steps=total_steps
        )
    else:
        scheduler = get_linear_schedule_with_warmup(
            optimizer,
            num_warmup_steps=warmup_steps,
```

```
            num_training_steps=total_steps
        )

    best_f1 = 0
    best_epoch = 0
    patience_counter = 0
    thresholds = None

    history = {
        'train_loss': [],
        'val_f1': [],
        'class_f1': {technique: [] for technique in TECHNIQUES}
    }

    for epoch in range(CONFIG['epochs']):
        logger.info(f"\nEpoch {epoch+1}/{CONFIG['epochs']}")
        print(f"\nEpoch {epoch+1}/{CONFIG['epochs']}")

        train_loss    =    train_epoch(model,    train_loader,    optimizer,    scheduler,
CONFIG['device'], class_weights)
        logger.info(f"Training loss: {train_loss:.4f}")
        print(f"Training loss: {train_loss:.4f}")
        history['train_loss'].append(train_loss)

        val_results = evaluate(model, val_loader, CONFIG['device'], thresholds=thresholds
if thresholds else None)
        val_f1 = val_results["macro_f1"]
        class_metrics = val_results["class_metrics"]

        logger.info(f"Validation Macro F1: {val_f1:.4f}")
        print(f"Validation Macro F1: {val_f1:.4f}")
        history['val_f1'].append(val_f1)

        if "optimal_thresholds" in val_results and val_f1 > best_f1:
             thresholds = val_results["optimal_thresholds"]
             logger.info(f"Updated optimal thresholds based on best F1: {thresholds}")
             print(f"Updated optimal thresholds based on best F1: {thresholds}")

        for technique, f1 in class_metrics.items():
            logger.info(f"{technique}: {f1:.4f}")
            print(f"{technique}: {f1:.4f}")
            history['class_f1'][technique].append(f1)

        if val_f1 > best_f1:
            best_f1 = val_f1
            best_epoch = epoch
            patience_counter = 0

            torch.save({
                'model_state_dict': model.state_dict(),
                'thresholds': thresholds,
                'config': CONFIG,
                'class_metrics': class_metrics
            }, 'best_model.pt')

            logger.info("Saved best model!")
            print("Saved best model!")
        else:
            patience_counter += 1
            logger.info(f"No improvement for {patience_counter} epochs.")
```

```
            print(f"No improvement for {patience_counter} epochs.")

        if patience_counter >= CONFIG['patience']:
            logger.info(f"Early stopping triggered after {epoch+1} epochs.")
            print(f"Early stopping triggered after {epoch+1} epochs.")
            break

        torch.cuda.empty_cache()
        gc.collect()

    logger.info(f"Best validation Macro F1: {best_f1:.4f} at epoch {best_epoch+1}")
    print(f"Best validation Macro F1: {best_f1:.4f} at epoch {best_epoch+1}")

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history['train_loss'], label='Training Loss')
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history['val_f1'], label='Validation Macro F1')
    plt.title('Validation Macro F1')
    plt.xlabel('Epoch')
    plt.ylabel('F1 Score')
    plt.legend()

    plt.tight_layout()
    plt.savefig('training_history.png')
    print("Training history plot saved to 'training_history.png'")

    plt.figure(figsize=(15, 8))
    for technique in TECHNIQUES:
        plt.plot(history['class_f1'][technique], label=technique)

    plt.title('Class-wise F1 Scores')
    plt.xlabel('Epoch')
    plt.ylabel('F1 Score')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('class_f1_scores.png')
    print("Class F1 scores plot saved to 'class_f1_scores.png'")

    checkpoint = torch.load('best_model.pt')
    final_thresholds = checkpoint['thresholds']
    logger.info(f"Using thresholds from best epoch for final reporting: {final_thresholds}")
    print(f"Using thresholds from best epoch for final reporting: {final_thresholds}")

    return model, tokenizer, final_thresholds

def predict_test_data(model, tokenizer, test_file, thresholds=None):
    logger.info(f"Loading test data from {test_file}")
    test_df = process_data(test_file)

    test_dataset = ManipulationDataset(
        texts=test_df['content'].values,
        tokenizer=tokenizer,
```

```
        max_length=CONFIG['max_length']
    )

    test_loader = DataLoader(
        test_dataset,
        batch_size=CONFIG['batch_size'],
        shuffle=False,
        num_workers=2,
        pin_memory=True
    )

    results = evaluate(model, test_loader, CONFIG['device'], thresholds)
    predictions = results["predictions"]

    submission_df = pd.DataFrame()
    submission_df['id'] = test_df['id']

    for i, technique in enumerate(TECHNIQUES):
        submission_df[technique] = [pred[i] for pred in predictions]

    return submission_df

def main():
    train_path                              =                              '/kaggle/input/manip-
dataset/data/techniques_classification/train.parquet'
    test_path = '/kaggle/input/manip-dataset/data/techniques_classification/test.csv'

    logger.info("Configuration:")
    print("Configuration:")
    for key, value in CONFIG.items():
        logger.info(f"{key}: {value}")
        print(f"{key}: {value}")

    model, tokenizer, thresholds = train_model(train_path)

    checkpoint = torch.load('best_model.pt')
    model.load_state_dict(checkpoint['model_state_dict'])
    thresholds = checkpoint['thresholds']

    submission_df = predict_test_data(model, tokenizer, test_path, thresholds)

    submission_df.to_csv('submission.csv', index=False)
    logger.info("Submission file created!")
    print("Submission file created!")

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