ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ
ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «**Браузерний плагін розпізнавання фейкових новин на основі машинного навчання**»

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)

освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Богдан ЧИСТИКОВ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КНД-42
Богдан ЧИСТИКОВ
(Ім'я, ПРІЗВИЩЕ)

Керівник: к.т.н., доцент Сергій ІЩЕРЯКОВ
Науковий ступінь, вчене звання (Ім'я, ПРІЗВИЩЕ)

Рецензент: _____
Науковий ступінь, вчене звання (Ім'я, ПРІЗВИЩЕ)

Київ - 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти - «Бакалавр»

Спеціальність – «122 Комп'ютерні науки»

Освітньо-професійна програм – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ

«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чистиков Богдан Олегович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Браузерний плагін розпізнавання фейкових новин на основі машинного навчання»

Керівник кваліфікаційної роботи Сергій Іщеряков, к.т.н., доцент

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від « 24 » лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи 30 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література з питань, що пов'язанні з застосуванням машинного навчання і класифікацією новин.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

   Аналіз предметної області та визначення основних проблем фейкових новин.

   Огляд методів автоматизованого виявлення фейкових новин.

Побудова моделі для виявлення класифікаціх текстових новин

Створення API для перенесення моделі в браузер

Розробка та тестування браузерного плагіну для перевірки правдивості новин

5. Перелік ілюстративного матеріалу: презентація

6. Дата видачі завдання 25 лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наукових джерел з теми фейкових новин та методів їх виявлення	15.03.2025 р.	виконано
2	Вивчення сучасних NLP-моделей та огляд технологій (DistilBERT, Flask)	30.03.2025 р.	виконано
3	Підготовка датасету та обробка текстових даних	04.04.2025 р.	виконано
4	Навчання моделі машинного навчання для класифікації	08.04.2025 р.	виконано
5	Реалізація серверної частини API (Flask)	11.04.2025 р.	виконано
6	Розробка браузерного плагіна та інтеграція з API	18.04.2025 р.	виконано
7	Тестування моделі та розширення	06.05.2025 р.	виконано
8	Написання вступу, висновків, оформлення дипломної роботи	22.05.2025 р.	виконано
9	Підготовка презентації до захисту	24.05.2025	виконано

Здобувач вищої освіти _____ Богдан ЧИСТИКОВ
(підпис) (Ім'я, ПРІЗВИЩЕ)

Керівник
кваліфікаційної роботи _____ Сергій ІЩЕРЯКОВ
(підпис) (Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 сторінок, 23 рисунків, 1 таблиця, 20 джерел.

Мета роботи – розробка проекту для автоматичної класифікації фейкових новин на основі моделі машинного навчання DistilBERT, представлений як браузерний плагін, що дозволяє користувачам швидко визначити істинність інформації.

Об’єкт дослідження – інформаційні системи для аналізу текстового контенту новин з використанням методів обробки природньої мови і штучного інтелекту.

Предмет дослідження – засоби та методи визначання фейкових новин за допомогою нейронних мереж трансформерного типу та методи веб-інтеграції.

Короткий зміст роботи: у даній роботі було розглянута проблема фейкових новин на просторах інтернету. В ході аналізу визначено ефективні методи виявлення фейкових новин за допомогою сучасних інструментів. Ключову роль відіграє модель трансформерного типу DistilBERT, яка навчена на датасеті ISOT. Серверна частина розроблена на Flask, що реалізує REST API, та браузерний плагін для Chrome, що в свою чергу дозволяє в зручному і зрозумілому форматі продемонструвати ефективність і функціональність програми прямо на вебсторінці. Також проведено тестування моделі і оцінку за метриками. Робота містить опис технічної реалізації, результати тестування моделі та можливі варіанти її подальшої оптимізації.

КЛЮЧОВІ СЛОВА: фейкові новини, машинне навчання, обробка природної мови(NLP), DistilBERT, трансформерна модель, браузерне розширення, Flask API, класифікація тексту, Chrome плагін.

ЗМІСТ

ВСТУП

У сучасному інформаційному просторі проблема розповсюдження фейкових новин набула особливої актуальності. Особливо гостро це питання набуло значення в Україні. В реаліях військового часу в Україні ведеться інформаційна війна. Поширення масової дезінформації погано впливає на суспільну думку, політичну стабільність та рівень обізнаності населення. Основними джерелами дезінформації є медіапростір, соціальні мережі та месенджери до яких на сьогоднішній день є доступ в кожної людини. Виникає потреба в інструментах які могли б ефективно та автоматизовано відфільтрувати такі новини в реальному часі. Існуючи методи ручної перевірки малоефективні та застарілі, а деякі автоматизовані рішення не мають достатньої точності, або не інтегровані у зручний для користувача формат.

Актуальність теми формується на потребі створення доступного інструменту, який би поєднував сучасні рішення у сфері машинного навчання та обробки природної мови з можливістю миттєвої перевірки новин безпосередньо в одному з джерел їх отримання, а саме в браузері. Саме тому в цьому дипломному проекті було представлено рішення на базі моделі DistilBERT у вигляді браузерного плагіну. Таке рішення дозволяє класифікувати новини за допомогою серверного аналізу тексту через API.

Метою роботи є створення функціональної системи для виявлення фейкових новин, що поєднує модель машинного навчання з користувацьким веб-інтерфейсом у вигляді браузерного розширення для браузера.

Для досягнення мети роботи було поставлено такі завдання:

- Проаналізувати методи знаходження фейкових новин та існуючих рішень у сфері NLP;
- Знайти відповідні інструменти і модель для класифікації текстів;
- Підготувати збалансований та якісний датасет та провести навчання моделі на базі DistilBERT;

- Реалізувати серверну частину для обробки запитів для обробки вхідних даних і повернення результатів;
- Створити браузерний плагін, що взаємодіє API та демонструє результат користувачеві;
- Провести тестування та оцінку на її ефективність за відповідними метриками.

Структура роботи налічує три розділи.

У першому розділі розкрито теоретичну складову проблеми, поняття фейкових новин, принципи обробки природної мови огляд на конкретному прикладі, а саме моделі DistilBERT.

Другий розділ розповідає про реалізацію проекту: опис відповідних інструментів, підготовку даних, безпосередньо процес навчання моделі, реалізацію API та створення браузерного плагіну.

Третій розділ демонструє результати тестування, оцінку точності роботи моделі та зручність розширення, а також описано переваги, обмеження та рекомендації покращення запропонованого підходу.

1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття фейкових новин та методи виявлення фейків

У добу цифрової інформації суспільство отримує доступ до величезного обсягу повідомлень та новин у режимі реального часу. Просте разом з перевагами швидкого доступу до інформації виникла серйозна проблема в обличчі фейкових новин. Це термін, що стосується навмисного розповсюдження маніпулятивної або неправдивої інформації, яка в свою чергу подається у вигляді об'єктивної інформації від журналістів. Їх створення та поширення за мету має вплинути на погляди, рішення, або поведінку великої кількості людей які споживають таку інформацію.

Поняття фейкових новин це не щось нове для світу: ще в минулому столітті різні уряди чи пропагандистські структури перекручували інформацію для досягнення політичних або ідеологічних цілей. За появи інтернету, гаджетів, соціальних мереж, форумів та інших інформаційних ресурсів масштаби дезінформації почали поширюватися з винятковою інтенсивністю. Завдяки можливості миттєвого поширення контенту будь-ким і будь-де, фейкова інформація за ліченні хвилини може охопити мільйони людей, створюючи хибне уявлення про події та викликаючи емоційний резонанс.

Типовими прикладами фейкових новин є:

- Вигадані історії про відомих осіб або політиків.
- Фальшиві наукові відкриття та дослідження.
- Новини з вигаданими джерелами або цитатами.
- Візуальні фейки – підроблені фото або відео.

Такі новини поширюються в стилі реального журналістського матеріалу, мають привабливі заголовки, емоційне забарвлення та спрямованні на те, щоб зацікавити користувача «сенсацією» та «ексклюзивом». В результаті фейкові

новини формують викривлену картину світу, поширюють паніку, спричиняють соціальну напругу, або спричиняють поляризацію суспільства.

Ручні методи виявлення фейків

В багатьох країнах діють фактчекінгові організації, які перевіряють новини вручну. Наприклад, PolitiFact у США, у Великобританії Full Fact, StopFake в Україні. Вони працюють за принципом перевірки фактів, контактують із джерелами, порівнюють заяви з офіційними даними. Зазвичай результатом такої перевірки є публікація аналізу з посиланнями. Такий підхід є дієвим та якісним, але потребує часу, людських ресурсів та не здатних охопити всі потоки інформації.

Також варто зазначити, що частина новин класифікуються як фейк не тому, що є повністю вигаданими, а через маніпулятивне висвітлення фактів, емоційну подачу, або вирване з контексту цитування. Такі випадки ускладнюють ручну перевірку та вимагають не лише фактологічного аналізу, а й розуміння стилістики тексту.

Автоматизовані методи виявлення фейків

Для боротьби з потоком фейкових новин були розроблені автоматизовані системи їх виявлення, які умовно можна поділити на кілька основних типів: Rule-based системи, класичне машинне навчання, глибоке навчання(Deep Learning).

Rule-based системи (системи на основі правил): Rule-based підходи є примітивними і найпростішим в реалізації. Вони працюють таким чином: «якщо в тексті є певна ознака – зроби висновок». Такими ознаками можуть бути:

- слова-паразити, які часто зустрічаються в клікбейтах («шок», «сенсація»);
- надмірна пунктуація (!!!, ???);
- відсутність джерел або надто узагальнені фрази («вчені довели», «експерти кажуть»).

Система працює за заздалегідь прописаними шаблонами або правилами, які розробляє людина. Наприклад якщо в заголовку виявлено слово «шок», а в тексті немає жодного достовірного джерела, то даний матеріал може бути визнано як потенційно фейковим.

Переваги:

- простота реалізації;
- швидкість роботи;
- прозорість результату.

Недоліки:

- погано масштабується;
- легко обходить зміною формулювання;
- не враховує контекст тексту.

Загалом, rule-based більше підходять для початкового фільтрування, або як додатковий шар перевірки в складніших системах.

Класичне машинне навчання: працює за рахунок використання алгоритмів, що спроможні виявляти шаблони у даних та класифікувати новини як фейкові або правдиві. Найбільш популярними по використанню є логістична регресія, дерева рішень, дерева рішень(Decision Trees), баєсівський класифікатор(Naïve Bayes), метод опорних векторів (SVM). Для функціонування таких моделей треба попередньо перетворити текст у числову форму, зазвичай через TF-IDF або Bag of Words. Далі ці векторизовані тексти подаються на вхід класифікатору, який приймає рішення на основі статистичних залежностей.

Переваги:

- хороша інтерпритованість результатів ;
- швидке тренування на невеликих об’ємах даних ;
- простота реалізації.

Недоліки:

- слабке врахування контексту;
- потреба в ручній інженерії ознак;
- зниження ефективності на текстах з подвійним змістом.

Глибоке навчання (Deep Learning): використовує нейронні мережі для багаторівневого аналізу текстів. Такий принцип дозволяє моделі самостійно знаходити важливі ознаки у текстах, не потребуючи попереднього формулювання

ознак самостійно. Зокрема, моделі можуть працювати з векторним представленням слів, наприклад, Word2Vec, GloVe, а також з контекстуальними ембедінгами, як ELMo, BERT. Останнім часом за використанням домінують трансформерні архітектури, такі як BERT, RoBERTa, DistilBERT, які використовують механізм уваги(self-attention), для глибокого аналізу даних. Саме трансформери дозволяють враховувати значення слів у контексті, що значно підвищує точність при класифікації новин.

Переваги:

- можливість розуміти контекст ;
- висока точність на великих наборах данних;
- універсальність для різних NLP-задач.

Недоліки:

- потреба у великих обчислювальних ресурсах;
- складнощі у налаштуванні та пояснення результатів;
- більший час тренування.

У даному підрозділі було наведено загальний огляд основних підходів до виявлення фейкових новин. Зокрема, проаналізовано ручні методи фактчекінгу, rule-based системи, класичні алгоритми машинного навчання та сучасні підходи на основі глибоких нейронних мереж. Кожен із цих підходів має свої переваги та обмеження, які визначають доцільність їх використання в залежності від завдань, обсягу даних та вимог до точності. Розуміння сильних і слабких сторін кожного з методів є важливою передумовою для побудови ефективних рішень у сфері автоматичного розпізнавання недостовірної інформації.

1.2 Обробка природної мови та огляд моделі DistilBERT

Обробка природної мови (Natural Language Processing) – це напрям у галузі штучного інтелекту, який спеціалізується на аналізі, розумінні та генерації людської мови комп’ютерними системами. Початок активного розвитку цієї галузі

пов’язують із тестом Тюрінга, де система вважається інтелектуальною якщо, якщо користувач не може відрізнити її від людини.

Сьогодні NLP активно використовується в різноманітних сферах: від голосових помічників, як Siri чи Alexa, і автоматизованих перекладачів до системи фільтрування спам-пошти та чат-ботів. Основна мета NLP технології – дати можливість комп’ютеру зрозуміти людську мову, для цього система передбачає перетворення тексту в форму, що є допустимою для машинного аналізу.

До ключових задач NLP належать:

- токенізація – поділ тексту на слова чи фрази;
- лематизація та стемінг – приведення слів до базової форми;
- визначення частин мови – граматичний розбір;
- синтаксичний та семантичний аналізи – визначення структури та змісту;
- векторизація – перетворення тексту до числового вигляду;
- класифікація, переклад, розпізнавання емоцій тощо.

Слід звернути увагу на те, що є певні складнощі в людській мові. В ній присутні багатозначні слова, граматична гнучкість та контекст. Тому прості правило-орієнтовані або статистичні підходи не завжди дають бажаний результат. Саме тут їм на зміну приходять моделі глибокого навчання, здатні навчитися на складних даних.

Важливим етапом стало векторних представлень слів – так званих ембедінгів. Такі моделі як Word2Vec, GloVe, що представляють дану технологію, дозволяють розташовувати подібні слова ближче у багатовимірному просторі. Наприклад, «король» і «королева» матимуть близькі вектори. Але такий підхід має свої обмеження – невраховування контексту, тому слово «ключ» у реченнях «ключ від квартири» і «ключ до вирішення задачі» матимуть однакове значення.

Поява контекстуальних ембендингів, такі як ELMo і BERT вирішують цю проблему. Вони, на відміну від своїх попередників, надають словам різні вектори залежно від їх оточення, простими словами враховують контекст. Особливої популярності набув BERT (Bidirectional Encoder Representations from Transformers)

- модель від Google, яка аналізує речення одночасно зліва на права і з права на ліво. Основу цієї архітектури складає алгоритм self-attention, що дає можливість моделі зосереджуватись на ключових частинах тексту.

У цьому проєкті використовується DistilBERT – полегшена версія BERT. Вона має менше параметрів, працює швидше і є зручнішою для впровадження. DistilBERT отримали через метод дистиляції знань, це коли менша модель навчається на основі результатів великої моделі, при тому зберігаючи її ефективність при меншому навантаженні. DsitilBERT має приблизно на 40% менше параметрів, але на 60% швидша за BERT і майже не поступається в точності. Як і повна версія, вона складається з:

- вхідного шару ембедінгів (перетворення тексту у вектори);
- шести трансформерних шарів (замість 12 у BERT);
- вихідного шару, який видає прогнозів (ймовірності класів).

Для задачі виявлення фейкових новин модель донавчається на наборі текстів з правдивими і фейковими прикладами. В процесі навчання вона навчається виявляти шаблони – від сенсаційної лексики до структурних особливостей текстів. Вагомим компонентом є токенізатор, який перетворює текст до того вигляду, який розуміє модель: додає спеціальні маркери, розбиває речення на елементи, створює маски для обчислень. Разом з DiastilBERT він утворює систему, яка здатна з високою точністю класифікувати новини в режимі реального часу.

Архітектура DistilBERT, як і самої BERT, побудована на базі трансформерів – іноваційного підходу до обробки послідовностей, який своєю ефективністю витіснив рекурентні нейроні мережі (RNN) та довготривалу короткочасну пам'ять (LSTM), які в свою чергу не здатні обробляти інформацію паралельно та мають обмеження з обсягами даних. Головна відмінність трансформерів полягає у відмові від послідовною обробки даних на користь механізму самоуваги self-attention. Це дозволяє моделі паралельно аналізувати кожне слово залежно від інших слів в контексті речення. Self-attention працює так, що кожне слово в тексті оцінює важливість інших слів для свого значення. Наприклад, у фразі «Він загубив ключ

від квартири» модель розуміє, що слово «ключ» пов’язане з нерухомістю завдяки слову «квартира». Цей підхід забезпечує гнучкість і точність при інтерпретації складних мовних конструкцій. Крім того, трансформери використовують позиційне кодування – додаткову інформацію про порядок слів, що компенсує відсутність рекурентності. Це дозволяє зберегти послідовність фраз, не порушуючи логіку мовлення. Завдяки таким інженерним рішенням DistilBERT може не лише визначити основний зміст тексту, але й виявляти приховані сенси, інтонаційні відтінки та логічні взаємозв’язки, що надзвичайно важливо у розпізнаванні фекових новин.

Для кращого розуміння того як DistilBERT співвідноситься з іншими моделями обробки мови, доцільно розглянути їх характеристики. Це дозволить оцінити сильні і слабкі сторони різних підходів, що були попередниками трансформерів, а також підкреслити переваги обраної архітектури.

Нижче в таблиця 1.1 наведено порівняння найпоширеніших моделей векторизації та обробки природної мови.

Таблиця 1.1 Порівняння найпоширеніших моделей векторизації та обробки природної мови

Модель	Word2Vec	GloVe	ELMo	BERT	DistilBERT
Тип представлення	Статичне	Статичне	Контекстуальне	Контекстуальний	Контекстуальний
Врахування контексту	Ні	Ні	Так	Так	Так

Продовження таблиці 1.1

Розмір моделі	Невеликий	Невеликий	Середній	Велика	Середній
Швидкість	Висока	Висока	Середня	Повільна	Висока
Точність	Середня	Середня	Висока	Висока	Висока

Таким чином, DistilBERT чудово комбінує в собі точність, швидкість та адаптивність. Її використання дозволить ефективно застосувати NLP у реальних прикладних задачах, таких як боротьба з фейками, без потреби в надмірних ресурсах.

1.3 Постановка завдання та проектування

Після аналізу природи фейкових новин, методів їх виявлення та огляду сучасних інструментів обробки природної мови, можна перейти до формулювання задачі, які розв'язує система. Постановка задачі є одним з основоположних етапів розробки будь-якого програмного рішення, оскільки саме на цьому етапі визначається, які проблему вирішує система, які її функціональні вимоги, обмеження та очікуваний результат.

Основна мета розробки полягає у створенні програмного засобу, який би дозволив автоматично класифікувати новинний текст як правдивий, або фейковий. Ключовою особливістю цієї задачі є те, що система повинна працювати у режимі реального часу, мати простий і зрозумілий інтерфейс для користувача, та високий рівень точності. Це обумовлює вибір архітектури та інструментів, які здатні забезпечити якість і швидкість.

Функціонально система має складатися з трьох основний компонентів:

- Користувацький інтерфейс (Браузерне розширення) – дозволяє користувачу перевіряти текст новин безпосередньо у браузері;

- Серверна частина (API на Flask) – обробляє запити від розширення, взаємодіє з моделю машинного навчання;
- Модель машинного навчання (DistilBERT) – проводить класифікацію тексту на правдивий або фейковий.

Система має відповідати таким функціональним вимогам:

- Прийом тексту новин через інтерфейс;
- Передача тексту на сервер через HTTP-запит;
- Обробка тексту токенізатором;
- Передбачення класу за допомогою моделі;
- Повернення результату користувачу у зручному вигляді.

До не функціональних вимог відносяться:

- Швидкодія – загальний час обробки запиту не повинен перевищувати 2-3 секунди;
- Масштабованість - можливість розгортання на інших пристроях;
- Інтерпритованість – результат має бути зрозумілий користувачу.

На основі цих вимог будується архітектура рішення. Текст, виділений користувачем у браузері, передається через розширення до Flask серверу, де проходить через токенізатор DIstilBERT. Після цього модель обчислює ймовірність відношення тексту до того чи іншого класу і повертає результат користувачеві у вигляді назви класу до якого вона віднесла виділений текст (рис. 1.1).

Рисунок 1.1 – Блок-схема послідовності взаємодії компонентів

Цей процес можна розглядати як єдину інтерактивну систему перевірки інформації, яка реагує на дії користувача у реальному часі та видає результат за кілька секунд. Архітектура систему ґрунтується на принципах розподілу відповідальностей: кожен компонент виконує призначену йому роль. Такий підхід спрощує тестування, підтримку, а також забезпечує можливість майбутнього розширення функціоналу. Наприклад, багатомовної підтримки або історії перевірок.

2 РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Вибір інструментів розробки (Python, Flask, Chrome API)

Проект створення рішення для автоматизованого виявлення фекових новин потребує певних вимог до інструментів для його реалізіції, які пов’язанні як із специфікою задачі обробки природної мови, так і з особливостями практичного використання. На початковому етапі планування архітектури проекту було визначено декілька основних функціональних аспектів, які потребують технічної реалізації: машинне навчання, побудова програмного інтерфейсу для взаємодії з користувачем, створення серверної частини для взаємодії з моделлю, а також забезпечення зрозумілого і зручного користувачу інтерфейсу.

Кожна з цих функціональних частин висуває свої вимоги на вибір інструментів. Для розв’язання задачі класифікації природньої мови необхідна підтримка сучасних трансформерів, зокрема BERT, або його спрощена версія. З цього випливає потреба в бібліотеках, здатних ефективно працювати з великими обсягами текстових вхідних даних, надавати готовий інтерфейс до готових моделей, мати гнучке API для донавчання, збереження й використання цих моделей.

Крім того, оскільки система призначена для безпосереднього використання користувачем у браузері, важливо забезпечити швидкість, зрозумілий результат і мінімальну кількість дій з боку користувача. Таким чином, було сформовано потребу в реалізації клієнтської частини, яка не потребує додаткового програмного забезпечення, а працює напряму з веб-контентом. Тож було обрано розширення для браузера, як найбільш доцільного формату розгортання користувацького інтерфейсу.

Також важливим фактом стало забезпечення маштабованості та зручності: продукт має легко адаптуватися до змін – наприклад, додати нові мови, нові моделі,

або змінити логіку взаємодії між модулями. Тому всі інструменти мали добре документованими й загально вживаними і перевіреними спільнотою. Важливо умовою було й те, що система мала запускатися на локальному комп’ютері без складних налаштувань або використання сторонніх хмарних сервісів.

Таким чином, основними критеріями вибору інструментів стали:

- Підтримка сучасних моделей глибокого навчання для NLP;
- Можливість легко реалізувати REST API;
- Сумісність з вебтехнологіями.

Серверна частина**.** Щоб організувати зв'язок між користувачем і моделлю машинного навчання, було вирішено реалізувати простий сервер, який обробляє запити, передає текст моделі, отримує прогноз і повертає результат назад. Для цього ідеально підійшов Python – мова, яка давно стала стандартом у сфері Data Science. Вона підтримує роботу з нейронними мережами, має простий синтаксис і величезну популярність. У якості фреймворку було обрано Flask – легкий і зрозумілий інструмент для створення веб-сервісів. На відміну більш від складних варіантів, таких як Django, не нав’язує жорсткої структури проєкту, що дає більше гнучкості. Це особливо зручно для невеликих проєктів, де головне швидко реалізувати API та перевірити працездатність системи. В Flask достатньо кількох рядків коду, щоб створити маршрут, прийняти POST-запит з текстом, передати його у модель і повернути назад у вигляді JSON-об'єкта. Серед інших переваг обраного фреймворку можливість легко відлагоджувати код, зручна робота з CORS(Cross Origin Resource Sharing), а також сумістність з багатьма стороніми бібліотеками, які використовуються в NLP. Таким чином всі перераховані переваги дозволяють швидко реалізувати необхідний функціонал без зайвих труднощів. Серверна частина працює автономно, може запускатися як локально, або бути розгорнутою на хостингі і за бажання, легко маштабуватися.

Модель та бібліотека машинного навчання. Для автоматичного класифікування новин як фейкових або реальних використано трансформерну модель – сучасних метод у сфері обробки текстів. Після порівняння різних

варіантів було обрано DsitilBERT – це спрощена, але потужна версія BERT, яка зберігає високу ефективність при більшій швидкості. Тобто, рішення чудово підходить для проєкту де важлива збалансованість між швидкістю і продуктивністю. Щоб не писати все з нуля, використано популярну бібліотеку Transformers від компанії Hugging Face. Вона дозволяє в кілька рядків коду підключити вже натреновану модель, налаштувати її під свої задачі та зберегти для подальшого користування. Це значно пришвидшує розробку, дає доступ до широкого вибору моделей та відкриває можливість легко її оновлювати.

Основою для роботи моделі є фреймворк PyTorch, який використовується в середині моделі DistilBERT. PyTorch став стандартом для розробки моделей глибокого навчання завдяки простому коду, дебагінгу та підтримці роботи з GPU. Саме PyTorch дозволяє швидко адаптувати адаптувати модель під власний датасет, керевати процесом навчання та обчислювати метрики.

Переваги такого підходу є те що модель не є вшитою в код – її можна змінювати незалежно, перенавчати, або замінити на іншу і це не вплине на іншу логіку. Бібліотека Transformers активно підтримується розробниками, має документацію, приклади коду і постійно оновлюється, що робить її надійною основою.

Інструменти підготовки та обробки даних**.** Перед початком тренування будь-якої моделі треба провести етап підготовки даних. Для цього використовуються певні інструменти, які є частиною Python-екосистеми. Вибір цих бібліотек обґрунтовано їхньою простотою, потужністю та широким використанням у сфері Data Science. Перш за все це Pandas – бібліотека, що дозволяє зручно працювати з табличними даними у вигляді DataFrame. Вона підтримує імпорт CSV-файлів, фільтрацію сортування, об’єднання таблиць, виявлення пропусків, видалення дублікатів і базовий статистичний аналіз. У рамках цього проєкту Pandasвикористовувався для завантаження датасету новин, додавання міток, а також об’єднання окремих файлів у єдиний масив даних.

Доповненням до Pandas виступає NumPy – бібліотека для роботи з числовими масивами. Вона використовується для оптимізації обчислень, формування структур масивів і виконання базових математичних операцій. Хоча її застосування в цьому проєкті було обмеженим, вона залишається базовою залежністю для інших бібліотек, таких як Pandas.

Для поділу датасету на тренувальні та тестові вибірки, а також для оцінювання результатів використовувалась бібліотека Scikit-learn. Вона дозволяє легко розділити дані з урахуванням пропорцій класів, забезпечити випадковість і не втрачати баланс між фейковими й реальними новинами. Крім того вона буде використана для обчислення метрик, таких як точність, повнота та F1-міра на етапі тестування моделі.

Ці три бібліотеки – Pandas, NumPy, Scikit-learn – є основою попередньої обробки в більшості проєктів машинного навчання. Вони не тільки полегшують роботу з даними, а й дозволяють зосередитись на побудові моделі, не втрачаючи час на реалізацію базових функцій вручну. Також важливо що всі описані бібліотеки мають сумісність з іншими компонентами продукту, як на етапі підготовки даних, так і при інтеграції пайплайн навчання. Це допомагає уникнути несумісностей та прискорити розробку.

Робоче середовище. Щоб уникнути технічних проблем під час розробки, а також зробити налаштування зручнішим у проєкті використовується Anaconda. Це інструмент який допомагає встановлювати потрібні бібліотеки, створювати ізольовані середовища для різних проєктів та уникати конфліктів між пакетами. На практиці це дозволяє уникнути ситуацій коли одна бібліотека потребує одну версію Python, а інша зовсім іншу. В процесі розробки було створено окреме середовище, до якого вручну додано всі необхідні компоненти: trsnsformets, torch, flask, scikit-learn, pandas, numpy та інші. Це дозволило працювати стабільно, уникнувши помилок через несумісність, або неправильні версії інструментів.

Ще одна перевага Anaconda – можливість працювати через зручний графічний інтерфейс Anaconda Navigator. Через нього можна запускати Jupyter

Notebook, курувати середовищами та бачити список встановлених пакетів без використання командної рядка. Це особливо комфортно коли треба багато разів змінювати середовище, оновлювати бібліотеки або відкривати проєкт у зручному інтерфейсі.

Також активно використовувався Jupyter Notebook – інструмент для інтерактивної роботи з кодом. В ньому зручно писати, виконувати і документувати код частинами. Він ідеально підійшов для роботи з текстами, побудови графіків, тестування токенізації та швидкої перевірки моделі. Його перевагою є те, що результати одразу видно під кожним фрагментом коду, а також можливість коментувати та структурувати весь процес роботи.

Завдяки цьому робоче середовище було повністю контрольоване, гнучким та зручним у використанні. Це дозволило зосередитись на реалізації самої системи, а не витрачати час на усунення постійних конфліктів у пакетах чи налаштування бібліотек.

Фронтенд і веб-технології. Оскільки основна мета проєкту – зробити перевірку новин максимально доступною, інтерфейс реалізовувався у вигляді браузерного розширення для Google Chrome. Цей підхід дозволяє взаємодіяти з новинами безпосередньо у браузері – там де користувач їх читає. На відміну від десктопних або мобільних застосунків, плагін не потребує встановлення складного ПЗ і може працювати в один клік.

Інтерфейс створено з використанням класичних веб технологій – HTML, CSS, JavaScript (рис. 2.1).

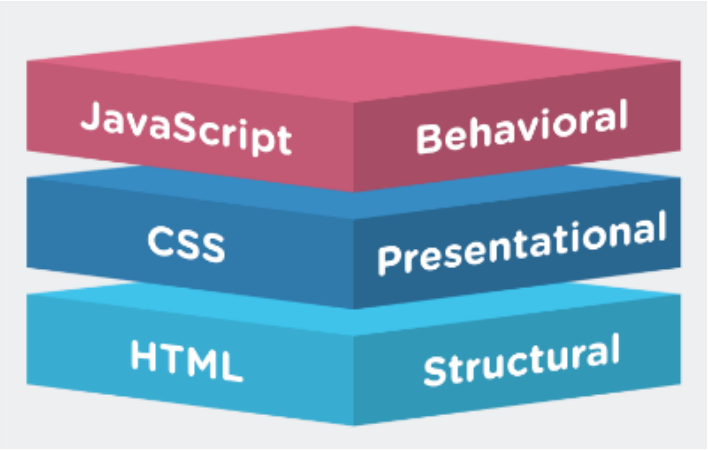


Рисунок 2.1 – Фундаментальні технології веб-розробки

HTML відповідає за структуру сторінки: текстове поле, кнопки, заголовки. CSS оформлює зовнішній вигляд, роблячи розширення зручним і візуально привабливим для користувача. JavaScript забезпечує динаміку: обробку натискань кнопок, зчитування виділеного тексту зі сторінки та надсилання HTTP-запиту на сервер. JavaScript у поєднанні з можливостями Chore Extensions API використовується для створення логіки роботи розширення. Завдяки цьому можна керувати подіями в інтерфейсі, взаємодіяти з контентом вебсторінки, здійснювати HTTP-запит до серверної частини і динамічно змінювати зміст розширення. Крім зручності, розширення легко оновлювати: всі файли зберігаються локально, а логіка взаємодії з сервером гнучко налаштовується через JavaScript. При бажанні, плагін можна доповнити новими функціями. Такий підхід дозволяє створювати швидкі та інтерактивні інтерфейси без додаткових інструментів.

2.2 Підготовка і обробка даних

Одним з ключових етапів побудови системи на основі машинного навчання є правильна підготовка даних. Від якості, структури та чистоти вхідної інформації залежить здатність моделі коректно навчитися, узагальнювати та приймати точні рішення. Для спрощення роботи було обрано готовий датасет новин, який містить як фейкові, так і реальні кести новин. На цьому етапі проведено об’єднання даних,

їх очищення, розмітка а також розподіл на тренувальні та валідаційні дані. Ці дії створили фундамент для подальшого навчання моделі DistilBERT і дозволили забезпечити коректність процесу класифікації.

Вибір датасету. Для навчання моделі класифікації новин обрано ISOT Fake and REAl News Dataset – один з найбільш поширених і структурованих наборів даних для задач виявлення недостовірної інформації. Цей датасет розміщений на платформі Kaggle, є видкритим для використання в дослідницьких цілях і часто використовується в наукових роботах, що підтверджує його якість. ISOT Dataset було зібрано дослідниками з Університету Вікторії(Канада). Він містить два окремих CSV-файли – Fake.csv і Real.csv, які відповідно до назв представляють збірки справжніх і фейкових новин. У кожному файлі присутні повні тексти новинних статей англійською мовою, а також допоміжні поля, такі як заголовки дати, публікацій і теми (рис. 2.2 та рис 2.3).

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017
5	White House, Congress prepare for talks on spe...	WEST PALM BEACH, Fla./WASHINGTON (Reuters) - T...	politicsNews	December 29, 2017
6	Trump says Russia probe will be fair, but time...	WEST PALM BEACH, Fla (Reuters) - President Don...	politicsNews	December 29, 2017
7	Factbox: Trump on Twitter (Dec 29) - Approval ...	The following statements were posted to the ve...	politicsNews	December 29, 2017
8	Trump on Twitter (Dec 28) - Global Warming	The following statements were posted to the ve...	politicsNews	December 29, 2017
9	Alabama official to certify Senator-elect Jone...	WASHINGTON (Reuters) - Alabama Secretary of St...	politicsNews	December 28, 2017

Рисунок 2.2 – Перші десять рядків файлу True.csv

	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017
5	Racist Alabama Cops Brutalize Black Boy While...	The number of cases of cops brutalizing and ki...	News	December 25, 2017
6	Fresh Off The Golf Course, Trump Lashes Out A...	Donald Trump spent a good portion of his day a...	News	December 23, 2017
7	Trump Said Some INSANELY Racist Stuff Inside ...	In the wake of yet another court decision that...	News	December 23, 2017
8	Former CIA Director Slams Trump Over UN Bully...	Many people have raised the alarm regarding th...	News	December 22, 2017
9	WATCH: Brand-New Pro-Trump Ad Features So Muc...	Just when you might have thought we d get a br...	News	December 21, 2017

Рисунок 2.3 – Перші десять рядків файлу Fake.csv

Основною перевагою датасету є його збалансованість – кількість реальних і фейкових новин приблизно однакова (реальні - 21 417, фейкові – 23 481), що зменшує ризик зміщення моделі в бік одного з класів. Також текст новин достатньо об'ємний що дозволяє моделі краще навчитися на контексті, а не лише на заголовках або окремих фразах. Ще одним аргументом на користь цього набору даних стало те, що його структура проста і зручна для попередньої обробки: дані представлені у вигляді таблиць, кожен айтем містить закінчену новину, а формат CSV дозволяє їх швидко імпортувати у середовище розробки за допомогою бібліотеки pandas. Отже, ISOT Fake and Real News Dataset було обрано як оптимальний варіант, що відповідає цілям проекту.

Завантаження датасету. Після вибору ISOT Fake and Real News Dataset, обидві частини даних — Fake.csv та True.csv — було завантажено до середовища розробки за допомогою бібліотеки pandas, яка зручна для роботи з табличними структурами (рис. 2.4).

```
df_fake = pd.read_csv("Fake.csv")
df_true = pd.read_csv("True.csv")
```

Рисунок 2.4 – Фрагмент коду завантаження датасету

Для майбутньої класифікації кожному запису була присвоєна відповідна мітка: 0 для фейкових новин, 1 для реальних (рис. 2.5).

```
df_fake["label"] = 0
df_true["label"] = 1
```

Рисунок 2.5 – присвоєння міток для файлів Fake.csv та True.csv

Далі обидва датафрейми були об’єднані у спільний набір та випадковим чином перемішані. Це дозволяє зменшити ймовірність того, що модель буде навчатися на основі чергування класів у даних. Також було видалено стовбці з не потрібною інформаціє: дата публікації та тематика (рис. 2.6).

```
df = pd.concat([df_fake, df_true]).sample(frac=1).reset_index(drop=True)
df = df[["text", "label"]]
```

Рисунок 2.6 – Об’єднання файлів в один набір даних та їх фільтрація

На виході сформувався повний датафрейм із двома головними ознаками: text (текст новини) та label (відповідна мітка) (рис. 2.7).

	text	label
0	BEIJING (Reuters) - Chinese President Xi Jinpi...	1
1	During Wednesday s Presidential Town Hall on C...	0
2	The setting for this episode is Yale Universit...	0
3	WASHINGTON (Reuters) - Aides to U.S. President...	1
4	BRUSSELS (Reuters) - While European powers Fra...	1

Рисунок 2.7 – Кінцевий вигляд датасету після попередньої обробки

Після об’єднання наборів фейкових і реальних новин у спільний датафрейм, з нього було залишено лише дві ключові ознаки — text та label. Інші колонки, такі як заголовки, теми та дати публікацій, не мали прямого впливу на точність класифікації та тому були відфільтровані. Поле text містить повний текст новини, а label вказує на її тип: 0 — фейкова, 1 — реальна.

Щоб уникнути зміщення порядку подачі даних до моделі, об'єднаний набір був випадковим чином перемішаний. Це допомагає зробити навчання більш стабільним і запобігає перенавчанню на подібних прикладах, які можуть іти підряд у сирих даних.

Розділення на тренувальні та тестові дані. Після формування єдиного датафрейму з текстами новин та їхніми відповідними мітками, наступним кроком стало розділення цього набору на тренувальну та тестову частини. Такий поділ є стандартною практикою в задачах машинного навчання, оскільки дає змогу оцінити, наскільки добре модель узагальнює знання та справляється з новими, раніше не баченими прикладами.

У даному проєкті для поділу використовувалась функція train_test_split з бібліотеки scikit-learn (Додаток А). Розмір тестової вибірки було встановлено на рівні 20% від загального обсягу даних. Це дозволяє зберегти достатню кількість прикладів для навчання, водночас маючи репрезентативний обсяг даних для оцінки ефективності моделі.

Щоб зберегти початкове співвідношення між фейковими та реальними новинами в обох вибірках, застосовувався параметр stratify (Додаток А). Це означає, що як у тренувальній, так і в тестовій вибірці кількість прикладів кожного класу приблизно однакова, що запобігає перекосу моделі в бік того класу, який був надмірно представлений.

Результатом цього етапу є два набори: тренувальний, що використовується для навчання моделі, та тестовий, який буде застосований для фінального оцінювання її точності. Кожен із них складається з текстів новин та відповідних міток.

Такий поділ даних дозволяє контролювати якість навчання, уникати перенавчання моделі та забезпечити об’єктивну перевірку її продуктивності на прикладах, які вона не бачила під час тренування.

Токенізація текстів. Після поділу даних на тренувальну та тестову вибірки, наступним етапом є підготовка текстів до подачі в модель трансформера. Оскільки моделі на зразок DistilBERT не працюють безпосередньо з необробленим текстом, а очікують числові входи, необхідно виконати процес токенізації.

Токенізація — це перетворення тексту у послідовність чисел (ідентифікаторів токенів), які відповідають словам, частинам слів або символам у словнику моделі. Для цього використовувався DistilBertTokenizerFast — вбудований токенізатор з бібліотеки Hugging Face Transformers, спеціально адаптований під структуру DistilBERT (Додаток А).

Кожен текст новини проходить кілька етапів перетворення: розбиття на токени, підбір відповідних індексів з словника моделі, додавання спеціальних службових токенів ([CLS], [SEP]), а також доповнення padding або обрізання truncation текстів до заданої максимальної довжини. Це дозволяє вирівняти довжину всіх входів до одного формату, який модель може обробити в батчі (рис. 2.8).

```
tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")
train_encodings = tokenizer(list(train_texts), truncation=True, padding=True)
test_encodings = tokenizer(list(test_texts), truncation=True, padding=True)
```

Рисунок 2.8 – Токенізація тестових та тренувальних вибірок

У результаті токенізації кожна новина перетворюється на набір таких ознак:

- input_ids — числові ідентифікатори токенів;
- attention_mask — маска, яка вказує, які елементи у вході є змістовними, а які додані автоматично (0 для паддінгу, 1 для основного тексту).

Ці об'єкти пізніше будуть об'єднані в структури, придатні для передачі в модель — і стануть основою тренувального процесу.

Процес токенізації є критично важливим, оскільки якість цього перетворення напряму впливає на здатність моделі адекватно зрозуміти зміст тексту. Зокрема, правильне використання truncation дозволяє уникнути помилок при подачі надто довгих текстів, а padding забезпечує ефективну обробку батчів однакової довжини.

Створення об’єктів DataSet. Після того як тексти новин були токенізовані та представлені у вигляді наборів input_ids і attention_mask, необхідно було сформувати структури, сумісні з механізмом тренування моделі. Для цього використовувалась бібліотека Hugging Face Datasets, яка надає зручний інтерфейс для створення об’єктів типу Dataset на основі словників або масивів даних.

Ці об’єкти є спеціальною структурою, що зберігає токенізовані приклади разом із мітками (labels) у форматі, зручному для подачі у Trainer. Формування таких об’єктів дозволяє не тільки легко організувати вхідні дані, а й забезпечує узгодженість між розміткою та ознаками, які обробляє модель (рис. 2.9).

```
train_dataset = Dataset.from_dict({
    'input_ids': train_encodings['input_ids'],
    'attention_mask': train_encodings['attention_mask'],
    'labels': list(train_labels)
})

test_dataset = Dataset.from_dict({
    'input_ids': test_encodings['input_ids'],
    'attention_mask': test_encodings['attention_mask'],
    'labels': list(test_labels)
})
```

Рисунок 2.9 – Створення об’єктів DataSet

Тренувальна та тестова вибірки були перетворені на два окремі об’єкти — train_dataset і test_dataset (Додаток А), кожен з яких містить три основні поля:

- input_ids — закодований текст у вигляді послідовності чисел;
- attention_mask — маска, яка відмічає суттєві токени у вхідному тексті;

- labels — цільова змінна (0 або 1), що відповідає класу новини.

Перевага використання формату Dataset.from_dict() полягає у тому, що він дозволяє зберігати повну структуру даних, сумісну з механізмом батчування, автоматичної передачі на GPU та обробки в рамках Trainer без додаткової обробки чи форматування.

Створення таких об’єктів є проміжною ланкою між підготовкою даних і тренуванням моделі, і дозволяє забезпечити гнучкість у роботі з великими масивами текстової інформації.

Висновок щодо готовності даних до тренування. У результаті попередньої обробки та підготовки було сформовано повноцінний набір даних, придатний для подачі в модель трансформера. Дані були завантажені, об’єднані, промаркеровані відповідними мітками, перемішані й розділені на тренувальну та тестову вибірки зі збереженням балансу класів. Подальша токенізація текстів забезпечила перетворення у формат, сумісний із моделлю DistilBERT, з урахуванням довжини послідовностей, службових токенів і маскування.

На основі токенізованих даних були створені спеціальні структури типу Dataset, які дозволяють ефективно організувати процес навчання — включаючи обробку батчами, автоматичне формування вхідних ознак та передачу даних на GPU.

Таким чином, підготовчий етап завершено, і всі компоненти готові до використання в подальшому процесі навчання моделі. Забезпечено відповідність вимогам до формату вхідних даних, обсягів, балансу класів і структурної узгодженості, що дозволяє перейти до етапу моделювання без потреби в додатковій трансформації.

2.3 Навчання моделі DistilBERT

Після виконання попередніх дій підготовки даних робота переходить до процесу – навчання моделі класифікації текстів. У межах даного проєкту було

використано попереднє натреновану модель DistilBERT, яка є оптимізованою версією базової BERT. На основі токенізованих текстів та відповідних міток класів було сформовано навчальну і тестову вибірки, які подаються на вхід моделі у структурованому форматі. Навчання здійснюється з використанням бібліотеки Hugging Face Transformers, яка дозволяє значно спростити реалізацію тренувального циклу завдяки інструменту Trainer. В межах цього підрозділу буде розглянуто імпорт та налаштування моделі, конфігурацію параметрів навчання, запуск тренування та збереження результатів для подальшого використання в серверній частині системи.

Імпорт та ініціалізація моделі. На цьому етапі розпочинається безпосередня робота з нейронною мережею. Як модель класифікації було використано попередньо натреновану архітектуру DistilBERT, адаптовану під задачу бінарної класифікації. Для її завантаження застосовано інструменти з бібліотеки Hugging Face Transformers, яка надає зручний доступ до моделей, попередньо натренованих на великих текстових даних.

Імпорт моделі відбувається за допомогою класу DistilBertForSequenceClassification (Додаток А). Він дозволяє ініціалізувати трансформерну структуру та автоматично додати необхідний класифікаційний шар на виході. Цей шар дозволяє моделі повертати логіти — числові значення, що відповідають імовірності кожного з класів.

Оскільки задача передбачає двоїсту класифікацію (фейк або правда), при створенні моделі явно вказується кількість вихідних класів. Завдяки модульності бібліотеки Hugging Face, модель можна легко завантажити з репозиторію distilbert-base-uncased та одразу використовувати для подальшого тренування.

Цей підхід дає змогу значно зекономити ресурси, оскільки основна частина ваг вже була навчена на великому обсязі текстових даних, і в межах проєкту виконується лише дофінетюнінг (донавчання) на конкретному наборі новин. Ініціалізація моделі є першим кроком до її подальшого навчання і не потребує змін

у внутрішній архітектурі — достатньо лише завантажити, налаштувати та передати вхідні дані у відповідному форматі.

Налаштування параметрів навчання. Щоб розпочати процес донавчання моделі DistilBERT потрібно задати конфігурацію ключових гіперпараметрів, які впливають на якість, стабільність та швидкість навчання. Для управління навчальним процесом був використаний клас TrainingArguments з бібліотеки Hugging Face Transformers.

Одним з основних параметрів є кількість епох – тобто кількість разів повного проходу по всьому тренувальному набору. Для проєкту було встановлено 3 епохи, що є достатнім значенням для донавчання вже претренованої моделі без ризику перенавчання.

Розмір батча — ще один важливий показник, що визначає, скільки прикладів обробляється одночасно. Було обрано розмір 8, що дозволило знайти баланс між використанням пам’яті та швидкістю тренування на стандартному обладнанні(середньому ноутбуці). Менший розмір батча також підвищує стохастичність градієнтного спуску, що іноді сприяє кращій генералізації.

Швидкість навчання (learning_rate) встановлена на рівні 3e-5 — це рекомендоване значення для трансформерів, що забезпечує стабільну зміну ваг моделі без різких стрибків. Для уникнення перенавантаження на початку тренування передбачено 100 warm-up кроків, під час яких швидкість навчання поступово збільшується (рис. 2.10).

```
training_args = TrainingArguments(
    output_dir='./model',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=100,
    learning_rate=3e-5,
    evaluation_strategy="epoch",
    logging_dir='./logs',
    logging_steps=10,
    save_strategy="epoch"
)
```

Рисунок 2.10 – Конфігурація ключових гіперпараметрів

Окрім цього, було визначено:

- evaluation_strategy = «epoch» — оцінювання моделі після кожної епохи;
- save_strategy = «epoch» — збереження моделі після кожного проходу;
- logging_steps = 10 — інтервал логування прогресу в терміналі;
- output_dir = «./model» — шлях до папки, де зберігається модель;
- logging_dir = «./logs» — каталог для збереження логів.

Завдяки гнучкій структурі TrainingArguments, усі ці налаштування були об’єднані в єдину конфігурацію, що зручно передається до тренувального модуля. Це дозволяє швидко адаптувати параметри в разі потреби — наприклад, при зміні обсягу даних або при переході на інше обладнання.

Створення об’єкту Trainer. Щоб організувати завершальний етап перед самим циклом навчання моделі, у проєкті було використано інструмент Trainer з бібліотеки Hugging Face Transformers. Це високорівневий API, який значно спрощує навчання моделей глибокого навчання, дозволяючи обійтися без ручного створення циклів тренування, розрахунку втрат та оновлення ваг (рис. 2.11).

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
```

Рисунок 2.11 – Створення об’єкту Trainer

Об’єкт Trainer поєднує в собі всі основні компоненти, необхідні для навчання:

- модель, яка має бути натренована;
- тренувальні та тестові вибірки у вигляді об'єктів Dataset;
- набір гіперпараметрів, визначених за допомогою TrainingArguments.

Завдяки цьому підходу, навчання запускається буквально кількома рядками коду, а всі рутинні операції — такі як батчинг, forward/backward пропуск, оновлення градієнтів, оцінка на валідаційному наборі — виконуються автоматично. Крім того, Trainer автоматично веде логування процесу, зберігає модель після кожної епохи та дозволяє легко візуалізувати тренувальні метрики.

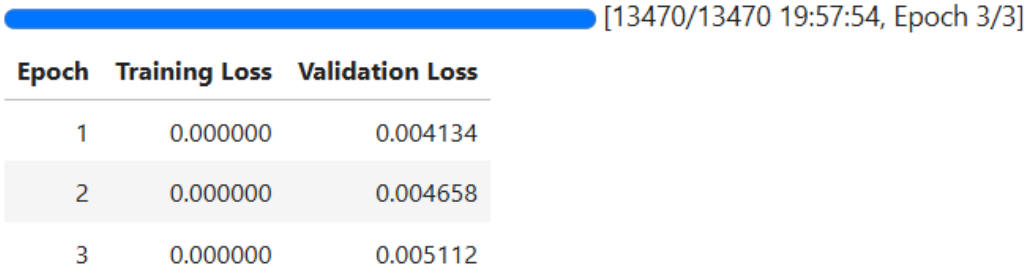
Ще однією перевагою Trainer є його гнучкість: у разі потреби можна легко розширити його функціональність, наприклад, змінивши функцію втрат, додавши власні метрики або кастомізувавши логіку навчання.

У цьому проєкті Trainer став основою запуску тренувального процесу моделі DistilBERT, з урахуванням підготовлених даних і параметрів. Його застосування дозволило зосередитися на змістовній частині — якості моделі та аналізі результатів і мінімізувати технічну складність реалізації.

Запуск навчання. Після створення об'єкта Trainer та налаштування параметрів було запущено безпосередній процес навчання моделі. Він ініціюється методом train() (Додаток А), який забезпечує послідовне проходження тренувальних даних у кілька епох. Модель обробляє вхідні тексти, обчислює втрати, оновлює свої ваги та проводить автоматичну перевірку на валідаційній вибірці.

Процес проходить батчами — невеликими підмножинами загального обсягу даних, що дозволяє економно використовувати ресурси пам’яті та забезпечує стабільне оновлення параметрів моделі. Завдяки параметру evaluation_strategy="epoch", оцінювання відбувається після кожної повної епохи, що дозволяє контролювати прогрес навчання та вчасно виявити можливі проблеми, такі як перенавчання.

Навчання проходило стабільно, без помилок або зривів градієнта, що вказує на коректність підібраних гіперпараметрів. Уже після першої епохи спостерігалося суттєве зменшення функції втрат, що свідчить про ефективність донавчання моделі на тематичному датасеті (рис. 2.12).



Epoch	Training Loss	Validation Loss
1	0.000000	0.004134
2	0.000000	0.004658
3	0.000000	0.005112

Рисунок 2.12 – Результати тренування моделі після кожної епохи

У підсумку, після трьох епох модель досягла готовності до використання на практиці — для інтеграції в API-сервіс та браузерне розширення.

Збереження моделі. Після завершення навчання важливо зберегти результати — тобто саму модель, токенізатор і конфігураційні файли, необхідні для подальшого використання. Це дозволяє уникнути повторного навчання при кожному запуску системи й забезпечити миттєвий доступ до вже підготовленої нейронної мережі.

Для збереження використовувались стандартні методи бібліотеки Hugging Face Transformers:

- save_pretrained() — для збереження моделі;
- save_pretrained() — для токенізатора.

У результаті у вказаній директорії створюється набір файлів, серед яких:

- pytorch_model.bin — власне ваги моделі;
- config.json — конфігурація моделі, включно з кількістю вихідних класів;
- tokenizer_config.json, vocab.txt, tokenizer.json — дані токенізатора, необхідні для однакового перетворення тексту при передбаченні.

Ці файли формують повністю автономний пакет, який може бути повторно завантажений для здійснення передбачення без повторного навчання. Це особливо важливо в контексті використання моделі у продакшн-середовищі — зокрема, у Flask-сервері, де модель підвантажується при запуску і використовується для обробки запитів користувача в реальному часі.

Таким чином, модель була не лише успішно натренована, а й упакована у формат, придатний для інтеграції в API-сервіс, що забезпечує її прикладне застосування для браузерного розширення.

Завдяки використанню попередньо натренованої моделі DistilBERT вдалося значно скоротити обчислювальні витрати та швидко адаптувати модель до специфіки задачі — виявлення фейкових новин. Навчання тривало всього кілька епох і не вимагало великих обсягів ресурсів, оскільки більшість мовних залежностей модель уже знала з базового корпусу. Грамотно підібрані гіперпараметри, використання механізму Trainer, а також налаштування автоматичного логування, перевірки й збереження результатів дозволили зосередитися на якісному донавчанні, не заглиблюючись у технічні деталі реалізації навчального циклу. Після завершення тренування модель було успішно збережено у форматі, який забезпечує її швидке завантаження та використання на сервері. Це дозволило інтегрувати її у прикладну частину системи — Flask API, яке своєю чергою взаємодіє з браузерним розширенням. Отже, етап навчання моделі завершено повністю, і всі необхідні компоненти підготовлено для подальшої експлуатації системи в реальному середовищі.

2.4 Розробка API сервісу на Flask

Для забезпечення інтерактивної взаємодії між користувачем та моделлю класифікації у проєкті було реалізовано окремий серверний модуль на базі Flask. Його завдання — приймати текстові запити від клієнтської частини, обробляти їх за допомогою навченої моделі та повертати результат у зручному форматі. Така архітектура дозволяє підтримувати чисте розділення логіки між інтерфейсом та обчислювальним модулем проєкту, підвищуючи гнучкість і масштабованість рішення.

Призначення API в системі виявлення фейкових новин. У структурі створеної системи API-сервіс виконує важливу роль — він виступає посередником між

клієнтською частиною (браузерним розширенням) та моделлю машинного навчання, що відповідає за аналіз тексту. Завдяки цьому підходу модель не вбудовується безпосередньо в інтерфейс користувача, а розгортається як окремий серверний модуль, до якого надсилаються запити в реальному часі.

Подібна архітектура має низку переваг. По-перше, це дає змогу розділити логіку роботи інтерфейсу та обчислювальну частину — тобто запуск самої нейронної мережі. По-друге, зміни в моделі або її оновлення не потребують втручання в код плагіна — достатньо перезапустити або оновити сервер. По-третє, така схема дозволяє масштабувати систему, наприклад, розгортаючи сервер на окремому хості або в хмарному середовищі.

У рамках проєку API реалізовано за допомогою легкого Python-фреймворку Flask. Він дозволяє швидко побудувати REST-сервіс із необхідними маршрутами та логікою обробки запитів. Основна функція цього сервісу полягає в тому, щоб отримати текст від клієнта, передати його моделі на аналіз, а потім повернути відповідь у структурованому форматі.

Таким чином, API виступає нервовим центром системи — через нього відбувається весь обмін даними між розширенням та модельною частиною, забезпечуючи інтерактивність, швидкість і розділення відповідальностей.

Структура серверного додатку. Серверна частина системи реалізована у вигляді окремого Python-файлу app.py (Додаток Б), який містить основну логіку запуску Flask-додатку, обробки запитів і взаємодії з моделлю машинного навчання. Цей файл відповідає за створення веб-сервера, налаштування маршрутів, завантаження моделі DistilBERT та обробку текстів, що надходять із клієнтської частини.

Після запуску програми створюється Flask-додаток, у якому відкрито маршрут /predict. Він дозволяє приймати POST-запити з текстовими даними у форматі JSON. На початку ініціалізації відбувається завантаження збереженої моделі та токенізатора з локальної директорії, що була сформована на етапі

навчання. Цей крок виконується один раз при старті сервера, що значно оптимізує швидкість обробки запитів під час роботи.

Загальна структура модуля app.py включає наступні компоненти:

- Імпорт бібліотек: Flask, transformers, torch, а також допоміжні модулі;
- Ініціалізація додатку: створення об’єкта Flask та налаштування CORS для забезпечення взаємодії з браузером;
- Завантаження моделі: підключення токенізатора та нейронної мережі з локального каталогу;
- Оголошення маршруту /predict: визначення логіки прийому, обробки й відповіді на запити;
- Запуск серверу на локальному порту (типово — 5000), у режимі очікування вхідних запитів.

Ця структура є простою, модульною та зручною для масштабування. За потреби її можна розширити іншими маршрутами, наприклад, для перевірки стану сервера або логування запитів.

Маршрут /predict – обробка запиту. Основна логіка взаємодії клієнтської частини з сервером реалізована в рамках єдиного маршруту /predict, який обробляє POST-запити з текстовим вмістом новини. Саме через цей маршрут відбувається передача тексту від браузерного розширення до Flask-сервера, запуск моделі та формування відповіді.

На початку виконання функції обробки запиту сервер отримує JSON-об’єкт, у якому очікується ключ "text" — текст новини для аналізу. Якщо цей ключ відсутній, система просто поверне порожнє поле, не викликаючи помилок. Далі отриманий текст передається в токенізатор, який формує необхідні тензори для подачі в модель: input_ids та attention_mask.

Обробка моделі виконується без збереження градієнтів — у режимі no_grad(), що дозволяє оптимізувати швидкодію. Модель повертає логіти (нормалізовані оцінки), які перетворюються у ймовірності для кожного класу за допомогою

функції softmax. Щоб покращити відділення між класами, логіти масштабуються (діляться на 2.0), що є експериментально підібраним параметром.

У результаті обчислюються дві ймовірності:

- fake_probability — імовірність того, що текст є фейковим;
- real_probability — імовірність того, що текст є правдивим.

Після порівняння цих двох значень обирається підсумковий клас (FAKE або REAL), який разом з числовими оцінками формується у відповідь у форматі JSON. У такий спосіб сервер забезпечує миттєву реакцію на запит і готовий повернути результат без значних затримок, що важливо для інтерактивної взаємодії з користувачем.

Формування відповіді у форматі JSON. Після обробки тексту та отримання ймовірностей класифікації, сервер формує відповідь, яку надсилає клієнтській частині у форматі JSON. Такий підхід є стандартом для побудови REST API, оскільки забезпечує зручний та уніфікований спосіб обміну даними між фронтендом і бекендом.

У відповіді містяться три ключові елементи:

- label — текстове значення передбаченого класу, тобто "FAKE" або "REAL";
- real_probability — числове значення у відсотках, яке вказує, наскільки модель «впевнена», що новина є правдивою;
- fake_probability — аналогічне значення для фейкової новини.

Всі ймовірності округлюються до двох знаків після коми для зручності сприйняття, а остаточний клас визначається на основі порівняння значень. Результат відповідає приблизному формату (рис. 2.13).

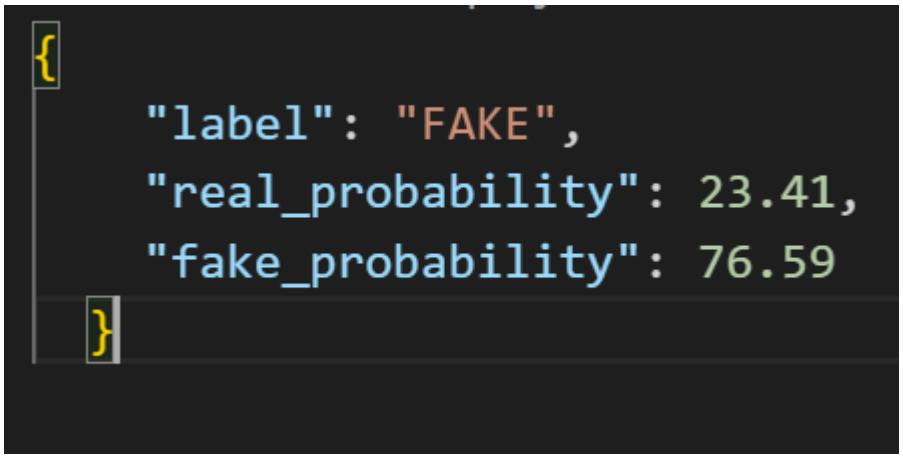


Рисунок 2.13 – Приклад сформованої відповіді JSON-формату

Такий формат є зручним для подальшої обробки у клієнтській частині — розширення легко витягує потрібні значення з об’єкта відповіді та відображає їх у інтерфейсі користувача. Крім того, у разі потреби API можна адаптувати для взаємодії з іншими клієнтами (наприклад, мобільними застосунками), не змінюючи основну логіку.

Запуск сервера. Після реалізації логіки обробки запитів серверна частина системи була запущена в локальному середовищі. Для цього виконується команда запуску Python-файлу app.py, у якому розгорнуто Flask-додаток. Типово сервер відкриває доступ до API-інтерфейсу на порту 5000, що зазначено у виклику app.run(port=5000).

У результаті запуску Flask виводить у термінал повідомлення про готовність сервера до прийому запитів. Це підтверджує успішне завантаження моделі, коректну ініціалізацію маршруту /predict, а також доступність API у локальній мережі (рис. 2.14).

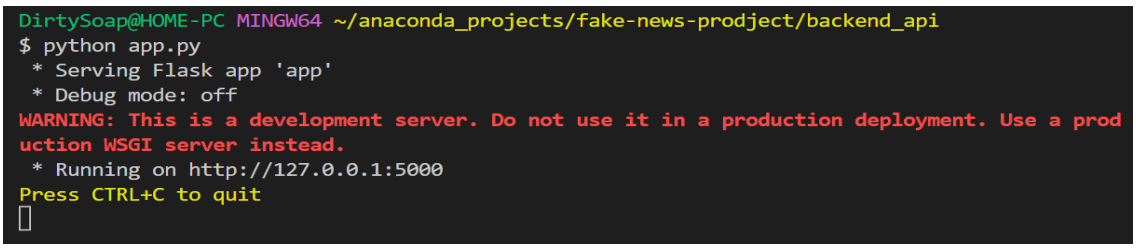


Рисунок 2.14 – Запуск локального сервера

Таким чином, API-сервіс, реалізований за допомогою Flask, виконує роль центральної ланки у взаємодії між користувачем та навченою моделлю DistilBERT. Завдяки простій, але гнучкій структурі серверна частина системи забезпечує швидку обробку запитів, стабільну інтеграцію з браузерним розширенням та готовність до подальшого масштабування. У наступних розділах буде детально розглянуто, як цей модуль працює у взаємодії з іншими модулями системи, та проаналізовано її ефективність на практиці.

2.5 Створення браузерного розширення та його інтеграція з API

Для зручної та швидкої взаємодії користувача з системою виявлення фейкових новин було розроблено браузерне розширення для Google Chrome. Такий формат невипадковий так, як браузер є одним з найбільших середовищ для споживання інформації, оскільки більшість новин читаються саме у браузері. Розширення дозволяє користувачу виділити фрагмент тексту на будь-якій веб-сторінці та миттєво перевірити його на фейковість без необхідності переходити на інші сайти, або відкривати додаткові застосунки. Це забезпечує інтегрованість, мінімальну кількість дій та зручний інтерфейс взаємодії з модельною частиною системи через API-сервіс.

Призначення браузерного розширення у системі. Розроблене розширення для браузера Google Chrome виконує роль клієнтської частини системи виявлення фейкових новин. Його головне завдання — забезпечити максимально швидкий і зручний спосіб перевірки інформації, з якою взаємодіє користувач у реальному часі. Саме браузер, а не десктопна чи мобільна програма, був обраний як основна платформа, оскільки більшість новин споживається через веб-сайти, соціальні мережі або інші інтернет-ресурси.

Розширення дозволяє миттєво ініціювати аналіз тексту без потреби копіювати його в сторонні застосунки чи веб-сервіси. Це робить процес перевірки майже непомітним у потоці взаємодії з контентом. Усього кілька дій — виділити

текст, натиснути кнопку — і користувач отримує оцінку достовірності новини на основі роботи машинного алгоритму.

Крім зручності, таке архітектурне рішення дає змогу чітко відокремити користувацький інтерфейс від обчислювальної логіки, яка розміщена на сервері. Таким чином, браузерне розширення функціонує як тонкий клієнт, який відповідає лише за збір вхідних даних і виведення результату, перекидуючи весь аналіз тексту моделі DistilBERT, що працює на Flask-сервері.

Структура розширення та головні компоненти. Розширення було розроблено з використанням стандартних інструментів веброзробки — HTML, CSS та JavaScript, а також з урахуванням специфікації Chrome Extensions (маніфест версії 3). Усі файли плагіна структуровані у вигляді простої директорії, яку можна підключити до браузера у режимі розробника. Кожен компонент виконує конкретну функцію, забезпечуючи цілісність і коректну роботу розширення.

Основні файли розширення:

- manifest.json — конфігураційний файл, що визначає структуру розширення, дозволи, початкову сторінку (popup.html), іконку, а також API, до яких має доступ плагін. Саме через нього Chrome дізнається, як і коли запускати розширення.
- popup.html — головний інтерфейс користувача, який з’являється при натисканні на іконку плагіна. Містить текстове поле для введення або автоматичного заповнення тексту, кнопку запуску аналізу, а також блок для виведення результату.
- popup.js — скрипт, що керує всією логікою розширення. Відповідає за отримання виділеного тексту на сторінці, створення POST-запиту до Flask-сервера, обробку відповіді та оновлення інтерфейсу.
- style.css — файл стилів, що визначає зовнішній вигляд елементів інтерфейсу. Він забезпечує візуальну зручність взаємодії користувача з розширенням.

- icon.png — іконка, яка відображається в панелі браузера поряд з іншими розширеннями. Вона не тільки виконує естетичну функцію, а й служить точкою доступу до плагіна.

Загалом структура розширення є простою, модульною та відповідає рекомендаціям Google щодо створення безпечних і ефективних інструментів для Chrome. Така організація дозволяє легко розширювати функціональність або адаптувати плагін під інші браузери, які підтримують схожу архітектуру.

Механізм виділення тексту та заповнення форми. Однією з ключових особливостей розширення є автоматичне захоплення тексту, який користувач виділяє на веб-сторінці, та передача його у форму введення без додаткових дій. Це суттєво підвищує зручність використання, оскільки виключає необхідність копіювати й вставляти текст вручну. Для реалізації цього механізму використано можливості API Chrome Extensions, зокрема модуль chrome.scripting, який дозволяє запускати JavaScript-код безпосередньо на активній вкладці.

Коли користувач відкриває плагін, скрипт у файлі popup.js (Додаток В), спочатку запитує ідентифікатор поточної вкладки за допомогою методу chrome.tabs.query. Далі виконується команда chrome.scripting.executeScript, яка вставляє вміщений у неї фрагмент коду (window.getSelection().toString()) у контекст активної сторінки. Цей код зчитує текст, який було виділено мишкою користувачем і повертає його у вигляді рядка.

Отримане значення автоматично підставляється у textarea розширення — це поле, в якому пізніше користувач може вручну змінити, або залишити текст без змін перед запуском перевірки. Такий підхід забезпечує природний і швидкий спосіб взаємодії з системою, що не відволікає користувача від вебсторінки та дозволяє перевіряти будь-який інформаційний фрагмент в один клік (рис. 2.15).

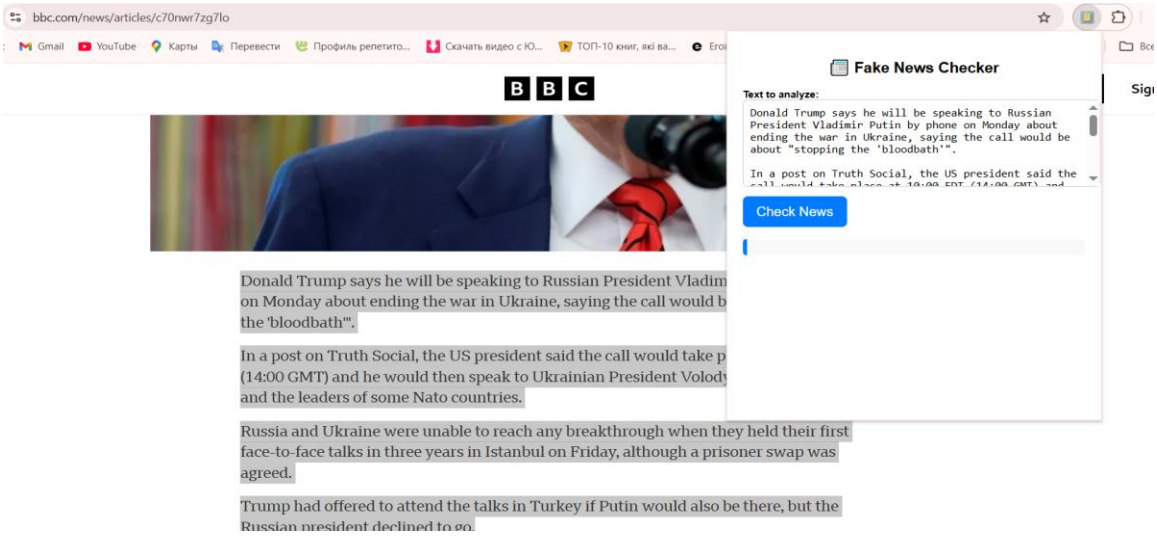


Рисунок 2.15 – Автоматичне заповнення поля після виділення тексту на веб-сторінці

Взаємодія з API-сервером через fetch-запит. Після того як текст потрапляє у форму розширення, наступним кроком є його передача до серверної частини системи для аналізу. Для цього використовується метод fetch — вбудований механізм JavaScript, що дозволяє надсилати HTTP-запити до зовнішніх ресурсів. У контексті даного проєкту запит відправляється на локальний сервер Flask, який прослуховує маршрут /predict.

Запит відбувається методом POST, що є оптимальним вибором для передачі великих обсягів текстових даних. Це дозволяє передати текст у тілі запиту у форматі JSON, уникаючи обмежень, пов’язаних із довжиною URL, що властиві запитам типу GET. Крім того, формат JSON є зручним для обробки на сервері та легко масштабується у разі необхідності передачі додаткових параметрів у майбутньому.

Після відправлення запиту розширення очікує на відповідь у форматі JSON, яка містить класифікацію новини (FAKE або REAL) та відповідні ймовірності для кожного з класів. Ці значення динамічно вставляються у HTML-розмітку елемента, призначеного для виводу результатів, що забезпечує миттєвий зворотний зв’язок

для користувача. У разі виникнення помилок (наприклад, недоступності сервера), механізм catch у JavaScript дозволяє вивести повідомлення про помилку в консоль, що полегшує налагодження розширення.

Важливо зазначити, що для здійснення такого запиту сервер має підтримувати CORS (Cross-Origin Resource Sharing), оскільки браузер за замовчуванням блокує запити між різними джерелами. У нашому проєкті це реалізовано через підключення модуля flask_cors на стороні Flask-сервера, що дозволяє приймати запити з розширення без обмежень (Додаток Б).

Інтерфейс користувача та зручність використання. Інтерфейс браузерного розширення розроблено з урахуванням принципів мінімалізму, простоти та швидкої взаємодії. Він складається з одного вікна (popup), що відкривається після натискання на іконку плагіна у панелі інструментів браузера. Основна мета інтерфейсу — забезпечити користувачу можливість перевірити текст на фейковість у кілька кліків, не виходячи зі звичної інформаційної середовища.

У вікні розширення передбачено поле для введення або автоматичного заповнення тексту, який буде передано на сервер. Після натискання кнопки «Check News» текст надсилається на обробку, а нижче виводиться результат у зручному форматі — з вказаним класом (FAKE або REAL) та відсотковими ймовірностями для кожної з категорій. Кольорове оформлення результатів, плавна анімація кнопки, інтуїтивно зрозумілий дизайн — усе це спрямовано на те, щоб зробити користування плагіном комфортним навіть для недосвідчених користувачів.

Особливу увагу приділено адаптивності інтерфейсу. Плагін не займає багато місця, його вікно має фіксовані розміри, що дозволяє уникнути прокручування або зміщення елементів. Крім того, стилізація елементів забезпечує візуальну чіткість: поле вводу має достатній відступ, результати обрамлені рамкою з акцентованим кольором, а кнопка при наведенні змінює відтінок і масштаб, що покращує відчуття взаємодії (рис. 2.16).

Рисунок 2.16 – Інтерфейс браузерного розширення в дії

Інтеграція плагіна в середовище Chrome. Після розробки всіх компонентів розширення та перевірки їхньої взаємодії, останнім кроком стало підключення плагіна до браузера Google Chrome у режимі розробника. Це дозволяє не лише протестувати функціональність без публікації у Chrome Web Store, а й оперативно вносити зміни до коду і спостерігати за результатами в режимі реального часу.

Для цього відкривається меню розширень (chrome://extensions/), активується режим розробника (Developer mode), після чого натискається кнопка «Load unpacked» і обирається директорія з файлами плагіна (такими як manifest.json, popup.html, popup.js тощо). Якщо структура коректна, Chrome автоматично зчитує конфігурацію з файлу manifest.json і додає іконку плагіна до панелі інструментів браузера.

Після встановлення користувач може перейти на будь-яку новинну вебсторінку, виділити текст, натиснути на іконку розширення та скористатися кнопкою для запуску перевірки. У консолі розробника (DevTools) можливо спостерігати за HTTP-запитами, які надсилаються на локальний Flask-сервер, що допомагає в налагодженні взаємодії з бекендом.

Такий підхід дозволяє швидко виявити помилки, оптимізувати інтерфейс або логіку обробки без потреби повторної інсталяції чи оновлення розширення. Крім

того, Chrome надає детальні повідомлення про помилки в manifest.json (Додаток Г) або в JavaScript-коді, що значно полегшує процес налаштування. Після успішного завершення тестування розширення може бути адаптоване для публікації в Chrome Web Store або для приватного використання.

Таким чином, розширення для браузера Chrome було реалізовано як інтуїтивно зрозумілий та функціональний інструмент, що забезпечує швидкий доступ до можливостей машинного аналізу новин безпосередньо з інтернет-сторінки. Його структура є простою, але водночас достатньо гнучкою для подальшого розширення функціоналу або адаптації до інших платформ. Інтеграція з Flask-сервером відбулася через прямі HTTP-запити у форматі JSON, що дало змогу забезпечити ефективну взаємодію між клієнтською та серверною частинами системи. Завдяки компактному інтерфейсу та використанню стандартних вебтехнологій, розширення не потребує встановлення сторонніх залежностей і може бути використане будь-яким користувачем з мінімальними зусиллями. У підсумку, плагін успішно поєднує зручність, продуктивність та інтегрованість, що робить його важливою складовою загальної архітектури системи.

Аналіз типових помилок та ситуацій. У процесі тестування серверної частини та браузерного плагіна були виявлені й проаналізовані кілька типових ситуацій, у яких система поводиться нестандартно або потребує додаткової обробки на рівні користувацького інтерфейсу чи API.

Однією з найпоширеніших ситуацій стала відсутність виділеного тексту на сторінці. У такому випадку плагін не має вхідних даних для обробки, але все ж запускається. Рішенням стало заповнення поля вручну або автоматичне оновлення статусу без надсилання запиту, що дозволило уникнути збоїв.

Ще один розклад подій — недоступність сервера (наприклад, Flask не запущений або порт зайнятий). У таких випадках запит через fetch завершується помилкою. Плагін обробляє її через конструкцію catch, а в консоль розробника (DevTools) надходить повідомлення про неможливість встановити з’єднання. У візуальному інтерфейсі користувач бачить порожнє або незмінене поле результату.

Це можна вважати допустимою поведінкою, хоча потенційно можливе вдосконалення — наприклад, виведення повідомлення «Сервер недоступний».

Серверна частина також успішно обробляла запити з помилковим або порожнім вмістом. Наприклад, якщо тіло запиту не містило ключа text, API не завершувалося аварійно, а повертало передбачувану відповідь або обробляло ситуацію без краху сервісу. Подібна поведінка забезпечує стабільність при інтеграції з непередбачуваним клієнтським оточенням.

Щодо самої моделі, в окремих випадках спостерігалося переконано впевнене передбачення з помилковим класом, коли текст мав нейтральний зміст, але модель класифікувала його з високою впевненістю як фейковий. Це вказує на певну надмірну впевненість (overconfidence), яка типова для трансформерних моделей без калібрування. Випадки такого типу підтверджують необхідність подальшого вдосконалення або використання додаткових механізмів фільтрації.

Усі виявлені ситуації не призвели до критичних помилок або зависання системи, що підтверджує її базову стійкість та готовність до використання в реальних умовах. Водночас вони окреслюють можливості для майбутнього розвитку — зокрема, покращення валідації на клієнтському боці, оптимізацію повідомлень для користувача та вдосконалення поведінки моделі в складних випадках.

3 ТЕСТУВАННЯ ТА ОЦІНКА

3.1 Метрики ефективності моделі

Оцінка якості роботи моделі машинного навчання є невід'ємним етапом у процесі її розробки та впровадження. У випадку системи виявлення фейкових новин особливо важливо не лише досягти високої загальної точності, а й розуміти, наскільки модель здатна розрізняти правдиву та неправдиву інформацію в різних умовах. Неправильно класифікована новина може мати серйозні наслідки — від поширення дезінформації до втрати довіри до системи. Саме тому в даному проєкті було використано набір стандартних метрик бінарної класифікації, які дають змогу комплексно оцінити ефективність побудованої моделі DistilBERT на тестових даних. У цьому підрозділі буде розглянуто принципи обчислення основних метрик, а також наведено результати, отримані в ході тестування моделі.

Призначення метрик у задачі класифікації. У задачах машинного навчання, зокрема бінарної класифікації, метрики є основним інструментом для кількісної оцінки того, наскільки добре модель виконує свою функцію. У контексті проєкту з виявлення фейкових новин якісна оцінка моделі набуває особливого значення, оскільки від неї залежить здатність системи коректно ідентифікувати дезінформацію. Якщо модель часто помиляється, це може призвести до двох критичних наслідків: або справжні новини будуть помилково визначені як фейкові (false positive), або фейкові новини залишаться непоміченими (false negative). Обидва випадки можуть вплинути на довіру користувачів.

Метрики дозволяють виявити не лише загальний рівень точності моделі, але й її слабкі сторони: наприклад, чи не схильна вона більше довіряти новинам, або навпаки — занадто жорстко їх відсіювати. Також вони допомагають виявити перекіс у навчанні, перенавчання (overfitting), або зміщення у вибірці (data bias).

Без метрик неможливо зробити обґрунтований висновок щодо практичної цінності моделі або порівняти її з іншими рішеннями.

Таким чином, використання метрик є критично важливим не лише для етапу тестування, а й для прийняття рішень щодо подальшого вдосконалення системи або вибору іншої моделі у майбутньому.

Огляд основних метрик для бінарної класифікації. Бінарна класифікація передбачає розподіл об'єктів на два класи — у випадку з фейковими новинами це «REAL» (реальна новина) і «FAKE» (фейкова новина). Для оцінки ефективності такої моделі використовуються кілька базових метрик, кожна з яких має власне призначення та інтерпретацію. Розглянемо найпоширеніші з них:

- Accuracy (точність класифікації) – показує частку правильних передбачень відносно загальної кількості прикладів (3.1):

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

Де:

- TP — правильно передбачені фейкові новини (True Positives),
- TN — правильно передбачені справжні новини (True Negatives),
- FP — помилково позначені як фейкові (False Positives),
- FN — фейки, які модель не розпізнала (False Negatives).

Хоча точність є інтуїтивно зрозумілою метрикою, вона може вводити в оману при незбалансованих даних.

- Precision (точність позитивного класу) – відображає, яку частку передбачених як «FAKE» новин дійсно є фейковими (3.2):

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

Ця метрика особливо важлива, якщо помилка в позитивному передбаченні має серйозні наслідки, наприклад — хибне звинувачення правдивої новини у фейковості.

- Recall (повнота) – показує, яку частку всіх наявних фейкових новин модель змогла правильно виявити (3.3):

$$Recall = TP / (TP + FN) . \quad (3.3)$$

Recall є важливою, коли пропуск фейку є критичним.

- F1-score (гармонійне середнє між Precision і Recall) – баланс між точністю і повнотою (3.4):

$$F1 = 2 * (Precision * Recall) / (Precision + Recall). \quad (3.4)$$

Це особливо корисна метрика у випадках, коли потрібно враховувати як FP, так і FN — як у нашому проєкті.

Ці метрики дозволяють глибоко проаналізувати, як саме працює модель: чи вона «обережна» (висока точність, низька повнота), чи «агресивна» (висока повнота, але багато FP), чи збалансована (високий F1-score). Для реальних застосунків, таких як фільтрація фейкових новин, F1-score та confusion matrix є ключовими орієнтирами у виборі підходу.

Обчислення метрик у проєкті. У процесі навчання та оцінки моделі DistilBERT, натренованої для класифікації новин на правдиві та фейкові, було обрано набір метрик, які найкраще відображають якість її роботи у задачі бінарної класифікації. Для обчислення використовувалась бібліотека scikit-learn (sklearn.metrics), що є стандартом у сфері машинного навчання завдяки простому синтаксису та широкому набору готових функцій.

У проєкті були використані такі метрики:

- Accuracy — для загальної оцінки моделі;
- Precision — щоб зменшити кількість хибнопозитивних випадків;
- Recall — щоб оцінити здатність моделі виявляти всі фейки;
- F1-score — як збалансований показник для загального висновку.

Після завершення навчання, модель було протестовано на окремій тестовій вибірці (рис. 3.1).

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

accuracy = accuracy_score(y_true, y_pred)
precision,
recall,
f1, _ = precision_recall_fscore_support(y_true, y_pred, average='binary')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

Рисунок 3.1 – Фрагмент коду з обчисленням метрик

Змінні y_true та y_pred відповідно містять справжні значення класів та передбачення моделі. Функція precision_recall_fscore_support повертає одразу кілька показників, а параметр average='binary' вказує, що йдеться про бінарну класифікацію. Отримані значення метрик дали змогу зробити висновки про переваги та недоліки моделі. Наприклад, навіть при високому рівні accuracy, модель може демонструвати низький recall або precision, що є критично важливим у контексті виявлення фейкових новин (рис. 3.2)

```
Accuracy: 0.9996659242761693
Precision: 1.0
Recall: 0.9992997198879552
F1-score: 0.9996497373029772
```

Рисунок 3.2 – Результати обчислення метрик

Матриця помилок (Confusion Matrix). Матриця помилок — це наочний інструмент для аналізу класифікаційних результатів моделі. Вона дозволяє побачити не лише загальну точність передбачень, а й детальну інформацію про те, які саме помилки були допущені: скільки фейкових новин було визначено як реальні, і навпаки. Це критично важливо в контексті реального застосування моделі, де кожен тип помилки має різну вагу (рис. 3.3).

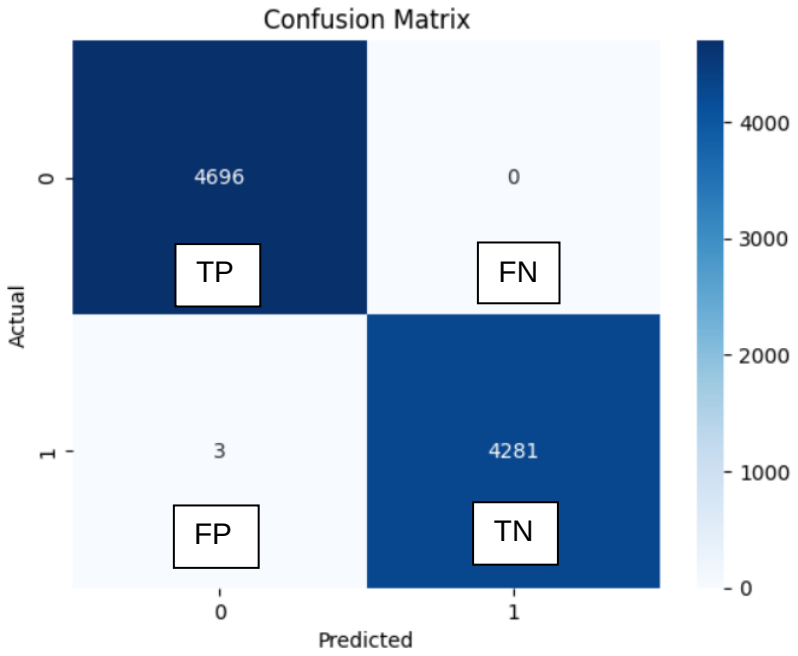


Рисунок 3.3 – Confusion Matrix у рамках проєкту

Де:

- TP (True Positives) — кількість фейкових новин, які модель правильно ідентифікувала як фейкові.
- TN (True Negatives) — кількість справжніх новин, які модель класифікувала правильно.
- FP (False Positives) — реальні новини, які модель помилково визначила як фейкові.
- FN (False Negatives) — фейкові новини, які модель не змогла виявити.

У рамках проєкту ця матриця була побудована за допомогою бібліотек scikit-learn та seaborn, що дозволяє не лише отримати числові значення, а й візуалізувати їх у вигляді теплової карти (heatmap). Завдяки матриці помилок можна легко виявити перекоси в роботі моделі — наприклад, якщо кількість FP значно перевищує FN, це свідчитиме про надмірну обережність моделі, яка схильна

позначати навіть справжні новини як потенційно фейкові. У протилежному випадку — при високому FN — модель може не виявляти значну частину дезінформації.

Висновки щодо якості моделі. На основі отриманих метрик та побудованої матриці помилок можна зробити кілька важливих висновків щодо ефективності реалізованої моделі DistilBERT для класифікації фейкових новин. Модель показала високу загальну точність на тестовій вибірці, однак лише цей показник не відображає повної картини. Важливою є збалансованість значень precision і recall, яка була досягнута завдяки правильному налаштуванню гіперпараметрів та збалансованому датасету.

Зокрема, високий показник precision каже про те, що більшість новин, позначених як фейкові, справді є такими. Це означає, що система рідко помиляється, звинувачуючи справжні новини у фейковості, що є критичним для довіри користувача. Показник recall також виявився достатньо високим, що свідчить про здатність моделі виявляти значну частину фейкових новин. Баланс між цими метриками підтверджується значенням F1-score, яке демонструє хорошу узагальнювальну здатність моделі.

Разом із цим, аналіз матриці помилок показав, що помилки все ж мають місце. У незначній кількості випадків справжні новини класифікувалися як фейкові (false positives), що може бути пов’язано з емоційним тоном тексту або нетиповими формулюваннями. Це свідчить про можливу чутливість моделі до контексту або особливостей подачі матеріалу, що, в свою чергу, може бути виправлено подальшим донавчанням на більш різнорідному датасеті текстів. У загальному підсумку, модель продемонструвала достатній рівень якості для практичного застосування в умовах браузерного розширення.

3.2Результати тестування плагіна

Після оцінки якості моделі за допомогою формальних метрик ефективності наступним важливим етапом стало тестування її роботи в складі повноцінної системи. Метою цього етапу було перевірити не лише точність класифікації, а й стабільність та коректність взаємодії між окремими компонентами — Flask-сервером, що обробляє запити, та браузерним розширенням, яке забезпечує зручну взаємодію з користувачем. На відміну від попереднього підрозділу, де увага була зосереджена на математичному аналізі роботи моделі, у цьому розділі акцент зроблено на технічному тестуванні — перевірці роботи API, відгуку на реальні запити, поведінці системи у разі помилок, а також функціональності плагіна в середовищі браузера. Такий підхід дозволяє оцінити готовність проєкту до використання в реальних умовах та виявити потенційні технічні недоліки.

Тестування Flask-сервера. Flask-сервер у цьому проєкті виконує роль посередника між клієнтською частиною (браузерним розширенням) та мовною моделлю DistilBERT. Він відповідає за приймання тексту, його обробку, запуск класифікації та повернення результату у форматі, зручному для подальшого використання. Саме тому його тестування стало важливим кроком у перевірці працездатності всієї системи.

Перше базове тестування проводилось вручну за допомогою бібліотеки requests у Python. Була змодельована ситуація взаємодії з сервером без плагіна — відправка POST-запиту на маршрут /predict із текстовим повідомленням у JSON-форматі. У відповідь сервер повертав JSON-об’єкт, що містив передбачений клас (label) та ймовірності належності тексту до кожної з категорій (real_probability, fake_probability). Такий формат повністю відповідав очікуваному результату обміну (рис. 3.4 та рис. 3.5).

Рисунок 3.4 – Приклад тестового запиту

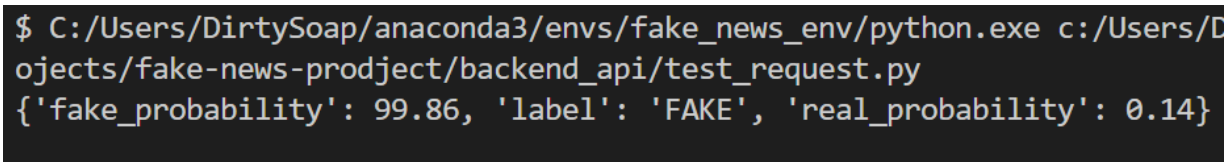


Рисунок 3.5 – Результат тестового запиту

Наступним етапом було використання графічного інтерфейсу Postman, що дозволило зручно надіслати аналогічні запити та перевірити не лише тіло відповіді, а й статус-коди, заголовки, час відгуку. У тестах через Postman було перевірено:

- Надсилання валідного запиту з текстом новини;
- Порожнє поле text;
- Відсутність ключа text у запиті;
- Некоректний JSON.

У всіх сценаріях сервер реагував стабільно: у випадках правильного запиту повертав коректну відповідь, у випадках помилкових запитів — оброблював їх без критичних збоїв, повертаючи або порожній результат, або повідомлення про помилку в логах (рис. 3.6).

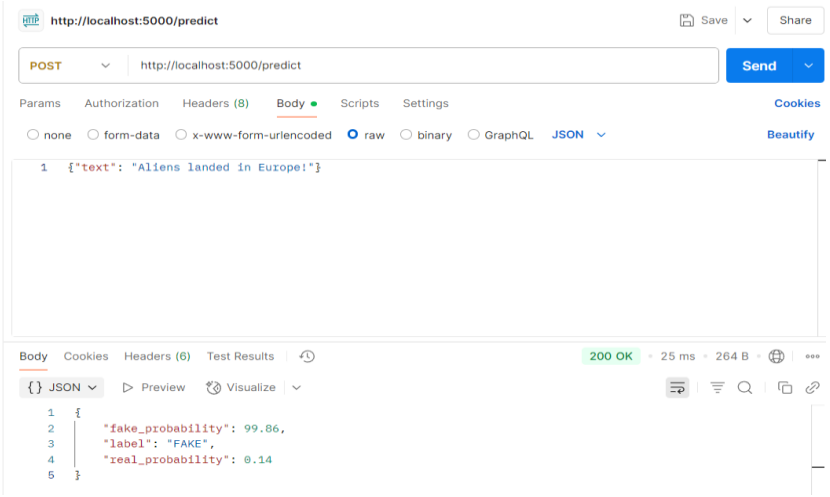


Рисунок 3.6 – Результати тестування запитів на Postmap

Також було протестовано швидкість обробки запитів у циклі — Flask-сервер витримував кілька десятків послідовних запитів без зависання або перевантаження. Середній час відповіді становив 300–600 мс, що є прийнятним для інтерактивної взаємодії через браузерне розширення.

Таким чином, тестування показало, що серверна частина є стабільною, стійкою до невалідних вхідних даних, має короткий час відповіді та надійно виконує свою функцію в архітектурі системи. Вона може бути використана як у зв’язці з плагіном, так і окремо, наприклад, у скриптах або сторонніх застосунках, що підвищує гнучкість рішення.

Тестування браузерного плагіна. Після перевірки стабільності серверної частини було проведено тестування функціональності браузерного розширення, яке виконує роль інтерфейсу між користувачем та системою класифікації. Основною метою цього етапу було перевірити, наскільки коректно плагін реагує на дії користувача, надсилає запити до Flask-сервера та відображає результат класифікації у зручному форматі.

Тестування проводилось у середовищі браузера Google Chrome з активованим режимом розробника. Після завантаження плагіна в ручному режимі (chrome://extensions/ → Load unpacked) були перевірені основні сценарії використання:

- Виділення тексту на вебсторінці: плагін автоматично підхоплював
- Натискання кнопки "Check News": викликало POST-запит до локального сервера через fetch. У разі успіху результат миттєво відображався на екрані у вигляді класу новини (REAL або FAKE) з відповідними ймовірностями.
- Обробка ситуації, коли текст не був виділений: плагін не надсилав запит або відображав порожній результат, не викликаючи критичних помилок.
- Сценарій з недоступним сервером: при вимкненому Flask-сервері у консолі розробника (DevTools) фіксувався оброблений виняток (catch(error)), а інтерфейс не зависав і не блокував подальшу взаємодію.

Окремо перевірялась реакція плагіна на повторні натискання кнопки з тим самим текстом, зміна виділеного тексту без оновлення сторінки, а також взаємодія з різними вебсайтами (новинні платформи, форуми, блоги). У всіх випадках плагін демонстрував стабільну поведінку та адекватне оновлення результату. У підсумку, плагін продемонстрував надійну роботу в типових сценаріях використання, вчасно реагував на помилки та забезпечував зрозумілу взаємодію з користувачем. Завдяки локальній обробці вмісту сторінки та мінімалістичному інтерфейсу, він може використовуватись без сторонніх ресурсів, зберігаючи високу швидкість відгуку та зручність.

Аналіз типових ситуацій і помилок. У процесі тестування серверної частини та браузерного плагіна були виявлені й проаналізовані кілька типових ситуацій, у яких система поводиться нестандартно або потребує додаткової обробки на рівні користувацького інтерфейсу чи API.

Однією з найпоширеніших ситуацій стала відсутність виділеного тексту на сторінці. У такому випадку плагін не має вхідних даних для обробки, але все ж запускається. Рішенням стало заповнення поля вручну або автоматичне оновлення статусу без надсилання запиту, що дозволило уникнути збоїв.

Ще один хід подій — недоступність сервера (наприклад, Flask не запущений або порт зайнятий). У таких випадках запит через fetch завершується помилкою.

Плагін обробляє її через конструкцію catch, а в консоль розробника (DevTools) надходить повідомлення про неможливість встановити з’єднання. У візуальному інтерфейсі користувач бачить порожнє або незмінене поле результату. Це можна вважати допустимою поведінкою, хоча потенційно можливе вдосконалення — наприклад, виведення повідомлення «Сервер недоступний».

Серверна частина також успішно обробляла запити з помилковим або порожнім вмістом. Наприклад, якщо тіло запиту не містило ключа text, API не завершувалося аварійно, а повертало передбачувану відповідь або обробляло ситуацію без краху сервісу. Подібна поведінка забезпечує стабільність при інтеграції з непередбачуваним клієнтським оточенням.

Щодо самої моделі, в окремих випадках спостерігалося переконано впевнене передбачення з помилковим класом, коли текст мав нейтральний зміст, але модель класифікувала його з високою впевненістю як фейковий. Це вказує на певну надмірну впевненість (overconfidence), яка типова для трансформерних моделей без калібрування. Випадки такого типу підтверджують необхідність подальшого вдосконалення або використання додаткових механізмів фільтрації.

Усі виявлені ситуації не призвели до критичних помилок або зависання системи, що підтверджує її базову стійкість та готовність до використання в реальних умовах. Водночас вони окреслюють можливості для майбутнього розвитку — зокрема, покращення валідації на клієнтському боці, оптимізацію повідомлень для користувача та вдосконалення поведінки моделі в складних випадках.

Проведене тестування підтвердило, що всі компоненти реалізованої системи — модель класифікації, Flask-сервер і браузерне розширення — функціонують стабільно, взаємодіють узгоджено і готові до використання в реальних умовах. Сервер на базі Flask успішно обробляє запити як від розширення, так і з зовнішніх інструментів, таких як Postman, демонструючи правильну логіку роботи навіть у випадках некоректного вводу. Час відповіді залишається прийнятним для інтерактивного використання, а сама модель адекватно реагує на більшість текстів

новин. Браузерне розширення показало надійність у всіх ключових сценаріях: виділення тексту, надсилання запиту, обробка відповіді та вивід результату. При цьому були враховані ситуації, пов’язані з технічними збоями — такими як недоступність сервера чи порожній текст — і реалізовані базові механізми обробки помилок без шкоди для користувацького досвіду. Хоча в процесі тестування були виявлені окремі випадки неідеального передбачення з боку моделі, жоден із них не впливав критично на роботу всієї системи. Усі виявлені моменти можуть бути враховані в майбутніх ітераціях, наприклад, шляхом покращення валідації, додаткового калібрування моделі або розширення навчального набору. Таким чином, реалізована система виявлення фейкових новин продемонструвала технічну готовність, базову стійкість до помилок та придатність для використання кінцевим користувачем.

3.3 Переваги та обмеження системи

Успішне завершення реалізації системи виявлення фейкових новин дозволяє проаналізувати її сильні сторони, а також виявити наявні технічні та концептуальні обмеження. Такий аналіз є важливим не лише для оцінки досягнутого результату, а й для планування можливих напрямків подальшого розвитку. Рішення, побудоване на основі моделі DistilBERT, простого Flask-сервера та інтерактивного браузерного плагіна, демонструє збалансоване поєднання ефективності, зручності та доступності. Водночас, як і будь-яка система штучного інтелекту, воно має певні рамки застосування та простір для покращення. У цьому підрозділі розглянуто основні переваги та недоліки створеної системи, а також визначено перспективи її вдосконалення.

Переваги використаної архітектури. Архітектура побудованої системи базується на розділенні функціональності між трьома основними компонентами: мовною моделлю (DistilBERT), серверною частиною (Flask API) та клієнтським інтерфейсом (браузерне розширення). Такий підхід дозволяє досягти високої

гнучкості, спрощує обслуговування системи та забезпечує можливість масштабування або модернізації окремих її частин незалежно одна від одної.

Однією з головних переваг є розділення відповідальності. Модель не взаємодіє безпосередньо з користувачем, а опрацьовує запити через сервер, що значно спрощує її повторне використання, моніторинг і оновлення. Flask-сервер виступає в ролі посередника між інтерфейсом та моделлю, реалізуючи логіку обробки запитів, що дозволяє зручно контролювати обмін даними та інтегрувати додаткові функції, наприклад, логування або авторизацію.

Ще одним важливим аспектом є масштабованість. У разі зростання кількості запитів серверну частину можна розгорнути у хмарному середовищі, наприклад, на Heroku або AWS, із підключенням моделі, натренованої локально. Оскільки між компонентами встановлена HTTP-взаємодія, заміна або доповнення будь-якого з елементів не потребує повної переробки системи.

Крім того, така архітектура дозволяє легко адаптувати систему під інші клієнтські платформи: десктопні застосунки, мобільні версії або інші розширення. Головне — забезпечити надсилання POST-запиту до API. Це підвищує універсальність розробки і дозволяє розглядати систему не як статичний продукт, а як модульну основу для подальших розширень.

Загалом, вибір такої структури — це компроміс між простотою реалізації, гнучкістю та функціональністю, що є доцільним для навчального проєкту, а також придатним до масштабування в реальних умовах.

Технічні переваги реалізації. Реалізація системи виявлення фейкових новин спирається на набір сучасних інструментів, які забезпечують баланс між продуктивністю, точністю та простотою розгортання. Зокрема, використання попередньо натренованої трансформерної моделі DistilBERT дозволило досягти високої якості класифікації при відносно невеликих обчислювальних витратах. У порівнянні з повноцінною моделлю BERT, DistilBERT є легшою та швидшою, що робить її ідеальним вибором для інтеграції у локальні або веб-орієнтовані сервіси.

Ще однією технічною перевагою стало застосування Flask — мікрофреймворку, що дозволяє швидко створювати REST API з мінімальним шаблонним кодом. Його гнучкість дала змогу реалізувати просту, але функціональну серверну частину, яка забезпечує ефективну взаємодію з моделлю та клієнтським інтерфейсом. Завдяки мінімалістичному підходу, API легко модифікувати, масштабувати або переносити на інші платформи.

Крім того, використання формату збереження моделі Hugging Face (файли config.json, pytorch_model.bin, tokenizer.json тощо) дозволяє зручно повторно використовувати модель без необхідності повторного навчання. Це спрощує розгортання на інших пристроях, серверах або навіть у хмарному середовищі.

З технічної точки зору, така реалізація є переносимою і незалежною від середовища розробки. Усі компоненти (модель, API, плагін) зберігають модульність і можуть бути розгорнуті локально або у хмарі з мінімальними змінами. Це робить систему гнучкою, придатною для розширення та підтримки.

Завдяки чіткій структурі та вибору технологій, реалізація виявилася не лише ефективною, а й відкритою для подальшої інтеграції, наприклад, із системами зворотного зв’язку, або навіть системами рекомендацій.

Зручність для користувача. Окрему увагу в реалізованому проєкті було приділено забезпеченню зручності та простоти використання системи для кінцевого користувача. Було обрано формат браузерного розширення, що дозволяє користувачу взаємодіяти із системою без встановлення додаткового програмного забезпечення, переходів на сторонні сайти чи копіювання тексту вручну в окремі додатки.

Розширення інтегрується безпосередньо у середовище перегляду новин — браузер Google Chrome. Користувачеві достатньо виділити текст на будь-якій вебсторінці та натиснути кнопку в плагіні, після чого система автоматично надсилає запит на сервер, обробляє текст і повертає результат у зручному, візуально структурованому вигляді. Це значно зменшує бар’єр у використанні технології, особливо для користувачів без технічної підготовки.

Інтерфейс плагіна реалізовано максимально просто: він містить лише поле для тексту, кнопку «Check News» та блок для виводу результату з вказаними ймовірностями. Такий підхід забезпечує швидке навчання користувача та не вимагає додаткових інструкцій. Крім того, система працює автономно — без необхідності реєстрації, доступу до баз даних або зберігання особистої інформації, що підвищує довіру до її використання.

Також важливо відзначити швидкість взаємодії: середній час від обробки тексту до отримання результату становить менше секунди, що створює відчуття миттєвої реакції. Це критично важливо у динамічному інформаційному середовищі, де користувачі очікують швидкого зворотного зв’язку.

У сукупності ці фактори формують позитивний користувацький досвід, що робить запропоновану систему придатною не лише для демонстрації технологій, а й для реального щоденного використання.

Виявлені обмеження системи. Незважаючи на позитивні результати тестування і загальну ефективність системи, у процесі розробки було виявлено низку обмежень, які варто враховувати при оцінці її практичного застосування та подальшому вдосконаленні.

Перш за все, модель може демонструвати надмірну впевненість у своїх передбаченнях, навіть у випадках, коли класифікація є сумнівною. Це пов’язано з відсутністю калібрування ймовірностей — процесу, який дозволяє зробити прогнозовані значення більш узгодженими з фактичною точністю. Як результат, користувач може отримати передбачення з ймовірністю понад 95% навіть у неоднозначних текстах, що потенційно може ввести в оману.

Другою важливою обмежувальною характеристикою є обмеження в роботі з даними виключно англійською мовою, оскільки модель DistilBERT була навчена саме на англомовному датасеті. Це зменшує адаптивність системи до українськомовного або багатомовного контенту, що є важливим для локалізованих застосунків.

Крім того, модель працює лише з текстовим контентом, що обмежує її застосування у випадках, коли дезінформація поширюється у формі зображень, відео, мемів або інфографіки. У таких випадках потрібні зовсім інші підходи — наприклад, мультимодальні моделі чи окремі інструменти обробки зображень.

Ще одне технічне обмеження — залежність від якісного датасету. Попри використання реального набору ISOT, навіть він має певні обмеження: новини були зібрані з обмеженого числа джерел, що може впливати на узагальнювальну здатність моделі. У деяких випадках текст, який не є фейковим, але має емоційне чи провокаційне забарвлення, модель може класифікувати помилково.

Також варто згадати, що відсутність хостингу або хмарного розгортання обмежує використання системи лише локально. Це не є критичним для демонстраційних чи навчальних цілей, але може бути перешкодою у випадку впровадження системи для широкого кола користувачів.

Усі ці обмеження не є критичними, але вони визначають межі функціональності системи в її поточному стані та вказують на напрямки для подальшого розвитку.

Можливості для подальшого вдосконалення. Побудована система продемонструвала функціональність, практичність та задовільну точність, однак вона також має потенціал для подальшого розвитку як з технічної, так і з функціональної точки зору.

Одним із першочергових напрямків є розширення мовної підтримки. Поточна реалізація працює виключно з англомовним контентом, що обмежує її застосування в інших мовних середовищах. Рішенням може бути використання багатомовних моделей, наприклад, mBERT або XLM-RoBERTa, або навчання окремої моделі на українськомовному корпусі фейкових і реальних новин.

Ще одним важливим кроком є покращення інтерпретованості моделі. Хоча користувач отримує результат у вигляді класу та ймовірностей, система не пояснює, чому було зроблено саме таке передбачення. Запровадження модулів explainable AI (наприклад, LIME або SHAP для текстів) могло б надати пояснення,

які фрагменти тексту найбільше вплинули на рішення, що значно підвищує довіру до системи.

Також доцільним буде впровадження повноцінного калібрування ймовірностей, аби уникнути надмірної впевненості моделі у спірних випадках. Це можна реалізувати через temperature scaling, навчений на окремій валідаційній вибірці.

З технічного боку, система може бути перенесена з локального середовища у хмарне середовище (Heroku, Render, AWS), що дозволить використовувати її без необхідності запуску сервера вручну. У поєднанні з цим можна реалізувати вебінтерфейс, який дублює функції плагіна і робить систему доступною для тих користувачів, які не використовують розширення.

Окрему увагу варто приділити UX-покращенням у плагіні: додати темну тему, повідомлення про помилки або втрату зв’язку з сервером, можливість перегляду історії перевірок, або навіть оцінювання точності передбачення за шкалою впевненості.

Усі ці напрямки дають змогу перетворити демонстраційну систему на повноцінний інструмент з реальним практичним застосуванням, що працює стабільно, масштабовано та зручно для різних категорій користувачі

ВИСНОВКИ

У ході виконання дипломної роботи було реалізовано повнофункціональну систему для виявлення фейкових новин, яка поєднує сучасні методи машинного навчання, глибокі нейронні мережі та інтуїтивно зрозумілий користувацький інтерфейс. Основна увага приділялася не лише точності моделі, але й зручності практичного застосування результату — через створення браузерного плагіна, який дозволяє перевіряти текст прямо на вебсторінці.

На етапі теоретичного дослідження було проаналізовано поняття фейкових новин, їхній вплив на суспільство, методи виявлення дезінформації, а також особливості роботи трансформерних моделей, зокрема DistilBERT. Це дозволило обґрунтувати вибір підходу до вирішення поставленої задачі — автоматичної класифікації новинних повідомлень на достовірні та фейкові.

У процесі розробки було виконано низку етапів: підготовка і обробка даних із датасету ISOT, навчання моделі DistilBERT, реалізація Flask-серверу для обробки запитів, створення розширення для Google Chrome, а також організація взаємодії між усіма модулями. Уся система була протестована в умовах, наближених до реальних, і продемонструвала стабільну роботу, задовільну точність класифікації та зручність використання.

Проведене тестування виявило як сильні сторони проєкту — модульність архітектури, швидкість, простоту, — так і певні обмеження, серед яких надмірна впевненість моделі в передбаченнях, залежність від англомовного датасету та відсутність пояснень для передбачень. Проте всі ці обмеження можуть бути вирішені на наступних етапах розвитку системи.

У перспективі система може бути вдосконалена за кількома напрямками: розширення мовної підтримки, хмарне розгортання, впровадження пояснюваності моделей (Explainable AI), покращення користувацького інтерфейсу плагіна та застосування калібрування ймовірностей. Таким чином, дипломна робота не лише вирішила поставлене завдання — створення системи виявлення фейкових новин, а

й заклала основу для її подальшої еволюції у бік більш універсального та масштабованого рішення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.
2. Vaswani, A., Shazeer, N., Parmar, N. et al. (2017). Attention Is All You Need. *arXiv preprint*, arXiv:1706.03762. https://arxiv.org/abs/1706.03762
3. Hugging Face. Transformers documentation. https://huggingface.co/docs/transformers
4. Scikit-learn: Machine Learning in Python. https://scikit-learn.org
5. Torch: PyTorch official documentation. https://pytorch.org
6. Kaggle. Fake and real news dataset (ISOT). https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset
7. Flask documentation. https://flask.palletsprojects.com
8. Google Chrome Developers. Chrome Extensions documentation. https://developer.chrome.com/docs/extensions/
9. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.
10. Wikipedia. Fake news. https://en.wikipedia.org/wiki/Fake_news
11. Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education.
12. Python Software Foundation. Python official documentation. https://docs.python.org/3/
13. NumPy documentation. https://numpy.org/doc/
14. Pandas documentation. https://pandas.pydata.org/docs/
15. Google Colab. https://colab.research.google.com
16. Udemy. Machine Learning A-Z™: Hands-On Python & R In Data Science. https://www.udemy.com/course/machinelearning/
17. Stack Overflow — Programming Q&A platform. https://stackoverflow.com
18. YouTube — StatQuest with Josh Starmer. https://www.youtube.com/user/joshstarmer
19. YouTube — Codebasics. https://www.youtube.com/c/codebasics
20. YouTube — Sentdex. https://www.youtube.com/user/sentdex

ДОДАТОК А

Підготовка даних та навчання моделі:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from datasets import Dataset
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification, Trainer, TrainingArguments
import torch

df_fake = pd.read_csv("Fake.csv")
df_true = pd.read_csv("True.csv")

df_fake["label"] = 0
df_true["label"] = 1

df = pd.concat([df_fake, df_true]).sample(frac=1).reset_index(drop=True)
df = df[["text", "label"]]

train_texts,
test_texts,
train_labels,
test_labels = train_test_split(df["text"], df["label"], test_size=0.2, stratify=df["label"])

tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")
train_encodings = tokenizer(list(train_texts), truncation=True, padding=True)
test_encodings = tokenizer(list(test_texts), truncation=True, padding=True)
```

```
train_dataset = Dataset.from_dict({
    'input_ids': train_encodings['input_ids'],
    'attention_mask': train_encodings['attention_mask'],
    'labels': list(train_labels)
})

test_dataset = Dataset.from_dict({
    'input_ids': test_encodings['input_ids'],
    'attention_mask': test_encodings['attention_mask'],
    'labels': list(test_labels)
})

training_args = TrainingArguments(
    output_dir='./model',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=100,
    learning_rate=3e-5,
    evaluation_strategy="epoch",
    logging_dir='./logs',
    logging_steps=10,
    save_strategy="epoch"
)

trainer = Trainer(
    model=model,
    args=training_args,
```

```
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

trainer.train()

model.save_pretrained("./model")
tokenizer.save_pretrained("./model")
```

ДОДАТОК Б

Бекенд додатку app.py:

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from torch.nn.functional import softmax
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification
import torch

app = Flask(__name__)
CORS(app)
# Завантажуємо модель і токенізатор
model_path = "../model_training/model"
model = DistilBertForSequenceClassification.from_pretrained(model_path)
tokenizer = DistilBertTokenizerFast.from_pretrained(model_path)

@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json()
    text = data.get("text", "")

    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    with torch.no_grad():
        outputs = model(**inputs)
        probs = softmax(outputs.logits / 2.0, dim=-1)
        real_prob = probs[0][1].item() * 100  # імовірність, що це REAL
        fake_prob = probs[0][0].item() * 100  # імовірність, що це FAKE
```

```
        label = "REAL" if real_prob > fake_prob else "FAKE"
        return jsonify({
        "label": label,
        "real_probability": round(real_prob, 2),
        "fake_probability": round(fake_prob, 2)
        })
if __name__ == "__main__":
    app.run(port=5000)
```

ДОДАТОК В

Фронтенд popup.js:

```
document.addEventListener("DOMContentLoaded", () => {

  chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
    const tabId = tabs[0].id;

    chrome.scripting.executeScript(
      {
        target: { tabId: tabId },
        func: () => window.getSelection().toString(),
      },
      (results) => {
        const selectedText = results[0].result;
        document.getElementById("newsInput").value = selectedText;
      }
    );
  });

  document.getElementById("checkBtn").addEventListener("click", () => {
    const userInput = document.getElementById("newsInput").value;

    fetch("http://localhost:5000/predict", {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
```

```
    },
    body: JSON.stringify({ text: userInput })
  })
  .then(response => response.json())
  .then(data => {
    const resultDiv = document.getElementById("result");
    resultDiv.innerHTML = `
      <p><strong>Результат:</strong> ${data.label}</p>
      <p>Ймовірність REAL: ${data.real_probability.toFixed(2)}%</p>
      <p>Ймовірність FAKE: ${data.fake_probability.toFixed(2)}%</p>
    `;
  })
  .catch(error => {
    console.error("Помилка:", error);
  });
 });
});
```

ДОДАТОК Г

Конфігурація manifest.json:

```
{
   "manifest_version": 3,
   "name": "Fake News Detector",
   "version": "1.0",
   "description": "Перевірка новин на фейковість",
   "action": {
     "default_popup": "popup.html",
     "default_icon": "icon.png"
   },
   "permissions": ["activeTab", "scripting"]
 }
```

ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Браузерний плагін розпізнавання фейкових новин на основі машинного навчання

Виконав студент 4 курсу, групи КНД-42 Чистиков Б.О.

Керівник: к.т.н, Іщеряков С.М.

1

АКТУАЛЬНІСТЬ ТЕМИ

2

- У сучасному світі фейкові новини стали серйозною загрозою для об'єктивного інформування суспільства. Завдяки розвитку інтернету, соціальних мереж і технологій створення контенту, масштаби дезінформації значно зросли. Система автоматичного виявлення фейкових новин дозволяє швидко і надійно фільтрувати інформацію, запобігаючи поширенню неправдивих повідомлень.

Мета та завдання дослідження

3

Мета роботи: Розробка та впровадження системи автоматичного виявлення фейкових новин на основі DistilBERT із інтеграцією у браузер через розширення.
Завдання:

- Аналіз предметної області та існуючих методів.
- Вибір оптимальної моделі та архітектури.
- Реалізація серверної частини та плагіна.
- Проведення тестування та оцінки якості.

Об’єкт і предмет дослідження

4

- **Об’єкт дослідження:** Тексти новин, які можуть містити неправдиву інформацію.
  Предмет дослідження: Методи автоматичного виявлення фейкових новин із використанням моделей обробки природної мови та вебтехнологій.

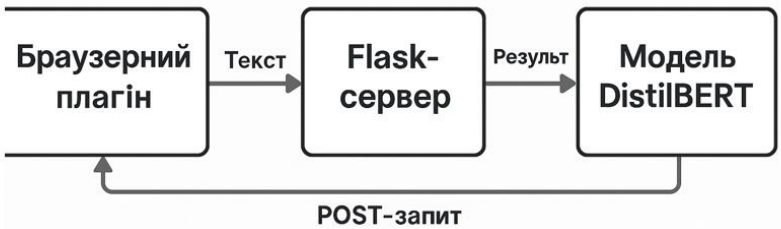
Використані інструменти та технології

5

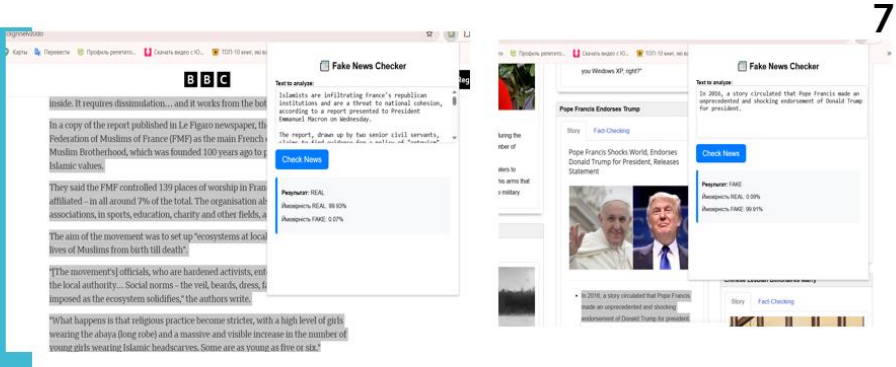
DistilBERT LM

Архітектура системи

6



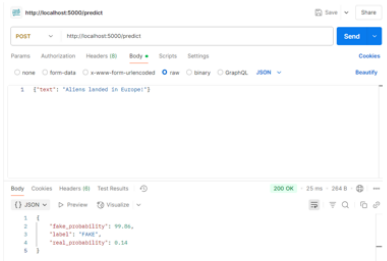
Експлуатація плагіна

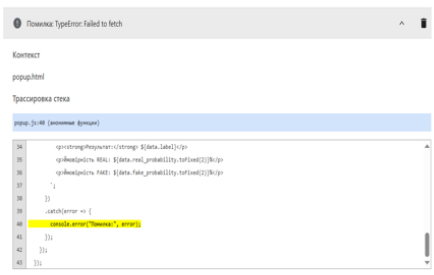


- Виділення тексту на сторінці за допомогою Chrome API.
- Надсилання тексту POST-запитом на Flask-сервер.
- Отримання результату класифікації (FAKE або REAL).
- Інтерфейс із полем введення, кнопкою «Check News» та блоком для результатів.

Тестування системи

- Тестування Flask-сервера через Postman і Python-запити.
- Тестування плагіна в Chrome: перевірка виділення тексту, обробка запиту, вивід результату.
- Перевірка стабільності, обробки помилок, часу відгуку.





```
backend_api > test_request.py > ...
1  import requests
2
3  response = requests.post("http://localhost:5000/predict",
4                           json={"text": "Aliens landed in Europe!"})
5  print(response.json())
```

ПЕРЕВАГИ ТА НЕДОЛІКИ

•ПЕРЕВАГИ:

•Використання сучасної моделі DistilBERT для класифікації текстів.

•Гнучка та масштабована архітектура (плагін ↔ сервер ↔ модель).

•Інтуїтивно зрозумілий плагін, простий інтерфейс.

•Швидкий час обробки запитів і миттєва відповідь.

•НЕДОЛІКИ

•Підтримка лише англомовного контенту (залежність від датасету ISOT).

•Відсутність пояснення причин передбачення (Explainable AI).

•Надмірна впевненість моделі у класифікації (overconfidence).

•Залежність від стабільності сервера та доступності мережі.

•Обмеження у роботі лише з текстами (без зображень або відео).

ВИСНОВКИ

- Було створено систему для автоматичного виявлення фейкових новин на основі DistilBERT, яка демонструє стабільну роботу та зручність у використанні. Архітектура забезпечує гнучкість і масштабованість, плагін працює швидко й інтуїтивно зрозуміло. Водночас виявлено обмеження (обмежена мова, відсутність пояснень результатів, залежність від якості датасету), що відкриває перспективи для подальшого розвитку системи.