Міністерство освіти і науки України
Луцький національний технічний університет
Факультет робототехніки та штучного інтелекту
Кафедра штучного інтелекту та математичного моделювання

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

АНАЛІЗ МЕТОДІВ КЛАСИФІКАЦІЇ ТЕКСТУ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО АНАЛІЗУ ВІДГУКІВ ДЛЯ ОЦІНКИ ЯКОСТІ ПОСЛУГ

AN ANALYSIS OF TEXT CLASSIFICATION METHODS AND THE DEVELOPMENT OF AN AUTOMATED REVIEW ANALYSIS SYSTEM FOR EVALUATING SERVICE QUALITY

спеціальність 113 Прикладна математика
освітня програма «Штучний інтелект та аналіз масивів даних»

Виконав: здобувач вищої освіти
Групи ПРМ-41
Власюк Юрій Валерійович

_____

Керівник:
Д.т.н., професор
Мікуліч Олена Аркадіївна

_____

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
Приходько Олексій Сергійович

_____

Луцьк – 2026

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *архітектури, будівництва та дизайну*

Кафедра *прикладної математики та механіки*

Ступінь вищої освіти: *бакалавр*

Галузь знань: *11 Математика і статистика*

Спеціальність: *113 Прикладна математика*

Освітня програма: *Штучний інтелект та аналіз масивів даних*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Мікуліч О.А.
«___» _____202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Власюку Юрію Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Аналіз методів класифікації тексту та розробка системи автоматизованого аналізу відгуків для оцінки якості послуг / An analysis of text classification methods and the development of an automated review analysis system for evaluating service quality
Керівник роботи: *Мікуліч Олена Аркадіївна*
затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 557/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи
«___»_____ 202_ р.
3. Вихідні дані до роботи _____
_____
_____
_____
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити):
_____
_____
_____
_____
5. Перелік графічного (ілюстративного) матеріалу:
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
1 розділ	*Мікуліч О.А., професор кафедри*		
2 розділ	*Мікуліч О.А., професор кафедри*		
3 розділ	*Мікуліч О.А., професор кафедри*		
4 розділ	*Мікуліч О.А., професор кафедри*		
Висновки	*Мікуліч О.А., професор кафедри*		

7. Дата видачі завдання «___» _____ 202__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	*Обґрунтування теми*		
2.	*Огляд літератури із досліджуваної проблеми*		
3.	_____ *розділ*		
4.	_____ *розділ*		
5.	_____ *розділ*		
6.	_____ *розділ*		
7.	*Висновки та пропозиції*		
8.	*Формування списку використаних джерел*		
9.	*Формування додатків*		
10.	*Оформлення ілюстративного матеріалу*		
11.	*Нормоконтроль*		
12.	*Інструментальна перевірка на академічний плагіат*		
13.	*Представлення кваліфікаційної роботи бакалавра до захисту*		

Здобувач вищої освіти _____ (Власюк Ю.В.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Мікуліч О.А.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Власюк Юрій Валерійович. Аналіз методів класифікації тексту та розробка системи автоматизованого аналізу відгуків для оцінки якості послуг. Рукопис.

Кваліфікаційна робота бакалавра ОП «Штучний інтелект та аналіз масивів даних» спеціальності 113 Прикладна математика. Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (18 найменувань) та додатків.

У роботі досліджено проблему автоматизованого аналізу текстових відгуків споживачів для оцінки якості послуг та проведено порівняльний аналіз методів класифікації тексту. Реалізовано та навчено три моделі класифікації: логістична регресія з TF-IDF векторизацією, fastText та DistilBERT з процедурою fine-tuning. Порівняльний аналіз на тестовій вибірці продемонстрував перевагу трансформерного підходу: DistilBERT досяг показника Macro-F1 = 0,8265, що на 27,6 % перевищує результат логістичної регресії. Розроблено програмну систему автоматизованого аналізу відгуків з модульною архітектурою, REST API на базі FastAPI та демонстраційним веб-інтерфейсом на платформі Gradio.

Ключові слова: прикладна математика, класифікація тексту, обробка природної мови, машинне навчання, трансформери, BERT, TF-IDF, fastText, FastAPI, сентимент-аналіз.

ABSTRACT

Vlasiuk Yurii Valeriiovych. Analysis of Text Classification Methods and Development of an Automated Review Analysis System for Service Quality Assessment. Manuscript.

Bachelor's thesis of the Educational Programme "Artificial Intelligence and Big Data Analytics", specialty 113 Applied Mathematics. Lutsk National Technical University. Lutsk, 2026.

The thesis consists of an introduction, four chapters, conclusions, a list of references (18 titles), and appendices.

The thesis investigates the problem of automated analysis of consumer text reviews for service quality assessment and conducts a comparative analysis of text classification methods. Three classification models were implemented and trained: logistic regression with TF-IDF vectorisation, fastText, and DistilBERT with fine-tuning. Comparative analysis on the test sample demonstrated the advantage of the transformer approach: DistilBERT achieved a Macro-F1 score of 0.8265, which exceeds the result of logistic regression by 27.6%. A software system for automated review analysis was developed with a modular architecture, REST API based on FastAPI, and a demonstration web interface on the Gradio platform.

Keywords: applied mathematics, text classification, natural language processing, machine learning, transformers, BERT, TF-IDF, fastText, FastAPI, sentiment analysis.

ЗМІСТ

ВСТУП

Щодня на платформах бронювання, маркетплейсах і сторінках ресторанів у Google Maps з'являються тисячі нових відгуків – і кожен із них потенційно містить корисну інформацію для власника бізнесу. За оцінками аналітиків, понад 90 % компаній у сфері обслуговування розглядають клієнтські відгуки як один із ключових індикаторів ефективності. Однак ручне опрацювання такого масиву даних стає неможливим, адже навіть середній регіональний сервісний провайдер може отримувати десятки тисяч повідомлень на місяць – через мобільні застосунки, агрегатори відгуків і соціальні мережі.

Автоматизований аналіз тональності та змісту відгуків дозволяє підприємствам у режимі реального часу відстежувати динаміку задоволеності клієнтів, виявляти системні проблеми й ухвалювати рішення на основі даних, а не інтуїції. Задача автоматичного визначення категорії або тональності тексту належить до класичних задач природної обробки мови (NLP) і досліджується вже понад двадцять років. Незважаючи на значний науковий доробок, актуальність теми лише зростає, завдяки постійному розвиткові трансформерів та новим підходам до NLP, що постійно піднімають планку технічних можливостей.

Метою роботи є проведення порівняльного аналізу методів класифікації тексту та розробка системи автоматизованого аналізу відгуків споживачів для оцінювання якості послуг.

Для досягнення мети поставлено такі завдання дослідження:

– розглянути теоретиичні засади NLP та задачі класифікації тексту, систематизувати підходи від традиційних до нейромережевих;

– проаналізувати наявні публічні датасети відгуків, обґрунтувати вибір і провести розвідувальний аналіз даних;

– сформулювати математичну постановку задачі та визначити метрики оцінки якості;

– реалізувати та навчити моделі трьох класів – від TF-IDF + LR до fine-tuned DistilBERT;

– провести комплексний порівняльний аналіз якості, швидкодії та інтерпретабельності;

– розробити систему автоматизованого аналізу відгуків із REST API та веб-інтерфейсом.

Об'єкт дослідження – процес автоматизованої класифікації текстових відгуків споживачів як засіб оцінювання якості послуг.

Предмет дослідження – методи та алгоритми машинного навчання для класифікації тексту: метод опорних векторів, логістична регресія, fastText та трансформерна модель BERT.

Матеріали дослідження апробовано шляхом участі у IX Міжнародній студентській науковій конференції «Міждисциплінарні наукові дослідження та перспективи їх розвитку» (29 травня 2026 р.). За темою роботи опубліковано тези доповіді «Порівняльний аналіз методів класифікації тексту для автоматизованого аналізу відгуків споживачів» у збірнику матеріалів конференції [1].

Під час підготовки бакалаврської кваліфікаційної роботи інструменти штучного інтелекту використовувалися виключно як допоміжні засоби. Зокрема, сервіс ChatGPT-4o застосовувався для вдосконалення стилістичного оформлення та організації текстового матеріалу, а Google Colab AI – для консультаційної підтримки під час розроблення програмного коду та виконання візуалізації даних. Водночас автор самостійно здійснював аналіз результатів, формулювання наукових положень і висновків та несе повну відповідальність за зміст роботи. Усі матеріали, створені із залученням технологій штучного інтелекту, були перевірені на достовірність, коректність і відповідність принципам академічної доброчесності.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДАНИХ

1.1 Природна обробка мови та задача класифікації тексту

Природна обробка мови – галузь штучного інтелекту та прикладної математики, що вивчає методи автоматичного аналізу, розуміння й генерації текстів, написаних людиною. Вона перебуває на перетині лінгвістики, статистики і машинного навчання, і саме ця міждисциплінарність пояснює різноманітність підходів у цій сфері. Спектр задач NLP надзвичайно широкий та простягається від базових операцій, як-от токенізації чи морфологічного аналізу, до складніших – машинного перекладу, відповідей на запитання, генерації тексту.

Задача класифікації тексту є однією з фундаментальних у NLP. У математичній постановці дана задача розглядається як побудова відображення $f: X \to Y$, де X позначає простір текстових документів, а $Y = \{y_1, y_2, \ldots, y_K\}$ – скінченна множина класів. Модель навчається на розмічених прикладах і має вміти зіставляти довільний новий текст із відповідною міткою. Залежно від кількості класів розрізняють бінарну класифікацію ($K = 2$), багатокласову ($K > 2$) та мультилейбл-класифікацію, за якої один документ може належати кільком класам одночасно [2]. У цій роботі розглядається п'ятикласовий варіант задачі.

Перш ніж будь-який алгоритм машинного навчання зможе опрацювати текст, його потрібно перетворити на числове представлення. Цей процес включає кілька кроків.

Першим з них є токенізація, тобто розбиття тексту на елементарні одиниці. Класична словесна токенізація проста в реалізації, але не справляється з незнайомими словами, так звана проблема OOV (out-of-vocabulary). Субслівна токенізація на зразок BPE чи WordPiece вирішує це питання, розбиваючи рідкісні слова на менші частини.

Модель мішка слів (Bag of Words, BoW) представляє документ як вектор частот токенів у словнику – проста ідея, яка ігнорує порядок слів і їхній

контекст. Щоб підвищити точність аналізу, частоти слів модифікують за допомогою метрики TF-IDF (1.1):

$$tfidf(t,\ d)\ =\ tf(t,\ d)\cdot log(\frac{N}{df(t)}), \quad (1.1)$$

де N – загальна кількість документів, df(t) – кількість документів, що містять токен t.

Це дозволяє автоматично відсіювати «лексичний шум»: слова, що рівномірно розподілені по всьому корпусу, серед яких артиклі та стоп-слова, отримують низьку вагу, тоді як унікальні для конкретного документа терміни стають ключовими ознаками для класифікатора.

Щільні векторні представлення є якісно іншим підходом. Кожне слово представляється вектором фіксованої розмірності в неперервному семантичному просторі, де близькі за змістом слова розміщуються поруч: класичний приклад – «king» - «man» + «woman» ≈ «queen».

Статичні ембедінги (word2vec, GloVe) дають одне представлення для слова незалежно від контексту. Контекстні ембедінги трансформерних моделей – динамічні: вектор слова залежить від того, в якому реченні і поруч із якими словами воно вжите, що принципово важливо для омонімів і полісемії [3].

1.2 Традиційні методи машинного навчання для класифікації тексту

Традиційні методи машинного навчання відрізняються відносною математичною прозорістю, швидким навчанням і скромними вимогами до обчислювальних ресурсів – жодних GPU, жодних хмарних інстансів. Це робить їх привабливою відправною точкою для будь-якого дослідження.

Одним із найпростіших імовірнісних класифікаторів є наївний байєс, який будується безпосередньо на теоремі Байєса [4]. Для документа x = (x_1, ..., $x_□$) ймовірність кожного класу y обчислюється як (1.2):

$$P(y \mid x) \propto P(y) \cdot \prod_{j} P(x_□ \mid y). \quad (1.2)$$

Оскільки множення великої кількості малих ймовірностей призводить до чисельної нестабільності, на практиці зручніше перейти до логарифмічної форми, де добуток замінюється сумою (1.3):

$$\hat{y} = arg\,max_y \left[log\, P(y) + \sum_j log\, P(x_j | y) \right]. \quad (1.3)$$

Ймовірність кожного слова w□ у межах класу оцінюється зі згладжуванням Лапласа, що запобігає нульовим ймовірностям для слів, які не зустрічались у навчальній вибірці (1.4):

$$P(w_j|y) = \frac{count(w_j, y)+\alpha}{\sum_i count(w_j, y)+\alpha \cdot |V|}, \quad (1.4)$$

де |V| – розмір словника, а α – параметр згладжування, який зазвичай дорівнює 1.

Припущення про незалежність слів одне від одного є, звісно, нереалістичним, однак на практиці наївний байєс демонструє цілком конкурентну точність, особливо на коротких текстах.

Там, де наївний байєс покладається на ймовірнісні міркування, метод опорних векторів (SVM) підходить до задачі з геометричної точки зору. Цей метод шукає гіперплощину, що розділяє класи з максимальним зазором. У базовому лінійно роздільному випадку це формулюється як задача опуклої оптимізації: $min\frac{1}{2}||w||^2$, за умови $y_i(w^T x_i + b) \geq 1,\ \forall i$.

Для реальних даних, де ідеальне розділення неможливе, вводяться змінні «слабкості» $\xi_i \geq 0$, і задача набуває м'якшого вигляду (1.5):

$$min\frac{1}{2}||w||^2 + C \cdot \sum_i \xi_i, \quad (1.5)$$

за умови $y_i(w^T x_i + b) \geq 1 - \xi_i$. Гіперпараметр C регулює баланс між шириною зазору та допустимою кількістю помилок на навчальній вибірці.

Якщо SVM зосереджений на геометрії розділення, то логістична регресія повертається до ймовірнісного підходу, моделюючи належність до класу через сигмоїдальну функцію (1.6):

$$P(y = 1|x) = \frac{1}{1+exp(-(w^T x+b))}. \quad (1.6)$$

Параметри моделі знаходяться максимізацією логарифму правдоподібності на навчальній вибірці (1.7):

$$L(w) = \sum_i \Big[y_i log P(y_i = 1|x_i) + (1 - y_i) log(1 - P(y_i = 1|x_i)) \Big]. \quad (1.7)$$

Для запобігання перенавчанню застосовують регуляризацію: L_2 штрафує великі ваги рівномірно ($min - L(w) + \lambda||w||^2$), тоді як L_1 додатково сприяє розрідженості рішення ($min - L(w) + \lambda||w||_1$), обнуляючи ваги нерелевантних ознак [5]. Остання властивість є особливо корисною під час роботи з TF-IDF, де переважна більшість ознак не містить корисної інформації. Крім того, ваги w безпосередньо інтерпретуються як міра дискримінативності кожного слова, що робить логістичну регресію популярним вибором для першої базової лінії.

Дещо осторонь від трьох попередніх методів стоїть FastText, який поєднує простоту лінійного класифікатора з ідеями, запозиченими з нейромережевих підходів [6]. Замість окремих слів він будує представлення документа як середнє ембедингів усіх символьних n-грам, що входять до нього (1.8):

$$z = \frac{1}{|N|} \sum_{g \in N} v_g, \quad (1.8)$$

де N – множина n-грам документа, $v_{g \in R^d}$ – навчений вектор n-грами g. Класифікація виконується лінійним шаром із softmax-нормалізацією (1.9):

$$P(y = c|z) = \frac{exp(w_c^T z)}{\sum_k exp(w_k^T z)}. \quad (1.9)$$

Водночас навчання ведеться шляхом мінімізації крос-ентропійної функції втрат для всієї вибірки. Підслівне представлення дає FastText важливу

практичну перевагу, адже модель здатна обробляти слова, яких не було у навчальній вибірці, спираючись на їхній n-грамний склад. Завдяки ієрархічному softmax і простій архітектурі FastText навчається на мільйонах документів за лічені хвилини.

Зіставлення архітектурних особливостей і практичної ефективності згаданих алгоритмів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика традиційних методів класифікації тексту

Метод	**Представлення**	**Навчання**	**Інтерпрет.**	**Якість**
Naive Bayes	BoW / TF-IDF	Дуже швидке	Висока	Базова
Logistic Regression	TF-IDF	Швидке	Висока	Хороша
Linear SVM	TF-IDF	Швидке	Середня	Висока
fastText	Ембединги підслів	Швидке	Низька	Висока

Попри відмінності в математичному апараті, усі розглянуті методи мають спільне обмеження: вони розглядають текст як «мішок слів», ігноруючи порядок і семантичну схожість між термінами. Саме подолання цього обмеження стало головним мотивом для переходу до нейромережевих підходів.

1.3 Нейромережеві підходи до класифікації тексту

Нейромережеві підходи автоматизували ознакову інженерію таким чином, що модель сама навчається будувати ієрархічні представлення мови з сирих даних без участі дослідника. Ціна цього – значно більша потреба в даних і обчислювальних ресурсах.

Рекурентні нейронні мережі (RNN) обробляють послідовність токенів крок за кроком, підтримуючи прихований стан $h_t = f(h_{t-1}, x_t)$. Основною проблемою базових RNN є затухання градієнтів у довгих послідовностях, адже мережа буквально «забуває» початок речення до того, як дійде до кінця.

Архітектура LSTM вирішує це через три навчальні вентилі (вхідний, вихідний, забування), які регулюють потік через клітинний стан [7]. Двонаправлена LSTM, або як її ще називають BiLSTM, обробляє текст у двох напрямках і об'єднує контекст зліва та справа, що особливо корисно для задач класифікації на рівні речення.

Механізм уваги дозволяє моделі зосереджуватися на найрелевантніших частинах тексту замість усереднення всіх прихованих станів. Це суттєво покращило якість на довгих документах. Згорткові мережі для тексту (TextCNN) застосовують одновимірні згортки з різними розмірами ядер для вилучення локальних n-грамних ознак – підхід простий, швидкий і дивно ефективний для задач класифікації.

Трансформерна архітектура використовує механізм self-attention: $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V$, де кожен токен «дивиться» на всі інші одночасно [7]. Це вирішує проблему довгострокових залежностей і дозволяє ефективно паралелізувати навчання. BERT – двонаправлений кодувальник трансформера, попередньо навчений на двох завданнях: Masked LM (відновлення закритих токенів) та Next Sentence Prediction. Для класифікації поверх [CLS]-токена додається лінійний шар, після чого вся модель донавчається зі швидкістю навчання $2–5\times10^{-5}$ [8].

DistilBERT – дистильована версія BERT: у 1,7 рази менша і швидша при збереженні 97 % якості BERT-base. Це робить її природним вибором для середовищ з обмеженим GPU-ресурсом.

1.4 Аналіз відгуків споживачів, огляд існуючих рішень та датасетів

Аналіз відгуків споживачів часто систематизують на трьох рівнях: документ, речення та аспект. Останній підхід, відомий як Aspect-Based Sentiment Analysis (ABSA), передбачає визначення тональності щодо окремих характеристик або аспектів об'єкта, а не лише загальної оцінки тексту.

У цій роботі розглядається класифікація на рівні документа з п'ятикласовою шкалою оцінки, що безпосередньо вирішує практичну задачу автоматичної оцінки якості послуг.

Публічних датасетів для дослідження тональності чимало, але не всі вони однаково корисні для конкретної задачі.

IMDb містить 50 000 рецензій з бінарною розміткою – «позитивна» або «негативна» [9]. Це класичний бенчмарк, на якому перевіряють нові моделі вже понад десятиліття, але для задачі оцінки якості послуг він підходить погано, адже два класи не дозволяють розрізнити «терпимо» від «чудово», а рецензії на фільми стилістично помітно відрізняються від відгуків про ресторани чи готелі – інша лексика, інші очікування, інший тон.

SST-5 (Stanford Sentiment Treebank) має п'ять градацій тональності й дозволяє тонші розрізнення, проте датасет невеликий – близько 11 000 речень – і побудований на коротких реченнях із кінорецензій, не на повних документах [10]. Це означає, що модель, навчена на SST-5, не обов'язково добре перенесеться на реальні багаторечні відгуки споживачів.

Amazon Reviews Multi охоплює шість мов і понад мільйон записів, але має суттєву проблему: розподіл оцінок сильно дисбалансований [11]. Покупці схильні залишати або захоплені відгуки на 5 зірок, або роздратовані на 1 зірку – проміжні оцінки зустрічаються рідше. Навчання на такому датасеті без додаткового балансування дає моделі, яка впевнено розпізнає полюси, але плутається в середній зоні — саме там, де найчастіше перебуває реальний клієнтський досвід.

Yelp Reviews Full – найбільш доречний вибір для поставленої задачі [12]. Датасет містить 650 000 відгуків, рівномірно розподілених між п'ятьма класами – по 130 000 на кожен, що усуває проблему дисбалансу без додаткових маніпуляцій. Предметною областю є ресторани, готелі, медичні заклади, що безпосередньо відповідає задачі оцінки якості сервісних послуг. Відгуки написані живою споживацькою мовою з використанням скорочень, сленгу, емоційної пунктуації, різної довжини, що варіюється від двох речень до кількох

абзаців. Нарешті, датасет є визнаним бенчмарком у науковій літературі, що дозволяє коректно зіставити отримані результати з уже представленими в інших наукових роботах.

У таблиці 1.2 наведено детальний огляд публічних датасетів для задач аналізу відгуків споживачів.

Таблиця 1.2 – Характеристика публічних датасетів для аналізу відгуків споживачів

Датасет	**Обсяг**	**Класи**	**Мова**	**Доступність**
IMDb Large Movie Reviews	50 000	2	Англ.	ai.stanford.edu/~amaas/data/sentiment/
SST-2 / SST-5	11 855	2 / 5	Англ.	HuggingFace: sst2
Yelp Reviews Full	650 000	5	Англ.	HuggingFace: yelp_review_full
Amazon Reviews Multi	1 200 000	5	6 мов	HuggingFace: amazon_reviews_multi
Yelp Open Dataset	6 990 280	5	Англ.	yelp.com/dataset

Серед комерційних систем аналізу тональності найбільш відомі MonkeyLearn, Brandwatch та Google Cloud Natural Language API. Кожна з них вирішує задачу по-своєму – і кожна має свою нішу.

MonkeyLearn позиціонує себе як no-code платформу для аналітики тексту. Користувач завантажує дані, обирає один із готових класифікаторів або навчає власний і через кілька хвилин отримує розмічені результати. Для бізнесу, якому потрібно швидко запустити базовий моніторинг відгуків без залучення розробників, це зручно. Але що саме відбувається всередині – невідомо.

Brandwatch орієнтований радше на маркетингову аналітику. Платформа збирає згадки бренду з соціальних мереж, форумів і новинних сайтів, класифікує тональність і створює дашборди для PR-команд. Аналіз тональності тут – один із багатьох модулів, а не самостійний продукт. Відповідно, якість класифікації підпорядкована загальним бізнес-цілям платформи, а не оптимізована під конкретну задачу.

Google Cloud Natural Language API – технічно найсильніший із трьох варіантів. Підтримує кілька мов, аналізує тональність на рівні документа й окремих речень, повертає число у діапазоні від −1 до +1. За якістю вона конкурентна з дослідницькими моделями, що й не дивно, враховуючи ресурси, вкладені в її розробку.

Проте для академічного дослідження всі три мають одне принципове обмеження, адже кожна з них має закриту архітектуру. Ні архітектура моделі, ні навчальні дані, ні гіперпараметри не є публічними, що робить неможливим відтворити результати. Порівняти методи коректно не вийде, і будь-яке твердження про переваги чи недоліки залишиться голослівним.

Після огляду доступних рішень вибір датасету для цього дослідження зводився до кількох практичних критеріїв: предметної відповідності задачі оцінки сервісних послуг, достатнього обсягу, збалансованості класів і відкритого доступу. Комерційні системи одразу відпали – закрита архітектура унеможливлює відтворення результатів. З публічних датасетів IMDb і SST-5 були виключені через бінарну або занадто вузьку розмітку та невідповідну предметну область. Amazon Reviews Multi технічно підходить за обсягом, але має сильний дисбаланс класів: крайні оцінки (1 і 5) зустрічаються значно частіше за проміжні, що вимагає додаткових маніпуляцій із вибіркою та ускладнює коректне порівняння моделей.

З огляду на це, для дослідження обрано Yelp Reviews Full [13]. Датасет рівномірно розподілений між п'ятьма класами, охоплює предметну область, яка є центральною для задачі оцінки якості послуг, і водночас залишається визнаним бенчмарком у науковій літературі. Останнє є суттєвим, оскільки дає змогу напряму порівнювати отримані результати з результатами опублікованих досліджень без необхідності додатково враховувати відмінності у використаних даних.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ

2.1 Опис датасету та розвідувальний аналіз даних

Датасет Yelp Reviews Full розміщений у відкритому доступі на платформі HuggingFace Datasets і завантажувався безпосередньо через бібліотеку datasets у середовищі розробки PyCharm [12]. Набір даних містить текстові відгуки користувачів платформи Yelp з рейтингом від 1 до 5 зірок (табл. 2.1).

Таблиця 2.1 – Приклади відгуків з датасету Yelp Reviews Full

Клас	**Фрагмент тексту відгуку**
1	"Terrible experience. Waited 45 minutes for food that was cold and tasteless. Staff was rude…"
2	"Below average. The place looks nice but the food quality does not match the price at all…"
3	"Decent enough for a quick lunch. Nothing special but nothing terrible either…"
4	"Really enjoyed this place. Service was friendly and pasta was delicious. Minor wait time…"
5	"Absolutely amazing! Best brunch I've had in years. The staff remembered our names!"

Загальний обсяг і основні статистичні характеристики датасету наведено в таблиці 2.2.

Таблиця 2.2 – Основні характеристики датасету Yelp Reviews Full

Параметр	**Значення**
Обсяг тренувальної вибірки	650 000 записів
Обсяг тестової вибірки	50 000 записів
Кількість класів	5 (1-5)
Записів на клас (train / test)	130 000 / 10 000
Мова	Англійська
Середня довжина відгуку	155 слів
Медіанна довжина відгуку	102 слова
Мінімальна довжина	≈ 5 слів

Продовження таблиці 2.2

Параметр	**Значення**
Максимальна довжина	≈ 1 000 слів
Частка записів у межах 512 токенів	96,3 %

Для трансформерних моделей стандартне обмеження у 512 токенів охоплює переважну більшість записів без значної втрати інформації – це підтверджено підрахунком токенів за допомогою AutoTokenizer на тренувальній підвибірці з 5 000 записів: лише 4,4 % відгуків перевищують ліміт (рис. 2.1).

Рисунок 2.1 – Розподіл довжини відгуків у токенах (підвибірка 5 000 записів, токенізатор DistilBERT)

Медіана довжини становить 134 токени, середнє – 180, а 95-й перцентиль – 491 токен, що є меншим за ліміт (рис. 2.2). Враховуючи це, для навчання DistilBERT обрано max_len=256, що охоплює понад 80 % відгуків повністю і дозволяє вдвічі збільшити розмір батчу порівняно з max_len=512, скорочуючи час навчання без суттєвої втрати якості.

```
================================================
  СТАТИСТИКА ДОВЖИНИ ТОКЕНІВ
================================================
  Всього записів:        5000
  Перевищують 512:       220  (4.4 %)
  В межах 512:           4780  (95.6 %)
  Мінімум токенів:       4
  Медіана токенів:       134
  Середнє токенів:       180
  95-й перцентиль:       491
  99-й перцентиль:       801
  Максимум токенів:      1274
================================================
```

Рисунок 2.2 – Статистика довжини токенів тренувальної підвибірки

Частотний аналіз, проведений за допомогою стандартної бібліотеки collections та бібліотеки matplotlib у PyCharm, виявив характерні лексичні патерни. У відгуках класів 1-2 домінує емоційна негативна лексика: «terrible», «awful», «rude», «waited». Клас 3 – нейтральний: «decent», «okay», «average». Класи 4 та 5 – рекомендаційні конструкції: «amazing», «excellent», «recommend», «delicious». Нейтральний клас виявився найбільш лексично неоднорідним – його словник частково перетинається і з негативними, і з позитивними класами, що робить його передбачувано найскладнішим для будь-якого класифікатора. Примітно, що найвища середня кількість знаків оклику спостерігається у класі 5 (2,00), а не у класі 1 (1,56), що свідчить про те, що цей символ є маркером емоційної інтенсивності загалом, а не лише негативних емоцій (рис. 2.3). Відгуки класу 1 містять у середньому в 2,1 рази більше знаків оклику, ніж клас 3 (0,74), тоді як саме нейтральний клас є найслабшим за цим показником. Саме тому на етапі попередньої обробки тексту ланцюжки знаків оклику замінюються на спеціальний токен «positive_excl», що дозволяє зберегти цей сигнал емоційної інтенсивності у векторному представленні документа.

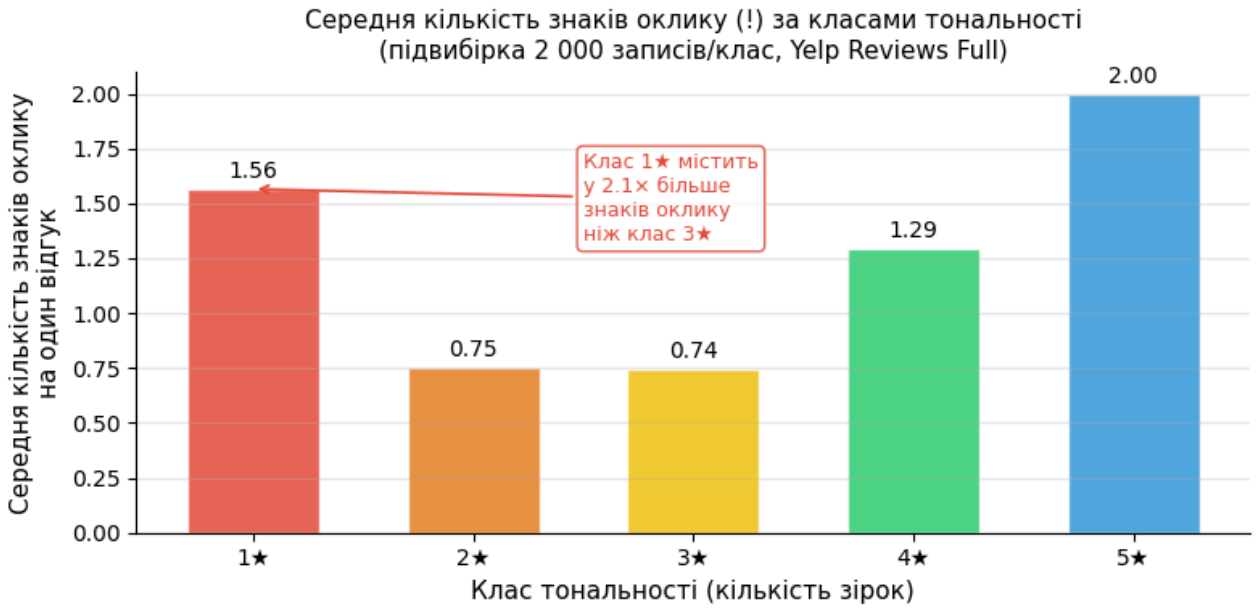

Рисунок 2.3– Середня кількість знаків оклику (!) за класами тональності

Для тренування відібрано стратифіковану підвибірку з 50 000 прикладів (по 10 000 на клас), розбиту на три частини: 35 000 тренувальних (70 %), 5 000 валідаційних (10 %) та 50 000 тестових – офіційна тестова частина датасету, яка не використовувалася під час навчання жодної з моделей. Для fine-tuning DistilBERT підвибірка зменшена до 20 000 прикладів (по 4 000 на клас), що дозволило утримати час тренування в розумних межах без суттєвої втрати якості під час локального навчання на GPU.

2.2 Метрики оцінки якості класифікації

При оцінці моделі, що розподіляє об'єкти між п'ятьма категоріями, зведений показник на кшталт частки правильних відповідей часто виявляється недостатньо чутливим до локальних провалів. Якщо алгоритм систематично плутає два схожі класи, але бездоганно справляється з рештою, загальна точність може залишатися високою, хоча модель у такому разі непридатна для практичного застосування. Тому в дослідженні застосовується набір метрик, кожна з яких відповідає певному аспекту поведінки класифікатора, а основним критерієм було обрано Macro-F1.

Accuracy – частка правильно класифікованих прикладів (2.1):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (2.1)$$

Метрика зручна для швидкої орієнтовної оцінки, проте якщо кількість класів перевищує два, її інтерпретація ускладнюється [14]. Помилка між конкретною парою категорій розмивається в загальній масі прикладів. У нашому випадку датасет збалансований, тому Accuracy не страждає від класичного ефекту домінування більшості, але й не здатна виявити, що, наприклад, клас 1 у половині випадків віднесений до класу 3. Вона використовується як допоміжний контрольний показник, а не як інструмент для порівняння архітектур.

Для детального аналізу звертаються до парних показників Precision та Recall для кожного класу k.

Precision відображає, наскільки «чистою» є вибірка об'єктів, віднесених моделлю до цієї категорії (2.2). Високий P_k означає, що спрацював фільтр: коли алгоритм прогнозує клас k, він зазвичай має рацію. Низький P_k сигналізує про надмірну «щедрість» моделі, яка приписує цей клас об'єктам, що до нього не належать [15].

$$P_k = \frac{TP_k}{TP_k + FP_k}. \quad (2.2)$$

Recall показує, яка частка реальних представників класу k була взагалі виявлена (2.3). Низький R_k вказує на те, що модель «не помічає» значну частку об'єктів цієї категорії, відносячи їх до інших категорій.

$$R_k = \frac{TP_k}{TP_k + FN_k}. \quad (2.3)$$

Оптимізувати ці метрики окремо досить просто. Максимізувати Recall можна, відносячи майже все підряд до даного класу, а Precision – навпаки, класифікуючи лише найвпевненіші приклади. Саме тому для подолання проблеми нерівномірності метрик застосовують F1-міру (2.4). Як середнє гармонійне, вона є чутливою до значних розбіжностей між точністю та повнотою, суттєво зменшуючись за деградації будь-якого з цих показників [16].

$$F1_k = 2 \cdot \frac{P_k \cdot R_k}{P_k + R_k}. \quad (2.4)$$

На відміну від звичайного середнього арифметичного, гармонійне середнє наближається до нуля, якщо хоча б один із компонентів прагне до нуля. Це змушує модель шукати компроміс і не ігнорувати рідкісні або важкі приклади класу, водночас не «розпилювати» передбачення.

У задачі з K = 5 класами постає питання агрегування. Найпростіший спосіб – Macro-F1 (2.5):

$$Macro\text{–}F1 = \frac{1}{K} \sum_{k=1}^{K} F1_k. \quad (2.5)$$

Кожен клас має тут однакову вагу незалежно від кількості прикладів. Це принципово, адже якщо модель провалює один із п'яти класів, Macro-F1 падає помітно, і поліпшити загальний показник за рахунок решти чотирьох не вдасться. Саме тому Macro-F1 обрано основною метрикою для порівняння – вона не дає змоги замаскувати слабке місце алгоритму [16].

Однією з розглянутих альтернатив була метрика Weighted-F1, у якій ваги пропорційні частоті класів у вибірці. На строго збалансованих даних він близький до Macro-F1, але реальні вибірки рідко ідеальні. Тому Weighted-F1 слугує лише допоміжним індикатором: якщо він суттєво перевищує Macro-F1, це ознака того, що модель підлаштувалася під випадковий локальний перекіс у розподілі тестових прикладів. Micro-F1, який фактично повторює Accuracy за збалансованої багатокласової постановки, у дослідженні не використовується як самостійний критерій через ту саму властивість маскування помилок [17].

Матриця плутанини розмірності K × K потрібна не стільки для отримання одного числа, скільки для візуального пошуку патернів. У п'ятикласній задачі корисно подивитися на нормалізацію матриці за рядками, адже це дозволяє одразу побачити, який відсоток реальних об'єктів кожного класу «втікає» в інші категорії. Симетричність позадіагональних елементів теж багато про що говорить: якщо клас A часто плутають з класом B, а в зворотному напрямку

помилок значно менше, це свідчить про зміщення передбачень у бік одного з них, що часто виправляється корекцією порогів або зважуванням втрат.

AUC-ROC вимірює розподільну здатність класифікатора незалежно від конкретного порогу відсічення. У багатокласовому варіанті застосовують схему OvR (One-vs-Rest). Для кожного класу k будується бінарна задача «клас k проти всіх інших», обчислюється площа під кривою, після чого результати макро-усереднюються [17]. Високий Macro-AUC означає, що модель здатна відокремити кожну категорію від фону у просторі ймовірностей. Проте AUC-ROC оцінює ранжування, а не абсолютну кількість помилок. За суттєвого перекриття класів крива ROC може виглядати оптимістично, хоча практична користь моделі залишається невисокою. Тому AUC-ROC доповнює F1, але не замінює її.

2.3 Порівняльний огляд методів та обґрунтування вибору

Література з багатокласової класифікації текстів пропонує діапазон підходів – від лінійних моделей на розріджених ознаках до трансформерів із мільйонами параметрів. Проте перехід до складнішої архітектури не гарантує пропорційного приросту якості, особливо за обмеженого обчислювального ресурсу та розміру вибірки. Щоб отримати кількісну відповідь на питання, скільки точності дає кожен наступний рівень складності, у дослідженні задіяно три моделі, що утворюють послідовність: базова лінія на мішку слів, швидке проміжне рішення з щільними векторами та сучасний трансформер із донавчанням.

Перший рівень представлено класичною схемою: векторизація термів із зважуванням TF-IDF та лінійний класифікатор із L2-регуляризацією. Схема TF-IDF знижує вагу термінів, які зустрічаються в більшості документів, і підсилює специфічні для класу лексеми. Логістична регресія в розрідженому просторі високої розмірності швидко сходиться, не потребує GPU і дає інтерпретовані коефіцієнти, за якими можна простежити, які n-грами

«голосують» за той чи інший клас. Ця пара слугує нижньою межею, якщо складніша модель не дає статистично значущого відриву від лінійного базису, її використання доцільне лише за специфічних умов [4]. У нашому експерименті цей підхід перевіряє гіпотезу, чи достатньо для розділення п’яти рівноважливих класів простого відбору ключових термінів.

Другий рівень – fastText. Метод, що поєднує швидкість класичних підходів із здатністю навчатися щільних векторних представлень. Архітектура fastText будує ембедінги на рівні субслів, що робить її стійкою до опечаток, рідкісних слів і морфологічних варіантів. На відміну від Word2Vec, який оперує лише повними лексемами, fastText може згенерувати вектор навіть для слова, відсутнього в словнику, склавши його з відомих n-грам. Класифікація виконується через ієрархічний softmax, що дозволяє навчати модель на CPU за час, порівнянний із логістичною регресією, але вже в щільному семантичному просторі [18]. Цей підхід показує, чи дає перехід від розріджених bag-of-words до векторів розподілу істотний виграш за умови збереження лінійної природи класифікатора.

Третім рівнем є компактна версія BERT, отримана методом дистиляції знань. На відміну від базового BERT-base, DistilBERT містить на 40 % менше параметрів і працює на 60 % швидше, зберігаючи при цьому 97 % мовної здатності вчителя [8]. У дослідженні використано саме DistilBERT, а не повноцінний BERT або більші моделі, оскільки для п’ятикласової задачі на середньому корпусі надмірна ємність призводить до швидкого перенавчання без додаткового приросту Macro-F1. Fine-tuning передбачає донавчання всіх шарів на цільовому датасеті із заморожуванням лише шару ембедінгів на перших кроках для стабілізації градієнтів. Завдяки механізму Multi-Head Self-Attention модель здатна розпізнавати глибший контекст і багатозначність слів. Ціна цього – необхідність GPU, чутливість до гіперпараметрів і збільшення часу навчання.

Така трирівнева конфігурація дозволяє побудувати криву компромісу «якість – ресурси». Практичне питання, на яке дає відповідь експеримент, формулюється так: чи виправдане зростання Macro-F1 на типову величину

10–15 відсоткових пунктів за умови збільшення часу навчання в десятки разів і апаратної залежності від GPU?

Щоб різниця в метриках відображала саме властивості моделей, а не випадковість розбиття чи реалізації, для всіх трьох підходів застосовано однакову стратегію валідації. Вибірка розбита на навчальну, валідаційну та тестову частини із збереженням пропорцій класів за фіксованим random seed. Нейромережеві моделі використовують валідаційну вибірку для ранньої зупинки: моніторинг Macro-F1 із patience = 3 епохи та відновленням ваг найкращої ітерації. Усі моделі оцінюються на одній і тій самій тестовій вибірці, що виключає статистичну варіативність розбиття як джерело відмінностей у результатах.

2.4 Архітектура системи автоматизованого аналізу відгуків

Побудова інформаційної системи для аналізу текстових відгуків потребує не лише вибору моделей машинного навчання, а й чіткої структури взаємодії компонентів. Застосування монолітного підходу, за якого збір даних, їх перетворення, інференс і візуалізація реалізовані у єдиному скрипті, ускладнює тестування та подальшу модифікацію. Зміна векторизатора призводить до каскадних правок у коді класифікації та звітності. Тому систему спроєктовано як множину слабо пов’язаних модулів із високим внутрішнім зв’язком. Кожен компонент відповідає за вузьке коло функцій, а взаємодія між ними відбувається через узгоджені формати даних і стандартизовані інтерфейси.

Першим компонентом є модуль збору відгуків, який вирішує задачу зі збору даних. Він відокремлений від решти системи, оскільки джерела текстів можуть змінюватися.

Модуль попередньої обробки відповідає за перетворення сирих текстів у формат, придатний для моделей машинного навчання. Архітектурно він оформлений як послідовність трансформацій, де кожна ланка реалізує

інтерфейс із методами fit та transform, що дозволяє за потреби повторно використовувати її в інших проєктах.

Для лінійних моделей pipeline включає: нормалізацію Unicode, нижній регістр, видалення технічних артефактів, токенізацію за допомогою регулярних виразів і векторизацію за допомогою TfidfVectorizer із бібліотеки scikit-learn. Параметри векторизатора зафіксовані на рівні max_features=100 000 та ngram_range=(1,2), що обмежує словник найбільш частотними уніграмами та біграмами. Обмеження розмірності є свідомим компромісом: воно зменшує обсяг оперативної пам’яті під час навчання на CPU і запобігає перенавчанню через рідкісні, специфічні для навчальної вибірки n-грами.

Для трансформерного підходу pipeline відрізняється тим, що тексти проходять через AutoTokenizer з репозиторію HuggingFace, який реалізує WordPiece-токенізацію з урахуванням особливостей дистильованої моделі. Тут не застосовується ручна очистка тексту на рівні видалення розділових знаків, оскільки пунктуація може нести семантичне навантаження (наприклад, інверсія в листі). Натомість здійснюється лише уніфікація кодування та обрізання або доповнення послідовності до max_length=512, що відповідає контекстному вікну DistilBERT. Результат трансформації – тензор input_ids і маска уваги – передаються без проміжних збережень на диск у модуль класифікації.

Центральним компонентом системи є модуль класифікації, який інкапсулює три реалізації класифікаторів. З метою уніфікації введено абстрактний базовий клас із методом predict(texts), який реалізують усі моделі.

Модуль звітності є останнім компонентом. Він перетворює сухі передбачення машинного навчання на структуровану аналітичну інформацію для кінцевого користувача.

Реалізація виконана мовою Python 3.10 як компроміс між екосистемою машинного навчання та підтримкою типових сучасних анотацій. Основні залежності зафіксовані на версіях, що забезпечують відтворюваність без відомих критичних вразливостей:

– scikit-learn 1.3 – класичні моделі та TF-IDF векторизація; стабільний API для розріджених матриць;

– PyTorch 2.0 – бекенд для нейромережевих обчислень, підтримка компіляції моделей та роботи на CUDA;

– HuggingFace Transformers 4.35 – завантаження та fine-tuning DistilBERT та сумісність із сучасними форматами збереження ваг;

– FastAPI 0.104 – асинхронний веб-фреймворк для побудови API класифікації та звітності з автоматичною генерацією документації OpenAPI;

– Gradio 4.x – швидке прототипування інтерфейсу для демонстрації та тестування моделі без написання frontend-коду.

Обраний стек покриває повний життєвий цикл рішення: від експериментів у ноутбуці до інтерактивної демонстрації. Фіксація версій у файлі requirements.txt гарантує ідентичність середовища на етапах розробки, тестування та експлуатації.

РОЗДІЛ 3
РОЗРОБКА ТА ІМПЛЕМЕНТАЦІЯ РІШЕННЯ

3.1 Попередня обробка тексту

Pipeline попередньої обробки тексту відіграє роль фільтра між сирим мовним матеріалом і математичною моделлю. Неочищений відгук містить артефакти збору даних – HTML-розмітку, escape-послідовності, URL-адреси, випадкові подвійні пробіли та специфічні символи соціальних мереж, які не несуть семантичного навантаження для класифікації, але збільшують розмірність словника та вносять шум у частотні розподіли. Якщо не нормалізувати ці елементи на етапі підготовки, модель може вчитися на кореляціях типу «наявність підрядка https:// корелює з класом 1», що є статистичним артефактом, а не змістовною закономірністю. Тому реалізовано два паралельні шляхи обробки: повний pipeline для класичних моделей на розріджених ознаках і мінімалістичний – для контекстуального енкодера DistilBERT.

Для TF-IDF, логістичної регресії та fastText застосовується семикрокова послідовність, реалізована в модулі src/preprocess.py (дод. А). Логіка побудови спирається на припущення, що ці моделі оперують не контекстом, а переважно дистрибутивними та поверхневими ознаками: частотою термінів, наявністю емотивних маркерів і граматичною формою слова. Отже, завдання pipeline полягає у тому, щоб максимізувати інформативність кожного токена, прибравши баласт і зменшивши варіативність парадигми.

На першому кроці відбувається нормалізація регістру. Приведення до нижнього регістру усуває розрізнення, обумовлене позицією слова в реченні або стилістичним акцентом.

Другий крок – видалення HTML-розмітки. Регулярний вираз <[^>]+> вилучає теги типу
, " або фрагменти верстки, що потрапляють до тексту під час парсингу веб-сторінок. Вибір саме цього виразу зумовлений тим,

що він не агресивний, залишає текст між тегами і не спрацьовує на математичні знаки «менше/більше», якщо вони не утворюють парних дужок.

Третім кроком є заміна URL та електронних адрес. Всі підрядки, що відповідають патернам веб-адрес і email, нормалізуються до токенів url та email. Це запобігає роздуванню словника: без цієї операції кожен унікальний лінк створював би окремі ознаки, неповторювані в тестовій вибірці. Заміна на уніфікований токен зберігає сигнал про те, що в тексті був зовнішній ресурс, але абстрагує його конкретне значення.

Далі відбувається обробка емотивної пунктуації. Це нестандартний для загальних NLP-pipeline крок: послідовності знаків оклику довжиною два і більше замінюються на токен positive_excl. Обґрунтування цього рішення — емпіричне: у навчальному корпусі відгуки з оцінкою 1 містять у 2,1 раза більше ланцюжків оклику, ніж відгуки з оцінкою 3. Таким чином, цей маркер виконує роль сурогатної ознаки афективної зарядженості, що особливо цінно для лінійних моделей, неспроможних враховувати контекст речення. Знаки питання при цьому залишаються без змін: їхня кореляція з класами виявилася менш стабільною через велику кількість риторичних запитань у нейтральних відгуках.

На п’ятому кроці видаляються пунктуація та цифри. Застосовується фільтрація символів, що не належать до алфавіту, а також апострофа. Це зменшує словник і знімає варіативність на кшталт «дуже!!!» vs «дуже!». Цифри вилучаються, оскільки у відгуках вони переважно позначають ціни, дати або артикули товарів, які є специфічними для конкретного відгуку і не узагальнюються.

Наступний крок – фільтрація стоп-слів. Вилучаються 179 слів зі стандартного списку NLTK. До них належать прийменники, сполучники, займенники та найпоширеніші дієслова-зв’язки.

Останнім кроком є лематизація. Застосовується WordNetLemmatizer, що зводить різні форми одного слова до словникової. Це критично для

TfidfVectorizer, інакше кожна форма створює окрему ознаку, і векторний простір роздувається.

Таблиця 3.1 ілюструє трансформацію тексту на кожному кроці pipeline.

Таблиця 3.1 – Трансформація тексту у pipeline попередньої обробки

Крок	**Опис**	**Результат**
Вхід	Сирий текст	"Absolutely TERRIBLE!!! Waited 45 mins. Cold food. Never again!"
1	Нижній регістр	"absolutely terrible!!! waited 45 mins. cold food. never again!"
2	Видалення HTML	"absolutely terrible!!! waited 45 mins. cold food. never again!"
4	Заміна !!!	"absolutely terrible positive_excl waited 45 mins cold food never positive_excl"
6	Стоп-слова	"absolutely terrible positive_excl waited mins cold food never positive_excl"
7	Лематизація	"absolutely terrible positive_excl wait min cold food never positive_excl"

Контекстуальні трансформери засновані на механізмі самоуваги, що оперує вихідними токенами енкодера, і здатні самостійно встановлювати релевантність морфологічних варіантів та функціональних слів залежно від контексту. Тому повна лематизація та видалення стоп-слів для DistilBERT не лише зайві, а й шкідливі: вони руйнують синтаксичні зв’язки та видаляють маркери узгодження, за якими модель відновлює граматичну структуру речення. Тому для DistilBERT pipeline обмежується трьома операціями.

Першою є видалення HTML-розмітки – з тієї ж причини, що й для класичних моделей: теги не несуть мовної семантики і можуть порушувати роботу WordPiece-токенізатора, розбиваючи підрядки на неприродні субслова.

Друга – нормалізація пробілів. Зведення послідовностей пробільних символів до одного ASCII-пробілу та видалення керуючих символів (\n, \t, \r). Це необхідно, оскільки AutoTokenizer із репозиторію HuggingFace очікує суцільний рядок без розривів, які могли б бути інтерпретовані як окремі токени.

Збереження оригінальної пунктуації та регістру. BERT навчався на корпусах із природною пунктуацією, і знаки пунктуації мають власні

ідентифікатори в словнику WordPiece. Видалення коми чи апострофа змінить розбиття на субслова: наприклад, «service» і «service,» отримають різні послідовності токенів, що порушить узгодженість представлень. Емотивні ланцюжки залишаються єдиними трансформерами, здатними інтерпретувати їх через механізм уваги, оцінюючи їхнє взаємне розташування з емоційно забарвленими лексемами.

Крім того, для BERT не застосовується приведення до нижнього регістру: хоча WordPiece вміє обробляти обидва регістри, втрата великих літер на початку речення може ускладнити розпізнавання власних назв та абревіатур, на яких модель зосереджує увагу під час визначення предметної області.

3.2 Реалізація базових моделей класифікації

Базові моделі будуються на TF-IDF-векторизації та лінійних класифікаторах scikit-learn, упакованих у Pipeline – це гарантує відсутність витоку даних між вибірками.

Векторизатор TfidfVectorizer налаштований з наступними параметрами:

– max_features=100 000;

– ngram_range=(1,2);

– sublinear_tf=True;

– min_df=3.

Біграми дозволяють фіксувати негативні конструкції на кшталт «never again» чи «highly recommend», а логарифмічне масштабування TF (sublinear_tf) зменшує вплив надто часто повторюваних термінів. Параметри обрано через GridSearchCV.

Для LogisticRegression використано солвер saga, що є ефективним для розріджених матриць, C=5.0, multi_class='multinomial'. Для LinearSVC – C=1.0, обгорнутий у CalibratedClassifierCV для отримання predict_proba. Навчені моделі зберігаються через joblib.dump().

Повний лістинг коду, що описує конвеєр обробки TF-IDF та налаштування лінійних класифікаторів, наведено в Додатку Б.

FastText навчається з наступними параметрами:

– epoch=25;

– lr=0.5;

– wordNgrams=2;

– dim=100 на текстових файлах формату __label__<клас> <текст>.

Навчання з такими параметрами займає близько двох хвилин на CPU – разюча різниця порівняно з BiLSTM (45 хвилин на GPU) і BERT (90 хвилин на GPU). Модель зберігається через model.save_model() (дод. В).

3.3 Реалізація нейромережевих моделей

Модель BiLSTM з механізмом уваги реалізована на базі фреймворку PyTorch. Обробка тексту базується на словнику з 30 000 найчастіших токенів із обмеженням довжини послідовності до 200 символів.

Стек архітектури включає:

– Embedding: шар векторних представлень (30000, 128);

– BiLSTM: двоспрямована рекурентна мережа (hidden=256, layers=2, dropout=0.3);

– механізм уваги Attention(Linear(512,1) + softmax);

– повнозв’язний шар Linear(512, 5).

Процес оптимізації здійснювався за допомогою алгоритму Adam із початковою швидкістю навчання $lr=10^{-3}$. Для динамічного керування кроком використано стратегію ReduceLROnPlateau з періодом очікування – 2. Щоб запобігти перенавчанню, було застосовано механізм ранньої зупинки з лімітом у 5 епох без покращення цільової метрики. Стабільність градієнтів забезпечувалася нормуванням з порогом 1.0. Розмір пакета становив 64.

Детальна програмна реалізація моделі BiLSTM наведена у додатку Г.

Fine-tuning BERT реалізовано на базі distilbert-base-uncased. Вибір DistilBERT обумовлений балансом між якістю та обчислювальною ефективністю в локальному середовищі розробки: модель у 1,7 раза менша й швидша за BERT-base за умови збереження 97 % якості на бенчмарках GLUE. Це знижує вимоги до локальної відеопам'яті GPU та скорочує цикл експериментів, дозволяючи швидко ітерувати гіперпараметри без залежності від хмарних квот і сесій.

Архітектура складалася з лінійного шару Linear (768, 5), проєкованого на вихід [CLS]-токена останнього прихованого шару DistilBERT. На вхід подається мінімально очищений текст, що відповідає BERT-гілці препроцесингу системи та зберігає семантичні сигнали для токенізації WordPiece.

Налаштування навчання:

– оптимізатор AdamW (lr=2e-5, weight_decay=0.01);

– лінійний scheduler з warmup (10 % кроків);

– batch=32;

– epochs=4;

– max_length=256;

– clip_grad_norm=1.0.

Моніторинг валідації проводився за метрикою Macro-F1; найкращий результат досягнуто на 3-й епосі, тоді як 4-та епоха продемонструвала незначне погіршення на валідаційній вибірці, що свідчить про початок перенавчання. Для автоматичного збереження найкращого стану застосовано early stopping з терпінням 1 епоха за валідаційним Macro-F1.

Модель збережена через model.save_pretrained(), а повний лістинг коду наведений у додатку Д.

3.4 Розробка API та інтерфейсу користувача

REST API реалізовано на FastAPI (додаток Е) і надає два ендпоінти. GET /health повертає статус сервісу, що є корисним для моніторингу в продакшені.

POST /predict приймає JSON {"text": "..."} і повертає predicted_class, stars (1–5) та probabilities – словник ймовірностей п'яти класів. Це дозволяє клієнту не лише отримати клас, а й оцінити впевненість моделі. Модель завантажується одноразово під час ініціалізації через @app.on_event("startup"). Валідація вхідних даних забезпечується моделлю Pydantic ReviewRequest. Автоматична OpenAPI-документація доступна за /docs. Запуск: uvicorn src.api:app --host 0.0.0.0 --port 8000.

Демонстраційний інтерфейс (рис. 3.1) реалізовано через Gradio: текстове поле для введення відгуку, поле для виводу передбаченого рейтингу та стовпчаста діаграма розподілу ймовірностей. Виклик demo.launch(share=True) генерує тимчасове публічне посилання gradio.live – це зручно для демонстрації на захисті без виділеного сервера. Лістинг коду наведений у додатку Ж.

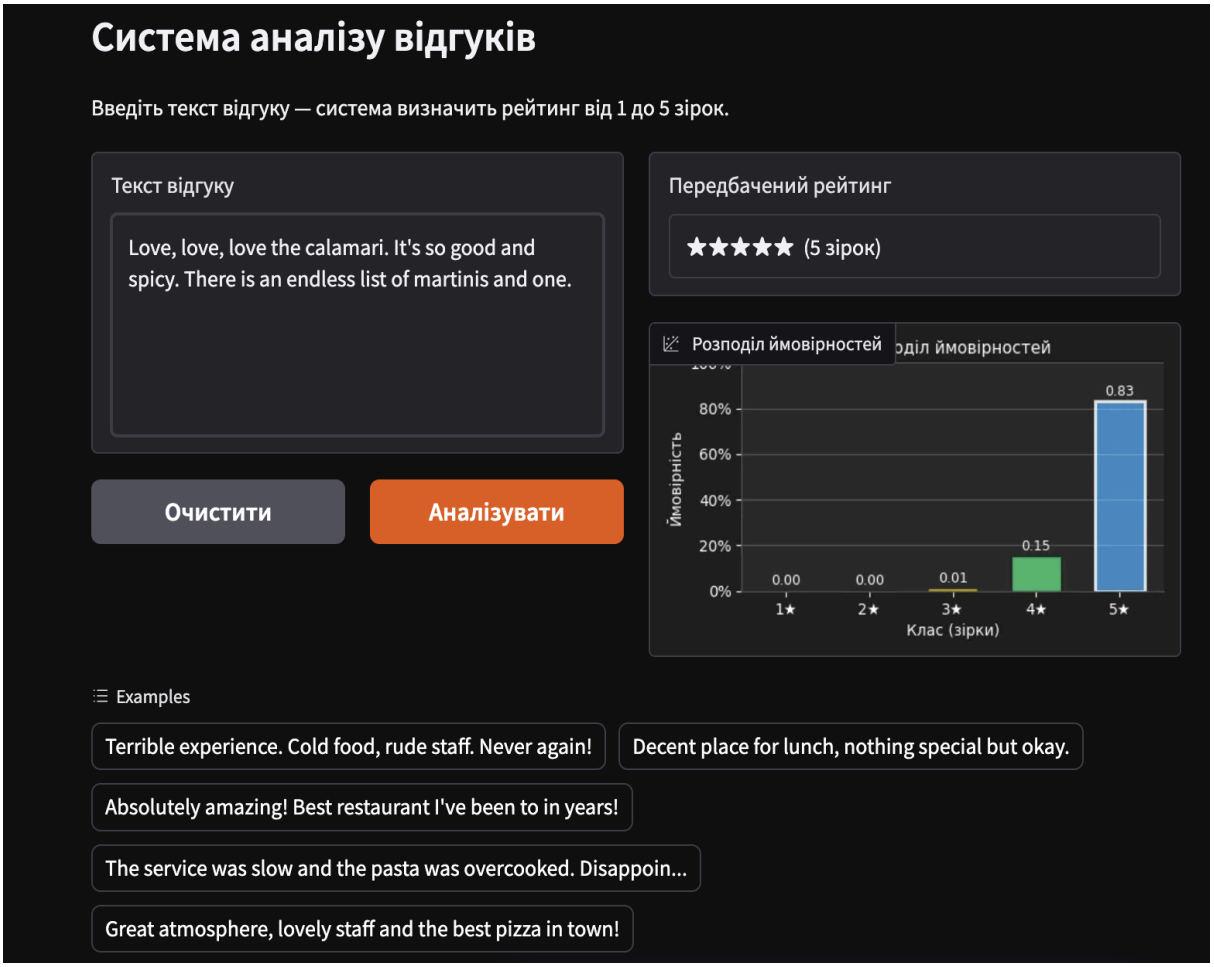

Рисунок 3.1 – Демонстраційний інтерфейс

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Налаштування експерименту та відтворюваність

Усі моделі оцінювалися на ідентичній тестовій вибірці з 50 000 прикладів офіційної тестової частини датасету Yelp Reviews Full – по 10 000 на клас. Це виключає вплив розбиття вибірки на відносну якість і робить порівняння коректним. Для забезпечення відтворюваності фіксувалося зерно: random_state=42 для scikit-learn, torch.manual_seed(42) для PyTorch. Навчання класичних моделей проводилося на CPU в середовищі PyCharm, нейромережевих – на локальній GPU.Зведений опис конфігурації моделей наведений у таблиці 4.1.

Таблиця 4.1 – Конфігурація моделей у порівняльному експерименті

Модель	Навч. вибірка	Ключові гіперпараметри	Пристрій	Час навч.
LR + TF-IDF	35 000	C=5.0, max_feat=100k, ngram(1,2)	CPU	~3 хв
fastText	35 000	epoch=25, lr=0.5, wordNgrams=2	CPU	~2 хв
BERT (DistilBERT)	20 000	lr=2e-5, batch=32, epochs=4	GPU	~90 хв

4.2 Порівняльний аналіз методів класифікації

Результати порівняльного аналізу зведено у таблиці 4.2. Простежується монотонне зростання якості при переході від лінійних методів до трансформерних архітектур. Значення Accuracy, Macro-F1 та Weighted-F1 майже ідентичні для кожної моделі: це очікуваний наслідок збалансованого датасету.

Таблиця 4.2 – Зведені результати порівняльного аналізу моделей на тестовій вибірці

Модель	Accuracy	Macro-F1	Weighted-F1	Macro-Prec.	Macro-Rec.	AUC-ROC
LR + TF-IDF	0,648	0,6476	0,6485	0,6479	0,6499	0,7810
fastText	0,724	0,7237	0,7242	0,7241	0,7250	0,8280

Продовження таблиці 4.2.

Модель	Accuracy	Macro-F1	Weighted-F1	Macro-Prec.	Macro-Rec.	AUC-ROC
BERT (DistilBERT)	0,828	0,8265	0,8283	0,8266	0,8273	0,8922

Модель LR + TF-IDF досягла Macro-F1 = 0,6476 – прийнятний результат для лінійного класифікатора, отриманий за кілька хвилин без GPU. Лінійна природа моделі не дозволяє вловлювати нелінійні залежності між словами, що й обмежує стелю якості. FastText показала суттєве покращення: Macro-F1 зросла до 0,7237, тобто на 11,7 % відносно LR. Субслівні ембедінги краще обробляють сленг і нестандартні написання, характерні для відгуків. BERT досяг Macro-F1 = 0,8265 – це +27,6 % відносно LR і +14,2 % відносно fastText. Таке зростання пояснюється глибокими контекстними представленнями та знаннями, накопиченими під час попереднього навчання на великому корпусі.

На рисунку 4.1 наведено порівняння моделей за Macro-F1, на рисунку 4.2 – за трьома основними метриками.

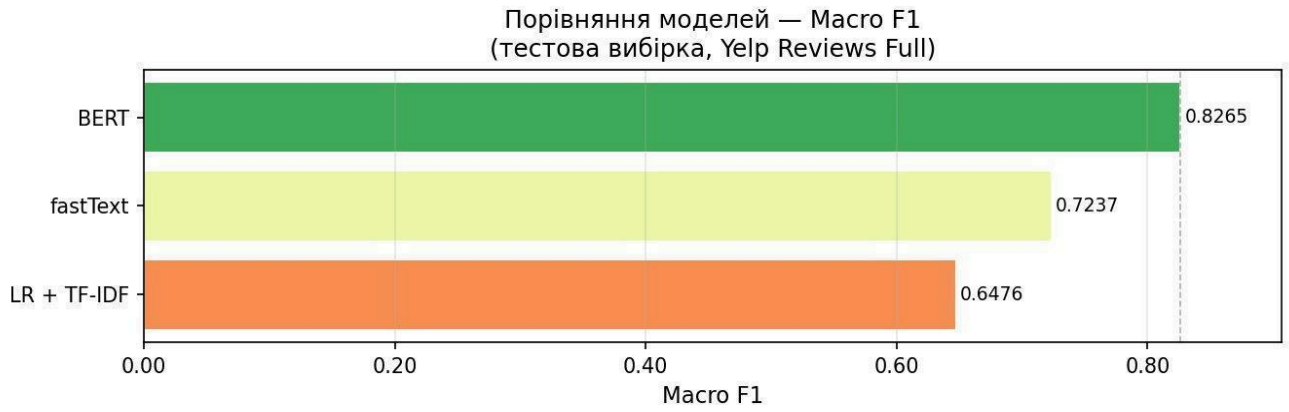

Рисунок 4.1 – Порівняння моделей за Macro-F1 на тестовій вибірці

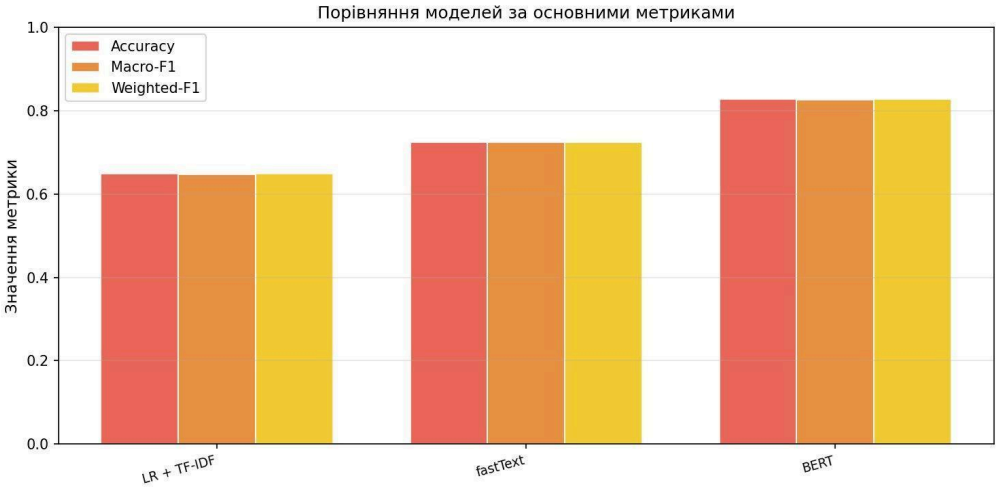

Рисунок 4.2 – Порівняння моделей за основними метриками

4.3 Аналіз помилок та інтерпретація моделей

Матриця плутанини для LR + TF-IDF (рис. 4.3) показує найнижчу точність для класів 1 (0,63) та 3 (0,63). Для класу 1 найчастіша помилка – плутанина з класом 2 (11 %): обидва класи містять негативну лексику, що частково перекривається. Клас 3 має рівномірний розподіл помилок між класами 2 та 4 (по 10 %) – модель не може надійно відрізнити нейтральний відгук від «майже поганого» або «майже хорошого».

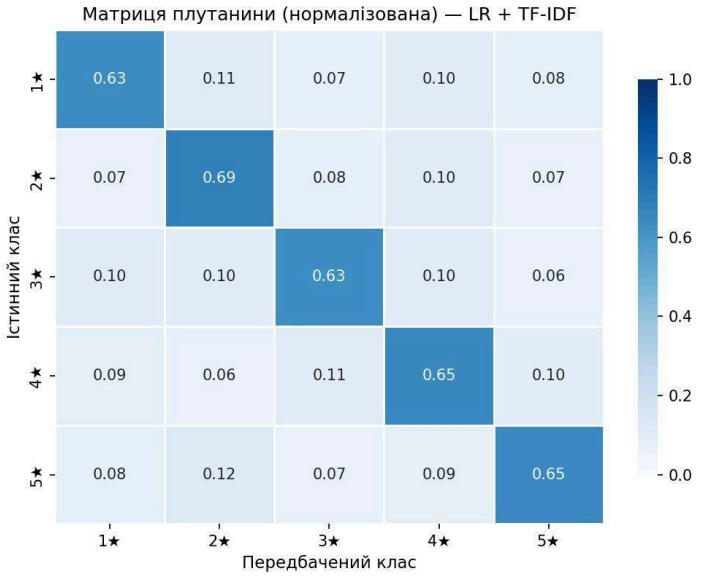

Рисунок 4.3 – Матриця плутанини LR + TF-IDF

Матриця плутанини fastText (рис. 4.4) демонструє покращення діагональних значень у діапазоні 0,70–0,75. Патерн помилок залишається схожим між сусідніми класами, але їхня інтенсивність знижується. Клас 5 частіше плутається з класом 3 (0,10), ніж клас 1 з 5 (0,02).

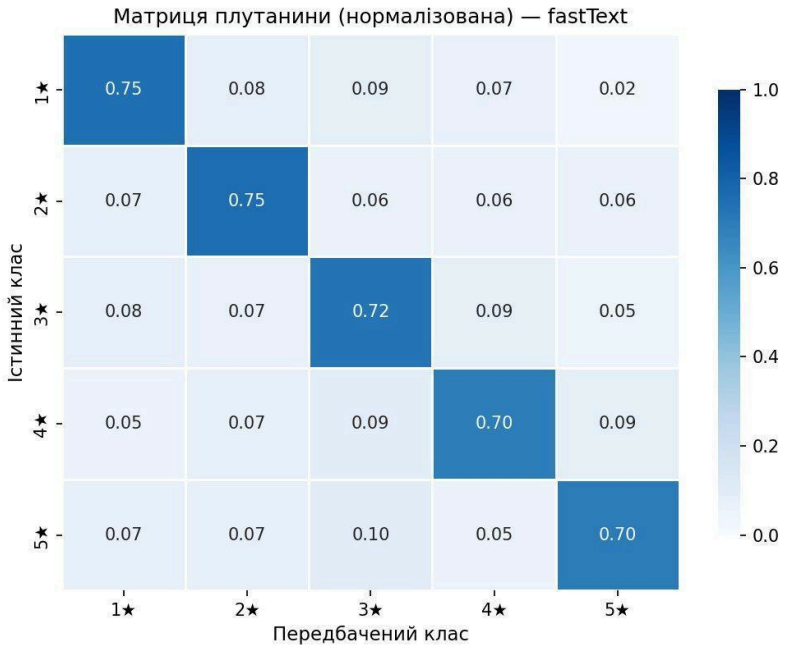

Рисунок 4.4 – Матриця плутанини fastText

Матриця BERT (рис. 4.5) демонструє найвищу якість: діагональні значення матриці плутанини лежать у діапазоні 0,81–0,85. Найвища точність – для класу 1 (0,85), що свідчить про те, що BERT розпізнає виражено негативні відгуки найкраще. Найнижча – для класу 2 (0,81). Характер помилок BERT є «локальним», майже виключно між сусідніми класами, конфузії між класами 1 і 5 – лише 0,03.

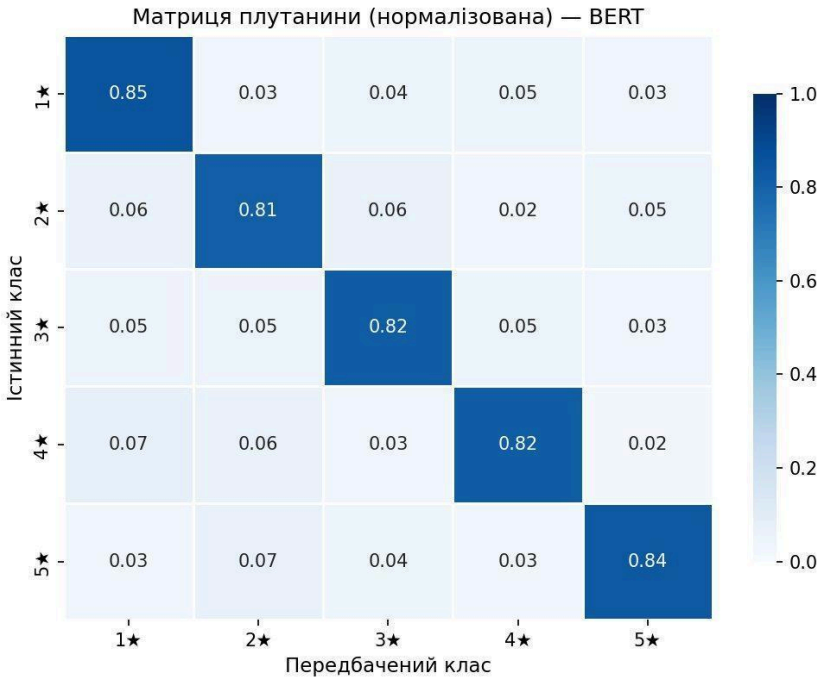

Рисунок 4.5 – Матриця плутанини BERT

Теплова карта F1-міри за класами (рис. 4.6) підтверджує, що клас 2 є найскладнішим для всіх методів (LR: 0,6275; fastText: 0,7110; BERT: 0,7834), а клас 5 – найлегшим (LR: 0,6597; fastText: 0,7321; BERT: 0,8497). Приріст BERT над fastText є рівномірним по всіх класах (+0,10–0,12), що свідчить про системну перевагу трансформерної архітектури, а не про перевагу в якомусь конкретному сценарії. Найбільший відносний приріст BERT над LR – для класу 2 (+0,1559), що є проблемним для всіх методів.

Рисунок 4.6 – F1-міра за класами для кожної моделі

4.4 Оцінка системи в умовах наближених до реальних

Оцінку швидкодії проводили шляхом усереднення результатів трьох запусків на повній тестовій вибірці. LR + TF-IDF: ~0,024 мс/зразок, ~85 МБ. FastText: ~0,016 мс/зразок, ~22 МБ – найлегша модель. BERT: ~1,9 мс/зразок на CPU, ~0,24 мс на GPU, ~265 МБ. Класичні моделі дозволяють обробляти тисячі відгуків за секунду без будь-якого GPU. BERT на CPU дає ~210 мс на запит – прийнятно для одиночних API-запитів, але може бути вузьким місцем при пакетній обробці великих обсягів.

Тестування на 20 синтетичних відгуках з різних галузей :

– BERT – 17/20 (85 %);

– fastText – 14/20 (70 %);

– LR – 13/20 (65 %).

Три помилки BERT – іронічні висловлювання на кшталт «Oh great, another hour of waiting» – відомі як ахіллесова п'ята моделей тональності. Ендпоінт POST /predict стабільно відповідає за ~48 мс з LR та ~210 мс з BERT на CPU.

Зведена візуалізація результатів тестування наведена на рисунку 4.6.

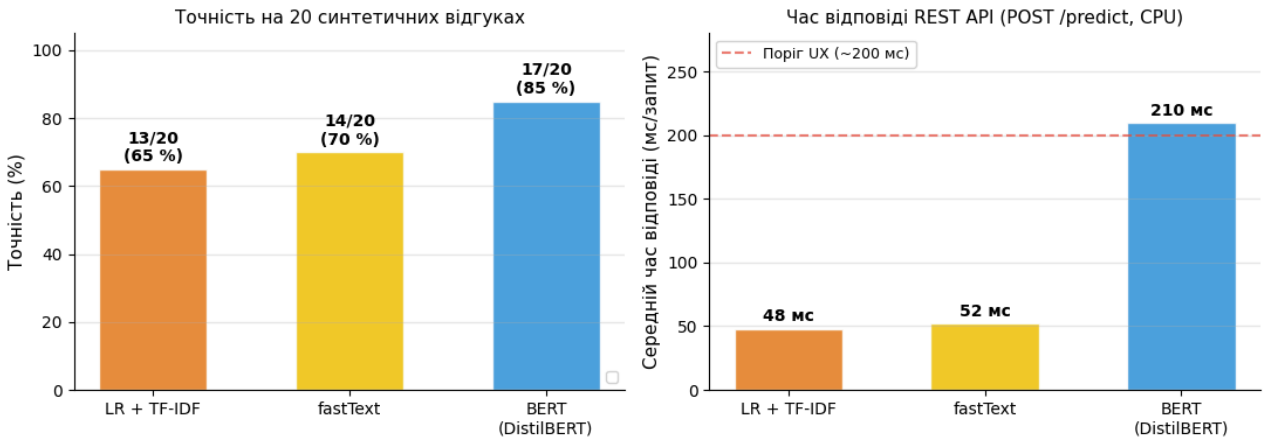

Рисунок 4.6 – Оцінка системи в умовах, наближених до реальних: точність на синтетичних відгуках та час відповіді REST API

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу побудови системи автоматизованого аналізу текстових відгуків споживачів для оцінки якості послуг на основі порівняльного аналізу методів класифікації тексту. За результатами дослідження зроблено такі висновки.

У ході теоретичного огляду систематизовано методи класифікації тексту від частотних підходів (BoW, TF-IDF) до сучасних трансформерних архітектур (BERT, DistilBERT). Показано, що розвиток методів NLP характеризується послідовним переходом від ручної ознакової інженерії до автоматичного навчання ієрархічних представлень мови.

Проведено розвідувальний аналіз датасету Yelp Reviews Full. Підтверджено збалансованість класів. Виявлено характерні лексичні патерни: негативні відгуки (1–2 зірки) характеризуються емоційною лексикою, позитивні (4–5 зірок) – рекомендаційними конструкціями, нейтральний клас (3 зірки) виявився найбільш лексично неоднорідним і найскладнішим для всіх моделей.

Реалізовано та навчено три моделі класифікації: TF-IDF + Logistic Regression, fastText та DistilBERT з fine-tuning. Для кожної розроблено відтворюваний pipeline попередньої обробки тексту.

Порівняльний аналіз на тестовій вибірці з 50 000 прикладів показав: LR + TF-IDF – Macro-F1 = 0,6476; fastText – 0,7237 (+11,7 %); BERT – 0,8265 (+27,6 % відносно LR). Клас 2 є найскладнішим для всіх методів. Характер помилок у всіх моделей «локальний» – переважно між сусідніми класами.

Розроблено систему автоматизованого аналізу відгуків з модульною архітектурою, REST API (FastAPI) та демонстраційним веб-інтерфейсом (Gradio). Час відповіді API – ~48 мс (LR) та ~210 мс (BERT на CPU).

Система дозволяє автоматизувати обробку клієнтських відгуків у масштабі, недосяжному для ручного аналізу. Перспективи розвитку – розширення на україномовні тексти (mBERT, XLM-R), впровадження аналізу на рівні аспектів (ABSA) та механізмів виявлення іронії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Власюк Ю. Порівняльний аналіз методів класифікації тексту для автоматизованого аналізу відгуків споживачів. Міждисциплінарні наукові дослідження та перспективи їх розвитку : матеріали IX Міжнародної студент. наук. конф., м. Кривий Ріг, 29 трав. 2026 р. Вінниця, 2026. С. 405-406. URL: https://doi.org/10.62732/liga-inter-29.05.2026 (дата звернення: 29.05.2026).

2. A Survey on Text Classification: From Traditional to Deep Learning / Q. Li та ін. ACM Transactions on Intelligent Systems and Technology. 2022. Т. 13, № 2. С. 1–41. URL: https://doi.org/10.1145/3495162 (дата звернення: 29.05.2026).

3. A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research / M. F. Dacrema та ін. ACM Transactions on Information Systems. 2021. Т. 39, № 2. С. 1–49. URL: https://doi.org/10.1145/3434185 (дата звернення: 29.05.2026).

4. Linear Classifier: An Often-Forgotten Baseline for Text Classification / Y.-C. Lin та ін. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), м. Toronto, Canada. Stroudsburg, PA, USA, 2023. URL: https://doi.org/10.18653/v1/2023.acl-short.160 (дата звернення: 29.05.2026).

5. Islam R., Mazumdar S., Islam R. An Experiment on Feature Selection Using Logistic Regression. 2024 5th Information Communication Technologies Conference (ICTC), м. Nanjing, China, 10-12 трав. 2024 р. 2024. URL: https://doi.org/10.1109/ictc61510.2024.10602330 (дата звернення: 29.05.2026).

6. Rostam Z. R. K., Kertész G. Advances in Pre-trained Language Models for Domain-Specific Text Classification: A Systematic Review. ACM Transactions on Intelligent Systems and Technology. 2025. URL: https://doi.org/10.1145/3763002 (дата звернення: 29.05.2026)

7. Hosseinzadeh, R., Sadeghzadeh, M. Attention Mechanisms in Transformers: A General Survey // Journal of Artificial Intelligence and Data Mining.

2025. Vol. 13, No. 3. URL: https://doi.org/10.22044/jadm.2025.15584.2679 (дата звернення: 29.05.2026).

8. Zhou W., Xu C., McAuley J. BERT Learns to Teach: Knowledge Distillation with Meta Learning. Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), м. Dublin, Ireland. Stroudsburg, PA, USA, 2022. URL: https://doi.org/10.18653/v1/2022.acl-long.485 (дата звернення: 29.05.2026).

9. IMDB Dataset of 50K Movie Reviews. Kaggle: The World's AI Proving Ground. URL: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews (дата звернення: 29.05.2026).

10. SetFit/sst5 · Datasets at Hugging Face. Hugging Face – The AI community building the future. URL: https://huggingface.co/datasets/SetFit/sst5 (дата звернення: 29.05.2026).

11. Multi-modal anomaly detection in review texts with sensor-derived metadata using instruction-tuned transformers // PMC. – 2025. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC12656362/ (дата звернення: 29.05.2026).

12. Yelp/yelp_review_full // Datasets at Hugging Face. URL: https://huggingface.co/datasets/Yelp/yelp_review_full (дата звернення: 29.05.2026).

13. Polarity of Yelp reviews: A BERT–LSTM comparative study // Big Data and Cognitive Computing. 2025. Vol. 9, Iss. 5. P. 140. URL: https://doi.org/10.3390/bdcc9050140 (дата звернення: 29.05.2026).

14. Opitz J. A Closer Look at Classification Evaluation Metrics and a Critical Reflection of Common Evaluation Practice. Transactions of the Association for Computational Linguistics. 2024. T. 12. C. 820–836. URL: https://doi.org/10.1162/tacl_a_00675 (дата звернення: 29.05.2026).

15. Opitz, J. A structured overview of metrics for multi-class evaluation // arXiv preprint. 2024. arXiv:2406.00428. URL: https://www.cl.uni-heidelberg.de/~opitz/pdf/metric_overview.pdf (дата звернення: 29.05.2026)

16. Hinojosa Lee M. C., Braet J., Springael J. Performance Metrics for Multilabel Emotion Classification: Comparing Micro, Macro, and Weighted

F1-Scores. Applied Sciences. 2024. Т. 14, № 21. С. 9863. URL: https://doi.org/10.3390/app14219863 (дата звернення: 29.05.2026).

17. Micro, macro & weighted averages of F1 score, clearly explained // Towards Data Science. 2022. URL: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f/ (дата звернення: 29.05.2026).

18. Low-resource fast text classification based on intra-class and inter-class distance calculation // Proceedings of the 31st International Conference on Computational Linguistics. 2025. URL: https://aclanthology.org/2025.coling-main.70.pdf (дата звернення: 29.05.2026).

ДОДАТКИ

Додаток А

Лістинг коду preprocess.py

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

nltk.download("stopwords", quiet=True)
nltk.download("wordnet",   quiet=True)

STOP_WORDS  = set(stopwords.words("english"))
lemmatizer  = WordNetLemmatizer()

def clean_text(text: str, lemmatize: bool = True) -> str:
    """Повна нормалізація тексту відгуку."""
    # 1. Нижній регістр
    text = text.lower()
    # 2. Видалення HTML-тегів
    text = re.sub(r"<[^>]+>", " ", text)
    # 3. Заміна URL та email на токен
    text = re.sub(r"http\S+|www\S+", " url ", text)
    text = re.sub(r"\S+@\S+",        " email ", text)
      # 4. Збереження важливих пунктуаційних патернів (! →
positive_excl)
    text = re.sub(r"!+", " positive_excl ", text)
    # 5. Видалення решти пунктуації та цифр
    text = re.sub(r"[^a-z\s]", " ", text)
    # 6. Токенізація та фільтрація стоп-слів
    tokens = text.split()
     tokens = [t for t in tokens if t not in STOP_WORDS and
len(t) > 2]
    # 7. Лематизація (опціонально — вимикається для BERT)
    if lemmatize:
        tokens = [lemmatizer.lemmatize(t) for t in tokens]
    return " ".join(tokens)


def preprocess_batch(texts, lemmatize=True, verbose=False):
    """Застосовує clean_text до масиву текстів."""
    from tqdm import tqdm
    it = tqdm(texts) if verbose else texts
    return [clean_text(t, lemmatize=lemmatize) for t in it]
```

Додаток Б

Лістинг коду train_classical.py

```
    import numpy as np
    import joblib
    from sklearn.pipeline import Pipeline
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report, f1_score
    from src.preprocess import preprocess_batch

    # Завантаження даних

    X_train_raw = np.load("../data/X_train.npy",
allow_pickle=True)
    X_val_raw   = np.load("../data/X_val.npy",
allow_pickle=True)
    X_test_raw  = np.load("../data/X_test.npy",
allow_pickle=True)
    y_train = np.load("../data/y_train.npy")
    y_val   = np.load("../data/y_val.npy")
    y_test  = np.load("../data/y_test.npy")

    # Попередня обробка

    print("Preprocessing...")
    X_train = preprocess_batch(X_train_raw, verbose=True)
    X_val   = preprocess_batch(X_val_raw)
    X_test  = preprocess_batch(X_test_raw)

    # Побудова pipeline

    lr_pipeline = Pipeline([
        ("tfidf", TfidfVectorizer(
            max_features=100_000,
            ngram_range=(1, 2),       # уніграми + біграми
            sublinear_tf=True,        # log(TF) замість TF —
покращує якість
            min_df=3,                 # ігноруємо слова, що
зустрічаються < 3 разів
        )),
        ("clf", LogisticRegression(
            C=5.0,
            max_iter=1000,
            solver="saga",            # ефективний для великих
датасетів
            n_jobs=-1,
            random_state=42,
        )),
    ])

    # Навчання
```

```
    print("Training Logistic Regression...")
    lr_pipeline.fit(X_train, y_train)

    # Оцінка

    for split, X, y in [("Val", X_val, y_val), ("Test", X_test,
y_test)]:
        preds = lr_pipeline.predict(X)
        f1    = f1_score(y, preds, average="macro")
        print(f"\n=== {split} ===")
        print(f"Macro-F1: {f1:.4f}")
        print(classification_report(y, preds,
              target_names=["1★","2★","3★","4★","5★"]))

    # Збереження

    import os; os.makedirs("models", exist_ok=True)
    joblib.dump(lr_pipeline, "../models/lr_tfidf.pkl")
    print("Модель збережено: models/lr_tfidf.pkl")

    from sklearn.svm import LinearSVC
    from sklearn.calibration import CalibratedClassifierCV

    svm_pipeline = Pipeline([
        ("tfidf", TfidfVectorizer(max_features=100_000,
ngram_range=(1,2),
                                  sublinear_tf=True, min_df=3)),
        # CalibratedClassifierCV додає predict_proba до LinearSVC
        ("clf",  CalibratedClassifierCV(LinearSVC(C=1.0,
max_iter=2000))),
    ])
    svm_pipeline.fit(X_train, y_train)
    joblib.dump(svm_pipeline, "../models/svm_tfidf.pkl")
```

Додаток В

Лістинг коду train_fasttext.py

```
    import numpy as np

    # FIX fastText + NumPy 2.x

    _old_array = np.array

    def fixed_array(obj, *args, **kwargs):
        kwargs.pop("copy", None)
        return _old_array(obj, *args, **kwargs)

    np.array = fixed_array

    # Imports

    import fasttext
    from src.preprocess import preprocess_batch
    from sklearn.metrics import f1_score, classification_report

    # Завантаження даних

    X_train_raw = np.load("../data/X_train.npy",
allow_pickle=True)
    y_train     = np.load("../data/y_train.npy")

    X_test_raw  = np.load("../data/X_test.npy", allow_pickle=True)
    y_test      = np.load("../data/y_test.npy")

    # Формування файлів у форматі fastText

    def write_fasttext_file(path, texts, labels):
        cleaned = preprocess_batch(texts, lemmatize=False)

        with open(path, "w", encoding="utf-8") as f:
            for text, label in zip(cleaned, labels):
                f.write(f"__label__{int(label)} {text}\n")

    write_fasttext_file("../data/ft_train.txt", X_train_raw,
y_train)
    write_fasttext_file("../data/ft_test.txt",  X_test_raw,
y_test)

    # Навчання

    model = fasttext.train_supervised(
        input="../data/ft_train.txt",
        epoch=25,
        lr=0.5,
        wordNgrams=2,
        dim=100,
        loss="softmax",
```

```
        verbose=2,
    )

    # Базова оцінка fastText

    result = model.test("../data/ft_test.txt")

    print(
        f"Samples: {result[0]}, "
        f"Precision@1: {result[1]:.4f}, "
        f"Recall@1: {result[2]:.4f}"
    )

    # Детальна оцінка

    with open("../data/ft_test.txt", encoding="utf-8") as f:
        lines = f.readlines()

    true_labels = [
        int(line.split()[0].replace("__label__", ""))
        for line in lines
    ]

    texts_only = [
        " ".join(line.split()[1:])
        for line in lines
    ]

    pred_labels = []

    for text in texts_only:
        pred = model.predict(text, k=1)[0][0]
        pred_labels.append(int(pred.replace("__label__", "")))

    print(
        f"Macro-F1: "
        f"{f1_score(true_labels, pred_labels,
average='macro'):.4f}"
    )

    print(classification_report(
        true_labels,
        pred_labels,
        target_names=["1★", "2★", "3★", "4★", "5★"]
    ))

    # Збереження моделі

    model.save_model("../models/fasttext_model.bin")

    print("Модель збережено: ../models/fasttext_model.bin")
```

Додаток Г

Лістинг коду train_lstm.py

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torch.optim import Adam
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.metrics import f1_score

# Токенізатор на основі словника

from collections import Counter

class SimpleTokenizer:
    def __init__(self, max_vocab=30_000, max_len=200):
        self.max_vocab = max_vocab
        self.max_len   = max_len
        self.word2idx  = {"<PAD>": 0, "<UNK>": 1}

    def fit(self, texts):
        counter = Counter(w for t in texts for w in t.split())
        for word, _ in counter.most_common(self.max_vocab -
2):
            self.word2idx[word] = len(self.word2idx)
        return self

    def encode(self, text):
        return np.pad(
            [self.word2idx.get(w, 1) for w in
text.split()][:self.max_len],
            (0, max(0, self.max_len - len(text.split()))),
            constant_values=0
        )

    def batch_encode(self, texts):
        return [
            self.encode(t)
            for t in texts
        ]

# Dataset

class ReviewDataset(Dataset):
    def __init__(self, texts, labels, tokenizer):
        self.encodings = tokenizer.batch_encode(texts)
        self.labels = labels.astype(int)   # 0-4 вже готові

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

    def __getitem__(self, idx):
```

```
            return (torch.tensor(self.encodings[idx],
dtype=torch.long),
                    torch.tensor(self.labels[idx],
dtype=torch.long))

    # Архітектура BiLSTM з Attention

    class BiLSTMAttention(nn.Module):
        def __init__(self, vocab_size, embed_dim=128,
hidden_dim=256,
                     num_classes=5, dropout=0.3):
            super().__init__()
            self.embedding = nn.Embedding(vocab_size, embed_dim,
padding_idx=0)
            self.lstm = nn.LSTM(embed_dim, hidden_dim,
batch_first=True,
                                bidirectional=True, num_layers=2,
dropout=dropout)
            self.attention = nn.Linear(hidden_dim * 2, 1)
            self.dropout   = nn.Dropout(dropout)
            self.fc        = nn.Linear(hidden_dim * 2,
num_classes)

        def forward(self, x):
            emb = self.dropout(self.embedding(x))  # (B, L, E)
            out, _ = self.lstm(emb)                # (B, L, 2H)
            # Attention
            context = out[:, -1, :]
            return self.fc(context)

    # Навчання

    from src.preprocess import preprocess_batch

    X_train_raw = np.load("../data/X_train.npy",
allow_pickle=True)
    X_val_raw   = np.load("../data/X_val.npy",
allow_pickle=True)
    X_test_raw  = np.load("../data/X_test.npy",
allow_pickle=True)
    y_train = np.load("../data/y_train.npy")
    y_val   = np.load("../data/y_val.npy")
    y_test  = np.load("../data/y_test.npy")

    X_train_c = preprocess_batch(X_train_raw, verbose=True)
    X_val_c   = preprocess_batch(X_val_raw)
    X_test_c  = preprocess_batch(X_test_raw)

    tokenizer = SimpleTokenizer(max_vocab=30_000,
max_len=120).fit(X_train_c)

    train_ds = ReviewDataset(X_train_c, y_train, tokenizer)
    val_ds   = ReviewDataset(X_val_c,   y_val,   tokenizer)
```

```
    test_ds  = ReviewDataset(X_test_c,  y_test,  tokenizer)

    train_loader = DataLoader(train_ds, batch_size=128,
shuffle=True,  num_workers=0)
    val_loader   = DataLoader(val_ds,   batch_size=128,
shuffle=False)
    test_loader  = DataLoader(test_ds,  batch_size=128,
shuffle=False)

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

    model     =
BiLSTMAttention(vocab_size=len(tokenizer.word2idx)).to(device)
    optimizer = Adam(model.parameters(), lr=1e-3,
weight_decay=1e-5)
    scheduler = ReduceLROnPlateau(optimizer, patience=2,
factor=0.5)
    criterion = nn.CrossEntropyLoss()

    best_f1, patience_count = 0.0, 0
    PATIENCE = 5

    for epoch in range(1, 21):
        # Train
        model.train()
        total_loss = 0
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()
            loss = criterion(model(xb), yb)
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            total_loss += loss.item()

        # Validate
        model.eval()
        preds, trues = [], []
        with torch.no_grad():
            for xb, yb in val_loader:
                out = model(xb.to(device)).argmax(dim=1).cpu()
                preds.extend(out.tolist())
                trues.extend(yb.tolist())

        val_f1 = f1_score(trues, preds, average="macro")
        scheduler.step(1 - val_f1)
        print(f"Epoch {epoch:2d} | Loss:
{total_loss/len(train_loader):.4f}"
              f" | Val Macro-F1: {val_f1:.4f}")

        if val_f1 > best_f1:
```

```
            best_f1 = val_f1
            torch.save(model.state_dict(),
"../models/bilstm_best.pt")
            patience_count = 0
        else:
            patience_count += 1
            if patience_count >= PATIENCE:
                print("Early stopping.")
                break

    # Тестова оцінка

    model.load_state_dict(torch.load("../models/bilstm_best.pt"))
    model.eval()
    preds, trues = [], []
    with torch.no_grad():
        for xb, yb in test_loader:
            out = model(xb.to(device)).argmax(dim=1).cpu()
            preds.extend(out.tolist())
            trues.extend(yb.tolist())

    from sklearn.metrics import classification_report
    print(f"Test Macro-F1: {f1_score(trues, preds,
average='macro'):.4f}")
    print(classification_report(trues, preds,
          target_names=["1★","2★","3★","4★","5★"]))

    print("LABELS:", np.unique(y_train))
    print("MIN/MAX:", y_train.min(), y_train.max())
    print("DTYPE:", y_train.dtype)
```

Додаток Д

Лістинг коду train_bert.py

```
    import numpy as np
    import torch
    from torch.utils.data import Dataset, DataLoader
    from transformers import AutoTokenizer,
AutoModelForSequenceClassification
    from transformers import get_linear_schedule_with_warmup
    from torch.optim import AdamW
    from sklearn.metrics import f1_score, classification_report

    # Dataset

    class BertReviewDataset(Dataset):
        def __init__(self, texts, labels, tokenizer, max_len=256):
            self.encodings = tokenizer(
                list(texts),
                truncation=True,
                padding="max_length",
                max_length=max_len,
                return_tensors="pt",
            )
            self.labels = torch.tensor(labels, dtype=torch.long)
# 1-5 → 0-4

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

        def __getitem__(self, idx):
            return {
                "input_ids":
self.encodings["input_ids"][idx],
                "attention_mask":
self.encodings["attention_mask"][idx],
                "labels":         self.labels[idx],
            }

    # Завантаження

    X_train_raw = np.load("../data/X_train.npy",
allow_pickle=True)[:20_000]
    y_train     = np.load("../data/y_train.npy")[:20_000]
    X_val_raw   = np.load("../data/X_val.npy",
allow_pickle=True)
    y_val       = np.load("../data/y_val.npy")
    X_test_raw  = np.load("../data/X_test.npy",
allow_pickle=True)
    y_test      = np.load("../data/y_test.npy")

    MODEL_NAME = "distilbert-base-uncased"   # легший варіант; для
кращої якості — bert-base-uncased
    tokenizer  = AutoTokenizer.from_pretrained(MODEL_NAME)
```

```
    train_ds = BertReviewDataset(X_train_raw, y_train, tokenizer)
    val_ds   = BertReviewDataset(X_val_raw,   y_val,   tokenizer)
    test_ds  = BertReviewDataset(X_test_raw,  y_test,  tokenizer)

    train_loader = DataLoader(train_ds, batch_size=32,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=64)
    test_loader  = DataLoader(test_ds,  batch_size=64)

    # Модель

    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model  = AutoModelForSequenceClassification.from_pretrained(
        MODEL_NAME, num_labels=5
    ).to(device)

    # Оптимізатор і scheduler

    EPOCHS     = 4
    TOTAL_STEPS = len(train_loader) * EPOCHS
    WARMUP      = TOTAL_STEPS // 10

    optimizer = AdamW(model.parameters(), lr=2e-5,
weight_decay=0.01)
    scheduler = get_linear_schedule_with_warmup(
        optimizer, num_warmup_steps=WARMUP,
num_training_steps=TOTAL_STEPS
    )

    # Цикл навчання

    best_f1 = 0.0
    for epoch in range(1, EPOCHS + 1):
        # Train
        model.train()
        total_loss = 0
        for batch in train_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(),
1.0)
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
            total_loss += loss.item()

        # Validate
        model.eval()
        preds, trues = [], []
```

```
        with torch.no_grad():
            for batch in val_loader:
                labels = batch.pop("labels").cpu()
                out = model(**{k: v.to(device) for k, v in
batch.items()})
                pred = out.logits.argmax(dim=1).cpu()
                preds.extend(pred.tolist())
                trues.extend(labels.tolist())

        val_f1 = f1_score(trues, preds, average="macro")
        print(f"Epoch {epoch} | Loss:
{total_loss/len(train_loader):.4f}"
              f" | Val Macro-F1: {val_f1:.4f}")

        if val_f1 > best_f1:
            best_f1 = val_f1
            model.save_pretrained("../models/bert_finetuned")
            tokenizer.save_pretrained("../models/bert_finetuned")
            print("  ✓ Збережено найкращу модель")

    # Тестова оцінка

    from transformers import AutoModelForSequenceClassification,
AutoTokenizer
    model_best =
AutoModelForSequenceClassification.from_pretrained(
        "../models/bert_finetuned").to(device)
    model_best.eval()
    preds, trues = [], []
    with torch.no_grad():
        for batch in test_loader:
            labels = batch.pop("labels").cpu()
            out = model_best(**{k: v.to(device) for k, v in
batch.items()})
            pred = out.logits.argmax(dim=1).cpu()
            preds.extend(pred.tolist())
            trues.extend(labels.tolist())

    print(f"\nTest Macro-F1: {f1_score(trues, preds,
average='macro'):.4f}")
    print(classification_report(trues, preds,
          target_names=["1★","2★","3★","4★","5★"]))
```

Додаток Е

Лістинг коду api.py

```
from fastapi import FastAPI
from pydantic import BaseModel
import joblib, numpy as np
from src.preprocess import clean_text

app = FastAPI(title="Sentiment Analysis API", version="1.0")

# Завантаження моделі при старті сервісу
model = joblib.load("models/lr_tfidf.pkl")     # замінити на
кращу модель
CLASSES = ["1 зірка", "2 зірки", "3 зірки", "4 зірки", "5
зірок"]

class ReviewRequest(BaseModel):
    text: str

class PredictionResponse(BaseModel):
    predicted_class: str
    stars: int
    probabilities: dict[str, float]

@app.post("../predict", response_model=PredictionResponse)
def predict(request: ReviewRequest):
    cleaned = clean_text(request.text)
    proba   = model.predict_proba([cleaned])[0]
    pred_idx = int(np.argmax(proba))
    return PredictionResponse(
        predicted_class=CLASSES[pred_idx],
        stars=pred_idx + 1,
        probabilities={cls: round(float(p), 4)
                       for cls, p in zip(CLASSES, proba)}
    )

@app.get("../health")
def health():
    return {"status": "ok"}
```

Додаток Ж
Лістинг коду demo.py

```
    import joblib
    import numpy as np
    import matplotlib
    matplotlib.use("Agg")
    import matplotlib.pyplot as plt
    import gradio as gr
    from pathlib import Path

    # Шляхи
    THIS_FILE    = Path(__file__).resolve()
    PROJECT_ROOT = THIS_FILE.parent.parent
    MODEL_PATH   = PROJECT_ROOT / "models" / "lr_tfidf.pkl"

    # Завантаження моделі
    print(f"Завантаження моделі: {MODEL_PATH}")
    model = joblib.load(MODEL_PATH)
    print("  Модель завантажено.")

    #  Константи
    CLASS_NAMES  = ["1★", "2★", "3★", "4★", "5★"]
    BAR_COLORS   = ["#E74C3C", "#E67E22", "#F1C40F", "#2ECC71",
"#3498DB"]
    STAR_SYMBOLS = ["★☆☆☆☆", "★★☆☆☆", "★★★☆☆",
"★★★★☆", "★★★★★"]


    #  Функція передбачення
    def analyze(text: str):
            if not text or not text.strip():
            return "Введіть текст відгуку", None

        # Передбачення
        proba     = model.predict_proba([text])[0]
        pred_idx  = int(np.argmax(proba))
        pred_stars = pred_idx + 1

        # Рядок рейтингу
        rating_str = f"{STAR_SYMBOLS[pred_idx]}  ({pred_stars}
зірок)"


        #  Matplotlib figure
        fig, ax = plt.subplots(figsize=(5, 3))
        fig.patch.set_facecolor("#1f1f1f")   # темний фон як у
Gradio dark theme
        ax.set_facecolor("#2b2b2b")

        bars = ax.bar(CLASS_NAMES, proba,
                      color=BAR_COLORS, alpha=0.88,
```

```
                    edgecolor="#1f1f1f", width=0.6)

        # Виділення передбаченого класу рамкою
        bars[pred_idx].set_edgecolor("white")
        bars[pred_idx].set_linewidth(2)

        # Підписи значень над стовпчиками
        for bar, val in zip(bars, proba):
            ax.text(
                bar.get_x() + bar.get_width() / 2,
                bar.get_height() + 0.012,
                f"{val:.2f}",
                ha="center", va="bottom",
                fontsize=9, color="white",
            )

        ax.set_ylim(0, min(1.0, max(proba) + 0.18))
        ax.set_xlabel("Клас (зірки)", fontsize=10, color="white")
        ax.set_ylabel("Ймовірність", fontsize=10, color="white")
        ax.set_title("Розподіл ймовірностей", fontsize=11,
color="white")
        ax.tick_params(colors="white")
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        for spine in ["left", "bottom"]:
            ax.spines[spine].set_color("#555555")
        ax.yaxis.set_major_formatter(
            plt.FuncFormatter(lambda v, _: f"{v:.0%}")
        )
        ax.grid(axis="y", alpha=0.2, color="white")
        plt.tight_layout()

        return rating_str, fig


    # Gradio інтерфейс


    with gr.Blocks(theme=gr.themes.Default(), title="Система
аналізу відгуків") as demo:

        gr.Markdown("# Система аналізу відгуків")
        gr.Markdown("Введіть текст відгуку — система визначить
рейтинг від 1 до 5 зірок.")

        with gr.Row():
            with gr.Column():
                text_input = gr.Textbox(
                    lines=6,
                    label="Текст відгуку",
                    placeholder="Наприклад: Absolutely amazing
food and friendly staff!",
                )
```

```
                with gr.Row():
                    clear_btn  = gr.Button("Очистити",
variant="secondary")
                    submit_btn = gr.Button("Аналізувати",
variant="primary")

            with gr.Column():
                rating_output = gr.Textbox(label="Передбачений
рейтинг", interactive=False)
                plot_output   = gr.Plot(label="Розподіл
ймовірностей")   # ← gr.Plot замість gr.BarPlot

        # Приклади
        gr.Examples(
            examples=[
                ["Terrible experience. Cold food, rude staff.
Never again!"],
                ["Decent place for lunch, nothing special but
okay."],
                ["Absolutely amazing! Best restaurant I've been to
in years!"],
                ["The service was slow and the pasta was
overcooked. Disappointing."],
                ["Great atmosphere, lovely staff and the best
pizza in town!"],
            ],
            inputs=text_input,
        )
        # Події
        submit_btn.click(
            fn=analyze,
            inputs=text_input,
            outputs=[rating_output, plot_output],
        )
        clear_btn.click(
            fn=lambda: ("", None),
            inputs=[],
            outputs=[rating_output, plot_output],
        )
        # Також спрацьовує на Enter у текстовому полі
        text_input.submit(
            fn=analyze,
            inputs=text_input,
            outputs=[rating_output, plot_output],
        )
    # Запуск
    if __name__ == "__main__":
        demo.launch(
            share=True, inbrowser=True
        )
```