ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної
та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

# Пояснювальна записка
# до кваліфікаційної роботи бакалавра

на тему: «ПРОЕКТУВАННЯ ДЕТЕКТОРА ТОКСИЧНОСТІ КОМЕНТАРІВ У СОЦІАЛЬНИХ МЕРЕЖАХ З ВИКОРИСТАННЯМ NLP ТА ML»

Виконав: здобувач вищої освіти
групи КН 2021-1
спеціальності
122 « Комп'ютерні науки»

 Микита ЗАГІНЕЙ

Керівник:
к.т.н., доц.  Марина БУЛАЄНКО

Рецензент:
к.т.н., доц.  Микола КАРПЕНКО

м. Харків – 2025 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 «Комп'ютерні науки»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«__24__» ___06____2025 року

# З А В Д А Н Н Я
# НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Загінею Микиті Олександровичу

1. Тема роботи Проектування детектора токсичності коментарів у соціальних мережах з використанням NLP та ML

керівник роботи к.т.н., доц. Булаєнко М.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «09» травня 2025 р. № 341-03

2. Термін подання студентом роботи 21.06.2025р.

3. Вихідні дані до роботи Рекомендації для проектування та впровадження детектора токсичності коментарів у соціальних мережах

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідження ринку та аналогів сервісів для проектування детектора токсичності коментарів у соціальних мережах; створення технічного завдання; вибір інструментів розробки і тестування застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація – 29 слайдів

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Розділ I | Марина БУЛАЄНКО | 15.05.2025 | 20.05.2025 |
| Розділ II | Марина БУЛАЄНКО | 21.05.2025 | 25.05.2025 |
| Розділ III | Марина БУЛАЄНКО | 26.05.2025 | 06.06.2025 |
| Розділ IV | Вікторія МАЛИШЕВА | 07.06.2025 | 11.06.2025 |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 15.05.2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Вибір теми кваліфікаційної роботи | 15.05.2025 | Виконано |
| 2 | Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи | 16.05.2025 | Виконано |
| 3 | Написання I розділу | 20.05.2025 | Виконано |
| 4 | Написання II розділу | 25.05.2025 | Виконано |
| 5 | Написання III розділу | 06.06.2025 | Виконано |
| 6 | Написання IV розділу | 11.06.2025 | Виконано |
| 7 | Подання кваліфікаційної роботи керівнику | 13.06.2025 | Виконано |
| 8 | Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до кваліфікаційної роботи | 16.06.2025 | Виконано |
| 9 | Подання доопрацьованого варіанту кваліфікаційної роботи керівнику | 18.06.2025 | Виконано |
| 10 | Захист матеріалів кваліфікаційної роботи на засіданні кафедри | 21.06.2025 | Виконано |
| 11 | Офіційний захист матеріалів кваліфікаційної роботи на засіданні екзаменаційної комісії | 25.06.2025 | Виконано |

Студент Микита ЗАГІНЕЙ

Керівник роботи Марина БУЛАЄНКО

# АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра здобувача вищої освіти групи КН 2021-1 спеціальності 122 - «Комп'ютерні науки» Микити Загінея на тему «Проектування детектора токсичності коментарів у соціальних мережах» складається з 4 розділів, містить 18 рисунків, 3 таблиці, 14 джерел, 1 додаток, сторінок 73. Головна мета даної роботи полягає у розробці системи автоматичного детектування токсичності у текстових коментарях соціальних мереж, її експериментальному дослідженні, а також у створенні програмного прототипу для демонстрації її роботи.

У розділі «Загальні положення» наведено опис предметного середовища, зокрема проблеми токсичності в онлайн-комунікаціях , описано процес модерації та функціональну модель системи. Проведено огляд наявних аналогів та сформульовано постановку задачі.

У розділі «Інформаційне та математичне забезпечення» наведено аналіз предметної області, визначено структуру вхідних та вихідних даних. Представлено математичне та алгоритмічне забезпечення, що включає алгоритми передобробки тексту та архітектуру моделі.

У розділі «Програмне та технічне забезпечення» наведено опис використаних засобів розробки (Python, PyTorch, FastAPI, Gradio) та вимог до апаратно-програмного забезпечення. Описано програмну реалізацію системи за допомогою UML-діаграм, а також надано детальне керівництво користувача для запуску та тестування прототипу.

У розділі «Охорона праці» приділено увагу питанню законодавства у сфері охорони праці для IT-спеціаліста , визначено потенційні небезпеки, характерні для його діяльності , та проведено їх оцінювання за допомогою матриці оцінки ризиків.

Реалізація даної роботи дозволила створити функціонуючий програмний прототип, який демонструє ефективність обраних методів машинного навчання для виявлення токсичності та може слугувати основою для розробки повноцінних систем модерації контенту.

Ключові слова: ДЕТЕКЦІЯ ТОКСИЧНОСТІ, ОБРОБКА ПРИРОДНОЇ МОВИ, МАШИННЕ НАВЧАННЯ, NEURAL BAG-OF-WORDS, PYTORCH, FASTAPI.

# ANNOTATION

The explanatory note to the bachelor's qualification thesis of Mykyta Zaginei, a student of group KN 2021-1, specialty 122 - "Computer Science," on the topic "Design of a toxicity detector for comments in social networks" consists of 4 chapters, contains 18 figures, 3 tables, 14 sources, 1 appendix, and 73 pages. The main goal of this bachelor's qualification thesis is to develop a system for the automatic detection of toxicity in text comments on social networks, to conduct its experimental research, and to create a software prototype to demonstrate its operation.

The "General Provisions" chapter describes the subject area, particularly the problem of toxicity in online communications , outlines the moderation process, and defines the functional model of the system. A review of existing analogues has been conducted, and the problem statement has been formulated.

The "Information and Mathematical Support" chapter provides an analysis of the subject area and defines the input and output data structures. The mathematical and algorithmic support is presented, including text preprocessing algorithms, the Neural Bag-of-Words model architecture, and methods for its improvement.

The "Software and Technical Support" chapter describes the development tools used (Python, PyTorch, FastAPI, Gradio) and the hardware/software requirements. The software implementation is detailed using UML diagrams, and a comprehensive user guide for running and testing the prototype is provided.

The "Occupational Safety" chapter focuses on legislation in the field of occupational safety for IT specialists , identifies potential hazards specific to their activities , and provides a risk assessment using a risk matrix.

The implementation of this work resulted in a functioning software prototype that demonstrates the effectiveness of the chosen machine learning methods for toxicity detection and can serve as a basis for the development of full-fledged content moderation systems.

Keywords: TOXICITY DETECTION, NATURAL LANGUAGE PROCESSING, MACHINE LEARNING, NEURAL BAG-OF-WORDS, PYTORCH, FASTAPI.

ЗМІСТ

# ВСТУП

Соціальні мережі, як один із ключових елементів сучасного інформаційного простору, утвердилися в якості фундаментальних платформ для глобальної комунікації, обміну думками та формування суспільної свідомості. Щоденно мільярди користувачів генерують величезні обсяги текстового контенту, що відкриває безпрецедентні можливості для взаємодії, але водночас створює значні виклики, пов'язані з поширенням негативних явищ. Онлайн-середовище, на жаль, нерідко стає простором для мови ворожнечі, кібербулінгу та іншої токсичної поведінки, що справляє негативний вплив на користувачів та загальну атмосферу цифрового простору.

Актуальність обраної теми зумовлена нагальною потребою у розробці та впровадженні сучасних технологічних рішень, спрямованих на забезпечення безпечного та конструктивного онлайн-спілкування. Автоматизоване виявлення токсичних коментарів здатне не лише зменшити навантаження на процеси ручної модерації, які часто виявляються недостатньо ефективними для обробки значних обсягів інформації, але й забезпечити оперативне реагування на шкідливий контент, тим самим захищаючи користувачів від його деструктивного впливу. Застосування ефективних детекторів токсичності має сприяти підвищенню якості онлайн-дискусій, зміцненню довіри до інтернет-платформ та формуванню більш сприятливого цифрового середовища.

Мета кваліфікаційної роботи полягає у розробці системи автоматичного детектування токсичності у текстових коментарях соціальних мереж, її експериментальному дослідженні, а також у створенні програмного прототипу для демонстрації практичного застосування розробленої системи та оцінки ефективності обраних підходів машинного навчання.

Об’єктом дослідження визначено процеси аналізу текстових коментарів у соціальних мережах та їх класифікації за ознаками токсичності.

Предметом дослідження є методи обробки природної мови (ОПМ) та алгоритми машинного навчання (МН), що застосовуються для виявлення токсичного контенту. Зокрема, розглядається модель Neural Bag-of-Words (NBoW) та методи підвищення її ефективності при роботі з незбалансованими наборами даних, такі як використання зваженої функції втрат та оптимізація порогів класифікації.

Завдання дослідження:

1. Провести аналіз сучасних підходів та інструментальних засобів для детекції токсичності в текстових даних.
2. Здійснити підготовку набору даних, а саме збір, аналіз та комплексну передню обробку текстових коментарів для навчання та тестування моделі.
3. Розробити та програмно реалізувати базову модель машинного навчання (Neural Bag-of-Words) для класифікації коментарів за рівнем токсичності.
4. Дослідити проблему дисбалансу класів та її вплив на якість класифікації, а також застосувати відповідні методи для її нівелювання, зокрема зважену функцію втрат.
5. Виконати експерименти з оптимізації порогів класифікації з метою покращення метрики F1-score для рідкісних класів токсичності.
6. Здійснити комплексну оцінку якості розробленої моделі на основі релевантних метрик (Precision, Recall, F1-score, ROC AUC).
7. Розробити програмний інтерфейс застосунку (API) для забезпечення взаємодії з навченою моделлю.
8. Створити користувацький веб-інтерфейс для наочної демонстрації функціональності розробленої системи детекції токсичності.

Для розробки моделі та програмного прототипу було використано мову програмування Python та набір спеціалізованих бібліотек: PyTorch – для побудови та навчання нейронних мереж; spaCy та модуль re – для обробки текстових даних; Pandas та NumPy – для операцій з даними; Scikit-learn – для

оцінювання якості моделі. Серверна частина API реалізована на базі фреймворку FastAPI, а користувацький інтерфейс створено з використанням бібліотеки Gradio. Експериментальна частина роботи та процес розробки здійснювалися в інтерактивному середовищі Jupyter Notebook.

Результатом даної кваліфікаційної роботи є функціонуючий програмний прототип системи детекції токсичності. Розроблений прототип демонструє ефективність обраних методів машинного навчання та є основою для подальшого вдосконалення і потенційного практичного впровадження у модераційні системи з метою підвищення безпеки та якості онлайн-комунікацій.

# РОЗДІЛ 1
# ЗАГАЛЬНІ ПОЛОЖЕННЯ

## 1.1 Опис предметного середовища

Епоха цифрових технологій ознаменувалася безпрецедентним зростанням ролі соціальних мереж та різноманітних онлайн-платформ, які перетворилися на невід'ємні атрибути суспільного життя та ключові канали комунікації для мільярдів людей по всьому світу. Ці платформи, що охоплюють широкий спектр форматів – від мікроблогів та форумів до мультимедійних сервісів обміну контентом та професійних мереж – надають користувачам унікальні можливості для самовираження, миттєвого обміну інформацією, формування спільнот за інтересами, отримання новин та участі у суспільно-політичних процесах. Щоденно генеруються терабайти текстового, візуального та аудіовізуального контенту, відображаючи динаміку та багатогранність сучасного інформаційного простору [1, 2]. Динаміка зростання аудиторії соціальних мереж приведена на рис.1.1. Ця цифрова революція, з одного боку, сприяє демократизації доступу до інформації та розширенню горизонтів спілкування, а з іншого – породжує низку серйозних викликів, пов'язаних із забезпеченням безпеки, етичності та конструктивності онлайн-взаємодій. Одним із найбільш гострих проявів цих викликів є поширення токсичного контенту.

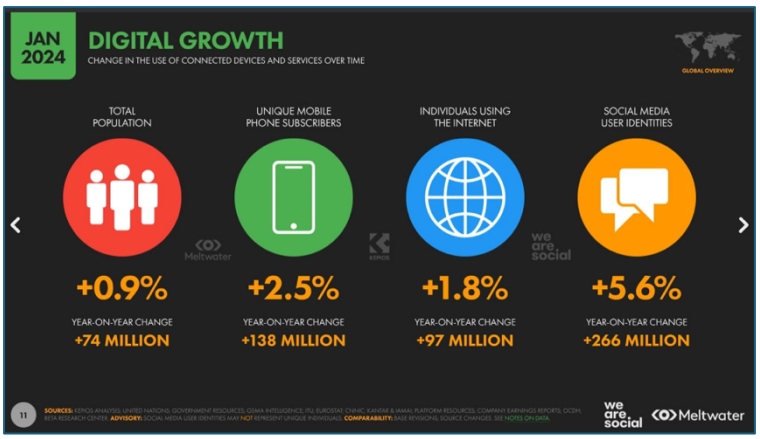


Рисунок 1.1 – Динаміка зростання аудиторії соціальних мереж

Під токсичним контентом у широкому сенсі розуміють будь-яку форму онлайн-комунікації, що має на меті або може призвести до завдання шкоди, образи, приниження, залякування чи дискримінації окремих осіб або груп, а також до дестабілізації конструктивного діалогу та створення ворожої атмосфери. Спектр токсичних проявів надзвичайно широкий і включає мову ворожнечі, кібербулінг, тролінг, погрози, поширення дезінформації та шкідливих наративів, сексуальні домагання та інші форми антисоціальної поведінки.

У контексті даної кваліфікаційної роботи, фокус зосереджено на виявленні наступних категорій токсичності в текстових коментарях користувачів соціальних мереж:

- Власне токсичні (toxic): загальні негативні, грубі або неповажні висловлювання.
- Виражено токсичні (severe toxic): крайні форми токсичності, що включають найбільш образливі та обурливі коментарі.
- Непристойні (obscene): коментарі, що містять вульгарну, нецензурну лексику або контент сексуального характеру.
- Погрози (threat): висловлювання, що містять намір завдати фізичної або іншої шкоди.
- Образи (insult): персональні атаки, спрямовані на приниження гідності іншої особи.
- Мова ворожнечі на основі ідентичності (identity hate): коментарі, що атакують або принижують групи людей на основі їх раси, етнічної належності, релігії, статі, сексуальної орієнтації, інвалідності тощо.

Визначення межі токсичності часто ускладнюється суб'єктивністю сприйняття, культурними відмінностями, використанням сарказму, іронії або контекстуально зумовлених висловлювань, що створює додаткові труднощі як для ручної, так і для автоматизованої модерації.

Нижче приведано таблицю (таблиця 1.1) , яка більш детально дає опис типів токсичності.

Таблиця 1.1 – Класифікація типів токсичного

| **Категорія токсичності** | **Характерні ознаки** | **Приклади (умовні, для ілюстрації типу)** |
|---|---|---|
| Токсичні (toxic) | Загаломгрубі, неповажні або негативні висловлювання, які можуть образити або створити негативну атмосферу, але не обов'язково містять явну лайку, погрози чи мову ворожнечі. | "Твоя логіка просто жахлива.", "Це абсолютно безглуздий коментар, перестань писати.", "Мені не подобається, як ти це робиш." |
| Виражено токсичні (severe_toxic) | Надзвичайно агресивні, вкрай образливі висловлювання, що часто включають лайку, виражають сильну ненависть, презирство або мають на меті глибоко образити. Зазвичай є посиленою формою категорії "toxic". | "[Надзвичайно агресивне висловлювання з використанням грубої лайки та побажанням шкоди]" |
| Непристойні (obscene) | Використання нецензурної лексики, вульгаризмів, лайливих слів, описів сексуального характеру або фізіологічних процесів у грубій, недоречній або образливій формі. | "[Коментар з недоречним, відверто сексуальним натяком або пропозицією]", "[Використання грубих слів для опису людини]" |
| Погрози (threat) | Висловлення явного або прихованого наміру завдати фізичної шкоди, насильства, переслідувати, завдати матеріальних збитків або іншої шкоди особі чи групі осіб. | "Краще б тобі не з'являтися [у певному місці], якщо цінуєш своє здоров'я.", "Якщо ти [зробиш щось], то [опис негативних наслідків для тебе]." |
| Образи (insult) | Прямі персональні атаки, принизливі коментарі щодо зовнішності, інтелекту, особистих якостей, здібностей, походження чи дій іншої людини, спрямовані на підрив її репутації або самоповаги. | Подивись на себе, [негативна оцінка зовнішності або розумових здібностей]." |
| Мова ворожнечі (identity_hate) | Агресивні, принизливі або дискримінаційні висловлювання, спрямовані проти окремих осіб або груп людей на основі їхньої раси, етнічної приналежності, національності, релігії, статі, сексуальної орієнтації, віку, інвалідності, соціального статусу чи іншої захищеної характеристики. | [Образливе узагальнююче твердження про всіх представників певної національності/релігії/соціальної групи]", |

Негативний вплив токсичного контенту є багатоаспектним та руйнівним. На індивідуальному рівні жертви онлайн-агресії можуть страждати від психологічних проблем, таких як тривожні розлади, депресія, зниження самооцінки, соціальна ізоляція, страх вільно висловлювати свою думку (ефект "самоцензури"). У найбільш важких випадках кібербулінг може

призводити до трагічних наслідків. На рівні онлайн-спільнот токсичність отруює атмосферу дискусій, призводить до поляризації думок, витіснення конструктивних учасників, зниження загальної якості обміну інформацією та руйнування довіри всередині спільноти. Для онлайн-платформ поширення токсичності становить серйозну загрозу, оскільки може призвести до відтоку користувачів, втрати рекламодавців, погіршення репутації та навіть до юридичної відповідальності та регуляторного тиску з боку державних органів. Масштаби цієї проблеми є глобальними, і дослідження регулярно фіксують високий рівень поширеності токсичних взаємодій в Інтернеті.

Традиційним методом боротьби з небажаним контентом є ручна модерація, що здійснюється штатними або залученими спеціалістами. Проте, цей підхід стикається з низкою фундаментальних обмежень, які стають дедалі очевиднішими на тлі експоненційного зростання обсягів користувацького контенту:

- Масштабованість. Кількість коментарів, постів та інших форм онлайн-взаємодій, що генеруються щосекунди, настільки велика, що жодна команда модераторів не здатна фізично переглянути та оцінити весь цей потік. Наприклад, на популярних платформах кількість щоденних публікацій може сягати сотень мільйонів.
- Оперативність. Токсичний контент, особливо вірусного характеру, може поширитися надзвичайно швидко, завдаючи шкоди значній аудиторії ще до того, як модератор встигне його виявити та відреагувати.
- Вартість. Утримання великих, глобально розподілених команд модераторів, які працюють у режимі 24/7 та володіють різними мовами, є надзвичайно ресурсозатратним для компаній. Сюди входять витрати на заробітну плату, навчання, програмне забезпечення та інфраструктуру.
- Суб'єктивність та узгодженість. Незважаючи на наявність детальних інструкцій та політик, процес прийняття рішень модераторами неминуче містить елемент суб'єктивності. Різні люди можуть по-різному інтерпретувати один і той самий контент, що призводить до

неузгодженості дій та помилок як першого (пропуск токсичного контенту), так і другого роду (помилкове блокування легітимного контенту).

- Психологічне навантаження. Постійна робота з контентом, що містить сцени насильства, мову ворожнечі, дитячу експлуатацію та інші деструктивні матеріали, справляє руйнівний вплив на психічне здоров'я модераторів, призводячи до емоційного вигорання, тривожних розладів, депресії та навіть посттравматичного стресового розладу (ПТСР).
- Лінгвістична та культурна складність. Токсичність може виражатися у неявний спосіб, використовуючи контекстуальні натяки, сарказм, іронію, кодові слова, неологізми, модифіковані слова (наприклад, з пропусками літер), емодзі та символи, що значно ускладнює її розпізнавання. Ефективна модерація також вимагає глибокого розуміння культурних особливостей та актуального сленгу, що є викликом для глобальних платформ.

Вказані обмеження ручної модерації роблять очевидною нагальну потребу у розробці та інтеграції автоматизованих інструментів, здатних ефективно допомагати у виявленні та нейтралізації токсичного контенту, підвищуючи таким чином безпеку та якість онлайн-середовища.

### 1.1.1 Опис процесу діяльності

Процес генерації та обробки користувацького контенту на типовій онлайн-платформі, в який інтегрується система детекції токсичності, є послідовністю взаємопов'язаних дій. Він починається з моменту, коли користувач створює текстове повідомлення (коментар, допис, відгук) та відправляє його для публікації.

Після надходження на сервер платформи, коментар проходить через низку етапів обробки. У традиційній моделі він може одразу ставати видимим для інших користувачів, або ж потрапляти у чергу на премодерацію, якщо така

політика діє на платформі. Інтеграція автоматизованої системи детекції токсичності модифікує цей процес, додаючи важливий етап аналізу.

Отже, оновлений процес діяльності може виглядати наступним чином: Користувач створює та надсилає текстовий коментар. Коментар приймається системою онлайн-платформи. Перед публікацією (або паралельно з нею, якщо використовується постмодерація) коментар передається на аналіз до автоматизованої системи детекції токсичності.

Система детекції проводить лінгвістичний аналіз тексту, застосовуючи навчені моделі машинного навчання для виявлення ознак токсичності за визначеними категоріями (наприклад, образа, погроза, мова ворожнечі тощо).

За результатами аналізу система видає оцінку токсичності коментаря, яка може бути представлена у вигляді бінарної мітки (токсичний/нетоксичний), набору міток для різних категорій або числових показників ймовірності належності до кожної категорії.

На основі отриманої оцінки онлайн-платформа автоматично приймає рішення щодо подальшої долі коментаря:

- Дозволити публікацію, якщо коментар визнано безпечним.
- Заблокувати або видалити коментар, якщо виявлено високий рівень токсичності.
- Тимчасово приховати коментар та/або надіслати його на перевірку людині-модератору, якщо система має низький рівень впевненості у своїй оцінці або коментар містить ознаки, що потребують експертного розгляду.
- Застосувати інші заходи, наприклад, надіслати попередження користувачеві.
- У випадку направлення коментаря на ручну перевірку, людина-модератор вивчає його зміст, контекст та оцінку автоматизованої системи, після чого приймає остаточне рішення. Дії модератора можуть також використовуватися для збору даних зворотного зв'язку з метою подальшого донавчання та покращення автоматизованої системи.

Більш детально можна побачити алгоритм на рис 1.2.

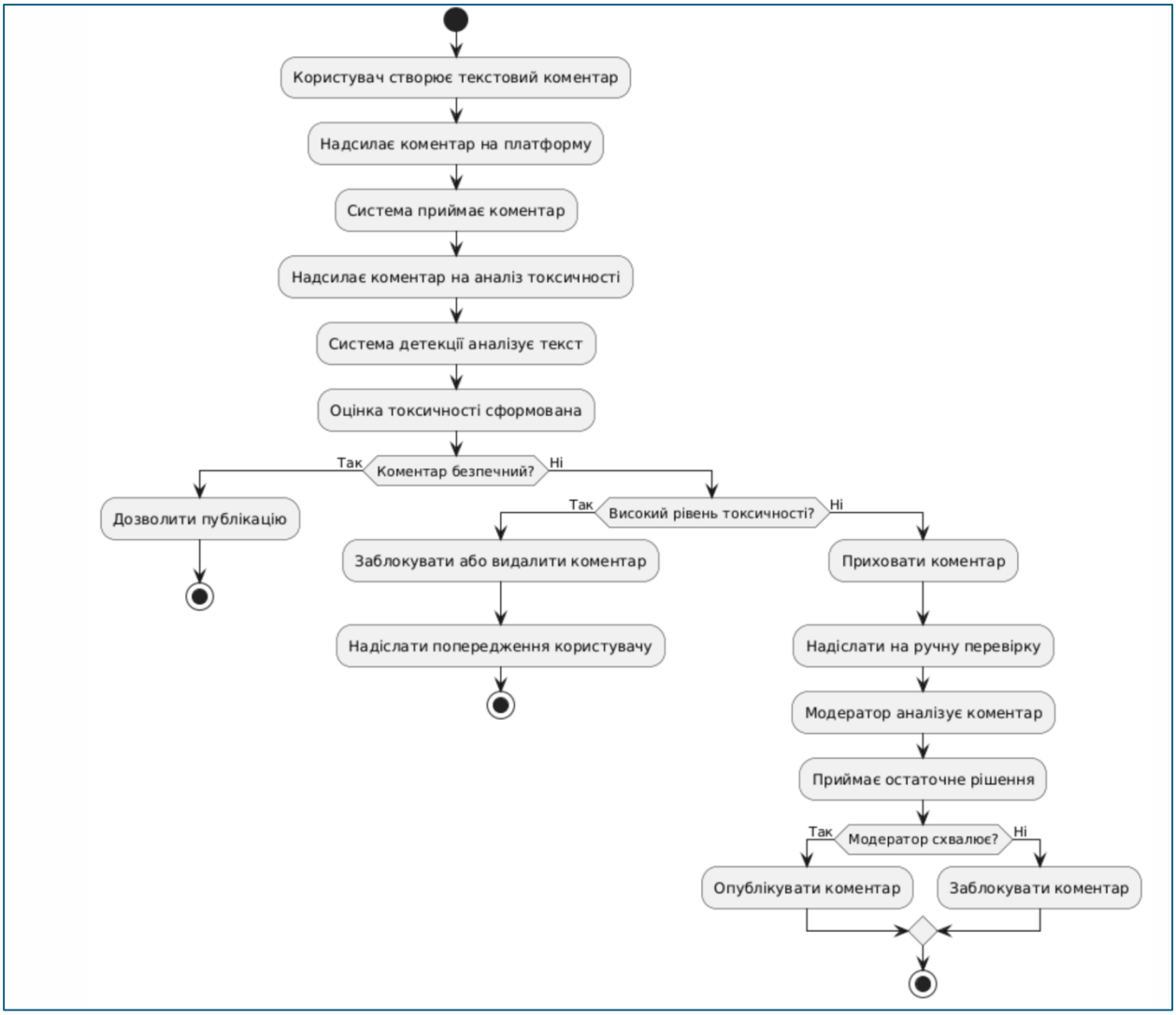


Рисунок 1.2 – Діаграма діяльності процесу обробки та модерації коментарів

Таким чином, система детекції токсичності функціонує як інтелектуальний фільтр, що допомагає оптимізувати процес модерації, підвищити його швидкість та ефективність, а також зменшити навантаження на людей-модераторів.

### 1.1.2 Опис функціональної моделі

Функціональна модель описує основні можливості та призначення розроблюваної системи детекції токсичності, а також визначає її взаємодію з зовнішніми сутностями (акторами). Розуміння функціональної моделі є критично важливим для проектування ефективної та корисної системи.

Основними акторами, що взаємодіють з системою або зазнають її впливу, є:

- Користувач онлайн-платформи. Генерує контент, який аналізується системою; є кінцевим бенефіціаром покращення якості комунікаційного середовища.
- Модератор контенту (людина). Використовує результати роботи автоматизованої системи для прийняття остаточних рішень щодо спірних коментарів, може брати участь у процесі донавчання моделі шляхом надання зворотного зв'язку.
- Адміністратор платформи. Відповідає за інтеграцію системи детекції з платформою, налаштування її параметрів (наприклад, порогів чутливості), моніторинг ефективності.
- Інтегрована система (ядро онлайн-платформи). Автоматично передає коментарі на аналіз через API та отримує результати для подальшої обробки.
- Розробник/дослідник системи. Здійснює розробку, тестування, оцінку якості моделі, її подальше вдосконалення та демонстрацію функціональності через спеціалізовані інтерфейси.

Ключові функції, які повинна виконувати система детекції токсичності, включають [7]:

1. Прийом текстових коментарів: Система повинна мати механізм для отримання текстових даних для аналізу. Це може бути реалізовано через програмний інтерфейс (API), який приймає текстові рядки або структуровані дані, що містять текст коментаря та, можливо, додаткову метаінформацію (наприклад, ідентифікатор користувача, час публікації).
2. Передобробка тексту: Ця функція є критично важливою для підготовки тексту до аналізу моделлю машинного навчання. Вона включає низку операцій: приведення тексту до нижнього регістру для уніфікації; видалення URL-адрес, згадок користувачів (типу @username), HTML-тегів та інших нерелевантних елементів; очищення тексту від символів,

що не є літерами (з можливим збереженням знаків пунктуації, якщо вони важливі для аналізу); лематизацію слів (приведення до базової форми) для зменшення розмірності простору ознак; видалення стоп-слів (часто вживаних слів, що не несуть значного смислового навантаження для задачі класифікації).

3. Побудова векторного представлення тексту (векторизація ознак): Оброблені тексти необхідно перетворити у числовий формат, зрозумілий для алгоритмів машинного навчання. У даній роботі як базовий підхід використовується модель Neural Bag-of-Words (NBoW), яка передбачає створення словника унікальних слів, нумерикалізацію текстів (заміну слів їхніми числовими індексами) та подальше представлення кожного коментаря як усередненого вектора ембеддингів слів, що до нього входять.
4. Класифікація токсичності: Ядром системи є навчена модель машинного навчання (у даному випадку – нейронна мережа на основі NBoW), яка приймає на вхід векторизоване представлення коментаря і видає оцінку його належності до однієї або декількох з шести визначених категорій токсичності. Це задача мультилейбл-класифікації.
5. Надання результатів аналізу: Система повинна повертати результати класифікації у чіткому та структурованому форматі. Це можуть бути бінарні мітки для кожної категорії та/або числові значення (ймовірності від 0 до 1), що вказують на ступінь впевненості моделі у належності коментаря до відповідної категорії.
6. Програмний інтерфейс (API): Для забезпечення можливості інтеграції детектора з іншими системами (наприклад, соціальними платформами, чат-ботами, інструментами модерації) необхідно реалізувати API. API має надавати ендпоінти для надсилання тексту на аналіз та отримання результатів у форматі JSON або аналогічному.
7. Користувацький інтерфейс (UI): Для наочної демонстрації роботи системи, проведення ручного тестування на окремих прикладах коментарів та візуалізації результатів класифікації розробляється

користувацький веб-інтерфейс. Він дозволяє вводити текст, отримувати оцінки токсичності та бачити їх у зручному для сприйняття вигляді.

Більш детально можна побачити алгоритм на рис 1.3.

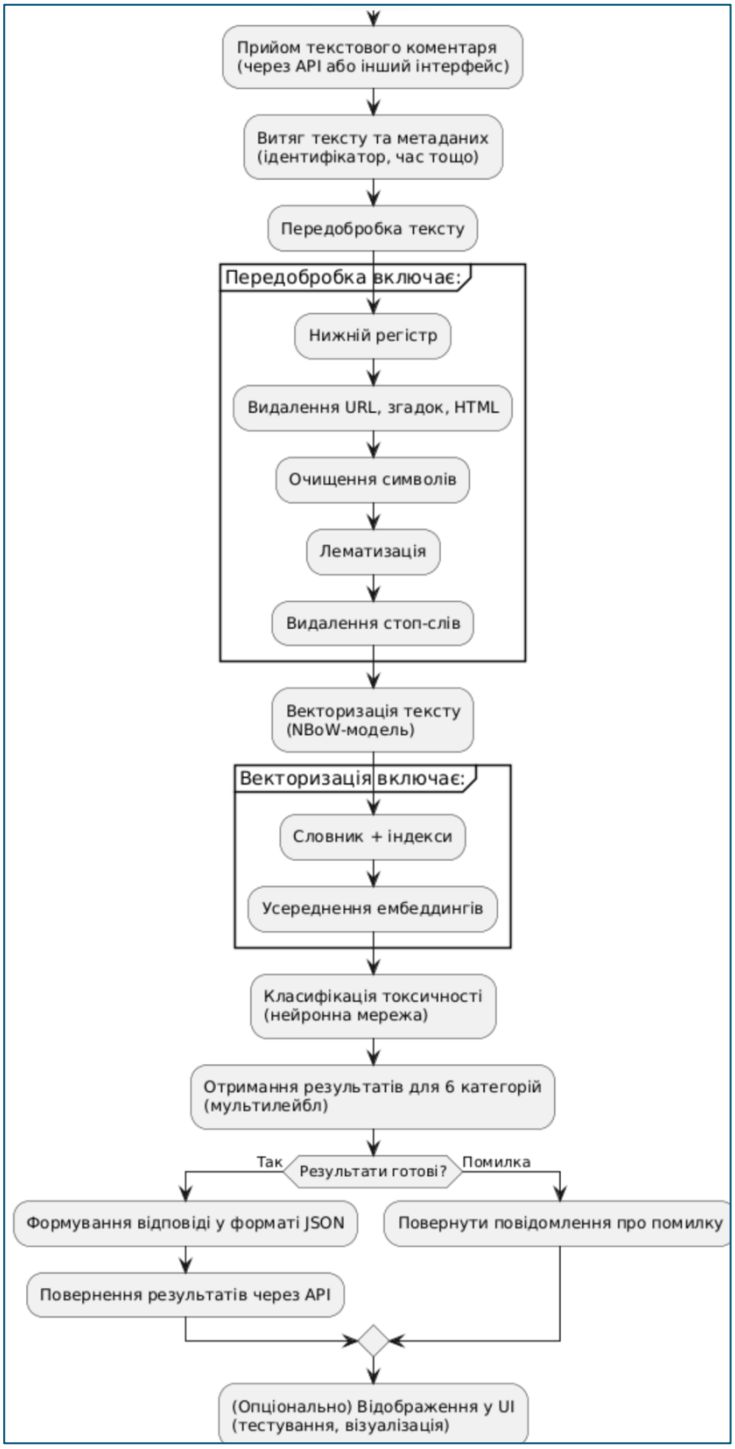


Рисунок1.3 – UML-діаграма діяльності детекції токсичності коментарів

Для реалізації цих функціональних можливостей пропонується архітектура системи, що складається з кількох взаємопов'язаних компонентів: модуль API на базі FastAPI для обробки зовнішніх запитів, модуль попередньої обробки тексту, що використовує spaCy та регулярні вирази, модуль машинного навчання з моделлю PyTorch (NBoW) для класифікації, та модуль користувацького інтерфейсу на базі Gradio. Дані, такі як словник та параметри моделі, зберігаються та завантажуються при старті системи.

Архітектуру програмного прототипу у вигляді UML діаграми можна побачити на рис 1.4.


Рисунок 1.4: Архітектура програмного прототипу системи детекції токсичності

Розробка системи з такою функціональною моделлю та архітектурою дозволить створити гнучкий та масштабований інструмент для автоматичного

аналізу токсичності текстових коментарів, що є важливим кроком на шляху до створення безпечнішого та більш конструктивного онлайн-середовища.

## 1.2 Огляд наявних аналогів

На даному етапі кваліфікаційної роботи було проведено дослідження та аналіз існуючих рішень – систем, сервісів та моделей, – призначених для автоматичного виявлення токсичності в текстовому контенті. Метою цього огляду є оцінка сучасних підходів, їхніх функціональних можливостей, переваг та обмежень, що дозволить краще позиціонувати розроблюваний програмний прототип детектора токсичності та визначити його потенційні переваги й відмінності.

Розглянемо низку актуальних аналогів, що варіюються від комерційних програмних інтерфейсів (API) до моделей з відкритим кодом. Оскільки детальна внутрішня архітектура та дані навчання багатьох комерційних систем є закритою інформацією, аналіз буде зосереджений на публічно доступній документації, заявлених функціональних можливостях, типах токсичності, що детектуються, та загальних принципах роботи, наскільки це можливо оцінити з точки зору потенційного користувача або дослідника.

До переліку обраних для аналізу рішень увійшли:

1. Perspective API (Jigsaw / Google)
2. Azure AI Content Safety (Microsoft)
3. OpenAI Moderation Endpoint
4. Amazon Comprehend

Perspective API (Jigsaw / Google) [3]

Є хмарним сервісом, розробленим підрозділом Jigsaw (що входить до складу Google). Його основне призначення – використання моделей машинного навчання для оцінки сприйнятого впливу текстового коментаря на онлайн-дискусію. Сервіс надає числові оцінки для різноманітних атрибутів, таких як загальна токсичність, виражена токсичність, образа, погроза,

сексуальна відвертість та інші, що дозволяє отримати багатогранну характеристику тексту. Важливою перевагою є розробка авторитетною компанією з великим досвідом у сфері штучного інтелекту та підтримка декількох мов, хоча якість аналізу для різних мов може варіюватися. Доступність через API спрощує інтеграцію з різними платформами. Водночас, як і багато подібних систем, Perspective API може демонструвати упередження, успадковані з навчальних даних, а його оцінки "сприйнятого впливу" не завжди ідеально корелюють з наміром автора чи специфікою токсичності в різних культурних або ситуативних контекстах. Деталі роботи моделей та точні дані навчання є закритою інформацією, що обмежує прозорість сервісу, а комерційне використання для великих обсягів запитів може бути пов'язане зі значними витратами. Приклад роботи Perspective API наведено на рис 1.5.

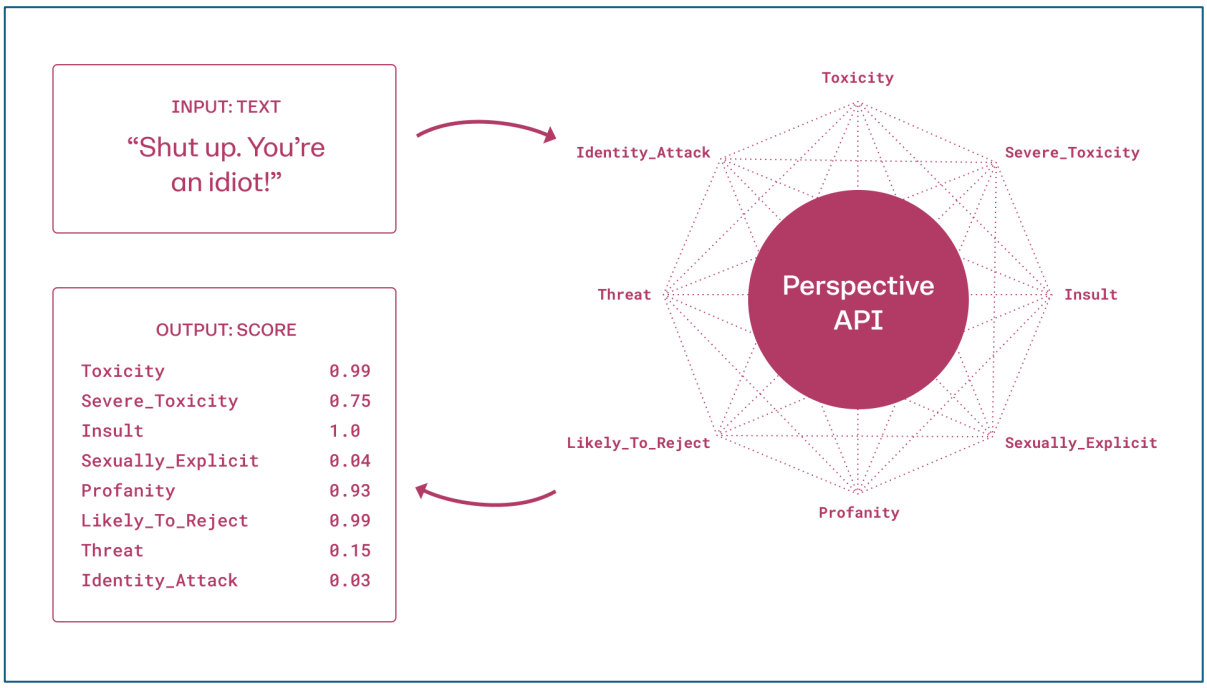


Рисунок 1.5 – Приклад роботи Perspective API

Azure AI Content Safety (Microsoft) [4]

Це комплексний сервіс від компанії Microsoft, інтегрований в екосистему хмарних рішень Azure. Він призначений для виявлення потенційно

образливого, ризикованого або іншого небажаного контенту не лише в текстах, але й у зображеннях.

Щодо текстового аналізу, сервіс здатен ідентифікувати мову ворожнечі, контент сексуального характеру, сцени насильства та заклики до самопошкодження.

Перевагою є надання рівнів серйозності для виявленого контенту, що дозволяє гнучкіше налаштовувати політику модерації, а також можливість встановлення порогів чутливості. Як продукт великої технологічної компанії, Azure AI Content Safety постійно оновлюється та підтримується. Однак, система не позбавлена недоліків, властивих подібним технологіям, зокрема можливі помилки класифікації та залежність вартості від обсягу оброблюваної інформації. Прозорість внутрішніх алгоритмів обмежена, а ефективність може варіюватися для різних мов та специфічних проявів токсичності. Приклад роботи Azure Content Safety приведено на рис 1.6.

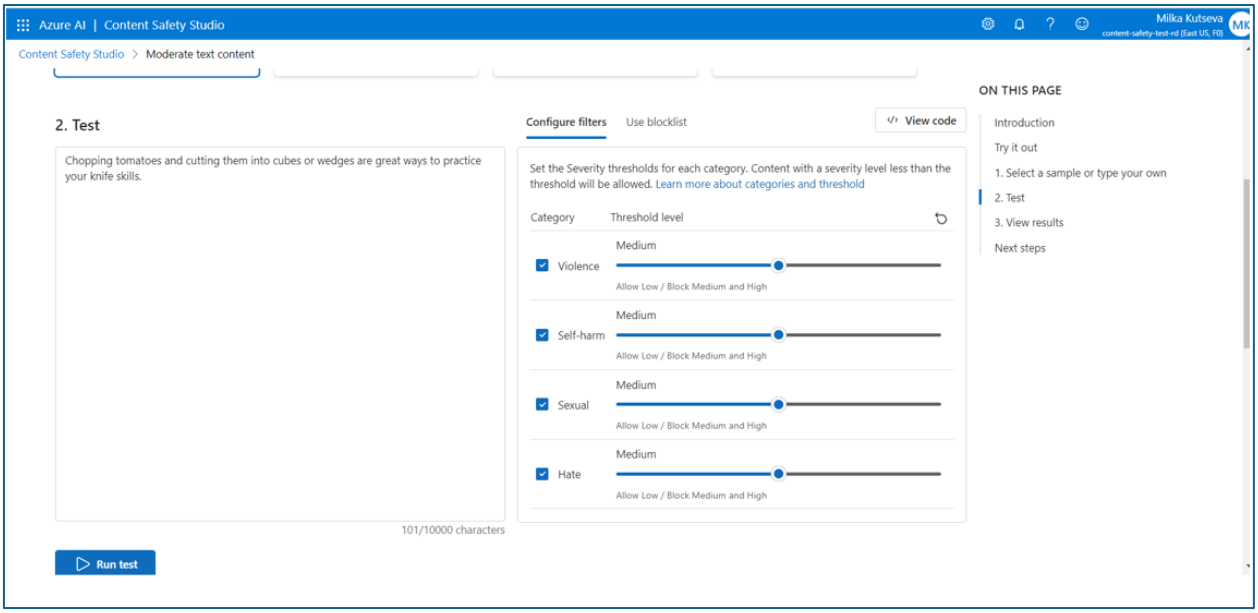


Рисунок 1.6 – Інтерфейс Azure Content Safety

OpenAI Moderation Endpoint [5]

Відомий розробник передових мовних моделей, пропонує програмний інтерфейс (API) Moderation Endpoint. Цей інструмент призначений для

перевірки текстового контенту на відповідність політиці використання сервісів OpenAI, класифікуючи його за такими категоріями, як мова ворожнечі, контент сексуального характеру, насильство, підбурювання до самопошкодження та деякі інші. В основі його роботи лежать потужні мовні моделі, що є значною перевагою. API є відносно простим у використанні та надає не лише мітки категорій, але й числові оцінки впевненості моделі. Певний обсяг запитів може бути доступний безкоштовно. До обмежень можна віднести те, що класифікація сфокусована переважно на політиках OpenAI, що може не завжди повністю збігатися з унікальними потребами модерації конкретної платформи. Моделі функціонують як "чорна скринька", і, як і інші великі мовні моделі, можуть бути схильні до певних упереджень або неточностей у розпізнаванні складних контекстуальних форм токсичності. Приклад інтерфейсу OpenAI API наведено на рис 1.7.

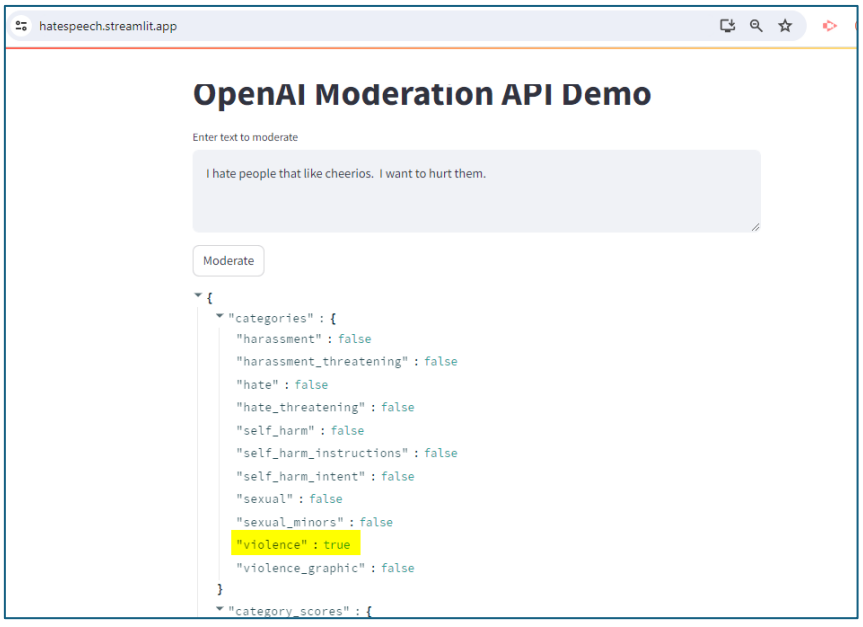


Рисунок 1.7 – Інтерфейс OpenAI Moderation Endpoint

Amazon Comprehend [6]

Є сервісом обробки природної мови (NLP) в рамках платформи Amazon Web Services (AWS). Він надає широкий набір інструментів для аналізу тексту, таких як виявлення сутностей, ключових фраз, визначення мови, аналіз

настрою (тональності) та синтаксичний аналіз. Хоча Amazon Comprehend не пропонує універсальної готової функції для детекції всіх типів токсичності "з коробки" для будь-яких завдань, його потужною стороною є можливість створення користувацьких (кастомних) моделей класифікації. Це дозволяє розробникам навчати власні моделі, специфічно налаштовані на виявлення тих чи інших видів токсичності, використовуючи власні набори даних та критерії. Така гнучкість є суттєвою перевагою, оскільки дає змогу адаптувати систему до конкретних потреб та зменшити вплив упереджень, властивих загальним моделям. Проте, розробка та навчання ефективної кастомної моделі вимагає значних зусиль, часу, наявності якісних та релевантних даних для навчання, а також певної експертизи в галузі машинного навчання. Вартість використання може бути вищою, особливо на етапі навчання та при обробці великих обсягів даних. Принцип роботи також наведено на рис 1.8.

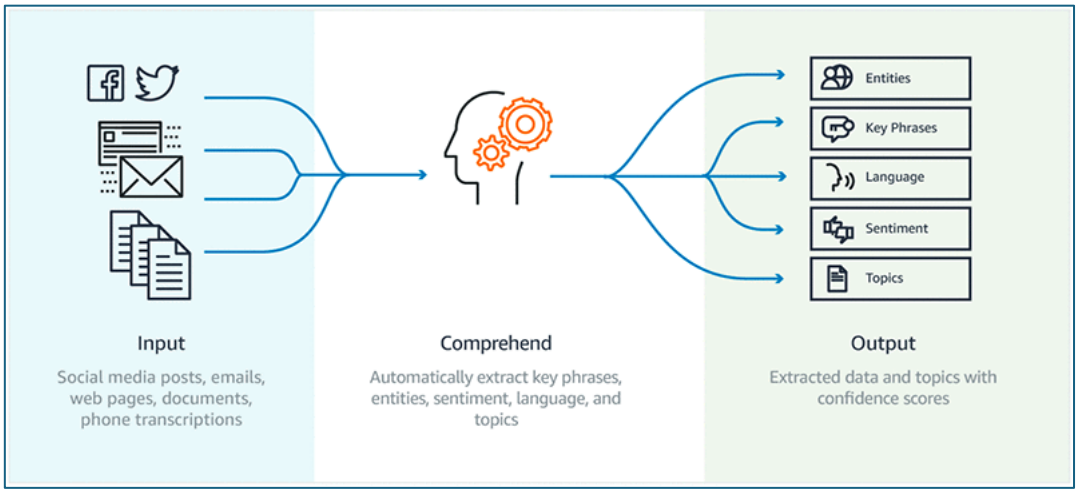


Рисунок 1.8 – Принцип роботи  Amazon Comprehend

Цей огляд показує різноманітність підходів до вирішення проблеми детекції токсичності. Кожен з аналогів має свої сильні та слабкі сторони, що необхідно враховувати при розробці власної системи. Важливо зазначити, що багато з розглянутих готових рішень, особливо комерційні API, часто пропонують обмежену гнучкість у налаштуванні під специфічні потреби конкретного сервісу чи бізнесу. Вони можуть не враховувати унікальний

контекст, специфіку аудиторії або особливості мови, що використовуються на конкретній платформі. У зв'язку з цим, для багатьох компаній та проектів оптимальним шляхом може бути розробка власної, кастомної системи детекції токсичності.

## 1.3Постановка задачі

### 1.3.1 Призначення розробки

Розроблюваний програмний прототип системи детекції токсичності призначений для автоматичного аналізу текстових коментарів у соціальних мережах з метою виявлення та класифікації різних типів токсичного контенту. Кінцевою метою такої системи є сприяння створенню безпечнішого та більш конструктивного онлайн-середовища шляхом надання інструменту для ефективнішої модерації контенту.

Об’єктом дослідження визначено процеси аналізу текстових коментарів у соціальних мережах та їх класифікації за ознаками токсичності.

Предметом дослідження є методи обробки природної мови (ОПМ) та алгоритми машинного навчання (МН), що застосовуються для виявлення токсичного контенту. Зокрема, розглядається модель Neural Bag-of-Words (NBoW) та методи підвищення її ефективності при роботі з незбалансованими наборами даних, такі як використання зваженої функції втрат та оптимізація порогів класифікації.

### 1.3.2 Мета та задачі розробки

Метою даної кваліфікаційної роботи є розробка системи автоматичного детектування токсичності у текстових коментарях соціальних мереж, її експериментальне дослідження, а також створення програмного прототипу для демонстрації практичного застосування розробленої системи та оцінки ефективності обраних підходів машинного навчання.

Для досягнення поставленої мети в рамках кваліфікаційної роботи необхідно вирішити такі завдання:

1. Провести аналіз сучасних підходів та інструментальних засобів для детекції токсичності в текстових даних.
2. Здійснити підготовку набору даних: збір, аналіз та комплексну передню обробку текстових коментарів для навчання та тестування моделі.
3. Розробити та програмно реалізувати базову модель машинного навчання (Neural Bag-of-Words) для класифікації коментарів за рівнем токсичності.
4. Дослідити проблему дисбалансу класів та її вплив на якість класифікації, а також застосувати відповідні методи для її нівелювання, зокрема зважену функцію втрат.
5. Виконати експерименти з оптимізації порогів класифікації з метою покращення метрики F1-score для рідкісних класів токсичності.
6. Здійснити комплексну оцінку якості розробленої моделі на основі релевантних метрик (Precision, Recall, F1-score, ROC AUC).
7. Розробити програмний інтерфейс застосунку (API) для забезпечення взаємодії з навченою моделлю.
8. Створити користувацький веб-інтерфейс для наочної демонстрації функціональності розробленої системи детекції токсичності.

Таким чином, дана робота спрямована не лише на створення функціонуючого програмного прототипу, але й на поглиблення знань та практичних навичок у сфері обробки природної мови та машинного навчання, що є важливим аспектом професійного становлення майбутнього фахівця з комп’ютерних наук.

Висновки до розділу

У даному розділі було детально описано предметне середовище, пов'язане з проблемою токсичності коментарів у соціальних мережах. Проаналізовано актуальність завдання автоматичного виявлення такого контенту, розглянуто його основні типи, характерні ознаки та негативний

вплив на користувачів і онлайн-спільноти, що було узагальнено у відповідній таблиці та проілюстровано прикладами можливих графічних матеріалів. Було представлено опис процесу діяльності, що включає обробку та модерацію коментарів із застосуванням системи детекції токсичності, а також наведено місце для відповідної UML-діаграми діяльності. Далі було описано функціональну модель розроблюваної системи, що охоплює ключових акторів, основні функціональні вимоги, представлені у табличній формі, та варіанти їх використання, для ілюстрації яких передбачено місця для UML-діаграми варіантів використання та загальної архітектурної схеми програмного прототипу. Крім того, проведено огляд та аналіз існуючих аналогічних рішень у сфері детекції токсичного контенту, визначено їхні ключові особливості, підходи до реалізації, переваги та недоліки. На завершення розділу сформульовано постановку задачі кваліфікаційної роботи: визначено призначення розроблюваного програмного прототипу, його основну мету та перелік конкретних завдань, що підлягають вирішенню.

# РОЗДІЛ 2
# ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

## 2.1 Аналіз предметної області

Із стрімким зростанням обсягів користувацького контенту в соціальних мережах та на онлайн-платформах, проблема поширення токсичних коментарів набуває все більшої гостроти. Необхідність підтримання здорового та безпечного комунікаційного середовища змушує розробників та власників платформ шукати ефективні інструменти для модерації. Ручна модерація, хоча й важлива, часто не в змозі впоратися з масштабами проблеми, що робить актуальною розробку автоматизованих систем виявлення токсичності. Такі системи, засновані на методах обробки природної мови та машинного навчання, покликані оперативно ідентифікувати та класифікувати небажаний контент, мінімізуючи його негативний вплив.

Система автоматичного виявлення токсичності – це програмний комплекс, призначений для аналізу текстових даних (коментарів, повідомлень) та ідентифікації в них різних форм токсичної поведінки. Така система зазвичай включає компоненти для попередньої обробки тексту, вилучення ознак, застосування моделі машинного навчання для класифікації та надання результатів аналізу. Для практичного застосування та демонстрації, подібні системи часто реалізуються у вигляді програмних прототипів, що включають серверний компонент (API) для інтеграції з іншими сервісами та клієнтський компонент (UI) для взаємодії з користувачем.

Ключові характеристики системи автоматичного виявлення токсичності:

1. Автоматизація процесу: Система виконує аналіз та класифікацію коментарів без прямого втручання людини на кожному етапі, хоча результати можуть використовуватися для підтримки ручної модерації.
2. Базування на машинному навчанні: Ядром системи є модель машинного навчання, навчена на великому наборі даних з розміченими прикладами токсичних та нетоксичних коментарів.
3. Обробка природної мови (NLP): Використовуються методи NLP для перетворення текстових даних у формат, придатний для аналізу моделлю (наприклад, векторизація).
4. Масштабованість: Архітектура системи (особливо її API) повинна бути розрахована на обробку значної кількості запитів.
5. Інтегрованість: Наявність API дозволяє легко інтегрувати систему з існуючими веб-платформами, форумами, чат-ботами та іншими онлайн-сервісами.
6. Спеціалізація: Система налаштована на виявлення конкретних типів токсичності, визначених у завданні (образи, погрози, мова ворожнечі тощо).

### 2.1.1 Вхідні дані

Для розробки, навчання, тестування та функціонування системи автоматичного виявлення токсичності використовуються різні типи вхідних даних.

1. Дані для навчання, валідації та тестування моделі:
   - Текстові коментарі. Набір текстових повідомлень, зібраних з онлайн-платформ (датасет з Kaggle "Jigsaw Toxic Comment Classification").
   - Мітки токсичності. Для кожного коментаря в навчальному наборі даних наявні бінарні мітки, що вказують на його належність до однієї або декількох з шести категорій токсичності (toxic, severe_toxic, obscene, threat, insult, identity_hate).

Дані для функціонування програмного прототипу (API та UI):

- Текстовий коментар для аналізу. Рядок тексту, що надсилається на обробку через API або вводиться користувачем в UI.

  Приклад запиту до API у форматі JSON :

```
{
  "text": " Here's a sample comment."
}
```

2. Конфігураційні дані та артефакти моделі :

- Файл стану навченої моделі: Збережені ваги та архітектура натренованої NBoW моделі (файли .pth та .json).
- Словник (nbow_vocab_config): Словник, що відображає унікальні слова з навчальної вибірки на їх числові індекси.
- Параметри моделі та передобробки: Фіксована довжина послідовності (SEQ_LENGTH), параметри ембеддингів тощо.
- Оптимальні пороги класифікації: Набір порогів для кожної категорії токсичності, розрахованих для покращення метрик.
- Компоненти бібліотеки spaCy: Завантажена мовна модель для лематизації та видалення стоп-слів.

### 2.1.2 Вихідні дані

Система генерує різні типи вихідних даних на етапі розробки (навчання та оцінки моделі) та під час її практичного використання через API та UI.

1. Результати навчання та оцінки моделі (на етапі розробки):

- Показники якості моделі: Значення метрик Precision, Recall, F1-score (для кожного класу та усереднені: micro, macro, weighted), ROC AUC (для кожного класу та середній), Exact Match Ratio.
- Збережені артефакти: Файл стану найкращої моделі, конфігураційний файл з параметрами та словником.

2. Результати роботи API:

- Класифікація токсичності: JSON-відповідь, що містить масив або об'єкт з ймовірностями (від 0 до 1) належності вхідного коментаря до кожної з шести категорій токсичності.

  Приклад відповіді API у форматі JSON :

```
{
  "toxic": 0.85,
  "severe_toxic": 0.15,
  "obscene": 0.60,
  "threat": 0.05,
  "insult": 0.40,
  "identity_hate": 0.10
}
```

3. Дані, що відображаються в Користувацькому Інтерфейсі (UI) (рис 2.1):

- Загальний статус коментаря. Текстове повідомлення, наприклад, "Коментар виглядає безпечним" або "Виявлено потенційну токсичність".
- Деталізована класифікація. Список виявлених категорій токсичності (якщо ймовірність перевищує встановлений поріг для даної категорії) та відповідні їм ймовірності, представлені у числовому та/або графічному вигляді (наприклад, кольорові смужки).
- Вхідний текст. Відображення тексту, який було проаналізовано.

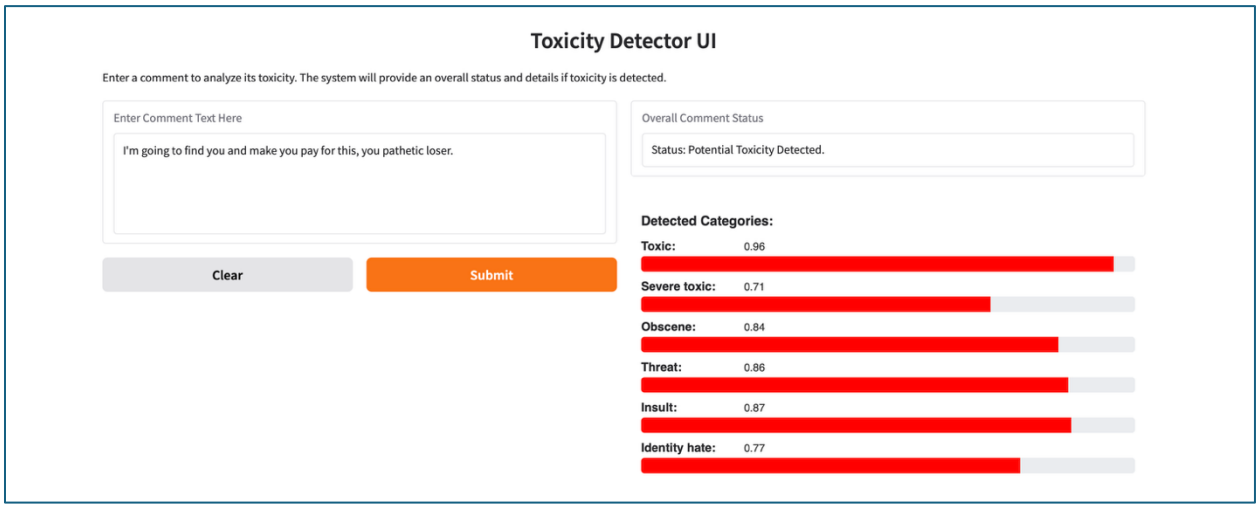


Рисунок 2.1 – Приклад роботи UI

Ці вхідні та вихідні дані визначають інформаційні потоки та основні аспекти функціонування системи детекції токсичності на всіх етапах її життєвого циклу.

## 2.2 Проєктування системи

Проєктування системи є фундаментальним етапом розробки будь-якого програмного продукту. Воно охоплює визначення архітектури, ключових компонентів, їх взаємодії, а також детальний опис структур даних, які система буде використовувати для своєї роботи та зберігання інформації. Для системи автоматичного виявлення токсичності коментарів це означає ретельне планування як алгоритмічної частини, пов'язаної з моделлю машинного навчання, так і програмних інтерфейсів (API та UI), через які здійснюється взаємодія з системою та її компонентами.

### 2.2.1 Проєктування структури даних та управління артефактами моделі

Для ефективної роботи системи машинного навчання, якою є розроблюваний детектор токсичності, критично важливим є чітке визначення структури вхідних даних, проміжних артефактів та вихідних результатів. Управління цими даними забезпечує відтворюваність експериментів, коректну роботу моделі та можливість її подальшого вдосконалення. На відміну від систем, що орієнтовані на транзакційні бази даних, у даному проекті основними інформаційними активами є навчальний набір даних та артефакти, згенеровані в процесі навчання моделі [8].

Процес проєктування структури даних для системи машинного навчання включає:

- Аналіз вимог до вхідних даних для навчання та роботи моделі.
- Визначення формату та структури навчального набору даних.

- Визначення формату та структури для зберігання артефактів моделі (словників, конфігурацій, ваг моделі).
- Опис механізмів завантаження та використання цих артефактів у робочому середовищі системи (API, UI).

Структура вхідного набору даних (Dataset) Для навчання та оцінки моделі детекції токсичності використовується публічний набір даних, наприклад, з платформи Kaggle (Jigsaw Toxic Comment Classification Challenge). Такі набори даних зазвичай представлені у форматі CSV-файлів, де кожен рядок відповідає одному коментарю.

Типова структура колонок включає:

- id: унікальний ідентифікатор коментаря.
- comment_text: текстовий зміст коментаря.
- toxic: бінарна мітка (0 або 1), що вказує на наявність/відсутність загальної токсичності.
- severe_toxic: бінарна мітка для вираженої токсичності.
- obscene: бінарна мітка для непристойності.
- threat: бінарна мітка для погроз.
- insult: бінарна мітка для образ.
- identity_hate: бінарна мітка для мови ворожнечі на основі ідентичності. Ця структурована інформація є основою для навчання класифікатора.

Артефакти моделі та конфігураційні файли у процесі передобробки даних та навчання моделі генеруються декілька ключових артефактів, які необхідно зберігати для подальшого використання системою:

а) Словник токенів (nbow_vocab_config.json) Цей файл зберігає відповідність між унікальними словами (токенами), виявленими у навчальному наборі даних, та їх числовими індексами. Він є критично важливим для перетворення нових текстових коментарів у числовий формат, який може обробляти модель. Структура файлу зазвичай є JSON-об'єктом, де ключами є токени, а значеннями – їхні індекси. Також словник включає

спеціальні токени, такі як <pad> для вирівнювання послідовностей та <unk> для невідомих слів.

Приклад nbow_model_state.json :

```
{
"<pad>": 0,
"<unk>": 1,
"hello": 2,
"world": 3,
}
```

б) Стан навченої моделі (nbow_model_state.pth). Це бінарний файл, що містить збережені ваги та параметри навченої нейронної мережі (у випадку PyTorch – результат model.state_dict()). Цей файл завантажується для відновлення навченої моделі без необхідності повторного навчання.

Управління артефактами. Усі зазначені артефакти (навчальний датасет, словник, файл стану моделі) зберігаються у файловій системі у спеціально відведеній директорії (../model_artifacts/). Програмні компоненти системи (API та UI) при старті завантажують конфігураційний файл, словник, стан моделі та інші необхідні ресурси (наприклад, мовну модель spaCy) для забезпечення своєї функціональності. Це дозволяє системі коректно обробляти нові вхідні коментарі відповідно до параметрів та знань, отриманих під час навчання.

Такий підхід до проєктування структури даних та управління артефактами забезпечує модульність, відтворюваність та можливість подальшого оновлення моделі та її компонентів. На відміну від традиційних баз даних, тут основний акцент робиться на версіонуванні та зберіганні файлів, що є типовим для багатьох проектів у галузі машинного навчання.

## 2.3 Математичне та алгоритмічне забезпечення

Функціонування розробленої системи детекції токсичності ґрунтується на сукупності алгоритмів обробки природної мови та математичних моделей. У цьому підрозділі детально розглянуто ключові алгоритмічні та

математичні аспекти, що забезпечують перетворення текстових даних, побудову та навчання класифікаційної моделі.

### 2.3.1 Алгоритми попередньої обробки тексту

Попередня обробка тексту є етапом нормалізації та очищення вхідних коментарів. Нехай вхідний коментар представлено як текстовий рядок Craw. Процес обробки можна описати як композицію функцій :

$$C_{clean} = f_{spacy}(f_{regex}(C_{raw})) \quad (2.1)$$

де $C_{raw}$ — початковий коментар, $f_{regex}$ — функція очищення тексту, $f_{spacy}$ — функція обробки слів, $C_{clean}$ — фінальний чистий текст.

1. Очищення за допомогою регулярних виразів (fregex): Ця функція послідовно застосовує операції для видалення нерелевантних елементів, таких як URL-адреси, згадки користувачів (@username) та HTML-теги. Текст також приводиться до нижнього регістру, а всі символи, окрім латинських літер та пробілів, видаляються.
2. Обробка за допомогою spaCy. Обробка токенізованого тексту включає:
   - Лематизація: Кожен токен (слово) w з очищеного тексту перетворюється на свою базову словникову форму (лему) $L(w_i)$ Наприклад, L("running")→"run".
   - Видалення стоп-слів: Токени, що належать до заздалегідь визначеної множини стоп-слів S, вилучаються. Оброблений список токенів Tfinal для коментаря виглядає так:

$$T_{final} = \{L(w_i) \mid w_i \in T_{regex}, L(w_i) \notin S\} \quad (2.2)$$

де $T_{final}$ — кінцевий набір слів (токенів), $T_{regex}$ — набір слів після очищення, $w_i$ — окреме слово (токен), $L(w_i)$ — базова форма слова (лема), S — множина стоп-слів (зайвих слів).

### 2.3.2 Алгоритми підготовки ознак для моделі Neural Bag-of-Words

Після обробки, послідовність токенів Tfinal перетворюється у числовий формат, придатний для моделі.

1. Створення словника (Vocabulary): На основі навчальної вибірки створюється словник V, що є відображенням кожного унікального токена на унікальний цілочисельний індекс.

$$V: token \rightarrow index \tag{2.3}$$

Словник також містить спеціальні токени: <pad> для вирівнювання (індекс 0) та <unk> для невідомих слів (індекс 1).

2. Нумерикалізація. Кожен токен у $T_{final}$ замінюється на відповідний індекс зі словника V. Якщо токен відсутній у V, використовується індекс <unk>.
3. Падинг (Padding). Усі нумеризовані послідовності приводяться до фіксованої довжини $L_{SEQ}$ (наприклад, 200).
   - Якщо довжина послідовності менша за $L_{SEQ}$, вона доповнюється індексами токена <pad> на початку.
   - Якщо довжина більша, послідовність обрізається з початку.

Результатом є цілочисельний вектор x фіксованої довжини $L_{SEQ}$ для кожного коментаря.

### 2.3.3 Математична модель класифікатора (Neural Bag-of-Words)

Модель NBoW, реалізована на PyTorch, перетворює вектор індексів x на вектор ймовірностей для 6 класів токсичності.

1. Шар вкладень (Embedding Layer). Кожен індекс з вектора x відображається у векторне представлення (ембеддинг).
2. Усереднення вкладень (Averaging). Усі вектори ембеддингів для одного коментаря усереднюються, що є реалізацією підходу "мішок слів".

$$\nu = \frac{1}{L_{SEQ}} \sum_{i=1}^{L_{SEQ}} e_i \quad (2.4)$$

де $\nu$ — усереднений вектор ознак для всього коментаря, $L_{SEQ}$ — фіксована довжина послідовності, $e_i$ — вектор-ембеддинг для і-го токена в послідовності.

3. Повнозв'язний шар (Fully Connected Layer). Усереднений вектор $\nu$ подається на вхід повнозв'язного шару, який виконує лінійне перетворення для отримання "сирих" оцінок (логітів).

$$z = \nu W^T + b \quad (2.5)$$

де z — вихідний вектор логітів (один логіт для кожного з 6 класів), $\nu$ — усереднений вектор ознак з попереднього кроку, W — матриця ваг повнозв'язного шару, $W^T$ — транспонована матриця ваг, b — вектор зсуву (bias) повнозв'язного шару.

4. Функція активації (Sigmoid): До кожного логіта zi застосовується сигмоїдальна функція σ для отримання ймовірності належності до і-го класу.

$$P_{(class\ i)} = \sigma(z_i) = \frac{1}{1 + e^{-z_i}} \quad (2.6)$$

де P(class i) — розрахована ймовірність того, що коментар належить до класу i, $z_i$ — логіт для класу i з вектора z, e — основа натурального логарифма (число Ейлера, e ≈ 2.718).

### 2.3.4 Алгоритм навчання моделі

Навчання моделі полягає у мінімізації функції втрат за допомогою оптимізатора.

1. Зважена функція втрат. Для боротьби з дисбалансом класів вага для позитивних прикладів кожного класу розраховується за формулою:

$$p_c = \frac{\text{кількість негативних прикладів для класу c}}{\text{кількість позитивних прикладів для класу c}}$$

де $p_c$ — вага для класу c, c — конкретний клас токсичності.

2. Оптимізатор. Алгоритм Adam використовується для ітеративного оновлення ваг моделі (W та b) з метою мінімізації загальної функції втрат.

Висновки до розділу

У даному розділі було закладено інформаційне та математичне підґрунтя для розробки системи детекції токсичності. Визначено структуру вхідних та вихідних даних, а також спроєктовано формати для зберігання ключових артефактів моделі: словника та стану навченої мережі. Детально розглянуто алгоритмічний пайплайн, що включає формалізований опис попередньої обробки тексту, його нумерикалізацію та перетворення у вектори фіксованої довжини. Представлено математичну модель класифікатора Neural Bag-of-Words, яка крок за кроком описує перетворення вхідного вектора на ймовірності належності до класів токсичності. Ключову увагу приділено обґрунтуванню алгоритму навчання, зокрема математичному розрахунку вагових коефіцієнтів для функції втрат, що спрямовано на вирішення проблеми незбалансованих даних. Таким чином, розділ надає повну теоретичну та алгоритмічну базу, необхідну для подальшої програмної реалізації та експериментального дослідження системи.

# РОЗДІЛ 3

# ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

## 3.1 Засоби розробки

Програмний прототип, створений у рамках даної кваліфікаційної роботи, був розроблений з використанням мови програмування Python та її багатої екосистеми спеціалізованих бібліотек для науки про дані та машинного навчання. Вибір Python обумовлений його гнучкістю, великою спільнотою та наявністю потужних інструментів для всіх етапів розробки.

1. PyTorch — провідний фреймворк для побудови та навчання моделей глибокого навчання, розроблений Facebook AI Research. Його було обрано за гнучкість динамічного обчислювального графа, що значно спрощує процес експериментів та налагодження архітектур нейронних мереж.
2. Pandas та NumPy — фундаментальні бібліотеки для роботи з даними в Python. Pandas використовувався для завантаження, аналізу та маніпуляції табличними даними (датасетом), а NumPy — для ефективних числових та векторних операцій.
3. spaCy — сучасна бібліотека для промислової обробки природної мови (NLP). У проєкті її було задіяно для виконання ключових завдань передобробки тексту, таких як токенізація, лематизація та видалення стоп-слів. Для первинного очищення тексту від специфічних патернів (URL, HTML-тегів) також використовувався вбудований модуль для роботи з регулярними виразами re.
4. Scikit-learn — одна з найпопулярніших бібліотек для машинного навчання, яку в даній роботі було використано для комплексної оцінки якості розробленої моделі, зокрема для розрахунку метрик Precision, Recall, F1-score та ROC AUC.

5. FastAPI та Uvicorn — зв'язка для створення високопродуктивних програмних інтерфейсів (API). FastAPI є сучасним вебфреймворком, що дозволяє швидко розробляти API з автоматичною генерацією документації. Uvicorn виступає в ролі ASGI-сервера, необхідного для асинхронного запуску додатку.
6. Gradio — бібліотека, що дозволяє швидко створювати прості та наочні користувацькі веб-інтерфейси для демонстрації роботи моделей машинного навчання. Її було використано для реалізації клієнтської частини прототипу.
7. Jupyter Notebook — інтерактивне середовище розробки, яке використовувалося на етапах дослідження даних, розробки та навчання моделі завдяки можливості покрокового виконання коду та візуалізації проміжних результатів.
8. Visual Studio Code — багатофункціональний редактор вихідного коду, Він підтримує сотні мов програмування через систему розширень.
9. Git та GitHub — розподілена система контролю версій та веб-сервіс. Git використовувався для керування версіями коду впродовж розробки, а GitHub — для централізованого зберігання коду проєкту, відстеження змін та організації роботи.

## 3.2 Вимоги до технічного та програмного забезпечення

Для розробки, навчання моделі та розгортання серверної частини (API), а також для кінцевого користувача, що взаємодіє з системою через веб-інтерфейс, визначено наступні вимоги до технічного та програмного забезпечення.

Для розробки, навчання моделі та функціонування серверної частини (API) рекомендується така конфігурація, на основі якої велася розробка:

- Операційна система. macOS, Windows 10/11 або сучасних дистрибутивах Linux (наприклад, Ubuntu 20.04+).
- Процесор. Робота проводилася на процесорі Apple M2. Для інших платформ рекомендується 4-ядерний процесор з тактовою частотою 2.5 ГГц або краще (наприклад, Intel Core i5, AMD Ryzen 5).
- Оперативна пам'ять: Система була оснащена 16 ГБ оперативної пам'яті, що є рекомендованим об'ємом для комфортної роботи з великими наборами даних. Мінімальною вимогою є 8 ГБ.
- Вільний дисковий простір: Мінімум 20 ГБ для зберігання операційної системи, середовища розробки, бібліотек, набору даних та артефактів моделі. Рекомендовано використовувати SSD для швидкого доступу до даних.
- Відеокарта (опціонально, але рекомендовано для прискорення навчання): Хоча розробка велася на Apple M2 з його вбудованим графічним ядром, для суттєвого прискорення навчання нейронних мереж на платформі PyTorch рекомендовано використовувати дискретну відеокарту NVIDIA з підтримкою CUDA (наприклад, серії GeForce RTX).
- Програмне забезпечення: Python версії 3.8 або вище, система контролю версій Git, а також встановлені бібліотеки проєкту, зазначені у розділі 3.1 (PyTorch, FastAPI, spaCy та інші).

Для функціонування робочого місця кінцевого користувача (доступ до веб-інтерфейсу Gradio) необхідна така мінімальна конфігурація:

- Операційна система: Будь-яка сучасна настільна ОС (Windows 10+, macOS, Linux).
- Процесор: 2-ядерний процесор з частотою 1.6 ГГц або краще.
- Оперативна пам'ять: 4 ГБ або більше.
- Веб-браузер: Остання стабільна версія Google Chrome, Mozilla Firefox, Safari або Microsoft Edge.
- Підключення до мережі Інтернет для взаємодії з розгорнутим API та інтерфейсом.

## 3.3 Опис програмної реалізації

Програмна реалізація проєкту складається з кількох ключових, взаємопов'язаних компонентів:

- навченої моделі машинного навчання.
- серверного програмного інтерфейсу (API) для взаємодії з моделлю.
- та клієнтського користувацького інтерфейсу (UI) для демонстрації її роботи.
- Взаємодія цих компонентів, а також внутрішня логіка їх роботи, детально описана за допомогою UML-діаграм.

### 3.3.1 Діаграма компонентів

Діаграма компонентів ілюструє загальну архітектуру програмного прототипу та зв'язки між його основними модулями. Вона показує, як окремі частини системи взаємодіють для виконання спільного завдання. Розроблена діаграма компонентів наведена на рис. 3.1.

Система складається з чотирьох основних компонентів. Користувацький інтерфейс (Gradio) є точкою входу для користувача; він надсилає текстові дані на аналіз. API модуль (FastAPI) приймає ці запити, оркеструє процес обробки та повертає результат. Модуль Передобробки тексту виконує очищення та векторизацію даних, використовуючи артефакти зі Сховища даних (словник). Модель машинного навчання (PyTorch NBoW) приймає оброблений текст, виконує класифікацію та повертає результат API модулю.

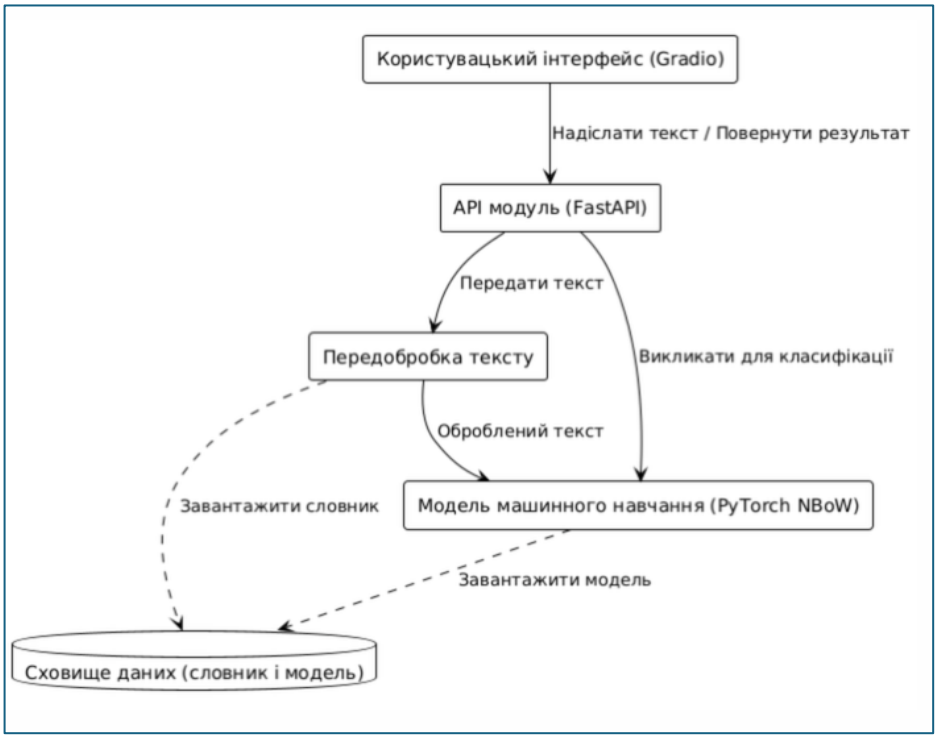


Рисунок 3.1 – Діаграма компонентів системи

3.3.2 Діаграма послідовності

Діаграма послідовності демонструє хронологічний порядок взаємодії між об'єктами системи під час виконання конкретного сценарію. На рисунку 3.2 наведено діаграму послідовності для сценарію "Аналіз токсичності коментаря".

Користувач вводить текст у веб-інтерфейс (Gradio). Інтерфейс надсилає HTTP POST-запит з текстом до API на FastAPI. API послідовно викликає модуль передобробки для перетворення тексту на числовий тензор, а потім передає цей тензор навченій моделі PyTorch для отримання логітів. Після застосування функції активації для отримання ймовірностей, API формує JSON-відповідь та повертає її користувацькому інтерфейсу, який, у свою чергу, візуалізує результат для користувача.

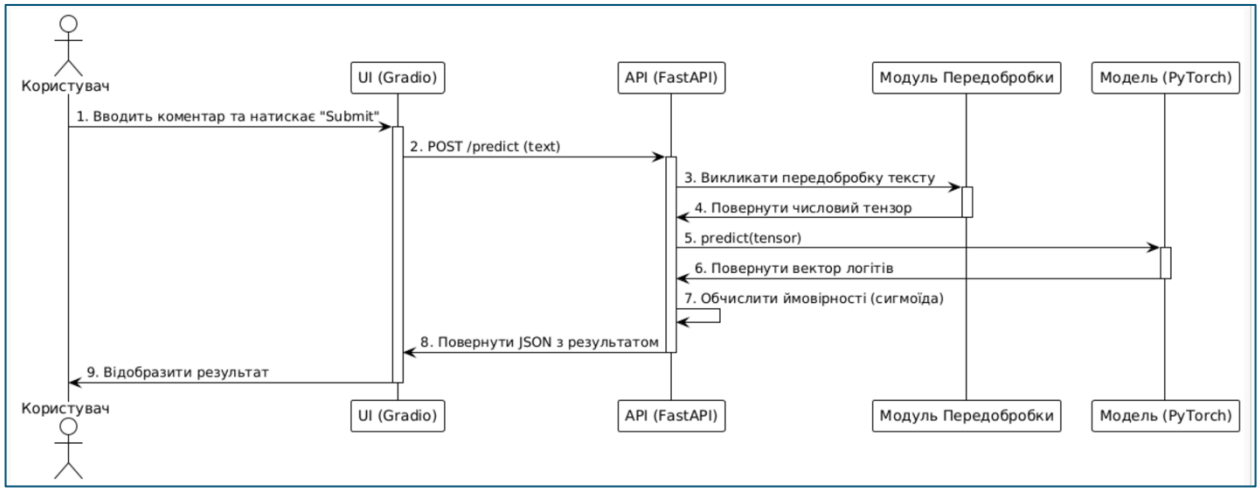


Рисунок 3.2 – Діаграма послідовності для аналізу коментаря

### 3.3.3 Діаграма діяльності

Діаграма діяльності візуалізує покроковий робочий процес (workflow) всередині системи. На відміну від діаграми послідовності, вона фокусується на послідовності дій та умовах їх виконання. На рисунку 3.3 наведено діаграму діяльності, що описує логіку роботи ендпоінту /predict в API.

Процес починається з отримання текстового коментаря через API. Далі виконується послідовна передобробка: приведення до нижнього регістру, видалення зайвих елементів, лематизація та видалення стоп-слів. Очищені токени перетворюються на числові індекси за допомогою словника, і послідовність вирівнюється до фіксованої довжини (падинг). На основі отриманого тензора нейронна мережа виконує класифікацію. На виході формується JSON-відповідь з ймовірностями для кожного класу, яка повертається клієнту.


Рисунок 3.3 – Діаграма діяльності процесу детекції токсичності

На основі розробленої архітектури та алгоритмів було створено програмний прототип. Фрагменти ключових програмних кодів, що реалізують описані модулі, наведені у Додатку А.

## 3.4 Керівництво користувача

Програмний прототип системи детекції токсичності складається з двох окремих, але взаємопов'язаних частин: серверного програмного інтерфейсу (API) та клієнтського користувацького інтерфейсу (UI). Для повноцінної роботи необхідно послідовно запустити обидва компоненти.

Перш за все, необхідно переконатися, що всі залежності проєкту, перелічені в розділі 3.1, встановлені у вашому середовищі Python. Також важливо, щоб директорія з артефактами моделі (model_artifacts) знаходилася у правильному відносному шляху до скриптів, як це було визначено під час розробки. Для запуску прототипу, треба виконати наступні кроки :

Крок 1: Запуск серверної частини (API)

Серверна частина на базі FastAPI є ядром системи, яке виконує обчислення. Її необхідно запустити першою. Для цього у терміналі, перебуваючи в кореневій директорії проєкту, потрібно виконати наступну команду:

```
uvicorn main:app --reload
```

Після успішного запуску в терміналі (рис. 3.4) з'явиться повідомлення, що інформує про запуск сервера на локальній адресі та порту, зазвичай http://127.0.0.1:8000. Це означає, що API готовий приймати запити.

```
mykytazaginei@MacBook-Air--Mykyta api % uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['/Users/mykytazaginei/Desktop/Toxic_Comment_Classification/api']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [88027] using WatchFiles
INFO:     Started server process [88029]
INFO:     Waiting for application startup.
Loading API resources...
API will use device: cpu
spaCy 'en_core_web_sm' model loaded successfully for API.
Vocabulary (size: 197521) and SEQ_LENGTH (200) loaded from ../model_artifacts/nbow_vocab_config.json.
Model state loaded from '../model_artifacts/nbow_model_state.pth' and moved to cpu.
API resources loaded in 0.57 seconds.
INFO:     Application startup complete.
```

Рисунок 3.4 – Результат запуску API-сервера у терміналі

Крок 2: Запуск користувацького інтерфейсу (UI)

Не закриваючи термінал з запущеним API, необхідно відкрити новий термінал. У ньому, перебуваючи в тій самій директорії проєкту, слід запустити скрипт додатку Gradio наступною командою:

```
python3 gradio_app.py
```

Після запуску цього скрипту в терміналі (рис. 3.5). з'явиться локальна URL-адреса, за якою буде доступний веб-інтерфейс, наприклад, http://127.0.0.1:7860.


Рисунок 3.5 – Результат запуску UI у терміналі

Крок 3: Робота з веб-інтерфейсом

Відкривши отриману на попередньому кроці URL-адресу в веб-браузері, користувач побачить головний інтерфейс додатку. Він містить текстове поле для введення коментаря, кнопки "Submit" (Надіслати) та "Clear" (Очистити), а також секцію для виведення результатів. Інтерфейс зображено на рисунку 3.6.

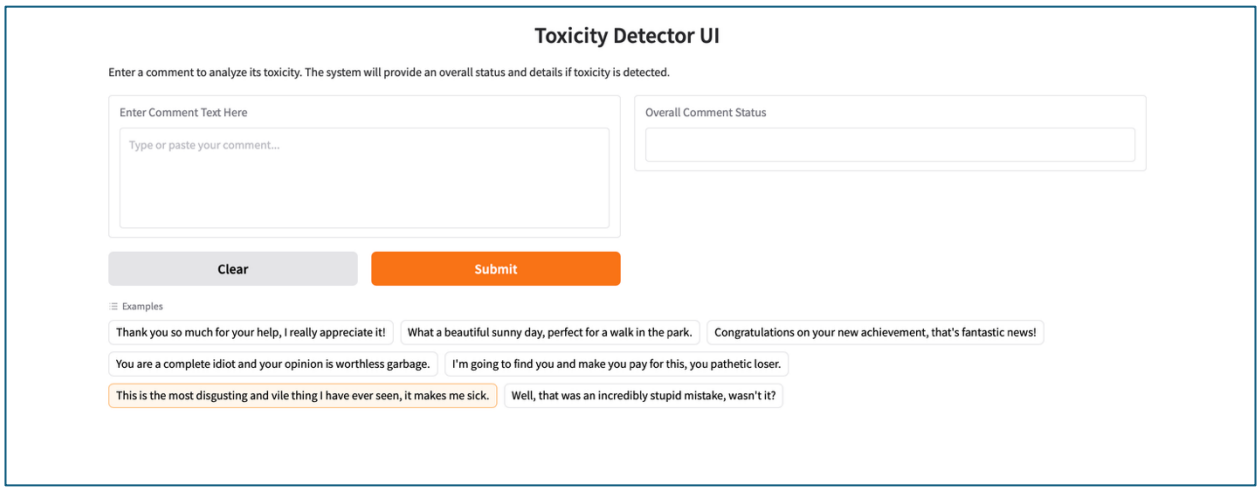


Рисунок 3.6 – Головний інтерфейс додатку для детекції токсичності

Для аналізу коментаря необхідно ввести текст у поле "Enter Comment Text Here" та натиснути кнопку "Submit". Інтерфейс надішле запит до запущеного API та відобразить результат.

Крок 4: Інтерпретація результатів

Результат аналізу відображається у двох блоках. У блоці "Overall Comment Status" виводиться загальний вердикт. Якщо коментар не є токсичним, з'явиться повідомлення "Status: This appears to be a Good Comment". Інтерфейс зображено на рисунку 3.7.

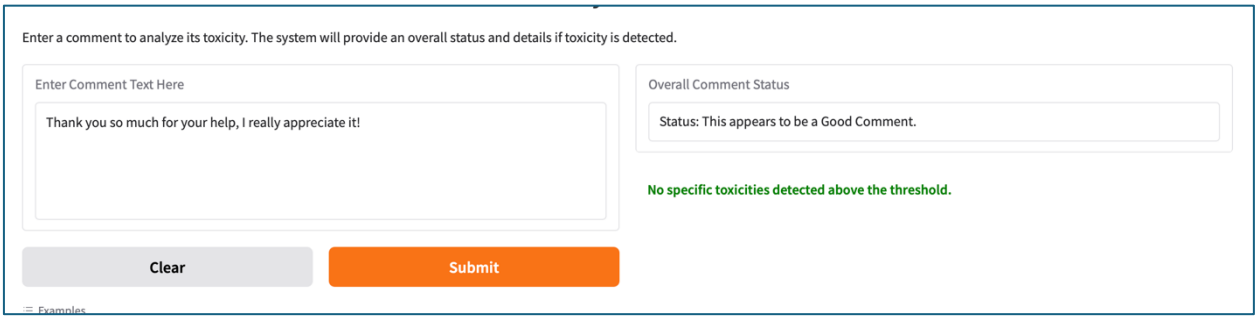


Рисунок 3.7 – Приклад результату для безпечного коментаря

Якщо в коментарі виявлено ознаки токсичності, статус зміниться на "Status: Potential Toxicity Detected" (рис. 3.8) Для кожної з шести категорій токсичності, яка перевищила свій поріг, буде показано її назву та розраховану ймовірність.

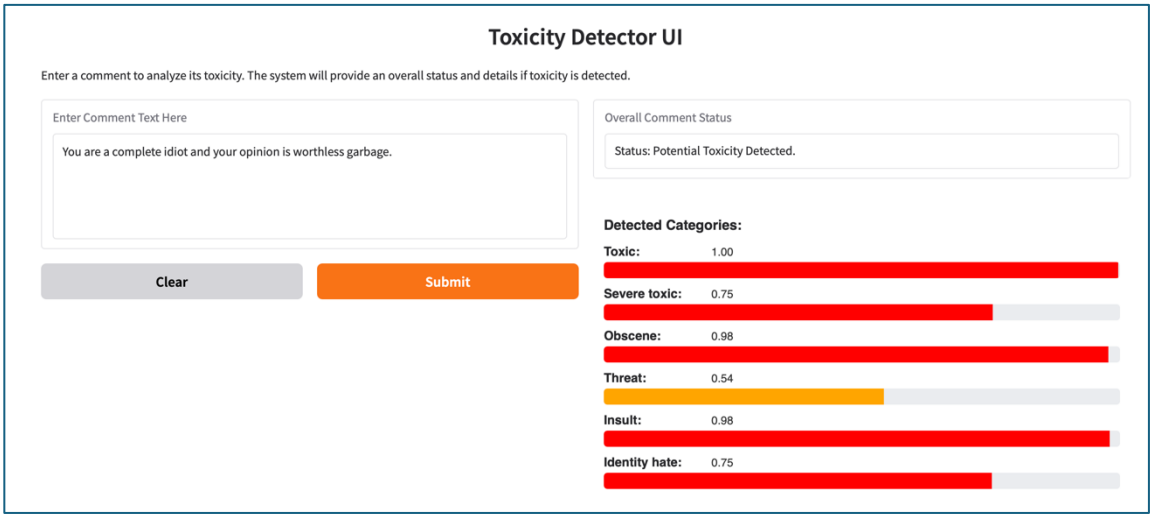


Рисунок 3.8 – Приклад результату для токсичного коментаря з деталізацією

Для зручності тестування, в інтерфейсі також передбачені готові приклади коментарів, які можна обрати для швидкого аналізу, натиснувши на один з них під полем вводу. Їх можна побачити на рисунку 3.8.

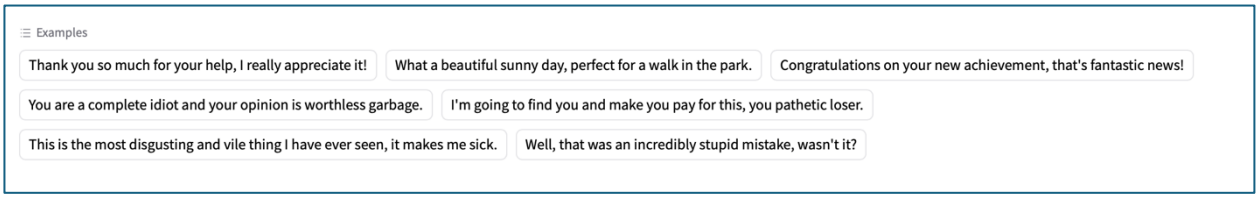


Рисунок 3.8 – Приклади коментарів

Висновки до розділу

У даному розділі було детально розглянуто аспекти програмної та технічної реалізації розробленого прототипу системи детекції токсичності. Було наведено перелік ключових засобів розробки, що включає мову програмування Python та її спеціалізовані бібліотеки, такі як PyTorch, FastAPI та Gradio. Разом з цим було сформульовано вимоги до технічного та програмного забезпечення, необхідного для розгортання середовища розробки та для кінцевого користувача.

Для наочної демонстрації архітектури та логіки роботи системи було описано її програмну реалізацію за допомогою набору UML-діаграм. Діаграма компонентів розкрила загальну структуру прототипу, діаграма послідовності проілюструвала покрокову взаємодію модулів під час обробки запиту, а діаграма діяльності деталізувала внутрішній робочий процес API-ендпоінту.

На завершення було надано детальне керівництво користувача, що пояснює процедуру запуску серверної та клієнтської частин додатку, а також процес взаємодії з інтерфейсом та інтерпретації отриманих результатів. Таким чином, цей розділ надає вичерпний опис практичної реалізації проєкту та підготовлює основу для аналізу експериментальних результатів.

# РОЗДІЛ 4
# ОХОРОНА ПРАЦІ

Діяльність у сфері інформаційних технологій, зокрема розробка програмного забезпечення, як і будь-яка інша професійна діяльність, вимагає створення безпечних та сприятливих умов праці. Дотримання норм охорони праці є не лише вимогою законодавства, але й ключовим фактором забезпечення здоров'я, високої продуктивності та творчого потенціалу IT-спеціалістів. У даному розділі, відповідно до отриманого завдання, буде проведено аналіз питань охорони праці стосовно об'єкта проєктування - робочого місця розробника програмного забезпечення, що працює над створенням детектора токсичності коментарів.

## 4.1 Регулювання питань охорони праці на законодавчому рівні

Забезпечення безпечних і здорових умов праці в Україні є пріоритетом державної політики та регулюється розгалуженою системою законодавчих та нормативно-правових актів. Фундаментальні права громадян на належні, безпечні й здорові умови праці закріплені в Конституції України. Основним документом, що регулює відносини у цій сфері, є Закон України «Про охорону праці» [10]. Цей закон встановлює єдиний порядок організації охорони праці для всіх підприємств, установ та організацій, незалежно від форми власності, та поширюється на всіх роботодавців і працівників. Його головна мета полягає у збереженні життя, здоров'я і працездатності людини в процесі трудової діяльності.

Об'єктом проєктування в даній кваліфікаційній роботі є процес розробки програмного продукту, що виконується фахівцем у галузі комп'ютерних наук. Характер його діяльності пов'язаний з тривалою роботою за персональним комп'ютером (ПК) та іншими відеодисплейними терміналами (ВДТ), що

визначає специфіку потенційних шкідливих та небезпечних факторів виробничого середовища [11].

Реалізація державної політики у сфері охорони праці покладається на різних суб'єктів, кожен з яких має свої чітко визначені обов'язки:

- Роботодавець**.** Несе повну відповідальність за створення безпечних умов праці на кожному робочому місці. Відповідно до Закону «Про охорону праці» [10], роботодавець зобов'язаний забезпечити функціонування системи управління охороною праці, проводити інструктажі та навчання працівників, організовувати обов'язкові медичні огляди для певних категорій працівників, а також забезпечувати робочі місця обладнанням та устаткуванням, що відповідає чинним нормам.
- Працівник (розробник ПЗ). У свою чергу, зобов'язаний дбати про особисту безпеку і здоров'я, а також про безпеку оточуючих. Це включає виконання вимог інструкцій з охорони праці, правильне використання комп'ютерної техніки та іншого обладнання, проходження медичних оглядів у встановленому порядку та співпрацю з роботодавцем у справі організації безпечних умов праці.
- Державний нагляд та контроль. Здійснюється центральними органами виконавчої влади, зокрема Державною службою України з питань праці. Ці органи контролюють дотримання законодавства, проводять розслідування нещасних випадків та мають право застосовувати санкції до порушників.

З соціально-економічного погляду, приділення належної уваги охороні праці в ІТ-сфері є надзвичайно важливим. Інвестиції у створення ергономічних та безпечних робочих місць безпосередньо впливають на зниження ризику професійних захворювань, пов'язаних із сидячою роботою та навантаженням на зоровий аналізатор [11]. Це, в свою чергу, сприяє підвищенню продуктивності праці, зменшенню економічних втрат через тимчасову непрацездатність та підвищенню лояльності співробітників і загальної привабливості компанії на ринку праці. Таким чином, ефективна система

охорони праці на робочому місці ІТ-спеціаліста базується на неухильному дотриманні законодавства всіма учасниками трудового процесу та є запорукою сталого розвитку як окремого фахівця, так і компанії в цілому.

## 4.2 Виявлення потенційних небезпек стосовно об'єкту проектування

Для розробки ефективних заходів з охорони праці необхідно провести детальний аналіз та ідентифікацію потенційних небезпек, притаманних робочому процесу розробника програмного забезпечення. Об'єктом аналізу є робоче місце ІТ-спеціаліста, обладнане персональним комп'ютером та периферійними пристроями. Діяльність розробника характеризується як розумова праця з високим рівнем концентрації уваги та тривалим перебуванням у статичному положенні.

Відповідно до загальноприйнятої класифікації, проаналізуємо небезпечні та шкідливі виробничі фактори, що можуть впливати на працівника.

1. Фізичні небезпечні й шкідливі виробничі фактори:

Ця група є найбільш представленою для робочого місця ІТ-спеціаліста.

- Підвищений рівень електромагнітних випромінювань. Джерелами є системний блок ПК, монітор, Wi-Fi роутер, мобільні пристрої та інша офісна техніка. Тривалий вплив цих полів може негативно позначатися на стані нервової, імунної та серцево-судинної систем [10].
- Недоліки освітлення. Недостатня освітленість робочої зони, підвищена яскравість, пряма та відбита блискість від екрана монітора та інших поверхонь. Причиною може бути неправильне розташування робочого стола відносно джерел природного світла (вікон) або несправність освітлювальних приладів. Це призводить до перенапруження зорового аналізатора та швидкої втоми.

- Невідповідність параметрів мікроклімату. До них належать підвищена або знижена температура, вологість та недостатня рухливість повітря. Причиною є переважно несправність або неправильне налаштування систем опалення, вентиляції та кондиціонування. Комфортні параметри мікроклімату для офісних приміщень чітко регламентуються державними санітарними нормами [12].
- Підвищений рівень шуму. В умовах офісу, особливо відкритого типу, джерелами шуму є розмови колег, робота оргтехніки та зовнішній шум. Це знижує концентрацію уваги та може спричиняти нервове напруження.
- Пожежна небезпека. Ризик виникнення пожежі пов'язаний з можливістю короткого замикання в електромережі через її перевантаження, використання несправного обладнання або порушення правил пожежної безпеки.

2. Хімічні небезпечні й шкідливі виробничі фактори:

Для офісних приміщень ці фактори є менш вираженими. До них можна віднести підвищену концентрацію пилу в повітрі, який накопичується на поверхнях оргтехніки через статичну електрику, а також можливе виділення незначної кількості шкідливих речовин від матеріалів, з яких виготовлені меблі та техніка.

3. Біологічні небезпечні й шкідливі виробничі фактори:

У місцях скупчення людей завжди існує ризик поширення патогенних мікроорганізмів (вірусів, бактерій), що передаються повітряно-крапельним шляхом, особливо в періоди сезонних епідемій.

4. Психофізіологічні небезпечні й шкідливі виробничі фактори:

Ця група факторів є однією з ключових для розробника програмного забезпечення.

- Фізичні перевантаження. Переважно статичні, викликані необхідністю підтримувати робочу позу протягом тривалого часу. Неправильна організація робочого місця (неергономічне крісло, невірна висота столу) призводить до захворювань опорно-рухового апарату [10].

- Нервово-психічні перевантаження. Це комплексний фактор, що включає:
  - Розумове перенапруження: робота вимагає високої концентрації та вирішення складних аналітичних завдань.
  - Перенапруження зорового аналізатора: тривала безперервна робота з монітором.
  - Емоційні перевантаження*:* стрес, пов'язаний з дотриманням жорстких термінів (дедлайнів) та високою відповідальністю.
- Загальні загрози безпеці. В умовах воєнного стану в Україні до психофізіологічних факторів додається постійний стрес, пов'язаний із загрозою життю через ракетні обстріли, що вимагає додаткових заходів безпеки.

Отже, було виявлено комплекс потенційних небезпек. Хоча повністю усунути їх вплив неможливо, його можна і потрібно мінімізувати. Завданням наступного підрозділу є проведення оцінки ризику реалізації найбільш значущих з виявлених небезпек та розробка конкретних заходів щодо зниження їхнього впливу до прийнятного рівня, як того вимагає Закон України «Про охорону праці» [11].

## 4.3 Дослідження ризику реалізації потенційних небезпек на об’єкті проектування та розробка заходів щодо їх попередження.

Оцінка ризиків є фундаментальним елементом системи охорони праці та здоров'я в будь-якій сучасній організації, зокрема у сфері інформаційних технологій. Ефективність цієї процедури напряму залежить від системного підходу та об'єктивності в аналізі потенційних загроз.

Матриця оцінки ризиків є корисним інструментом, що використовується для ідентифікації, аналізу та пріоритезації ризиків, а також для розробки адекватних заходів управління ними. До ключових переваг використання цього методу можна віднести:

- Допомога у визначенні пріоритетів. Для ІТ-проєкту матриця дозволяє сфокусувати увагу не лише на очевидних, але й на прихованих ризиках. Наприклад, вона допомагає збалансувати увагу між гострими, але малоймовірними подіями та менш драматичними, але постійними загрозами (ергономічні ризики), які мають значний кумулятивний вплив на здоров'я та продуктивність команди.
- Забезпечення проактивного управління ризиками. Використання матриці дає змогу компанії завчасно ідентифікувати та керувати ризиками. Це означає, що заходи, спрямовані на запобігання професійним захворюванням розробників (наприклад, закупівля ергономічних крісел, організація режиму праці та відпочинку), впроваджуються до того, як виникнуть скарги чи проблеми зі здоров'ям, а не як реакція на них.
- Сприяння розвитку культури безпеки та добробуту. Систематична оцінка ризиків демонструє турботу компанії про своїх співробітників, що є основою для формування позитивної корпоративної культури. Коли кожен розробник розуміє, що його безпека та комфорт є пріоритетом, це підвищує його залученість, лояльність та відповідальність.
- Зменшення непрямих витрат. Ефективне управління ризиками, пов'язаними зі здоров'ям працівників, дозволяє зменшити витрати, пов'язані з лікарняними, зниженням продуктивності через втому чи дискомфорт, та плинністю кадрів, спричиненою професійним вигоранням.
- Покращення результатів проєкту. Зрештою, здоровий, сконцентрований та мотивований розробник працює більш ефективно. Управління ризиками дозволяє уникнути просідань у продуктивності, підвищити якість коду та загалом досягти кращих результатів проєкту в установлені терміни.

Приклади використання матриці оцінки ризику для ключових небезпек, пов'язаних із діяльністю розробника програмного забезпечення, наведено у табл. 4.1, 4.2.

Таблиця 4.1 – Матриця оцінювання ризиків

| Психофізіологічне перевантаження (статичне та зорове) | | | | |
|---|---|---|---|---|
| Визначення категорії серйозності небезпеки | | Визначення рівня ймовірності небезпеки | | Індекс ризику небезпеки |
| Вид, категорія | Опис | Вид, рівень | Опис | |
| II – Критична | Розвиток стійких професійних захворювань (остеохондроз, синдром комп'ютерного зору) | A | Вплив є щоденним та гарантованим без профілактичних заходів | II A – неприпустимий (надмірний) рівень ризику. |

Таблиця 4.2 – Матриця оцінювання ризиків

| Недоліки освітлення | | | | |
|---|---|---|---|---|
| Визначення категорії серйозності небезпеки | | Визначення рівня ймовірності небезпеки | | Індекс ризику небезпеки |
| Вид, категорія | Опис | Вид, рівень | Опис | |
| III – гранична | Дискомфорт, головний біль, втома очей, зниження працездатності. | B | Може трапитися декілька разів при зміні робочого місця або погодних умов. | III B – небажаний (гранично допустимий) рівень ризику |

На основі проведеної оцінки, яка виявила один неприпустимий та один небажаний ризик, необхідно розробити комплекс заходів, спрямованих на їх мінімізацію, керуючись ієрархією контролю.

1. Заходи щодо управління ризиком психофізіологічного перевантаження Оскільки цей ризик є неприпустимим, заходи щодо його контролю мають бути впроваджені в першу чергу.

- Інженерні заходи: Ключовим заходом є створення повністю ергономічного робочого місця. Це включає:
    - Використання професійного крісла з можливістю регулювання висоти сидіння, глибини, кута нахилу спинки та висоти підлокітників для забезпечення правильної постави.

- Підбір столу відповідної висоти, що дозволяє тримати лікті під кутом 90 градусів.
- Правильне розташування монітора: на відстані 50-70 см від очей, з верхнім краєм екрана на рівні очей або трохи нижче.

- Адміністративні заходи:
  - Впровадження обов'язкового режиму праці та відпочинку, що включає короткі 5-10 хвилинні перерви кожну годину для виконання профілактичних вправ для очей та опорно-рухового апарату.
  - Проведення для співробітників навчальних семінарів з ергономіки та правильної організації робочого процесу.

2. Заходи щодо управління ризиком недоліків освітлення [13] :

- Інженерні заходи:
  - Забезпечення нормативних рівнів освітленості на робочому місці шляхом використання комбінованого освітлення (загальне + місцеве у вигляді настільної лампи).
  - Встановлення на вікнах сонцезахисних систем (жалюзі, ролети) для уникнення відблисків на екранах моніторів.
  - Використання ламп з нейтральним або теплим спектром світла, що є більш комфортним для очей.
- Адміністративні заходи:
  - Раціональне планування робочих місць в офісі, при якому монітори розташовуються боком до джерел природного світла.

Впровадження запропонованих заходів дозволить значно знизити рівень ідентифікованих ризиків, створивши безпечне, комфортне та продуктивне середовище для роботи над проєктом.

Висновки по розділу

У розділі з охорони праці було приділено увагу питанням законодавчого регулювання безпеки праці для ІТ-спеціаліста, визначено роль роботодавця у створенні належних умов, а також переваги, які отримують як компанія, так і розробник від дотримання вимог охорони праці.

Також у розділі було проведено аналіз потенційних небезпек, характерних для діяльності розробника програмного забезпечення. Для ключових ризиків було проведено оцінювання за допомогою матриці оцінки ризиків, яка базується на двох критеріях – імовірності впливу ідентифікованої небезпеки та тяжкості її наслідків.

# ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання даної кваліфікаційної роботи було спроєктовано та розроблено програмний прототип системи детекції токсичності для коментарів у соціальних мережах. Розроблена система здатна класифікувати текст за шістьма категоріями токсичності, а її ефективність була покращена за допомогою методів роботи з незбалансованими даними. Прототип включає програмний інтерфейс (API) для взаємодії та користувацький інтерфейс для наочної демонстрації.

Також були розглянуті наступні питання:

- Проведено аналіз предметного середовища проблеми токсичності в онлайн-комунікаціях, описано процес модерації контенту та функціональну модель системи за допомогою UML-діаграм.
- Здійснено огляд та аналіз існуючих аналогічних рішень, таких як Perspective API, Azure AI Content Safety та OpenAI Moderation Endpoint, визначено їхні переваги та недоліки.
- Сформульовано постановку задачі, визначено мету та основні завдання кваліфікаційної роботи.
- Проведено аналіз предметної області, визначено структуру вхідних даних та вихідних даних (ймовірності токсичності), а також артефактів моделі.
- Розроблено математичне та алгоритмічне забезпечення, що включає алгоритми передобробки тексту (spaCy, regex) та архітектуру моделі Neural Bag-of-Words (NBoW) на базі PyTorch. Описано методи покращення моделі, зокрема використання зваженої функції втрат та оптимізацію порогів класифікації.
- Визначено стек технологій для розробки (Python, PyTorch, FastAPI, Gradio) та сформульовано вимоги до програмного та технічного забезпечення.

- Описано програмну реалізацію системи за допомогою UML-діаграм компонентів, послідовності та діяльності, що ілюструють взаємодію API та UI.
- Надано детальне керівництво користувача для запуску та використання розробленого програмного прототипу.
- Проаналізовано питання охорони праці для IT-спеціаліста, визначено потенційні небезпеки та проведено їх оцінку за допомогою матричного методу.

Отримані результати та розроблений прототип можуть бути використані як основа для створення повноцінних систем модерації контенту для соціальних мереж, форумів, ігрових чатів та інших онлайн-платформ з метою покращення безпеки та якості онлайн-дискусій.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кемп, С. Глобальний огляд цифрових технологій у 2024 році. Звіт We Are Social та Meltwater. [Електронний ресурс]. – Режим доступу: https://datareportal.com/reports/digital-2024-global-overview-report

2. Statista. Кількість користувачів соціальних мереж у світі з 2017 по 2027 рік. [Електронний ресурс]. – Режим доступу: https://www.statista.com/topics/1164/social-networks

3. Perspective API. [Електронний ресурс]. – Режим доступу: https://perspectiveapi.com

4. Microsoft Azure. Azure AI Content Safety Documentation. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/ai-services/content-safety/

5. OpenAI. OpenAI Moderation. [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/guides/moderation

6. Amazon Web Services. Amazon Comprehend. [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/fr/comprehend/

7. A systematic review of hate speech automatic detection using natural language processing [Електронний ресурс] - Режим доступу: https://www.sciencedirect.com/science/article/pii/S0925231223003557

8. Natural Language Processing with PyTorch Режим доступу: https://www.oreilly.com/library/view/natural-language-processing/9781491978221/

9. Martinelli K. (2019). A Guide to the Most Common Workplace Hazards. – Режим доступу: https://www.highspeedtraining.co.uk/hub/hazards-in-the-workplace/.

10. Закон України «Про охорону праці» [Електронний ресурс] Офіційний сайт Верховної Ради України. Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12.

11. Бедрій Я.І. Основи охорони праці користувачів персональних комп'ютерів: навч. посіб. Тернопіль: Навчальна книга – Богдан, 2014. 144 с.

12. Санітарні норми мікроклімату виробничих приміщень: ДСН 3.3.6.042-99. [Чинний від 1999-12-01]. Київ: МОЗ, 1999. – 21 с.

13. Природне і штучне освітлення: ДБН В.2.5-28:2018. [Чинний від 2019-03-01]. Київ: ДП «НДІ БК», 2018. – 94 с.

14. Kaggle [Електронний ресурс]. Режим доступу: https://www.kaggle.com

# ДОДАТОК А

## Лістинг програми

Фрагменти програмного коду:

### Cell 1: Import Libraries and Load Initial Data

This cell imports necessary libraries and loads your train.csv dataset into a pandas DataFrame.

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, random_split
from collections import Counter
import spacy
import re
import json
import time
from sklearn.metrics import classification_report, roc_auc_score, accuracy_score

train_df = pd.read_csv('../data/train.csv')

print("Initial data loaded.")
print(f"Shape of train_df: {train_df.shape}")
print(train_df.head())
```

```
Initial data loaded.
Shape of train_df: (159571, 8)
                 id                                       comment_text  toxic  \
0  0000997932d777bf  Explanation\nWhy the edits made under my usern...      0
1  000103f0d9cfb60f  D'aww! He matches this background colour I'm s...      0
2  000113f07ec002fd  Hey man, I'm really not trying to edit war. It...      0
3  0001b41b1c6bb37e  "\nMore\nI can't make any real suggestions on ...      0
4  0001d958c54c6e35  You, sir, are my hero. Any chance you remember...      0

   severe_toxic  obscene  threat  insult  identity_hate
0             0        0       0       0              0
1             0        0       0       0              0
2             0        0       0       0              0
3             0        0       0       0              0
4             0        0       0       0              0
```

### Cell 2: Define Text Preprocessing Function

This cell defines your custom text preprocessing function using spaCy and regular expressions. It also loads the spaCy model.

```
nlp_preprocessor = spacy.load("en_core_web_sm", disable=["parser", "ner"])
print("spaCy 'en_core_web_sm' model loaded for preprocessing.")

def preprocess_text_custom_spacy(text_to_process, nlp_instance):
    if not nlp_instance:
        pass
    if not isinstance(text_to_process, str): text_to_process = str(text_to_process)

    text_to_process = text_to_process.lower()
    text_to_process = re.sub(r'https?://\S+|www\.\S+', '', text_to_process)
    text_to_process = re.sub(r'@\w+', '', text_to_process)
    text_to_process = re.sub(r'<.*?>', '', text_to_process)
    text_to_process = re.sub(r'[^a-z\s]', '', text_to_process)
    text_to_process = re.sub(r'\s+', ' ', text_to_process).strip()

    doc = nlp_instance(text_to_process)
    processed_tokens = [
        token.lemma_ for token in doc if token.is_alpha and not token.is_stop
    ]
    return ' '.join(processed_tokens)

print("Custom text preprocessing function 'preprocess_text_custom_spacy' defined.")
```

```
spaCy 'en_core_web_sm' model loaded for preprocessing.
Custom text preprocessing function 'preprocess_text_custom_spacy' defined.
```

### Cell 3: Apply Text Preprocessing

This cell applies the defined preprocessing function to your comment text, handles NaNs, and removes any rows that become empty after processing.

```
train_df_processed = train_df.copy()
train_df_processed['comment_text_processed'] = train_df_processed['comment_text'].fillna('').apply(
    lambda x: preprocess_text_custom_spacy(x, nlp_preprocessor)
)

initial_rows = train_df_processed.shape[0]
train_df_processed = train_df_processed[train_df_processed['comment_text_processed'] != ""]
rows_removed = initial_rows - train_df_processed.shape[0]

print(f"Text preprocessing applied. Rows after processing and removing empty: {train_df_processed.shape[0]} (removed {rows_removed})")
print(train_df_processed[['comment_text', 'comment_text_processed']].head())
```

```
Text preprocessing applied. Rows after processing and removing empty: 159434 (removed 137)
                                        comment_text  \
0  Explanation\nWhy the edits made under my usern...
1  D'aww! He matches this background colour I'm s...
2  Hey man, I'm really not trying to edit war. It...
3  "\nMore\nI can't make any real suggestions on ...
4  You, sir, are my hero. Any chance you remember...

                            comment_text_processed
0  explanation edit username hardcore metallica f...
1  daww match background colour m seemingly stick...
2  hey man m try edit war guy constantly remove r...
3  not real suggestion improvement wonder section...
4                    sir hero chance remember page s
```

### Cell 4: Build Vocabulary, Numericalize, and Pad Sequences

This cell creates the vocabulary from the processed text, converts text to numerical sequences, and pads/truncates them to a fixed length.

```
PROCESSED_TEXT_COLUMN_NAME = 'comment_text_processed'

all_processed_words = [word for comment in train_df_processed[PROCESSED_TEXT_COLUMN_NAME].astype(str) for word in comment.split()]
word_counts = Counter(all_processed_words)
sorted_words = sorted(word_counts, key=word_counts.get, reverse=True)
vocab_to_int = {word: i+2 for i, word in enumerate(sorted_words)}
vocab_to_int['<pad>'] = 0
vocab_to_int['<unk>'] = 1
SEQ_LENGTH = 200

train_df_processed['padded_features'] = train_df_processed[PROCESSED_TEXT_COLUMN_NAME].apply(
    lambda text: (
        lambda seq: seq[:SEQ_LENGTH] if len(seq) > SEQ_LENGTH else seq + [vocab_to_int['<pad>']] * (SEQ_LENGTH - len(seq))
    )([vocab_to_int.get(word, vocab_to_int['<unk>']) for word in str(text).split() if word])
)

print(f"Vocabulary created (size: {len(vocab_to_int)}), SEQ_LENGTH set to {SEQ_LENGTH}.")
print("Column 'padded_features' created.")
```

```
Vocabulary created (size: 197521), SEQ_LENGTH set to 200.
Column 'padded_features' created.
```

### Cell 5: Prepare PyTorch Tensors, Dataset, and DataLoaders

This cell extracts the padded features and labels, converts them to PyTorch tensors, defines a custom Dataset class, splits data into training and validation sets, and creates DataLoader instances.

```
X_list = train_df_processed['padded_features'].tolist()
X_np = np.array(X_list, dtype=np.int64)
features_tensor = torch.from_numpy(X_np)

label_columns = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
y_np = train_df_processed[label_columns].values.astype(np.float32)
labels_tensor = torch.from_numpy(y_np)

print(f"Shape of features_tensor: {features_tensor.shape}")
print(f"Shape of labels_tensor: {labels_tensor.shape}")

class ToxicityDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

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

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]

full_dataset = ToxicityDataset(features_tensor, labels_tensor)

validation_split = 0.2
dataset_size = len(full_dataset)
val_size = int(validation_split * dataset_size)
train_size = dataset_size - val_size
train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

print(f"Training samples: {len(train_dataset)}, Validation samples: {len(val_dataset)}")

BATCH_SIZE = 32
NUM_WORKERS = 0
```

### Cell 6: Define NBoW Model, Loss, and Optimizer

This cell defines the SimplerNBoWClassifier model class, sets up the device, instantiates the model, defines the loss function (BCEWithLogitsLoss without weights for this baseline), and the Adam optimizer.

```
VOCAB_SIZE_NBOW = len(vocab_to_int)
EMBEDDING_DIM_NBOW = 100
OUTPUT_DIM_NBOW = 6
padding_idx_nbow = vocab_to_int.get('<pad>', 0)

class SimplerNBoWClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, output_dim, padding_idx_val):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=padding_idx_val)
        self.fc = nn.Linear(embedding_dim, output_dim)

    def forward(self, text_batch):
        embedded = self.embedding(text_batch)
        averaged_embeddings = torch.mean(embedded, dim=1)
        logits = self.fc(averaged_embeddings)
        return logits

model_nbow = SimplerNBoWClassifier(VOCAB_SIZE_NBOW, EMBEDDING_DIM_NBOW, OUTPUT_DIM_NBOW, padding_idx_nbow)
model_nbow.to(device)

criterion = nn.BCEWithLogitsLoss()
LEARNING_RATE_NBOW = 0.001
optimizer_nbow = optim.Adam(model_nbow.parameters(), lr=LEARNING_RATE_NBOW)

print("NBoW Model, Loss function, and Optimizer defined.")
print(f"Model moved to device: {device}")
```

```
NBoW Model, Loss function, and Optimizer defined.
Model moved to device: mps
```

### Cell 7: Helper Functions for Training and Evaluation Epochs

Defines two helper functions: `train_epoch_func` for handling the logic of a single training epoch (forward pass, loss calculation, backpropagation, optimizer step) and `evaluate_epoch_func` for a single validation epoch.

```
def train_epoch_func(model, dataloader, criterion_fn, optimizer_fn, current_device):
    model.train()
    total_loss = 0
    for features, labels in dataloader:
        features = features.to(current_device)
        labels = labels.to(current_device)
        optimizer_fn.zero_grad()
        predictions = model(features)
        loss = criterion_fn(predictions, labels)
        loss.backward()
        optimizer_fn.step()
        total_loss += loss.item()
    return total_loss / len(dataloader)

def evaluate_epoch_func(model, dataloader, criterion_fn, current_device):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for features, labels in dataloader:
            features = features.to(current_device)
            labels = labels.to(current_device)
            predictions = model(features)
            loss = criterion_fn(predictions, labels)
            total_loss += loss.item()
    return total_loss / len(dataloader)

print("Helper functions 'train_epoch_func' and 'evaluate_epoch_func' defined.")
```

```
Helper functions 'train_epoch_func' and 'evaluate_epoch_func' defined.
```

## Cell 8: Calculate Class Weights and Define Weighted Loss Function

Calculates positive class weights ( `pos_weight` ) for each toxicity category based on the training set distribution to address class imbalance. Defines a new `BCEWithLogitsLoss` criterion ( `criterion_nbow_weighted` ) using these calculated weights.

```
import torch

train_labels_np = labels_tensor[train_dataset.indices].cpu().numpy()

pos_weights_list = []
print("Calculating positive weights for BCEWithLogitsLoss:")
for i, col_name in enumerate(label_columns):
    num_total_train_samples = train_labels_np.shape[0]
    num_pos = train_labels_np[:, i].sum()
    num_neg = num_total_train_samples - num_pos

    if num_pos > 0 and num_neg > 0:
        weight = num_neg / num_pos
    elif num_pos == 0 and num_neg > 0:
        weight = num_total_train_samples
        print(f"Warning: No positive samples found for class '{col_name}' in training data. Using large weight: {weight:.2f}")
    else:
        weight = 1.0
        print(f"Warning: No negative samples or no samples for class '{col_name}'? Using weight: {weight:.2f}")

    pos_weights_list.append(weight)
    print(f"  Class '{col_name}': num_pos={num_pos}, num_neg={num_neg}, calculated_pos_weight={weight:.2f}")

pos_weights_tensor = torch.tensor(pos_weights_list, dtype=torch.float32).to(device)
print("\npos_weights_tensor created and moved to device.")
print(pos_weights_tensor)

criterion_nbow_weighted = nn.BCEWithLogitsLoss(pos_weight=pos_weights_tensor)
print("\nWeighted BCEWithLogitsLoss ('criterion_nbow_weighted') defined.")
```

```
Calculating positive weights for BCEWithLogitsLoss:
  Class 'toxic': num_pos=12280.0, num_neg=115268.0, calculated_pos_weight=9.39
  Class 'severe_toxic': num_pos=1282.0, num_neg=126266.0, calculated_pos_weight=98.49
  Class 'obscene': num_pos=6823.0, num_neg=120725.0, calculated_pos_weight=17.69
  Class 'threat': num_pos=379.0, num_neg=127169.0, calculated_pos_weight=335.54
  Class 'insult': num_pos=6357.0, num_neg=121191.0, calculated_pos_weight=19.06
```

## Cell 9: NBoW Model Training Loop (with Weighted Loss)

Executes the main training loop for the NBoW model ( `model_nbow` ) for a specified number of epochs. This loop uses the `train_epoch_func` , `evaluate_epoch_func` , the NBoW optimizer, and the `criterion_nbow_weighted` (weighted loss function).

```
NUM_EPOCHS_NBOW = 10

print(f"Starting NBoW model training for {NUM_EPOCHS_NBOW} epochs...")
for epoch in range(NUM_EPOCHS_NBOW):
    epoch_start_time = time.time()
    avg_train_loss = train_epoch_func(model_nbow, train_loader, criterion_nbow_weighted, optimizer_nbow, device)
    avg_val_loss = evaluate_epoch_func(model_nbow, val_loader, criterion_nbow_weighted, device)
    epoch_end_time = time.time()
    epoch_duration = epoch_end_time - epoch_start_time
    print(f"Epoch [{epoch+1}/{NUM_EPOCHS_NBOW}] (NBoW) - Train Loss: {avg_train_loss:.4f}, Val Loss: {avg_val_loss:.4f}, Duration: {epoch_dura

print("\nNBoW Training finished.")
```

```
Starting NBoW model training for 10 epochs...
Epoch [1/10] (NBoW) - Train Loss: 1.0911, Val Loss: 0.8907, Duration: 172.11s
Epoch [2/10] (NBoW) - Train Loss: 0.7660, Val Loss: 0.7242, Duration: 161.23s
Epoch [3/10] (NBoW) - Train Loss: 0.6169, Val Loss: 0.6802, Duration: 161.79s
Epoch [4/10] (NBoW) - Train Loss: 0.5379, Val Loss: 0.6630, Duration: 155.48s
Epoch [5/10] (NBoW) - Train Loss: 0.4857, Val Loss: 0.6536, Duration: 158.95s
Epoch [6/10] (NBoW) - Train Loss: 0.4433, Val Loss: 0.6495, Duration: 161.01s
Epoch [7/10] (NBoW) - Train Loss: 0.4080, Val Loss: 0.6775, Duration: 160.94s
Epoch [8/10] (NBoW) - Train Loss: 0.3768, Val Loss: 0.7067, Duration: 154.91s
Epoch [9/10] (NBoW) - Train Loss: 0.3485, Val Loss: 0.7069, Duration: 153.63s
Epoch [10/10] (NBoW) - Train Loss: 0.3238, Val Loss: 0.7275, Duration: 151.61s

NBoW Training finished.
```

### Cell 10: Initial NBoW Model Evaluation (Weighted Loss, Default 0.5 Threshold)

Performs an initial evaluation of the NBoW model (trained with weighted loss) on the validation set. It calculates and prints a classification report, ROC AUC scores, and Exact Match Ratio using the default 0.5 threshold for binary predictions.

```
print("Starting NBoW model evaluation on the validation set...")
model_nbow.eval()
all_true_labels_nbow = []
all_predicted_probs_nbow = []
all_predicted_labels_nbow = []

with torch.no_grad():
    for features, labels in val_loader:
        features = features.to(device)
        logits = model_nbow(features)
        probabilities = torch.sigmoid(logits).cpu().numpy()
        binary_predictions = (probabilities >= 0.5).astype(int)
        all_true_labels_nbow.extend(labels.numpy().astype(int))
        all_predicted_probs_nbow.extend(probabilities)
        all_predicted_labels_nbow.extend(binary_predictions)

all_true_labels_nbow_np = np.array(all_true_labels_nbow)
all_predicted_labels_nbow_np = np.array(all_predicted_labels_nbow)
all_predicted_probs_nbow_np = np.array(all_predicted_probs_nbow)

print("\n--- NBoW Model: Classification Report ---")
report_nbow = classification_report(all_true_labels_nbow_np, all_predicted_labels_nbow_np, target_names=label_columns, zero_division=0)
print(report_nbow)

print("\n--- NBoW Model: ROC AUC Score (per class and average) ---")
roc_auc_per_class_nbow = []
for i in range(all_true_labels_nbow_np.shape[1]):
    try:
        score = roc_auc_score(all_true_labels_nbow_np[:, i], all_predicted_probs_nbow_np[:, i])
        print(f"ROC AUC for class '{label_columns[i]}': {score:.4f}")
        roc_auc_per_class_nbow.append(score)
    except ValueError:
        print(f"ROC AUC for class '{label_columns[i]}': Not computable (likely only one class present in y_true).")
        roc_auc_per_class_nbow.append(float('nan'))
```

### Cell 11: Optimal Threshold Tuning

Using the probabilities predicted by the NBoW model (from Cell 10), this cell finds the optimal classification threshold for each toxicity class that maximizes its F1-score on the validation set. It then prints these optimal thresholds and a new classification report and Exact Match Ratio based on these tuned thresholds.

```
import numpy as np
from sklearn.metrics import f1_score, classification_report, accuracy_score

print("Starting threshold tuning for NBoW model (trained with weighted loss)...")

optimal_thresholds = {}
best_f1_scores = {}

threshold_candidates = np.arange(0.01, 1.00, 0.01)

for i, class_name in enumerate(label_columns):
    best_threshold_for_class = 0.5
    best_f1_for_class = 0.0

    true_labels_for_class = all_true_labels_nbow_np[:, i]
    pred_probs_for_class = all_predicted_probs_nbow_np[:, i]

    if np.sum(true_labels_for_class) == 0:
        print(f"Class '{class_name}': No positive samples in validation set. Skipping threshold tuning, using default 0.5.")
        optimal_thresholds[class_name] = 0.5
        temp_preds = (pred_probs_for_class >= 0.5).astype(int)
        best_f1_for_class = f1_score(true_labels_for_class, temp_preds, zero_division=0)
        best_f1_scores[class_name] = best_f1_for_class
        continue

    for threshold in threshold_candidates:
        binary_predictions_for_class = (pred_probs_for_class >= threshold).astype(int)
        current_f1 = f1_score(true_labels_for_class, binary_predictions_for_class, average='binary', zero_division=0)

        if current_f1 > best_f1_for_class:
            best_f1_for_class = current_f1
```

Фрагмент коду блока для API :

```
import torch
import torch.nn as nn
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import json
ick to add a breakpoint
import spacy
import time
from contextlib import asynccontextmanager

loaded_model = None
vocab_to_int_map = None
SEQ_LENGTH_API = None
nlp_spacy = None
device_api = None
LABEL_COLUMNS_API = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
EMBEDDING_DIM_API = 100
OUTPUT_DIM_API = 6

class SimplerNBoWClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, output_dim, padding_idx_val):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=padding_idx_val)
        self.fc = nn.Linear(embedding_dim, output_dim)

    def forward(self, text_batch):
        embedded = self.embedding(text_batch)
        averaged_embeddings = torch.mean(embedded, dim=1)
        logits = self.fc(averaged_embeddings)
        return logits

def preprocess_text_spacy_api(text, nlp_processor):
    if not nlp_processor:
        raise RuntimeError("spaCy model not loaded for API preprocessing.")
    if not isinstance(text, str): text = str(text)

    text = text.lower()
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'[^a-z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()

    doc = nlp_processor(text)
    processed_tokens = [
        token.lemma_ for token in doc if token.is_alpha and not token.is_stop
    ]
    return ' '.join(processed_tokens)

def numericalize_text_api(text, vocab_map):
    return [vocab_map.get(word, vocab_map.get('<unk>', 1)) for word in str(text).split() if word]

def pad_sequence_api(numerical_sequence, seq_len, pad_idx):
    if len(numerical_sequence) < seq_len:
        return numerical_sequence + [pad_idx] * (seq_len - len(numerical_sequence))
    elif len(numerical_sequence) > seq_len:
        return numerical_sequence[:seq_len]
    else:
        return numerical_sequence

@asynccontextmanager
async def lifespan(app: FastAPI):
    global loaded_model, vocab_to_int_map, SEQ_LENGTH_API, nlp_spacy, device_api, EMBEDDING_DIM_API, OUTPUT_DIM_API
    print("Loading API resources...")
    start_load_time = time.time()

    device_api = torch.device("cpu")
    print(f"API will use device: {device_api}")
```

```
    try:
        nlp_spacy = spacy.load("en_core_web_sm", disable=["parser", "ner"])
        print("spaCy 'en_core_web_sm' model loaded successfully for API.")
    except OSError:
        print("Critical Error: spaCy model 'en_core_web_sm' not found. API cannot start.")
        raise RuntimeError("spaCy model not found. Please run: python -m spacy download en_core_web_sm")

    vocab_config_path = '../model_artifacts/nbow_vocab_config.json'
    try:
        with open(vocab_config_path, 'r') as f:
            vocab_config = json.load(f)
        vocab_to_int_map = vocab_config['vocab_to_int']
        SEQ_LENGTH_API = vocab_config['SEQ_LENGTH']
        print(f"Vocabulary (size: {len(vocab_to_int_map)}) and SEQ_LENGTH ({SEQ_LENGTH_API}) loaded from {vocab_config_path}.")
    except FileNotFoundError:
        print(f"Critical Error: Vocabulary config file '{vocab_config_path}' not found. API cannot start.")
        raise RuntimeError(f"File not found: {vocab_config_path}")
    except KeyError:
        print(f"Critical Error: 'vocab_to_int' or 'SEQ_LENGTH' not found in '{vocab_config_path}'. API cannot start.")
        raise RuntimeError(f"Invalid format in: {vocab_config_path}")

    padding_idx_val_api = vocab_to_int_map.get('<pad>', 0)
    loaded_model = SimplerNBoWClassifier(len(vocab_to_int_map), EMBEDDING_DIM_API, OUTPUT_DIM_API, padding_idx_val_api)
    model_state_path = '../model_artifacts/nbow_model_state.pth'
    try:
        loaded_model.load_state_dict(torch.load(model_state_path, map_location=torch.device('cpu')))
        loaded_model.to(device_api)
        loaded_model.eval()
        print(f"Model state loaded from '{model_state_path}' and moved to {device_api}.")
    except FileNotFoundError:
        print(f"Critical Error: Model state file '{model_state_path}' not found. API cannot start.")
        raise RuntimeError(f"File not found: {model_state_path}")
    except Exception as e:
        print(f"Critical Error loading model state: {e}. API cannot start.")
        raise RuntimeError(f"Error loading model state: {e}")

    end_load_time = time.time()
    print(f"API resources loaded in {end_load_time - start_load_time:.2f} seconds.")
    yield
    print("API shutting down...")

app = FastAPI(lifespan=lifespan)

class CommentInput(BaseModel):
    text: str

class PredictionOutput(BaseModel):
    label_probabilities: dict[str, float]

@app.post("/predict", response_model=PredictionOutput)
async def predict_toxicity(comment: CommentInput):
    if not all([loaded_model, vocab_to_int_map, SEQ_LENGTH_API, nlp_spacy]):
        raise HTTPException(status_code=503, detail="Model or resources not loaded. API not ready.")

    try:
        processed_text = preprocess_text_spacy_api(comment.text, nlp_spacy)
        numerical_sequence = numericalize_text_api(processed_text, vocab_to_int_map)
        padded_sequence = pad_sequence_api(numerical_sequence, SEQ_LENGTH_API, vocab_to_int_map.get('<pad>', 0))
        input_tensor = torch.LongTensor([padded_sequence]).to(device_api)

        with torch.no_grad():
            logits = loaded_model(input_tensor)

        probabilities = torch.sigmoid(logits).squeeze().cpu().numpy()
        label_probs_dict = {LABEL_COLUMNS_API[i]: float(probabilities[i]) for i in range(len(LABEL_COLUMNS_API))}

        return PredictionOutput(label_probabilities=label_probs_dict)

    except Exception as e:
        print(f"Error during prediction: {e}")
        raise HTTPException(status_code=500, detail=f"Error processing request: {str(e)}")

@app.get("/")
async def read_root():
    return {"message": "Toxicity Detection API is running!"}
```