МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота
на тему

Розробка програмного забезпечення для технологій обробки тексту

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконав: студент групи МгЗІТ-23
Седляр А. О.
Науковий керівник:
к.т.н., доц. Астістова Т.І.
Рецензент:
к.т.н., доц. Мельник Г.В.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Наталія ЧУПРИНКА
«_____» _____ 2024 року

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

Седляру Андрію Олександровичу

1. Тема кваліфікаційної роботи: Розробка програмного забезпечення для технологій обробки тексту, науковий керівник роботи к.т.н., доц. Астістова Тетяна Іванівна, затверджені наказом КНУТД від "3" вересня 2024 року № 118-уч.
2. Строк подання студентом роботи 22.11.2024р.
3. Вихідні дані до кваліфікаційної роботи . Розробки кафедри комп'ютерних наук
4. Зміст кваліфікаційної роботи: Розділ 1. Аналіз предметної області, Розділ 2. Проектування програмної реалізації, Розділ 3. Розробка системи
   Додатки - програмні коди системи, презентація.

5. Дата видачі завдання 03. 09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів	Терміни виконання етапів	Примітка про виконання
1	Вступ	03.10.2024	
2	Розділ 1 Аналіз предметної області	08.10.2024	
3	Розділ 2 Проектування програмної реалізації	15.10.2024	
4	Розділ 3 Розробка системи	28.10.2024	
5	Висновки	30.10.2024	
6	Оформлення кваліфікаційної роботи	01.11.2024	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку	12.11.2024	
8	Подача кваліфікаційної роботи для рецензування	14.11.2024	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату	17.11.2024	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	20.11.2024	

Студент _____ Андрій СЕДЛЯР

Науковий керівник _____ Тетяна АСТІСТОВА

АНОТАЦІЯ

СедлярА.О. Розробка програмного забезпечення для технологій обробки тексту.

Кваліфікаційна робота за спеціальністю 122 – «Комп'ютерні науки» , Київський національний університет технологій та дизайну, Київ, 2024 рік.

Кваліфікаційна робота присвячена дослідженню використання технологій штучного інтелекту для створення інструменту оцінки оригінальності текстів в контексті академічної доброчесності. Метою роботи є дослідження технологій штучного інтелекту та розробка застосунку що буде поєднувати звичайні методи пошуку плагіату з нейронною мережею навченою аналізувати текст для виявлення фрагментів написаних штучним інтелектом.

У роботі використовувалася нейронна мережа, яка була налаштована на класифікацію текстів за шаблонами, характерними для текстів, згенерованих людиною або штучним інтелектом. Використання цієї моделі дозволяє швидко і правильно розрізняти тексти неавторського походження, що сприяє підвищенню точності сучасних систем перевірки на плагіат.

Практичне значення роботи полягає у розробці інструмента, який може бути легко інтегрований у програми для оцінки академічної доброчесності, що стане важливим кроком у протидії несанкціонованому використанню ШІ у навчальному процесі. Результати можуть стати основою для подальших досліджень у сфері автоматичного аналізу текстів та підвищення ефективності перевірки на оригінальність.

Для створення програмного продукту було обрано: середовище програмування Visual Studio; мову програмування C#.

Ключові слова: академічна доброчесність, штучний інтелект, нейронна мережа, алгоритми, Visual Studio, C#, DB Browser (SQLite), JSON, ReadDocFile, API, сервіс Grammarly.

,

ANNOTATION

Andrii Sedliar. AI technology in the creation of a tool for assessing the originality of texts.

Master's degree project in specialty 122 - “Computer Science” - Kyiv National University of Technology and Design, Kyiv, 2024.

The thesis is devoted to the study of the use of artificial intelligence technologies to create a tool for assessing the originality of texts in the context of academic integrity. The purpose of the work is to study artificial intelligence technologies and develop an application that will combine conventional plagiarism detection methods with a neural network trained to analyze text to identify fragments written by artificial intelligence.

The work used a neural network that was configured to classify texts according to patterns typical of texts generated by humans or artificial intelligence. The use of this model allows to quickly and correctly distinguish between texts of non-authoritative origin, which helps to improve the accuracy of modern plagiarism checking systems.

The practical significance of the work is to develop a tool that can be easily integrated into programs for assessing academic integrity, which will be an important step in countering the unauthorized use of AI in the educational process. The results can serve as a basis for further research in the field of automatic text analysis and improving the effectiveness of originality checks.

The following programming environment was chosen to create the software product: Visual Studio programming environment; C# programming language.

Keywords: academic integrity, artificial intelligence, neural network, originality of texts, Visual Studio, C#, DB Browser (SQLite), JSON, ReadDocFile, API Grammarly.

ЗМІСТ

ВСТУП

Актуальність проблеми. Питання штучного інтелекту в роботах стає дедалі складнішим і тривожнішим. По суті, це питання про те, чого повинна досягати освіта. Коли студенти дозволяють штучному інтелекту писати їхні есе, розв'язувати за них задачі або навіть давати відповіді на тести, ми втрачаємо зв'язок з автентичністю навчання. Це зводить нанівець сенс того, чи дійсно учень займався вивченням матеріалу, чи дозволив машині думати за нього. Вся передумова освіти зміщується з подорожі, в якій студенти поступово набувають знань і навичок, на транзакцію, в якій результат важливіший за розуміння.

Особливо іронічно, однак, те, що надмірне покладання на ШІ ставить під загрозу набуття тих самих навичок, в яких він найслабший і які, отже, не можуть бути передані машинам: критичне мислення, розв'язання проблем, творчість. Адже коли студенти відчувають, що будь-яку відповідь можна знайти за кілька кліків, то зникає будь-який стимул досліджувати, ставити запитання і помилятися. Без такого практичного досвіду вони позбавлені автентичного навчання, яке зростає зі спроб і помилок, що призводить до розчарування в осмисленні нових ідей.

Мета роботи. Метою даного дипломного проекту є дослідження проблеми порушення академічної доброчесності методом зловживання сучасними розвиненими штучним інтелектом (ШІ) помічниками та впровадження в існуючі програми пошуку плагіату методи розпізнавання тексту, написаним штучним інтелектом. Вирішення проблеми може критися в використанні нейронної мережі навченої пошуку відповідних закономірностей та патернів написання тексту людиною та штучним інтелектом.

Завдання роботи. Задля виконання поставленої мети було сформовано низку задач, які повинна бути виконані:

- Провести огляд наукової літератури щодо проблем академічної доброчесності та способів її порушення із використанням ШІ.

- Дослідити сучасні методи розпізнавання текстів, згенерованих ШІ, і можливості їхнього застосування для виявлення академічного шахрайства.
- Дослідити, які характеристики можуть слугувати індикаторами штучного написання
- Обрати або створити модель нейронної мережі, яка здатна аналізувати текстові дані на наявність шаблонів, притаманних ШІ-текстам.
- Впровадити модель у систему або програму для перевірки на плагіат, щоб додати функцію розпізнавання ШІ-згенерованих текстів.
- Провести тестування точності роботи моделі, оцінюючи її ефективність і точність у реальних випадках.

Методика досліджень. Методика дослідження базується на аналізі наукової літератури щодо академічної доброчесності та методів розпізнавання текстів, згенерованих штучним інтелектом, виборі та навчанні нейронної мережі для виявлення таких текстів, а також інтеграції цієї моделі в програмне забезпечення для перевірки плагіату. Етапи включають теоретичне дослідження, розробку інструменту з використанням API, реалізація технологій аналізу тексту, та створенні інтерфейсу користувача..

Предмет дослідження. У дипломній роботі досліджуються технології та методи, які можуть бути використані для ідентифікації текстів, згенерованих штучним інтелектом, з метою забезпечення академічної доброчесності. Зокрема, вона зосереджується на характеристиках текстів, що вказують на їхнє штучне походження, а також на застосуванні нейронних мереж для аналізу та класифікації текстів. Тема тісно пов'язана з дослідженням алгоритмів, моделей та інструментів, які дозволяють інтегрувати механізми виявлення текстів, згенерованих штучним інтелектом, у програми виявлення плагіату.

Наукова новизна. Наукова новизна дипломної роботи полягає у створені інтегрованої системи яка має поєднувати класичні методи перевірки на плагіат з розпізнаванням тексів створених штучним інтелектом, що є важливим кроком

на шляху до доказу академічної доброчесності в контексті, коли штучний інтелект, ймовірно, буде використовуватися все ширше.

Практичне значення отриманих результатів. Практичне значення роботи полягає в тому, що буде розроблено інструмент для виявлення текстів, які генерує штучний інтелект, який можна буде впровадити у вже існуючі системи перевірки на плагіат. Функціонал таких систем буде розширено: замість того, щоб просто виявляти випадки звичайного плагіату, вони зможуть виявляти випадки зловживання технологіями штучного інтелекту, які студенти все частіше використовують для створення текстів, що здаються оригінальними на перший погляд.

Апробація результатів роботи в наступних виступах на конференціях та статті

1. Astistova T.I., Sedlyar A.O.AItechnology in the creation of a tool for assessing the originality of texts / Астістова Т. І. Седляр А. О. // мехатронні системи:інновації та інжиніринг тези доповідей VIII міжнародної науково-практичної конференції / Науковий редактор М.М. Рубанка Відповідальний за поліграфічне виконання Л.Л. Овечкіна – КНУТД, 2024 – С 280.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття штучний інтелект

Генеративний штучний інтелект (ШІ) став революційною парадигмою в галузі машинного навчання, що дозволяє синтезувати складні та реальні дані за допомогою обчислювальних засобів. Його вплив знаходить відгук у різних сферах, де генерація синтетичних даних сприяла прогресу. Здатність генеративного ШІ створювати дані, які відображають характеристики реальної інформації, знайшла застосування в таких різноманітних сферах, як комп'ютерний зір, обробка природної мови, написання музики та пошук нових ліків [1].

У комп'ютерному зорі генеративний ШІ проклав шлях до реалістичного синтезу зображень і маніпулювання ними, покращуючи такі завдання, як доповнення даних і передача художнього стилю. В обробці природної мови моделі генерації тексту виявилися корисними в різних додатках, від діалогових систем до створення контенту. Крім того, музична індустрія стала свідком появи генеративного ШІ для створення нових музичних композицій. Навіть фармацевтичний сектор використовує генеративний ШІ для прискорення відкриття ліків шляхом створення молекулярних структур з потрібними властивостями. Крім того, генеративний ШІ демонструє багатообіцяючий потенціал у сфері охорони здоров'я, допомагаючи у створенні медичних зображень та діагностиці [2].

Універсальність генеративного ШІ поширюється на такі сфери, як дизайн одягу, дизайн відеоігор та архітектурна візуалізація. Ці застосування підкреслюють глибокий вплив і широке впровадження генеративного ШІ в різних галузях. Далі ми детально розглянемо архітектурні компоненти, цілі та робочі механізми кожної моделі.

Як показано на рис. 1, генеративні моделі та їхні архітектури є наріжним каменем сучасного штучного інтелекту, революціонізуючи те, як ми розуміємо і використовуємо дані. Ці моделі, які охоплюють різноманітні методи, такі як генеративні змагальні мережі (GAN), варіаційні автокодери (VAE) та потокові

моделі, призначені для вивчення основних закономірностей і структур складних наборів даних [3].

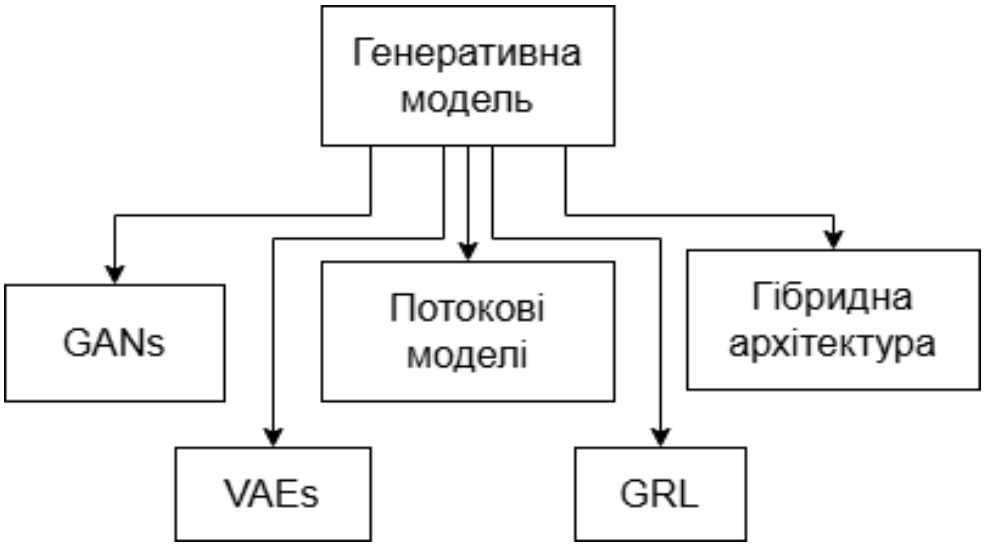


Рисунок 1.1 – Архітектури генеративних моделей

Наприклад, GAN-мережі протиставляють генератор дискримінатору в стратегічному змаганні, що призводить до створення надзвичайно реалістичних екземплярів даних. З іншого боку, VAE вводять імовірнісні кодування, які дозволяють генерувати нові точки даних шляхом вибірки з вивчених латентних просторів. Моделі на основі потоків, навпаки, зосереджуються на обернених перетвореннях для відображення простих розподілів на складний простір даних, що полегшує ефективну вибірку та оцінку ймовірності.

Генеративні змагальні мережі (GAN) - це революційна структура, запропонована Гудфеллоу, яка складається з двох нейронних мереж, генератора (G) і дискримінатора (D), що беруть участь у змагальній грі для двох гравців. Функція (1.1) описує гру між генератором і дискримінатором. Генератор створює синтетичні екземпляри даних для імітації реальних розподілів даних, тоді як дискримінатор намагається розрізнити реальні та згенеровані дані. Процес навчання передбачає одночасну оптимізацію обох мереж у мінімаксній грі, при цьому генератор покращує свою здатність створювати реалістичні дані,

які можуть обдурити дискримінатор. У кінцевій рівновазі генератор виробляє дані, які неможливо відрізнити від реальних [4].

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}\left[\log\left(D(x)\right)\right] + E_{z-p_z(z)}\left[\log(1-d\left(G(z)\right))\right] \quad (1.1)$$

Варіаційні автокодери (VAE): Варіаційні автокодери, запропоновані Кінгмою та Веллінгом, поєднують концепції автокодерів та варіаційного виведення для вивчення ймовірнісного представлення даних, функція втрат для варіаційного автокодувальника (VAE). VAE складаються з кодера, який відображає вхідні дані в імовірнісний латентний простір, і декодера, який генерує дані з вибіркових латентних змінних. Модель навчається для мінімізації втрат при реконструкції, одночасно регулюючи латентний простір за допомогою розбіжності Кульбака-Лейблера між вивченим латентним розподілом і попереднім розподілом формула [5]. Це дозволяє VAE генерувати нові дані шляхом вибірки з вивченого латентного простору. Функція втрат для варіаційного автокодера показана формулою (1.2).

$$\mathcal{L}_{VAE}(\theta, \phi; x) = -E_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right] + KL(q_\phi(z|x)||p(z) \quad (1.2)$$

Потокові моделі: моделі на основі потоків, введені Діном, наближаються до генеративного моделювання за допомогою інверсійних перетворень даних. Ці моделі визначають серію бієктивних перетворень, які відображають дані від простого розподілу (наприклад, гауссового) до бажаного складного розподілу даних. Функція залежності розподілу показана формулою (1.3)

$$z=f(x);$$

$$\text{де: } x\sim p(x),\ z\sim p(z) \quad (1.3)$$

Оскільки як прямі, так і зворотні перетворення можна використовувати, потокові моделі дозволяють ефективно проводити вибірку та обчислення

ймовірностей. Ця архітектура особливо підходить для даних високої розмірності і використовується для генерації вибірок шляхом перетворення шуму за допомогою вивчених перетворень [6].

Генеративне навчання з підкріпленням (GRL): Генеративне змагальне імітаційне навчання (GAIL), запропоноване Хо і Ермоном, поєднує в собі генеративні змагальні мережі (GAN) з навчанням з підкріпленням (RL) для імітації політики. У GAIL генератор створює траєкторії дій у середовищі, прагнучи імітувати поведінку експерта, формула функції втрат для алгоритму GAIL. Функція втрат для алгоритму GAIL показана формулою (1.4). Дискримінатор, що представляє політику експерта, розрізняє траєкторії експерта та згенеровані траєкторії. Генератор навчений генерувати траєкторії, які неможливо відрізнити від експертних, що покращує продуктивність політики. GAIL долає розрив між RL і GAN, дозволяючи навчати політику через імітацію [7].

$$\mathcal{L}_{GAIL} = E_{\pi E}[logD(s,a)] + E_{\pi}[\log(1 - D(s,a))] \qquad (1.4)$$

Гібридні та вдосконалені архітектури: гібридні архітектури поєднують різні моделі генерації, щоб використовувати їхні сильні сторони. Одним з таких прикладів є поєднання GAN і VAE, де поєднується здатність VAE кодувати дані і здатність GAN генерувати високоякісні зразки.[8] Функція втрат - це зважена комбінація втрат при реконструкції VAE і втрат при боротьбі з противником GAN, функція втрат, яка комбінує втрати варіаційного автоенкодера має вигляд (1.5) . Ця гібридна архітектура генерує дані, які є одночасно когерентними і точними до вхідних даних, таким чином долаючи обмеження окремих моделей.

$$GAN = VAE\ Loss = \mathcal{L}_{VAE} + \lambda * \mathcal{L}_{GAN} \qquad (1.5)$$

1.2 Нейронна мережа

Штучна нейронна мережа (ШНМ) - це система обробки інформації або сигналів, що складається з великої кількості простих елементів, з'єднаних між собою прямими зв'язками, які співпрацюють для виконання паралельної розподіленої обробки з метою вирішення бажаної обчислювальної задачі. Нейронні мережі обробляють інформацію подібно до того, як це робить людський мозок. ANN натхненний тим, як працюють біологічні нервові системи, такі як мозок - нейронні мережі навчаються на прикладі. ANN використовує інший підхід до вирішення проблем, ніж традиційні обчислення.

Звичайні комп'ютерні системи використовують алгоритмічний підхід, тобто слідують набору інструкцій, щоб вирішити проблему [11]. Це обмежує можливості вирішення проблем тими проблемами, які ми вже розуміємо і знаємо, як їх вирішувати. Однак нейронні мережі та звичайні алгоритмічні обчислення не конкурують, а доповнюють один одного. Є завдання, які більше підходять для алгоритмічного підходу, наприклад, арифметичні операції, і завдання, які більше підходять для підходу нейронних мереж.

Нейронну мережу можна уявити як мережу «нейронів», організованих пошарово. Кількість типів штучних нейронних мереж («ANN») та їхніх застосувань потенційно може бути дуже великою. З часу створення першої нейронної моделі Мак-Каллохом і Піттсом було розроблено сотні різних моделей, які вважаються ANN [12]. Вони можуть відрізнятися функціями, прийнятими значеннями, топологією, алгоритмами навчання тощо. Також існує багато гібридних моделей.

Оскільки функція ANN полягає в обробці інформації, вони використовуються переважно в галузях, пов'язаних з нею. ANN формується з окремих одиниць (штучних нейронів або обробних елементів - PE), з'єднаних між собою коефіцієнтами (вагами), які складають нейронну структуру і організовані пошарово.

Потужність нейронних обчислень досягається завдяки з'єднанню нейронів у мережу. Кожен нейрон має зважені входи, передатну функцію та

один вихід. Поведінка нейронної мережі визначається передавальними функціями її нейронів, правилами навчання і самою архітектурою. Ваги є регульованими параметрами, і в цьому сенсі нейронна мережа є параметризованою системою. Зважена сума входів є активацією нейрона.

Розглянемо популярні структури нейронних мереж:

1. Багатошаровий персептрон (MLP) є базовою архітектурою штучних нейронних мереж, що складається з трьох основних типів шарів: вхідного, одного або кількох прихованих та вихідного, дана структура зображена на рис.1.2. Вхідний шар приймає дані у вигляді числових векторів, приховані шари виконують нелінійні перетворення для навчання ознак, а вихідний шар генерує результат, відповідний поставленому завданню, наприклад класифікації або регресії [13].

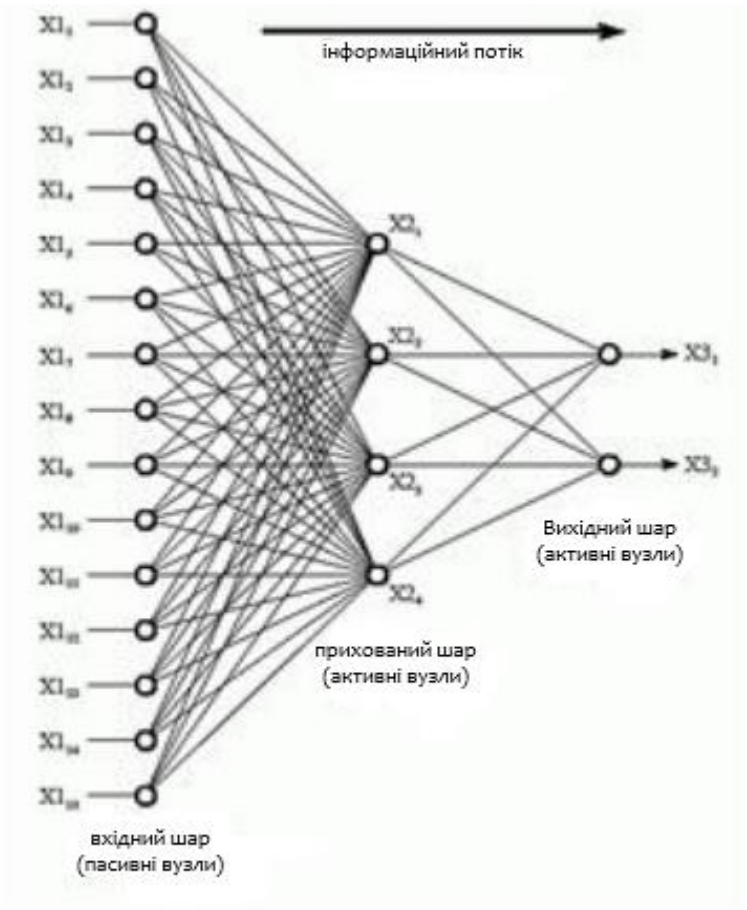


Рисунок 1.2 – Модель MLP

Ключовими елементами цієї архітектури є повнозв’язна структура між шарами, яка забезпечує зв’язок кожного нейрона з усіма нейронами суміжних шарів, а також функції активації, що додають нелінійність до моделі, дозволяючи їй навчатися складних залежностей. Навчання MLP здійснюється методом зворотного поширення помилки, який включає оптимізацію вагових коефіцієнтів для зменшення похибки між прогнозом і реальними даними.

2. Адаптивний лінійний нейрон ADALINE або пізніший адаптивний лінійний елемент (рис. 1.3) - це рання одношарова штучна нейронна мережа і назва фізичного пристрою, який реалізував цю мережу.

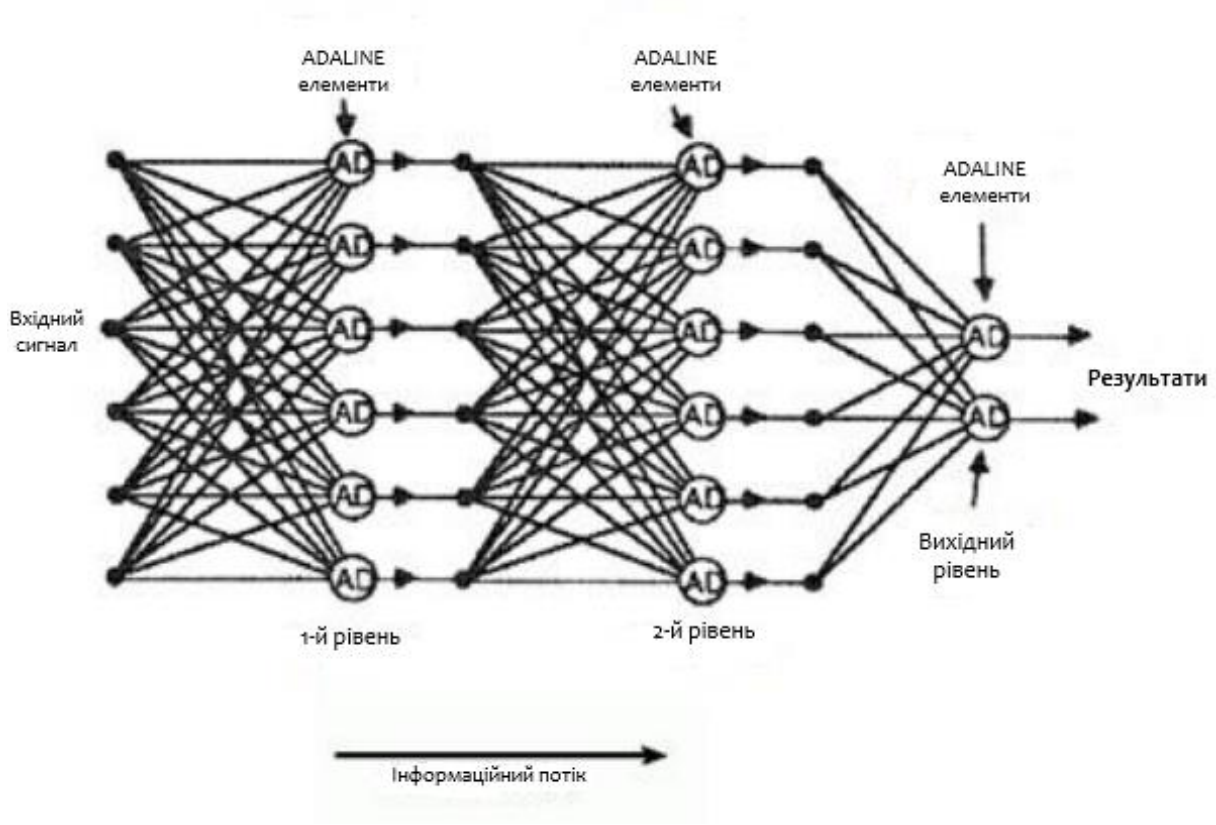


Рисунок 1.3 – Модель ADALINE

Вона була розроблена Бернардом Відроу і Тедом Хоффом зі Стенфордського університету в 1960 році. В її основі лежить нейрон Мак-Каллока-Піттса. Він складається з ваги, зсуву та функції підсумовування. Різниця між Adaline і стандартним персептроном (Мак-Каллока-Піттса) полягає в тому, що на етапі навчання ваги коригуються відповідно до зваженої суми

вхідних даних (мережі). У стандартному персептроні нейромережа передається на активаційну (передавальну) функцію, а вихід функції використовується для налаштування ваг. Існує також розширення, відоме як Madaline [14].

3. Модель ART. Основна ідея моделі ART (рис. 1.4) полягає в тому, що ідентифікація та розпізнавання об'єктів, як правило, відбувається в результаті взаємодії очікувань спостерігача «зверху-вниз» з сенсорною інформацією «знизу-вгору».

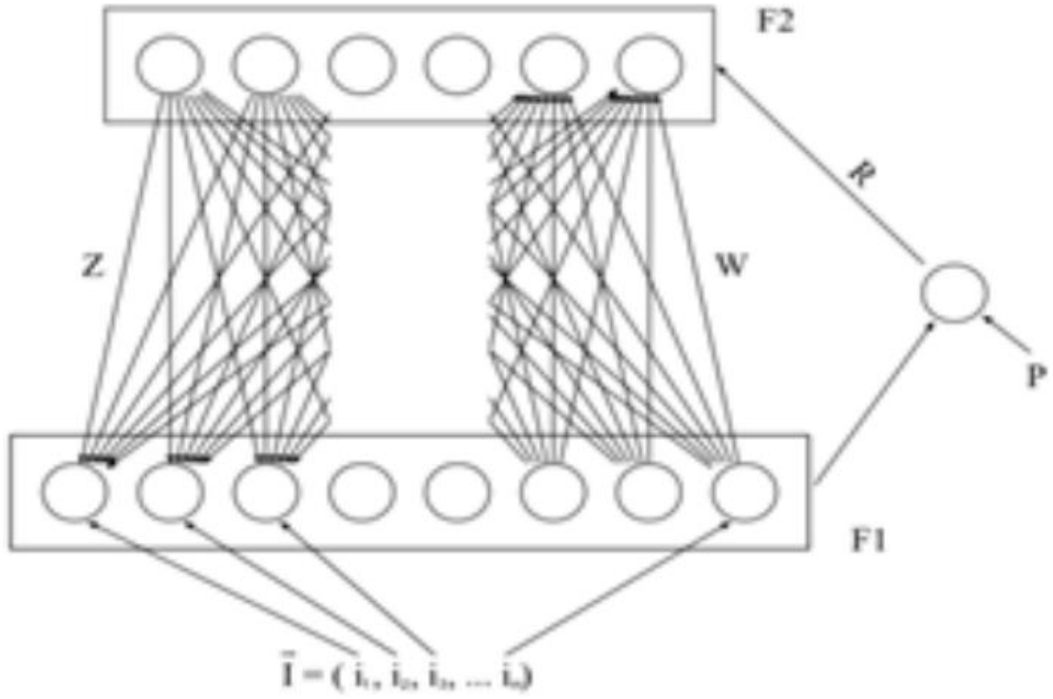


Рисунок 1.4 – Модель ART

Модель постулює, що «низхідні» очікування приймають форму шаблону пам'яті або прототипу, який потім порівнюється з реальними характеристиками об'єкта, виявленими органами чуття. Це порівняння породжує міру приналежності до категорії. Поки різниця між відчуттям і очікуванням не перевищує встановленого порогу, який називається «параметром пильності», об'єкт, що сприймається, буде вважатися членом очікуваного класу.

Таким чином, система пропонує вирішення проблеми «пластичності/стабільності», тобто проблеми набуття нових знань без порушення існуючих знань [15].

4. Архітектура SOM .Самоорганізуюча карта ознак (Self-Organizing Feature Map, SOM) — це архітектура штучної нейронної мережі, розроблена Тейво Кохоненом (рис. 1.5). Вона базується на принципах неконтрольованого навчання і використовується для кластеризації та візуалізації багатовимірних даних. SOM зменшує розмірність даних, перетворюючи їх у двовимірну решітку, яка зберігає топологічні властивості початкового простору.

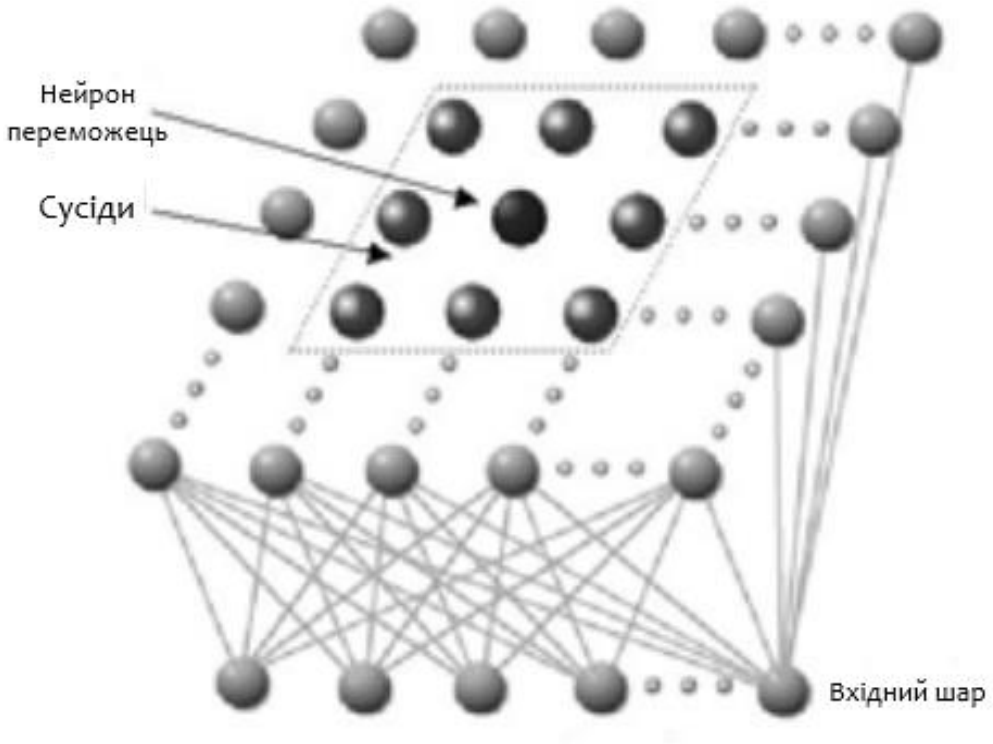


Рисунок 1.5 – Архітектура SOM

Основні характеристики:

- Вхідний шар приймає багатовимірні вектори даних.
- Вихідний шар складається з нейронів, організованих у вигляді решітки (зазвичай двовимірної).
- Навчання здійснюється через пошук нейрона-переможця (Best Matching Unit, BMU), а потім корекцію ваг цього нейрона і його сусідів.

SOM знаходить застосування у сфері аналізу даних, сегментації зображень, кластеризації текстів і створення карт подібностей у фінансових і біоінформатичних дослідженнях [16].

В той час як більшість інших мереж призначені для завдань навчання під контролем, мережі SOFM призначені в першу чергу для навчання без контролю. В той час, як в контрольованому навчанні набір навчальних даних містить випадки, що містять вхідні змінні разом з відповідними виходами (і мережа повинна зробити висновок про відображення від входів до виходів), в неконтрольованому навчанні набір навчальних даних містить тільки вхідні змінні.

5. Мережа Хопфілда– це один з типів рекурентних нейронних мереж, які застосовуються для створення асоціативної пам'яті. Основна ідея цього підходу полягає в тому, щоб зберігати певні патерни і відновлювати їх за пошкодженими або частковими вхідними даними. (рис. 1.6) .

У цій архітектурі всі нейрони пов'язані між собою, отже, вона утворює повністю взаємопов'язану мережу. Ваги зв'язків між нейронами симетричні, що дозволяє мережі працювати стабільно і мінімізувати спеціальну енергетичну функцію. Це ґрунтується на ключовій особливості використання дискретних станів нейронів, що приймають значення +1 або -1.

Таким чином, мережа може стабілізуватися в часі в один зі станів, що відповідають запам'ятовуваним шаблонам. Мережі Хопфілда слугують системами пам’яті з контент-адресацією, які використовують бінарні вузли з пороговою функцією [17].

Мережі Хопфілда слугують системами пам’яті з контент-адресацією, які використовують бінарні вузли з пороговою функцією. Вони гарантовано сходяться до локального мінімуму, проте може статися збіжність до хибного патерну (невірного локального мінімуму) замість збереженого патерну (очікуваного локального мінімуму). Мережі Хопфілда також пропонують модель для розуміння роботи людської пам’яті.

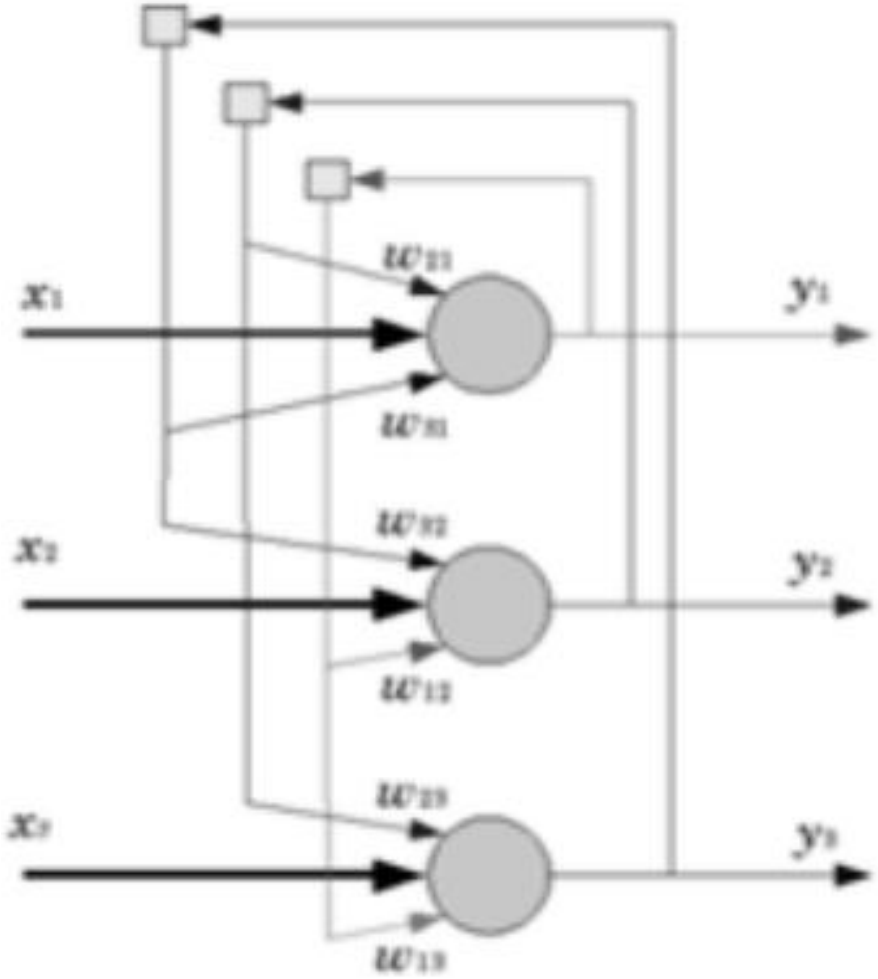


Рисунок 1.6 – Модель мережі Хопфілда

6. Згорткова нейронна мережа Згорткова нейронна мережа (рис. 1.7) є типом прямого штучного нейронного мережевого з'єднання, в якій окремі нейрони організовані таким чином, щоб реагувати на перекриваючі області, які утворюють візуальне поле.

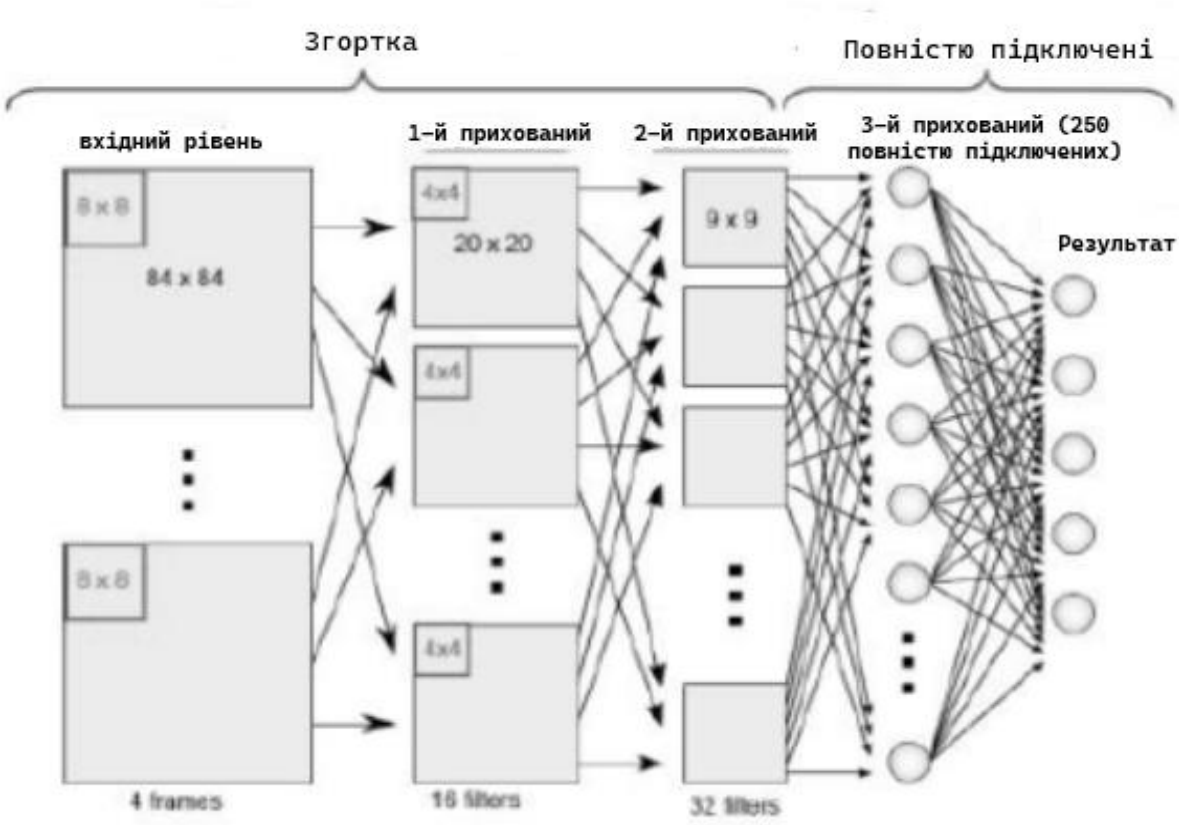


Рисунок 1.7 – Згорткова нейронна мережа

Основна ідея полягає в тому, щоб аналізувати зображення локально за допомогою невеликих фільтрів (згорток), що знижує потребу в обробці всього зображення одночасно. Згорткові нейронні мережі складаються з кількох шарів, кожен з яких містить невеликі колекції нейронів, які обробляють окремі частини вхідного зображення (підобласті). Ці підобласті називають рецептивними полями. Виходи цих нейронних колекцій (тобто результат згортки) потім комбінуються, щоб їх вхідні області перекривалися, забезпечуючи отримання більш детального й узагальненого представлення оригінального зображення.

Цей процес повторюється на кількох шарах, де кожен наступний шар виявляє дедалі складніші й узагальнює ознаки. Наприклад, перший шар може виділяти прості контури, такі як вертикальні або горизонтальні лінії, тоді як наступні шари будуть розпізнавати складніші форми, наприклад, текстури або цілі об'єкти. Такий підхід дозволяє згортковим нейронним мережам ефективно

обробляти великі вхідні дані (наприклад, зображення високої роздільної здатності), зберігаючи при цьому інформацію про просторові залежності між пікселями [18].

Нейронні мережі широко використовуються в неконтрольованому навчанні (рис. 1.8) для того, щоб навчитися краще представляти вхідні дані. Наприклад, маючи набір текстових документів, НМ може навчитися відображення від документа до вектора дійсних значень таким чином, щоб результуючі вектори були схожими для документів зі схожим змістом, тобто зі збереженням відстані.

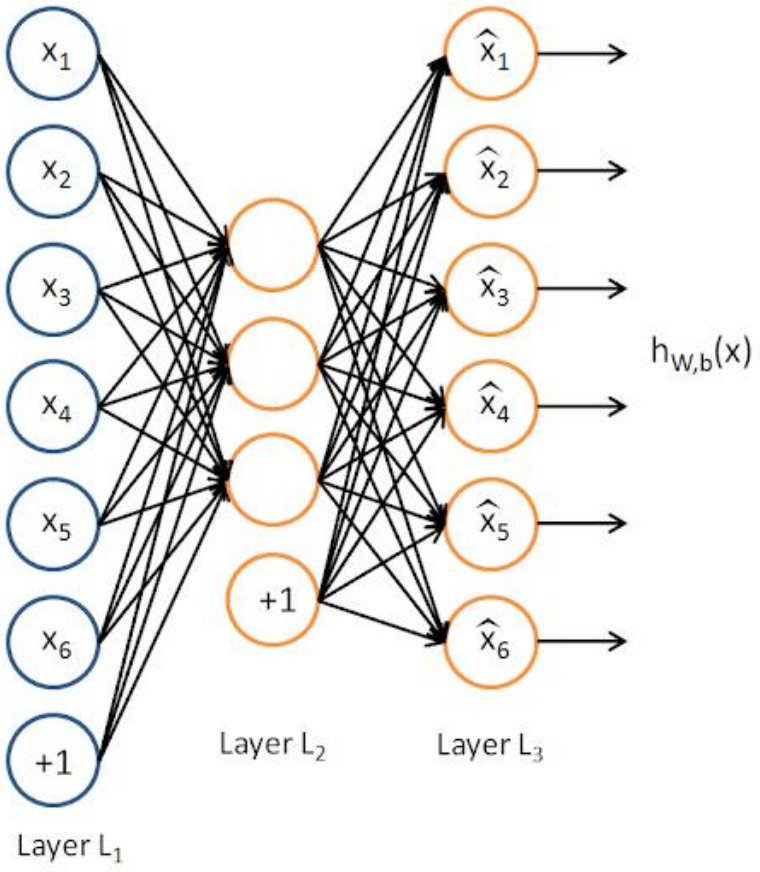


Рисунок 1.8 – Приклад неконтрольованого навчання нейронної мережі

Цього можна досягти за допомогою, наприклад, автокодерів - моделі, яка навчена відновлювати оригінальний вектор з меншого представлення (активації прихованого шару) з помилкою реконструкції (відстань від функції ідентифікації) як функцією вартості. Цей процес не дає вам кластерів, але він створює змістовні представлення, які можна використовувати для

кластеризації. Наприклад, ви можете запустити алгоритм кластеризації на активаціях прихованого шару.

Кластеризація: Існує низка різних архітектур NN, спеціально розроблених для кластеризації. Найвідомішою з них є, мабуть, самоорганізаційні карти. SOM - це NN, яка має набір нейронів, з'єднаних у топологічну сітку (зазвичай прямокутну). Коли деякий шаблон подається на ШНМ, нейрон з найближчим ваговим вектором вважається переможцем, і його ваги адаптуються до шаблону, а також до ваг його околиць. Таким чином, SOM природним чином знаходить кластери даних. Дещо схожий алгоритм - вирощування нейронного газу (він не обмежується наперед визначеною кількістю нейронів) [9].

Інший підхід - адаптивна резонансна теорія, де ми маємо два шари: «поле порівняння» і “поле розпізнавання”. Поле розпізнавання також визначає найкращу відповідність (нейрон) вектору, переданому з поля порівняння, а також має бічні гальмівні зв'язки. Деталі реалізації та точні рівняння можна легко знайти, загугливши назви цих моделей, тому я не буду їх тут наводити.

Навчання під контролем (рис 1.9), також відоме як контрольоване машинне навчання, є підкатегорією машинного навчання та штучного інтелекту. Воно визначається використанням маркованих наборів даних для навчання алгоритмів, які класифікують дані або точно прогнозують результати.

Коли вхідні дані подаються в модель, вона коригує свої ваги доти, доки модель не буде налаштована належним чином, що відбувається в рамках процесу перехресної валідації. Контрольоване навчання допомагає організаціям вирішувати різноманітні реальні проблеми в масштабі, наприклад, класифікувати спам в окремій папці від вашої поштової скриньки. Його можна використовувати для побудови високоточних моделей машинного навчання.

Навчання з контролем використовує навчальний набір даних для того, щоб навчити моделі генерувати бажаний результат. Цей навчальний набір даних містить вхідні значення та правильні вихідні результати, що дозволяє моделі вчитися з часом. Алгоритм оцінює свою точність за допомогою функції

втрат, коригуючи параметри до тих пір, поки помилка не буде зменшена до прийнятного рівня [10].

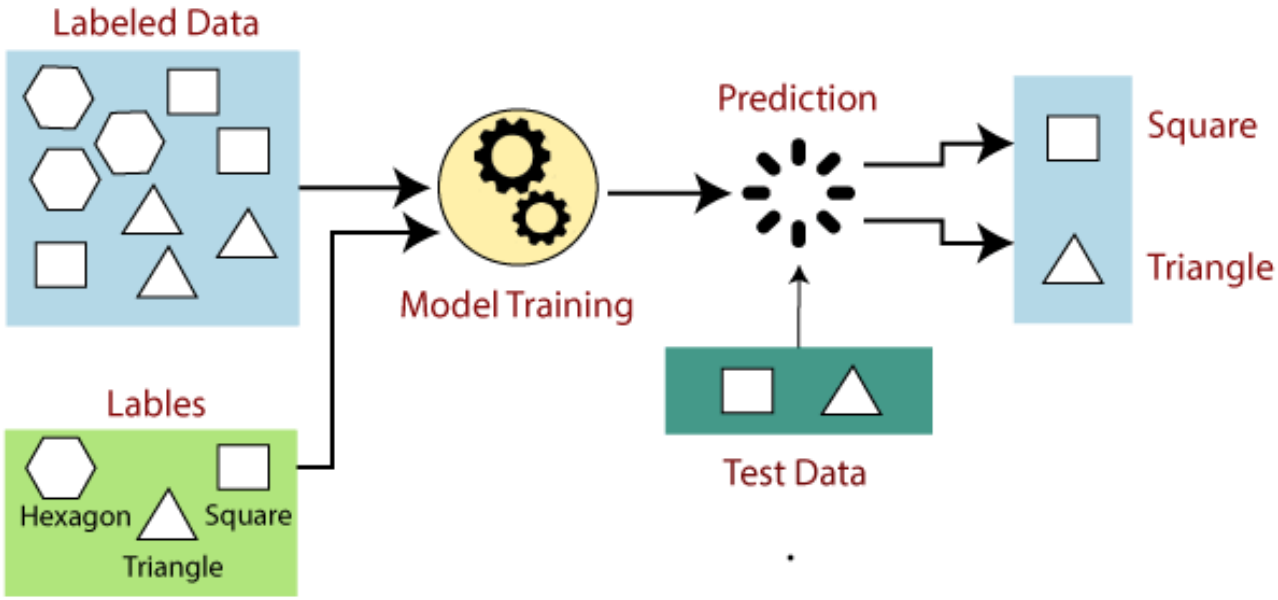


Рисунок 1.9 – Приклад контрольованого навчання нейронної мережі

Навчання з контролем поділяється на два типи задач у контексті пошуку даних: класифікацію та регресію:

- **Класифікація** використовує алгоритм для точного призначення тестових даних певним категоріям. Вона розпізнає конкретні об’єкти в наборі даних і намагається зробити висновки про те, як ці об’єкти слід позначити чи визначити. Поширені алгоритми класифікації включають лінійні класифікатори, метод опорних векторів (SVM), дерева рішень, метод k-ближчих сусідів і випадковий ліс, які описані більш детально нижче.
- **Регресія** використовується для розуміння зв’язку між залежними та незалежними змінними. Її зазвичай застосовують для створення прогнозів, наприклад, для передбачення доходів від продажів у певному бізнесі. Популярними алгоритмами регресії є лінійна регресія, логістична регресія та поліноміальна регресія.

1.3 Плагіат

Óксфордський університет визначає плагіат наступним чином: «Представлення роботи або ідей з іншого джерела як власних, за згодою або без згоди оригінального автора, шляхом включення їх у свою роботу без повного визнання. Під це визначення підпадає весь опублікований і неопублікований матеріал у рукописній, друкованій або електронній формі, а також використання матеріалів, створених повністю або частково за допомогою штучного інтелекту (за винятком випадків, коли використання ШІ для оцінювання отримало попередній дозвіл, наприклад, як розумне пристосування до інвалідності студента). Плагіатом також може бути повторне використання власної роботи без посилання на джерело. Згідно з правилами проведення іспитів, навмисний або необережний плагіат є дисциплінарним порушенням».

Необхідність визнавати чужу роботу або ідеї стосується не лише тексту, але й інших носіїв інформації, таких як комп'ютерний код, ілюстрації, графіки тощо. Це однаково стосується як опублікованих текстів і даних, взятих з книг і журналів, так і неопублікованих текстів і даних, наприклад, з лекцій, дисертацій або інших студентських робіт. Ви також повинні вказувати посилання на текст, дані або інші ресурси, завантажені з веб-сайтів [19].

Плагіат ідей є неприйнятним у жодному разі. В очах майже всіх науковців він прирівнюється до крадіжки. Залежно від галузі дослідження, формулювання тексту, а отже, і плагіат тексту, може мати першорядне значення. Наприклад, у гуманітарних науках і літературознавстві, де суть новизни й оригінальності твору справді залежить від вибору слів і способу їхнього поєднання, дослівний переклад майже в усіх культурах вважається кричущою помилкою. Проте в деяких культурах певна ступінь дослівності не тільки допустима, але й заохочується! Наприклад, у перській літературі існують прийоми, які використовують дослівність, щоб нібито надати більшої краси та красномовності тексту чи віршам. Один з таких прийомів полягає в тому, щоб запозичити фрагмент тексту, зазвичай у видатного ерудованого вченого, і

використати його точно так, як він є (технічно текстовий плагіат), безпосередньо у власному тексті без чіткого відокремлення запозиченого тексту від решти тексту (наприклад, вставивши його в подвійні лапки або з відступом) або навіть виключно з посиланням на першоджерело або на те, хто є оригінальним автором.

Передбачається, що запозичений текст є настільки відомим, що читачі зрозуміють, звідки запозичений фрагмент. Це було звичайною практикою навіть для всесвітньо відомих великих іранських поетів, таких як Седі та Хафез. Це не вважалося проступком. Це також не супроводжується соромом. У перській літературі це здавна вважалося мистецькою майстерністю. Ця практика не обмежується перською грамотністю.

Плагіат не завжди був шкідливим. Schola Medica Salernitana, перша сучасна медична школа, заснована в Західній Європі, процвітала завдяки існуванню кількох підручників. Однією з таких впливових книг була «Liber Pantegni», написана туніським лікарем Константином Африканським (1020-1087). Однак, за твердженням Стефана Антіохійського (перша половина 12 століття), Liber Pantegni є перекладом Liber Regalis (Kitāb al-mālikī), книги, написаної іранським лікарем Халі Аббасом (Алі ібн Аббасом Аль-Маджусі). За сьогоднішніми стандартами, те, що зробив Константин Африканський, було кричущим проступком, проте, якби він не плагіатував книгу Халі Аббаса, ми, ймовірно, не мали б Schola Medica Salernitana і сучасних університетів по всьому світу [20].

Плагіат як феномен можна поділити на 7 найпоширеніших типів, розглянемо дані типи разом з умовними прикладами та можливими порадами усунення:

1. Повний плагіат

Плагіат означає копіювання чужої роботи і використання її так, ніби ви є її оригінальним автором. Це схоже на визначення повного плагіату. У цьому типі дублювання ви копіюєте весь вихідний матеріал.

Наприклад, вам необхідно здати есе про технології шиття для наукового дослідження. Ви копіюєте всю статтю про чорні діри з Вікіпедії і використовуєте її у своєму есе. Якщо ви не вказуєте, звідки ви взяли інформацію, це буде навмисним плагіатом, який підпадає під категорію повного плагіату.

2. Прямий плагіат

Прямий плагіат - це тип дублювання, коли ви використовуєте розділи, речення або абзаци чужої роботи. Ви копіюєте все дослівно і використовуєте його, не вносячи жодних змін і не вказуючи джерело. Наприклад, ви натрапили на наукову роботу і використовуєте її висновки у своїй роботі.

3. Перефразування

Перефразування — це вид плагіату, коли автор змінює слова або структуру тексту, але зберігає ідеї чи зміст оригінального джерела без належного посилання на нього. Це обманливий спосіб уникнути прямого копіювання тексту, який може виглядати як власна робота, хоча фактично базується на чужих ідеях або дослідженнях.

Перефразування зазвичай виникає в ситуаціях, коли студент чи автор прагне уникнути прямого копіювання тексту, щоб обійти системи перевірки на плагіат. Водночас основна проблема полягає в тому, що навіть за зміненого формулювання зміст залишається чужим.

4. Клаптиковий плагіат

У цій формі плагіату ви відтворюєте у своїй роботі матеріал з декількох джерел. Ось приклад плагіату - вам потрібно написати реферат на уроці літератури. Ви знаходите кілька джерел в Інтернеті і запозичуєте їхні резюме у своєму огляді. Ви також включаєте до звіту свої думки про книгу. Будь-яка програма перевірки на плагіат позначить скопійовані розділи як плагіат, тобто клаптикове копіювання.

Плагіат на основі джерела

Плагіат за джерелом виникає, коли ви вказуєте вторинні джерела, навіть якщо вони походять з першоджерела.

Якщо ви подивитеся на будь-яку сторінку Вікіпедії, то помітите, що після певних речень стоять надрядкові знаки. Ці надрядкові знаки є посиланнями на джерела, які Вікіпедія використовувала у своїй статті.

Якщо ви посилаєтесь у своїй роботі лише на Вікіпедію, це стає плагіатом за джерелом.

5. Випадковий плагіат

Випадковий плагіат є поширеним явищем і трапляється тоді, коли ви не перевіряєте свою роботу перед відправкою. Це також може статися, якщо ви припускаєтеся помилок, вказуючи автора. У більшості випадків це ненавмисно, оскільки ви не знали про джерело. Це може статися підсвідомо, коли ви нещодавно читали оригінальний матеріал.

Наприклад, вам потрібно зробити рецензію на фільм в рамках вашого завдання. Ви переглядаєте кілька сайтів, щоб дізнатися про фільм. Під час написання рецензії ви несвідомо включили деякі з цих матеріалів і здали роботу. Оцінювач вважатиме це плагіатом, навіть якщо ви не знали про свої дії.

Плагіат часто обговорюють пліч-о-пліч з питаннями академічної доброчесності. У багатьох випадках вони використовуються як взаємозамінні. І хоча плагіат дійсно є актом академічної нечесності та академічного проступку, він не є повною мірою академічною доброчесністю.

Академічна доброчесність - це набір принципів, спрямованих на розвиток і просування академічної культури, вільної від академічних порушень і корупції. Академічна доброчесність зазвичай описується як зобов'язання, що включає шість фундаментальних цінностей: чесність, довіру, справедливість, повагу, відповідальність і сміливість, але не обмежується ними. Ці цінності передбачають відповідальність кожного в академічному світі за дотримання високих стандартів академічної доброчесності.

Міжнародний центр академічної доброчесності «визначає академічну доброчесність як зобов'язання, навіть перед обличчям негараздів, дотримуватися шести фундаментальних цінностей: чесності, довіри, справедливості, поваги, відповідальності та сміливості. З цих цінностей

випливають принципи поведінки, які дозволяють академічним спільнотам втілювати ідеали в життя»

Розглянемо фундаментальні цінності академічної доброчесності згрупувавши відповідальність та сміливість як взаємопов’язані аспекти в один пункт, таким чином ми має п’ять основних принципів зображених на рис. 1.10.

- Чесність: правдивість є фундаментальним принципом академічної доброчесності, і в письмовій формі це означає, що потрібно вказувати авторство роботи у формі атрибуції. Це також означає бути об'єктивним; для викладачів це означає оцінювати студентські роботи без упереджень.
- Довіра: віра в надійність студентських робіт є критично важливою для академічної доброчесності. У класі це демонструється встановленням чітких очікувань і підтримкою цих очікувань в оцінюванні.
- Справедливість: уникнення фаворитизму - ще один аспект академічної доброчесності. Це означає послідовне застосування правил і відповідальність за власні дії у вигляді рубрик та інших актів освітньої справедливості.
- Повага: повага до кожного є частиною академічної доброчесності. Повага проявляється в тому, що студенти серйозно ставляться до завдань і навчання та отримують зворотній зв'язок. Викладачі, у свою чергу, також повинні надавати зворотний зв'язок і виявляти емпатію до студентів.
- Відповідальність і сміливість: бути надійним і гідним довіри є фундаментальним для академічної доброчесності. Студенти повинні протистояти неправомірним діям, тоді як викладачі створюють і підтримують політику в класі та інституції.

Плагіат, або використання ідей чи слів іншої людини та видача їх за власну оригінальну ідею, порушує всі складові академічної доброчесності. Плагіат, зокрема, є різновидом академічної недоброчесності та одним із способів порушення академічної доброчесності.

Серед причин поширення плагіату можна виділити недостатнє розуміння принципів академічного письма, тиск дедлайнів, страх отримати незадовільну оцінку, а також легкий доступ до інформації в епоху цифрових технологій. Особливої уваги заслуговує роль технологій, які полегшують як створення, так і виявлення плагіату.

В академічному середовищі наслідки плагіату можуть бути значними: від анулювання результатів роботи до виключення з навчальних закладів. Для викладачів і науковців це може призвести до втрати репутації та професійної кар'єри. Тому університети й наукові установи активно розробляють політики боротьби з плагіатом, впроваджуючи технологічні рішення, такі як системи перевірки текстів, та освітні ініціативи з підвищення рівня обізнаності студентів.

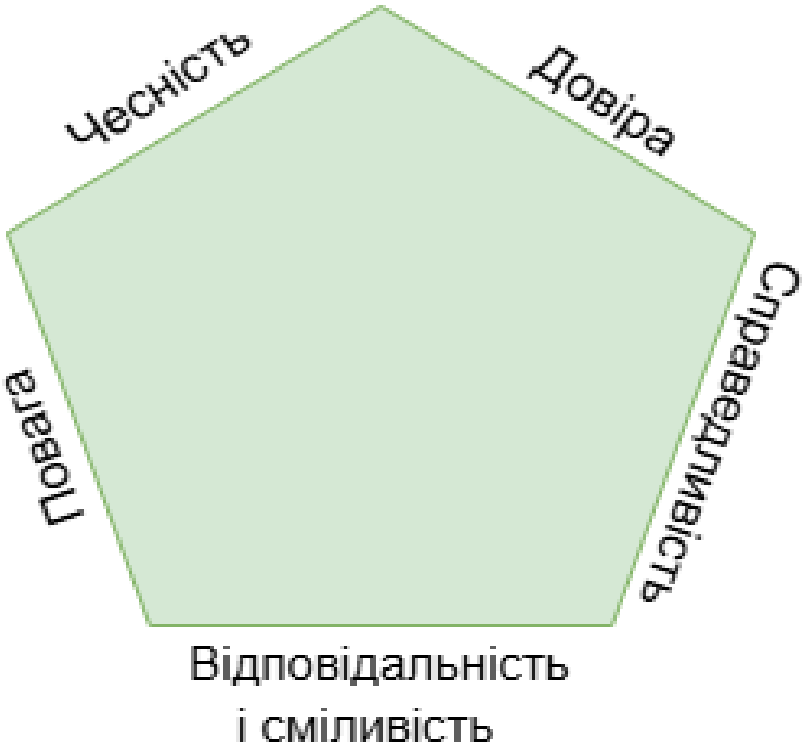


Рисунок 1.10 – фундаментальні цінності академічної доброчесності»

Одним із ключових інструментів у протидії плагіату є впровадження чітких інструкцій щодо академічного письма та навчання студентів етичним принципам роботи з джерелами. Наприклад, у багатьох університетах створюються курси, спрямовані на роз'яснення правил цитування, а також організовуються тренінги щодо уникнення ненавмисного плагіату.

Плагіат залишається серйозною загрозою для академічної доброчесності, але боротьба з ним можлива через поєднання освітніх, правових та технологічних підходів. Запобігання плагіату повинно стати невід'ємною частиною академічної культури, яка ґрунтується на етиці, чесності й повазі до інтелектуальної власності. Важливо, щоб студенти, викладачі та науковці не тільки дотримувалися правил доброчесності, але й розуміли їхню цінність для розвитку науки та освіти.

Висновки до першого розділу

В даному розділи було проведено теоретичні та методичні дослідження теми технологія ШІ в створенні інструмента для оцінки оригінальності текстів. Об'єктами аналізу стали такі поняття як сам штучний інтелект, поняття нейронної мережі та поняття академічної доброчесності та плагіату в цілому.

В аналізі присвяченому темі штучного інтелекту було розглянуто саме визначення даної системи, а також досліджено архітектури та моделі які являються основою цієї тематики. Розгляд було проведено з достатнім описом та візуальним підкріпленням. Це є важливим аспектом даної роботи, розпізнання результатів такої системи являється одним з основних її частин.

В частині розділу присвяченій нейронним мережам було визначено саме поняття такої мережі, аспекти її розробки та побудови з урахування різних етапів життя подібної моделі. Також були розглянуті та наочно продемонстровані основні існуючі моделі нейронних мереж з описом достатнім для поверхневого розуміння принципів їх роботи та додатковими візуальними аспектами які допомагають підняти це розуміння на більш високий рівень.

В останній частині розділу були опрацьовані питання академічної доброчесності та явища плагіату. Були розглянуті як самі визначення цих

питань так і технічні аспекти та принципи цього питання. Розгляд проводився з урахуванням різних точок зору, тобто були розглянуті як очевидні негативні приклади плагіату як порушення академічної доброчесності з можливими результатами такого недоброчесного підходу, так і приклади коли подібні дії призводили до позитивних наслідків. Також важливо зазначити що в аналізі було розглянуто і різні культурні особливості, що робить його більш багатогранним.

Тобто можна підсумувати що було проведено достойну роботу над аналізом предметної області, який в свою чергу являється достатньо глибоким та різноманітним.

РОЗДІ 2. ПРОЕКТУВАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Визначення мети та завдань програмного забезпечення.

Як приклад реалізації технологій ШІ в створенні інструмента для оцінки оригінальності текстів повинна бути написана програма , що буде поєднувати систему порівняння текстів з технологіями штучного інтелекту.Це буле нейронна мережа, яка навчена розпізнавати патерни та закономірності що використовує штучний інтелект для написання текстів, які в подальшому можуть бути використаними недобросовісними учасниками навчального процесу для написання наукових робіт.

Нажаль навіть при наявності доступних та реалізованих архітектур нейронних мереж для створення власної системи пошуку тексту, що написаний електронними помічниками, такими як chat GPT або Copilot, потрібно провести процес навчання нейронної мережі, а саме збір та збереження величезної бази даних робіт написаних як людьми так і штучним інтелектом, опрацювання процесу навчання мережі методом проведення розрахунків, кількість яких стандартно непомірно велика та не має точної кількості, адже повторюється до моменту отримання запланованих результатів, корегування вагів нейронної мережі та тестування наслідків роботи цієї системи.

Оскільки для реалізації настільки масштабного проекту з адекватними результатами необхідно залучити незмірні кількості обчислювальних потужностей, залучити команду спеціалістів з глибокими знаннями окремих аспектів робочого процесу які неможливо зосередити в окремому індивіді та зібрати базу робіт гігабайт них,а краще терабайтних маштабів, що остаточно та обов’язково означає залучання значних фінансових та інших ресурсів, і призводить до закономірного рішення використання готової системи використовуючи механізми API.

Враховуючи вище зазначену мету та особливості системи яку необхідно розробити можна сформувати деякі вимоги які необхідно виконати для демонстрації об’єднання системи розпізнавання плагіату з методами ШІ, а саме:

- Реалізація алгоритму перевірки текстів на плагіат;
- Мати можливість інтеграції з API для виявлення текстів, створених штучним інтелектом;
- Здійснення локального збереження робіт для подальшого використання в аналізі;
- Розробити інтуїтивно зрозумілий графічний інтерфейс користувача;
- Забезпечити сумісність з операційною системою Windows.

Однією з очевидних вимог до будь-якого проекту є наявність існуючого попиту на результати створеної системи. Для цього програмного забезпечення цільова аудиторія складається з двох основних груп користувачів:

- Першу групу представляють викладачі та адміністративний персонал навчальних закладів, які прагнуть забезпечити академічну доброчесність, перевіряючи роботи студентів на наявність плагіату та визначаючи тексти, створені штучним інтелектом.
- Друга група включає розробників програмного забезпечення та дослідників, які можуть використовувати створену програму як прототип або основу для впровадження подібних систем у своїх розробках чи наукових дослідженнях.

Програмне забезпечення орієнтоване на тих, хто потребує локальних, швидких і зручних інструментів для перевірки текстів без складних технічних налаштувань.

2.2 Вибір технологій і середовища розробки

Перед початком написання програмного коду необхідно визначитись з тим яку мову програмування, середовище розробки й формат для збереження даних буде використано. Правильний вибір мови програмування та середовища розробки є ключовим для ефективного створення програмного забезпечення, оскільки забезпечує відповідність технічним вимогам проєкту, спрощує реалізацію функцій, оптимізує продуктивність і масштабованість. Від обраної

технології залежить інтеграція з іншими системами, зручність підтримки, швидкість розробки та адаптивність програми до майбутніх змін. Крім того, вибір знайомих розробникам технологій скорочує час навчання і підвищує ефективність роботи.

2.2.1 Вибір мови програмування.

C#, яку також називають C Sharp - це мова програмування, яку створив Андерс Хьельсберг (рис. 2.1). Раніше C Sharp називали «Cool», але в якийсь момент в минулому цю назву було змінено. Це високорівнева об'єктно-орієнтована мова програмування, яка побудована на C, так само як і C++. C - це будівельний блок, з якого зроблено C#.

Рисунок 2.1 – Андерс Хьельсберг – розробник мови C#

Microsoft випустила C# у 2000 році (рис. 2.2). Це було зроблено, щоб задовольнити зростаючий попит на онлайн додатки, який Visual Basic (VB) та C++ не могли задовольнити. Архітектура мови поєднує в собі найкращі частини Java та C++. Завдяки цьому програмісти, які знають, як працювати з C та C++, можуть швидко навчитися використовувати C# [21].

Рисунок 2.2 – «Мова програмування C#»

C# від самого початку була створена за правилами об'єктно-орієнтованого програмування, яке іноді називають ООП. Щоб ця ідея програмування працювала, ви повинні вміти описувати тип даних і те, як вони організовані, щоб над ними можна було використовувати стандартний набір функцій. Об'єктно-орієнтоване програмування перетворює дані на об'єкти, що полегшує розбиття програми на менші частини, які легше створювати, керувати та збирати разом.

Об'єктно-орієнтоване програмування дозволяє керувати об'єктами без необхідності мати справу безпосередньо з їхніми внутрішніми властивостями (рис 2.3). Замість цього використовується оголошення класів для опису поведінки об'єктів. ООП мови полегшують тестування і читання програм. Вони також дають можливість виправляти будь-які проблеми, що виникають. Загалом, вони роблять процес кодування більш впорядкованим [22].

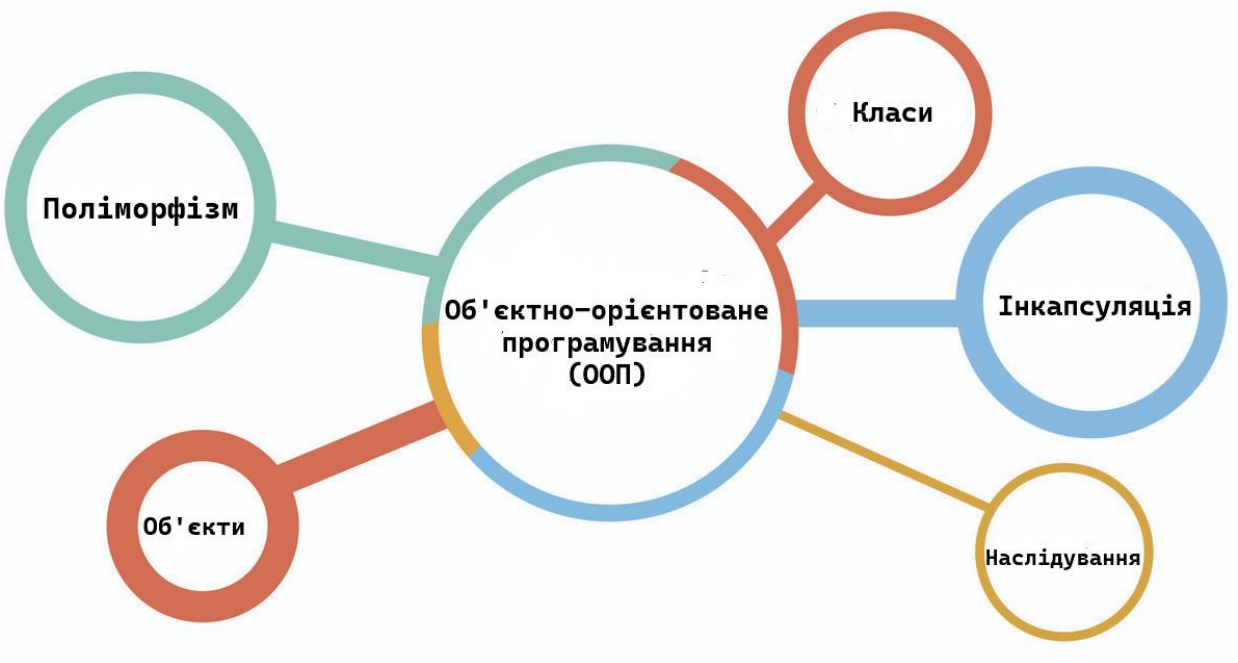


Рисунок 2.3 – ООП та його парадигми

На C# вплинули такі мови, як C, C++ та Java. Однак її творці взяли найкраще з цих мов і зробили C# своєю власною, додавши такі речі, як типи значень, властивості та події. Наприклад, мова програмування C# не дозволяє використовувати сирі вказівники, які вказують безпосередньо на пам'ять, а також не дозволяє успадковувати більше одного класу. У C# ви можете використовувати так званий «збирач сміття», який подбає про керування пам'яттю за вас. Управління пам'яттю - це не те, про що вам потрібно турбуватися для переважної більшості завдань, що робить роботу з C# набагато простішою.

C# є чудовим вибором для спрощення складності. Бувають випадки, коли ви можете успішно використовувати функції C#, навіть якщо не знаєте всього про те, як вони працюють за лаштунками.

Можна роками працювати з ітераторами і так і не зрозуміти, як вони працюють. Ви можете використовувати асинхронізацію та очікування, навіть якщо не знаєте всього про те, як компілятор реалізує цю функцію. Це базовий принцип інкапсуляції для об'єктно-орієнтованого програмування, вбудований у мову.

Мова програмування C# є хорошим вибором для цього проєкту, оскільки вона забезпечує зручну розробку Windows-додатків завдяки інтеграції з Visual Studio і платформою .NET. Її багатий набір бібліотек та інструментів спрощує створення графічного інтерфейсу (Windows Forms App) і роботу з файлами, наприклад, JSON.

Крім того, C# має потужні можливості для взаємодії з API, що важливо для реалізації перевірки текстів на оригінальність за допомогою сторонніх сервісів. Висока продуктивність, зрозумілий синтаксис і наявність великої спільноти розробників роблять C# ідеальним вибором для стабільного та масштабованого рішення.

2.2.2 Вибір середовища розробки.

Середовище розробки — це важливий набір інструментів, конфігурацій і процесів, які розробники використовують для створення, тестування та розгортання програмного забезпечення. Незалежно від того, чи це локальне налаштування на локальній машині розробника, чи віддалене хмарне рішення, основна мета середовища розробки полягає в тому, щоб забезпечити бездоганний простір, де можна писати, тестувати та ефективно виконувати код .

Для реалізації цього проекту було обрано середовище програмування Visual Studio, в цьому розділі ми розглянемо переваги які призвели до діного рішення.

Visual Studio - це потужний інструмент розробника, за допомогою якого ви можете виконати весь цикл розробки в одному місці (рис.2.4).

Це комплексне інтегроване середовище розробки (IDE), за допомогою якого ви можете писати, редагувати, налагоджувати та збирати код.

Visual Studio включає компілятори, інструменти для завершення коду, контроль вихідного коду, розширення та багато інших функцій для покращення кожного етапу процесу розробки програмного забезпечення [23].

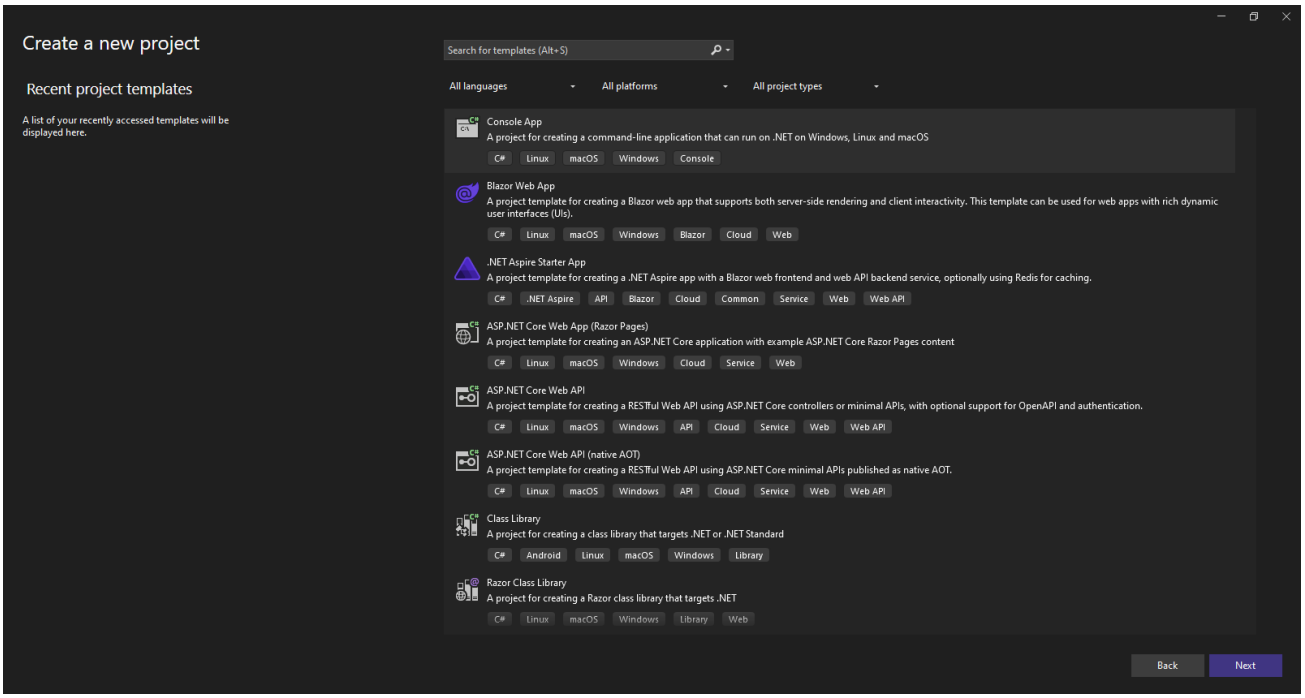


Рисунок 2.4 – Інтерфейс обраного середовище розробки Visual Studio

Visual Studio надає розробникам багатофункціональне середовище розробки для ефективного та спільного створення високоякісного коду, перерахуємо надані можливості:

- Інсталятор на основі робочого навантаження – варіативність у виборі необхідних систем що дозволяє гнучко налаштовувати систему під користувача;
- Потужні інструменти та функції кодування - все необхідне для всіх процесів написання програм в одному місці;
- Підтримка багатьох мов, перспектива писати код на C++, C#, JavaScript, TypeScript, Python та інших;
- Крос-платформна розробка здатність створювати додатки для будь-якої платформи;
- Інтеграція контролю версій;
- Розробка зі штучним інтелектом, можливість писати код ефективніше за допомогою штучного інтелекту.

Обрання середовища розробки Visual Studio для створення програми мовою C# є оптимальним рішенням завдяки його широкому функціоналу та підтримці сучасних технологій.

Visual Studio забезпечує зручний інтерфейс і багатий набір інструментів, які прискорюють розробку, включаючи інтеграцію з API, підтримку різних бібліотек та розширень, а також потужний редактор для роботи з кодом.

Крім того, Visual Studio пропонує вбудовані засоби для налагодження та тестування, що значно підвищує ефективність створення якісного програмного забезпечення. Використання C# в поєднанні з можливістю працювати з JSON-файлами легко реалізується завдяки готовим бібліотекам та інструментам у Visual Studio, що робить цей вибір практичним і раціональним.

2.3 Алгоритми аналізу тексту

Косинусна подібність - це метрика, яка використовується для визначення косинуса кута між двома ненульовими векторами в багатовимірному просторі. Це міра орієнтації, а не величини, в діапазоні від -1 до 1. В контексті схожості тексту ця метрика забезпечує надійний спосіб оцінити схожість між двома наборами текстових даних.

Математичне визначення: Коефіцієнт косинусної подібності обчислюється як точковий добуток двох векторів, поділений на добуток їхніх величин [24].

Математичне рівняння косинусної подібності між двома ненульовими векторами показано функцією (2.1) [25].

$$\text{подібність} = \cos(\theta) = \frac{A*B}{\|A\|\|B\|} \qquad (2.1)$$

Давайте конкретніше розглянемо це рівняння і визначимо що означають його конкретні елементи:

1. «Подібність» – це значення косинусної подібності між двома векторами A і B. Воно варіюється в межах від -1 до 1, де:
   - 1 означає максимальну подібність векторів, їх також називають співнаправлені;
   - 0 означає, що вектори ортогональні, це означає не схожі;
   - -1 означає, що вектори максимально відрізняються і не схожі між собою, тобто протилежно спрямовані.
2. $\cos\theta$ – це косинус кута θ між двома векторами A і B. Менший кут відповідає більшій схожості.
3. A*B – це скалярний добуток двох векторів. обчислюють за формулою 2.2, де A_i і B_i – компоненти векторів A і B відповідно.

$$A * B = \sum_{i=1}^{n} A_i B_i \qquad (2.2)$$

4. ||A|| і ||B|| – це довжини або норми векторів A і B, що обчислюють норми векторів A і B так, як показано згрупованих під один номер (2.3)

$$\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2}\,, \quad \|B\| = \sqrt{\sum_{i=1}^{n} B_i^2} \qquad (2.3)$$

5. n – кількість вимірів у векторах A і B. Це визначає розмірність простору, в якому проводиться обчислення.

Простіше кажучи, і в контексті обробки природної мови - це міра того, наскільки схожі ідеї та концепції, представлені в двох фрагментах тексту.

Загалом, одним з основних завдань систем оцінювання оригінальності текстів є перетворення технічних результатів роботи алгоритмів у формат, зрозумілий кінцевому користувачеві. У випадку аналізу на основі косинусної схожості результати, отримані в діапазоні [0,1], слід інтерпретувати як показник схожості тексту. Для цього результати нормалізуються і представляються відповідно.

Коефіцієнт косинусної подібності коливається від 0 (немає подібності) до 1 (повністю подібний) після проведення порівняння. Для того, щоб різниця значень сприймалася більш наочно, вони перетворюються у відсоткові значення шляхом множення результату на 100. Перетворення коефіцієнту косинусної подібності у відсотковий результат показано (2.4) :

$$\text{Схожість}(\%) = \cos\theta * 100 \qquad (2.4)$$

»

Наприклад, якщо значення cos(θ) дорівнює 0.75, це означає, що текст має 75% схожості з базовим зразком.

Після переведення у відсотки результати подаються в інтерфейсі користувача як показник, який дозволяє швидко оцінити можливість плагіату. Для цього встановлюються умовні пороги, наприклад:

- Менше 30%: Низький рівень схожості; висока ймовірність того, що текст є оригінальним;
- 30-70%: Як правило, середній рівень схожості, який вимагає більш детального аналізу для підтвердження;
- Понад 70%: Висока ймовірність плагіату.

2.4 Формат зберігання даних для аналізу робіт

Проміжний формат даних для перетворення з одного файлу або структури бази даних в інший. Також називається «форматом обміну даними», вихідні дані перетворюються у формат обміну однією програмою, а дані у форматі обміну перетворюються у цільовий формат іншою програмою.

Формати обміну даними — це стандартизовані засоби структурування та передачі даних між системами. Вони відіграють вирішальну роль у розробці сучасного програмного забезпечення, забезпечуючи взаємодію та безперебійну інтеграцію на різноманітних платформах і середовищах програмування [26].

Правильний вибір формату обміну даними є критично важливим у багатьох аспектах розробки систем і програм, оскільки впливає на ефективність, зручність і масштабованість роботи із даними.

Деякі формати набагато краще підходять для перевірки даних, наприклад, XML зі своїми схемами, або забезпечують підвищену безпеку, як, наприклад, веб-токени JSON для авторизації. Таким чином, вибір формату може сприяти або перешкоджати зусиллям, спрямованим на те, щоб дати системі більше шансів відповідати галузевим стандартам або регуляторним вимогам [27].

Формат обміну даними може впливати на оцінку обслуговування системи. Застосовування складного формату вимагає набавних ресурсів для навчання персоналу, розробки та налагодження системи.

Формат обміну даними впливає на швидкість передачі, серіалізації (перетворення об'єкта у формат передачі) та десеріалізації (зворотне перетворення). Наприклад:

- JSON легший за XML, тому його обробка відбувається швидше у веб-додатках;
- Бінарні формати, такі як Protocol Buffers або Apache Avro, забезпечують ще більшу швидкість і компактність, але складніші у використанні.

Для цієї роботи було обрано текстовий формат для представлення структурованих даних JSON.

JavaScript Object Notation (JSON) - це стандартний текстовий формат для представлення структурованих даних на основі об'єктного синтаксису JavaScript. Він зазвичай використовується для передачі даних у веб-додатках (наприклад, для надсилання деяких даних з сервера на клієнт, щоб їх можна було відобразити на веб-сторінці, або навпаки).

JSON існує у вигляді рядка - це зручно, коли ви хочете передати дані через мережу.

JSON побудований на двох структурах:

- Колекція пар ім'я/значення. У різних мовах це реалізується як об'єкт, запис, структура, словник, хеш-таблиця, список з ключами або асоціативний масив;
- Впорядкований список значень. У більшості мов реалізується як масив, вектор, список або послідовність.

Це універсальні структури даних. Практично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Логічно, що формат даних, взаємозамінний з мовами програмування, також має базуватися на цих структурах [28].

У JSON вони набувають таких форм:

Об'єкт - це невпорядкований набір пар ім'я/значення. Об'єкт починається з лівої дужки і закінчується правою дужкою. Після кожного імені ставиться двокрапка, а пари ім'я/значення розділяються комою. Приблизна структура такого об’єкта показана на рис. 2.5.

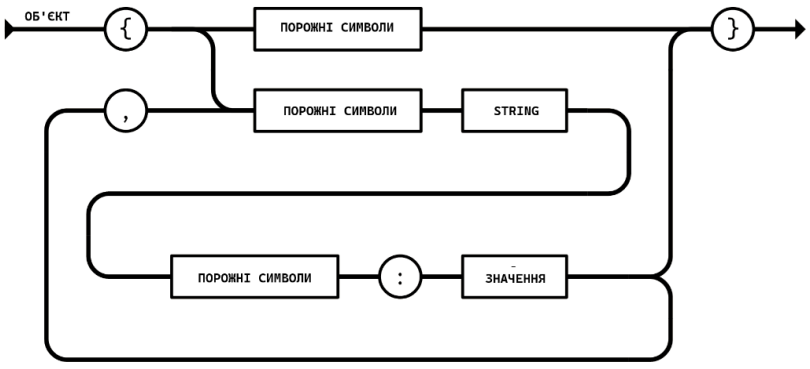


Рисунок 2.5 – Приблизна структура об’єкта JSON

Масив - це впорядкований набір значень. Масив починається з лівої дужки і закінчується правою дужкою. Значення відокремлюються комою. Приблизна структура масиву у форматі JSON показана на рисунку 2.6.

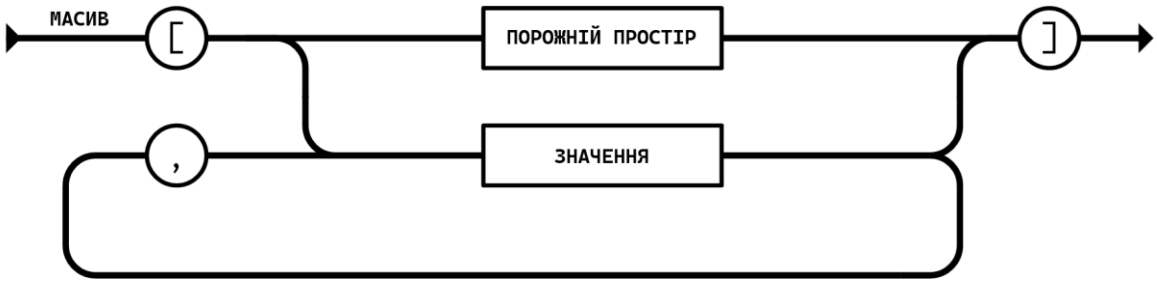


Рисунок 2.6 – Приблизна структура масиву JSON

Значення може бути рядком у подвійних лапках, числом, значенням true, false або null, об'єктом або масивом. Ці структури можуть бути вкладеними.

Можливі значення візуалізовані на рисунку 2.7.

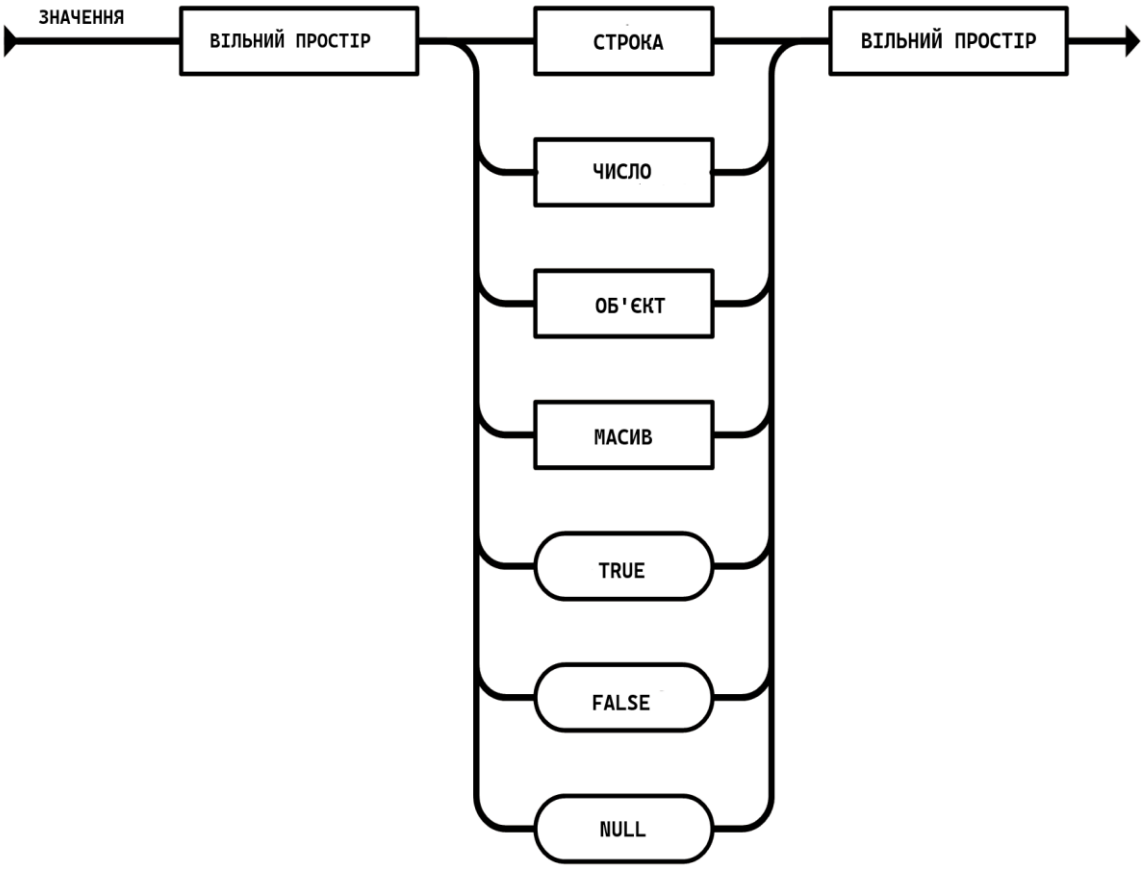


Рисунок 2.7 – Візуалізація можливих значень

Рядок - це послідовність нуля або більше символів Unicode, взятих у подвійні лапки з використанням символів зворотної косої риски. Символ представляється у вигляді рядка з одного символу. Рядок дуже схожий на рядок C або Java. Візуалізація прикладу можливих значень що можуть рахуватись рядком продемонстровано на рисунку 2.8.

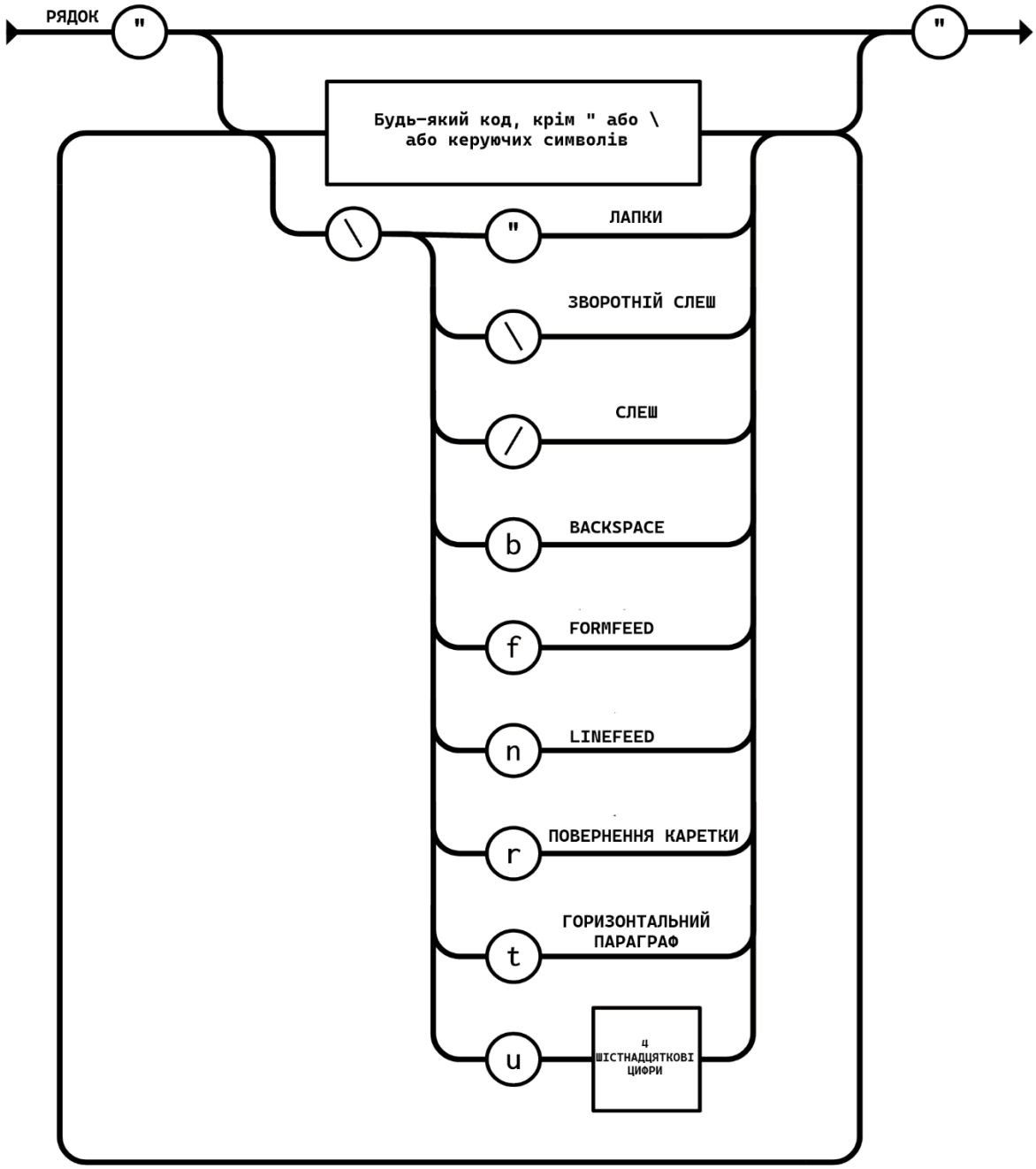


Рисунок 2.8 – Значення та відповідні токени для рядка JSON

Число дуже схоже на число в C або Java, за винятком того, що не використовуються вісімковий і шістнадцятковий формати. У форматі JSON числові дані можуть бути представлені у вигляді цілих чисел, чисел з плаваючою комою або в експоненційній нотації.

На рисунку 2.9 показано логіку, за якою JSON-парсер обробляє числові значення відповідно до стандарту JSON.

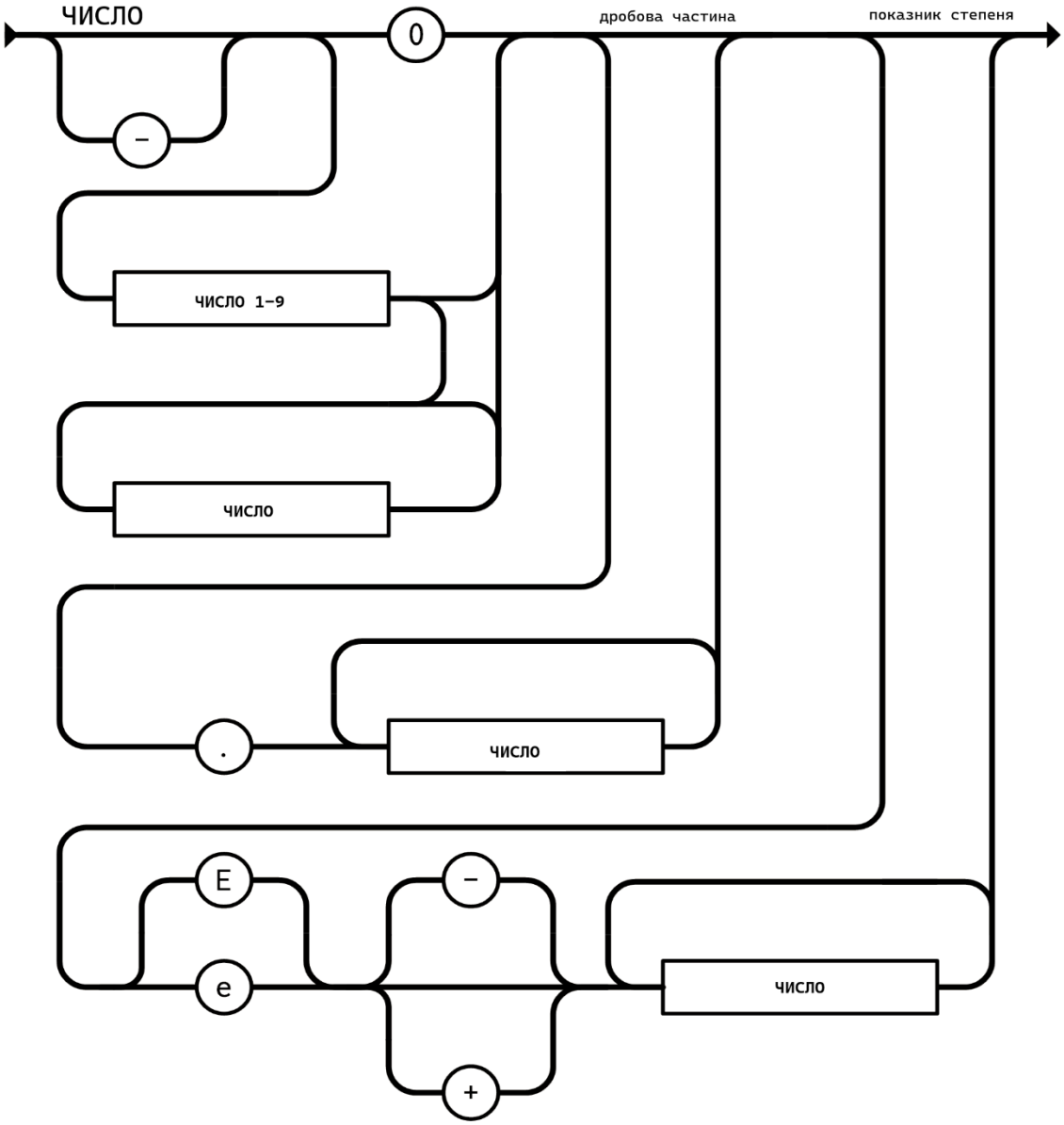


Рисунок 2.9 – Логіка обробки числових значень

Формат чисел у JSON включає наступні елементи:

- Стан "0": Початковий стан. Розпізнає число, яке починається з цифри 0 або відразу переходить до наступних допустимих символів;
- Символи число і число 1-9: Розпізнаються як будь-які допустимі цифри (від 0 до 9), причому окремо виділяються цифри від 1 до 9 для урахування можливих форматів чисел;
- Символ . (крапка): Вказує на перехід до дробових чисел, що вводить формат числа з плаваючою комою;
- Символи E або e: Позначають використання експоненційної нотації (наприклад, числа виду 1.23e+4);
- Символи + та -: Задають знаковість числа (позитивне або негативне), також можуть використовуватися в експоненційній нотації.

2.5 Інтеграція API для аналізу текстів, створених ШІ

API розшифровується як Application Programming Interface , який допомагає програмним програмам спілкуватися одна з одною для обміну даними, функціями та функціями. Дані, якими обмінюються, мають форму HTML , JSON , XML і Text [29].

API, або інтерфейс прикладного програмування, відіграє важливу роль у сучасному програмуванні та системній інтеграції. API надають можливість взаємодії між програмним забезпеченням за допомогою стандартизованого обміну даними між компонентами або сервісами.

Так, веб-API дозволяють взаємодіяти з віддаленими сервісами, такими як бази даних, аналітичні системи або сторонні платформи, не розкриваючи внутрішню логіку цих систем. Це дозволяє створювати гнучкі, масштабовані та модульні додатки, оскільки компоненти можна легко змінювати або оновлювати, не впливаючи на загальну архітектуру системи.

Крім того, API використовують вже розроблені функції - наприклад, платіжні, автентифікаційні або картографічні сервіси - таким чином

прискорюючи процес розробки чогось нового. Це економить час і ресурси, які витрачаються на розробку нових систем. API уможливлюють інтеграцію екосистем додатків по всьому світу, стимулюючи цифрову трансформацію та інновації в багатьох галузях: фінансах, охороні здоров'я, електронній комерції та інших [30].

API-запит містить наступні компоненти

- Кінцева точка: Кінцева точка API - це URL-адреса, яка надає доступ до певного ресурсу. Наприклад, кінцева точка статті в GeeksforGeeks міститиме логіку обробки всіх запитів, пов'язаних зі статтями.
- Методи: Метод запиту включає в себе тип операції, яку клієнт повинен виконати над даним ресурсом. REST API доступні за допомогою стандартних HTTP-методів, які виконують звичайні дії, такі як отримання, створення, оновлення та видалення даних.
- Параметри: Параметри - це змінні, які передаються кінцевій точці API для надання конкретних інструкцій для обробки API. Ці параметри можуть бути включені в запит до API як частина URL-адреси, в рядок запиту або в тіло запиту. Наприклад, кінцева точка /articles API для блогів може прийняти параметр «topic», який буде використовуватися для доступу і повернення статей на певну тему.
- Заголовки запитів : Заголовки запиту - це пари ключ-значення, які надають додаткову інформацію про запит, наприклад, тип вмісту або облікові дані для автентифікації.
- Тіло запиту: Тіло - це основна частина запиту, яка містить фактичні дані, необхідні для створення, оновлення або видалення ресурсу. Наприклад, якщо ви створюєте нову статтю – тіло запиту, ймовірно, міститиме вміст статті, її назву та автора.

В даній роботі використання API зумовлено нераціональною кількістю фінансових та інших ресурсів які потрібно використати для навчання власної нейронної мережі призначеною для пошуку фрагментів тексту створених за

допомого штучного інтелекту. Оскільки для реалізації програмного забезпечення що розкриває тему технологія ШІ в створенні інструмента для оцінки оригінальності текстів потрібно впровадити систему пошуку ШІ, але розробка даної системи не є раціональною – було знайдено відповідний ресурс, який виконую поставлену задачу. Підключення цієї системи до написаної програми, що має слугувати прикладом подібних систем, втілено з використанням Application Programming Interface.

Висновки до другого розділу

У вищеописаному розділі було проведено роботу по проектуванню програмної реалізації розробленого продукту, мета якого продемонструвати важливість перевірки будь-яких академічних робіт як на наявність плагіату так і фрагментів створеним не конкретно учасником наукового процесу, а розвинутими електронними помічниками на базі штучного інтелекту.

Завдяки аналізу різноманітних аспектів було остаточно обрано як мову програмування на якій буде написано проект так і середовище розробки. Оскільки було наведено коректний опис та переваги зазначених аспектів можна наочно зрозуміти результати прийнятих рішень.

Було послідовно та детально описано алгоритм який використовується для вирішення проблеми виявлення плагіату в розробленому продукті та наведено можливості впровадження його в дану роботу.

Також в розділі було описано методи впровадження системи для визначення фрагментів тесту написаних за допомогою штучного інтелекту та спосіб і формат зберігання доданих робіт для подальшого їх використання в майбутніх аналізах.

Проведений аналіз та обраний формат зберігання файлів що будуть використовуватись для збереження робіт та порівняння їх між собою для подальшого виявлення плагіату.

Маємо можливість зазначити що в цьому розділі було виділено достатню кількість уваги для розкриття та пояснення як вибору інструментів, що будуть

використані в роботі, так і алгоритмам й рішенням на яких ґрунтується програмна частина реалізація поставлених завдань.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ

3.1. Підключення бази даних

Спочатку варто зазначити, що для збереження даних в проєкті використовується база даних SQLite, початок роботи полягає в створені самої бази даних, для цього використаємо DB Browser (SQLite). На рисунку 3.1 зображено інтерфейс програми.

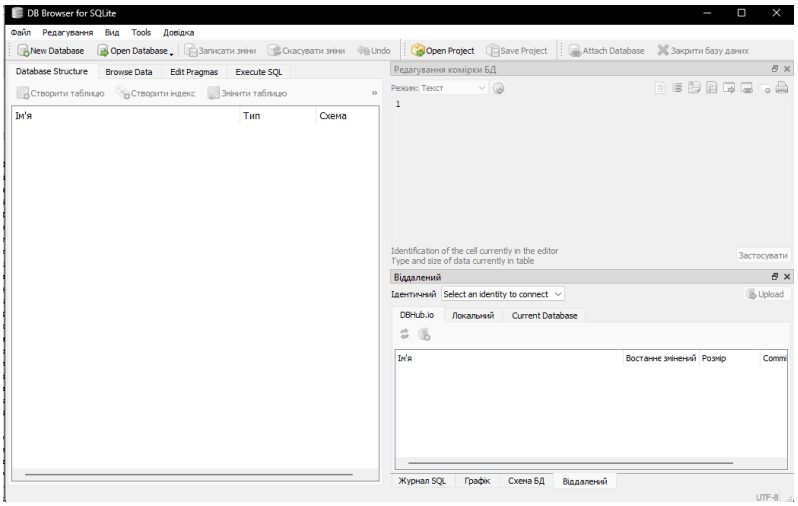


Рисунок 3.1 – Інтерфейс програми DB Browser (SQLite)

З минулих розділів, а саме «2.2.2 Вибір середовища розробки», зазначено що для роботи обране середовище розробки Visual Studio, інтерфейс якої показано на рисунку 2.4. По цій причині в описах дій впродовж всього розділу будуть зустрічатися елементи пов’язані з технічними особливостями конкретно виду середовища. Так для прикладу на рисунку 3.2 зображено процес створення проекту, який є обов’язковим і ключовим моментом створення програм в показаній системі.

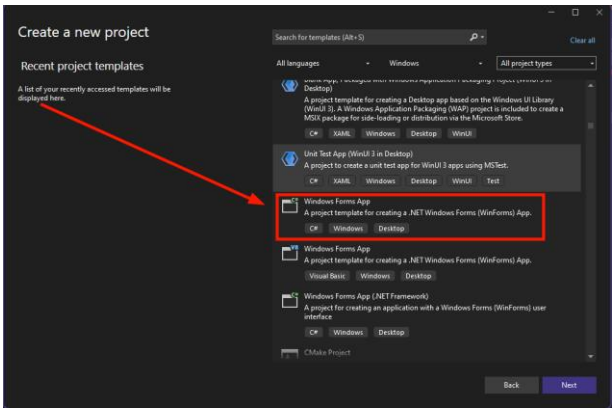


Рисунок 3.2 – Створення проекту

Створюємо саму базу, продемонстровану на рисунку 3.3, даних в якій будуть зберігатись роботи для аналізу. Оскільки вона призначення для збереження текстів в відформатованому форматі обміну даними JSON в ній ми створюємо лише три поля, а саме:

- Id формату integer для запису ідентифікатора роботи;
- name формату text для зберігання назви документа, зазвичай це ПІБ автора та назва роботи;
- JSON формату text для збереження самих робіт.

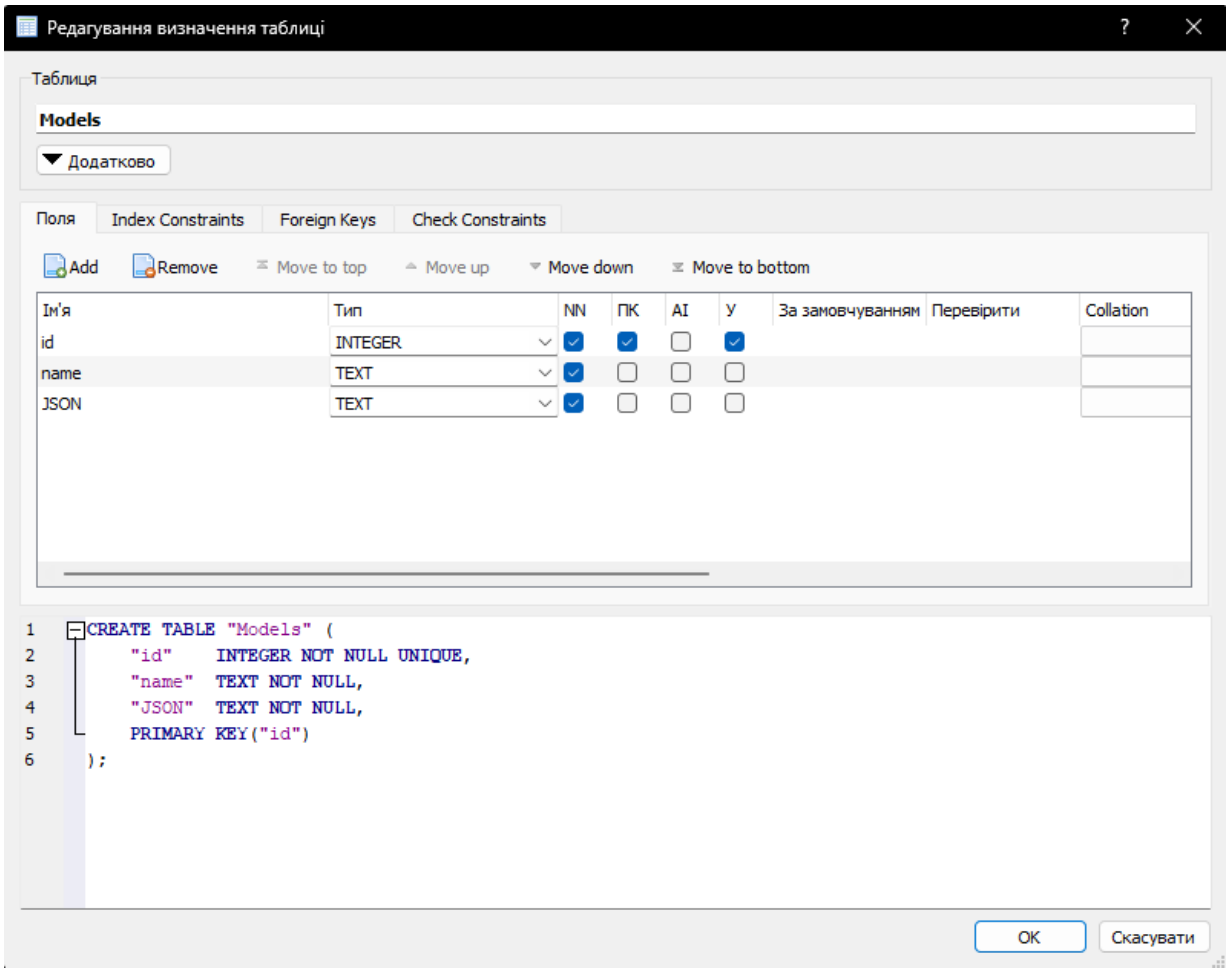


Рисунок 3.3 – Створена база даних

Тепер для того аби мати можливість використовувати базу даних, мається на увазі записувати дані в середину та зчитувати їх для подальшого використання, в самій розробленій програмі нам необхідно провести деякі маніпуляції з проектом.

Конкретніше додати пакет SQLite, використовуючи вбудований інструмент обраного середовища, який носить назву «NuGet», що надасть в майбутньому можливість імпортувати створену базу. Процес додавання пакетів на рисунку 3.4.

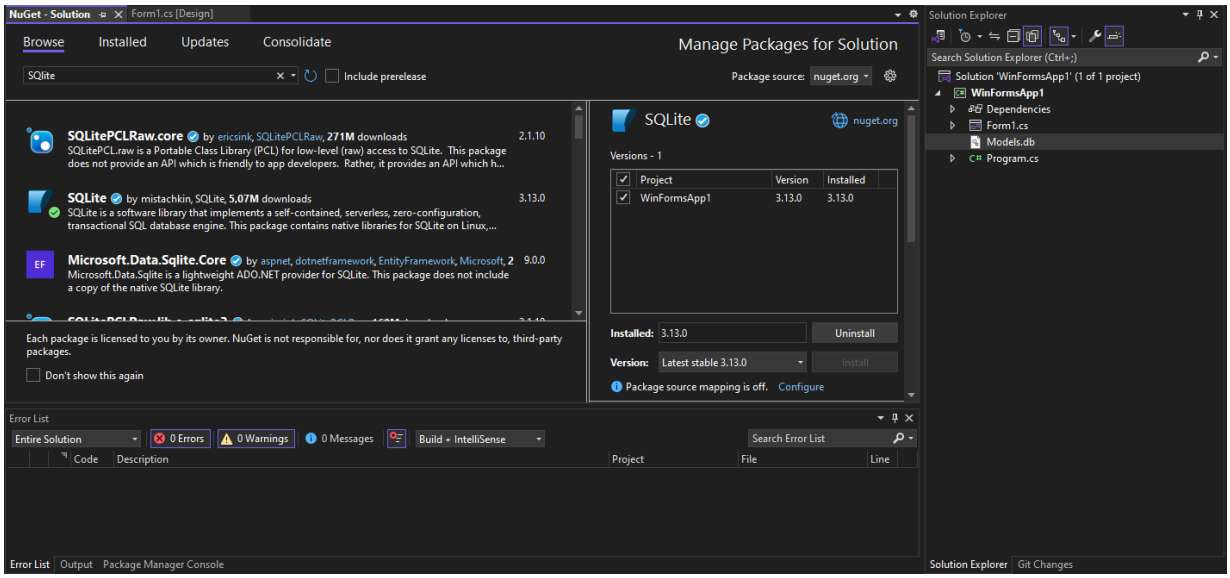


Рисунок 3.4 – Додавання потрібного пакета

Тепер, коли усі обов'язкові етапи налаштування пройдені ми можемо перейти до підключення БД до проекту, шляхом створення відповідного програмного коду. Код показано на рисунку 3.5. В цій частині ми повинні додати відповідний простір імен та створити рядок підключення БД.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;

namespace WinFormsApp1
{
    0 references
    internal class BD
    {
        string connectionString = "Data Source=database.sqlite;Version=3;";
    }
}
```

Рисунок 3.5 – Підключення БД до проєкту

Додамо на головну форму сторінки невеликий тул, що призначений для того щоб користувач міг побачити існуючі в базу даних записи. Приблизний вигляд показано на рис. 3.6.

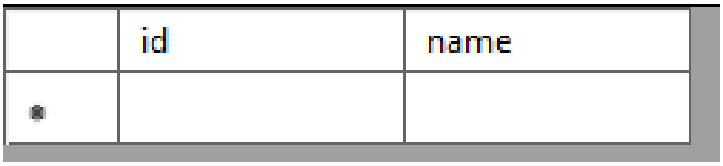


Рисунок 3.6 – Приблизний вигляд записів БД

3.2 Зчитування docX файлу

Для зручності використання застосунку клієнтом потрібно реалізувати систему яка зможе надати можливість зручно обирати файл формату docX і зчитувати наведену в ньому інформацію в змінну типу string для подальших маніпуляцій.

Для більш ефективного та якісної реалізації цього аспекту обрано бібліотеку під назвою «DocX», момент додавання її до проекту показаний на рисунку 3.7.

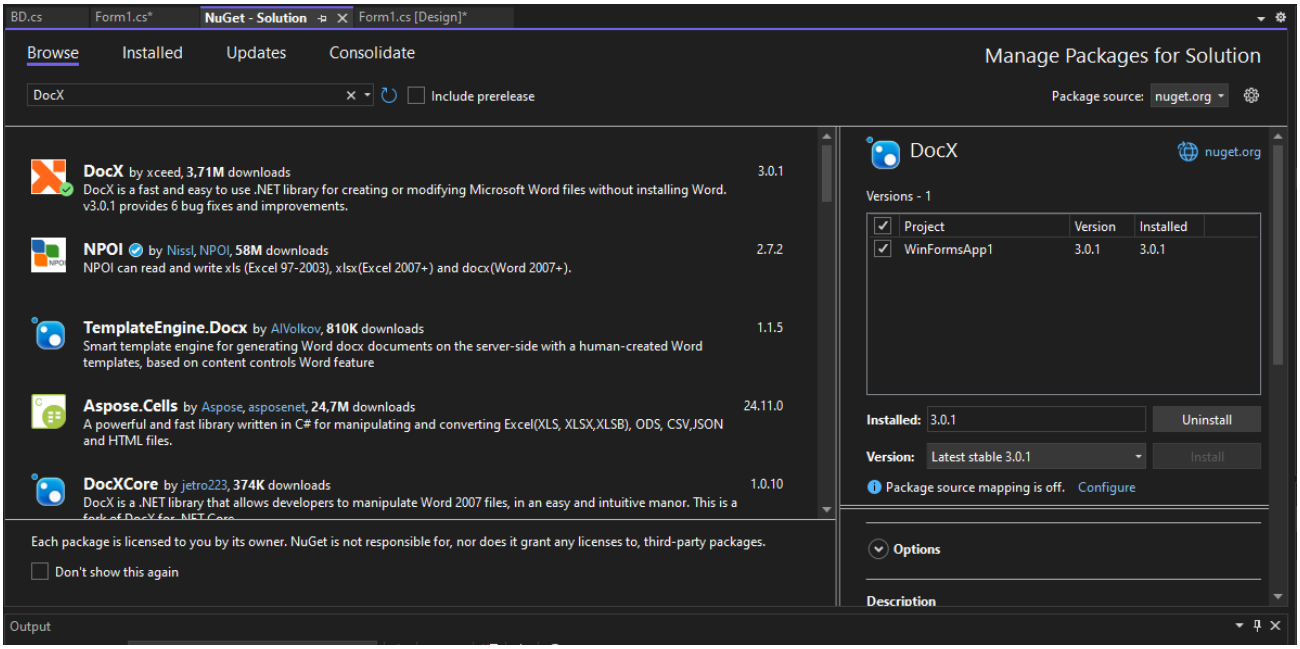


Рисунок 3.7 – Додавання бібліотеки DocX

Наступним кроком являється реалізація програмної частини, що відповідає за зчитування файлу та його запису. Для цього напишемо функцію ReadDocFile, яка прийматиме обрані документ та проведе вище зазначені дії. Скріншот функції ReadDocFile показано на рисунку 3.8.

```
private void ReadDocFile(string filePath)
{
    try
    {
        // Відкриваємо документ
        using (var document = DocX.Load(filePath))
        {
            // Отримуємо весь текст документа
            string content = document.Text;

            // Виводимо в консоль або записуємо в змінну
            MessageBox.Show(content);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка: {ex.Message}");
    }
}
```

Рисунок 3.8 – Функція ReadDocFile

Для користувача створимо візуальний інтерфейс у вигляді клавіши «Вибрати документ для аналізу» яку можна побачити на рисунку 3.9.

Рисунок 3.9 – Клавіша Вибрати документ для аналізу

Функціоналом для цієї клавіши буде окрема функція, яка при натисканні на клавішу викликає подію, що відкриває вікно обрання файлу(рис 3.9).

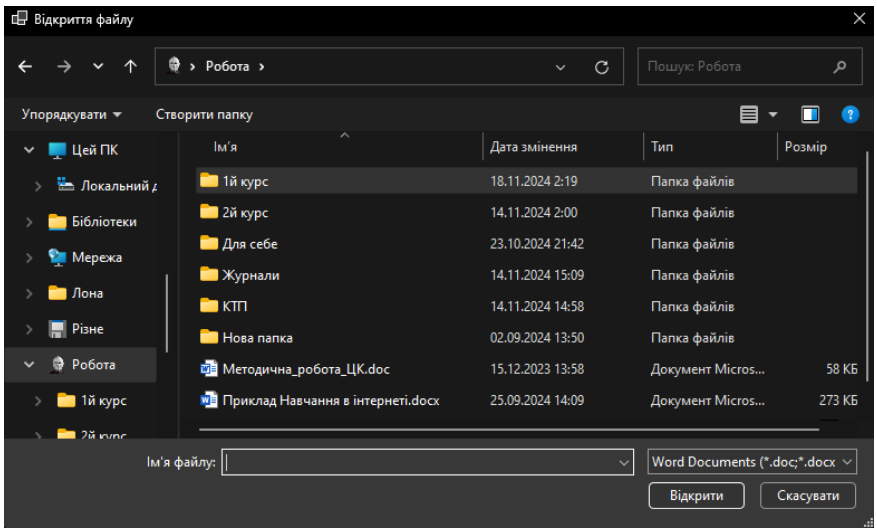


Рисунок 3.9 – Скріншот коду Вікно відкриття файлу

3.3 Робота з JSON – конвертація, запис та зчитування

Для збереження текстів наукових робіт в БД, потрібно перевести string в формат JSON. Для цього до проекту потрібно підключити бібліотеку Newtonsoft.Json та розробити новий метод, який буде виконувати вище зазначені задачі.(рис. 3.10).

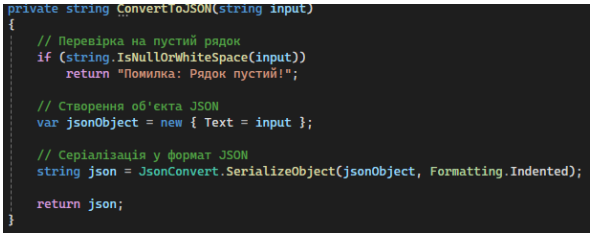


Рисунок 3.10 – Скріншот коду перетворення в JSON

Також для порівняння робіт між собою неможливо обійтись без можливості функціонального вирішення питання зчитування файлів з бази даних наукових робіт призначених для перевірки. Реалізуємо цю задачу за допомогою функції, що повинна брати інформацію з створеної бази даних та почергово записує цю інформацію в окремий масив даних для подальшого зручного використання при порівнянні текстів.

Останнім, але не менш важливим аспектом роботи з JASON, є можливість запису створених об’єктів в розроблену базу даних, цей процес має автоматично проводитись при закінченні аналізу й порівняння обраної роботи.

Такий підхід забезпечить те, що при перевірці наступної роботи, які частіше за все перевіряються одразу великим набором, робота що перевіряється в даний момент буде використовуватись для порівняння, при чому не змусить користувача напружено контролювати цей процес і покращить досвід роботи з створеним проектом.

3.4 Реалізація методу косинусної подібності

Як зазначено у розділі 2 «Проектування програмної реалізації» для рішення задачі пошуку плагіату є метод косинусної подібності. Це означає що даний метод буде реалізований в розробленій демонстративній програмі, а алгоритм аналізу тексту був розглянутий в розділ 2. Побудова цього алгоритму в мові C# показана в додаткуА.

Так як для виконання цієї частини роботи в деякій мірі буде використовуватись бібліотека RegularExpressions одразу додамо її до проекту використовуючи відповідні інструменти. Додавання бібліотеки зображено на рисунку 3.11.

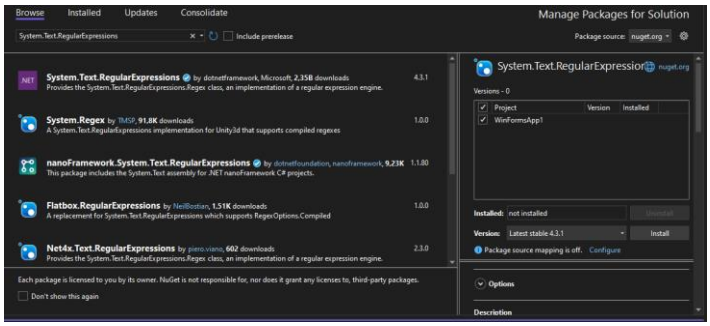


Рисунок 3.11 – Додання відповідної бібліотеки

Виконавши програмну частину, що відповідає нашому завданню цього пункту розділу, проведемо тестування і подивимось на косинус кута двох тестових фраз що в майбутньому буде перетворений у відсоткове значення.

Тестова фраза номер 1 – «This is a sample document.», тестова фраза 2 – «This document is a sample.». Результат показано на рис 3.12.Тобто ми отримали плагіат тестової фрази.- косинусної подібності дорів1

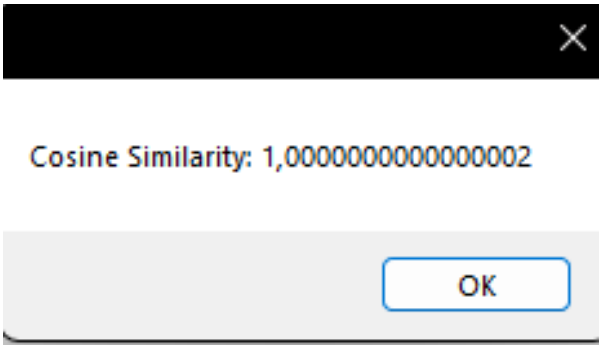


Рисунок 3.12 – Результат розрахунків

3.5 Використання API

Напишемо код, який розділяє текст на частини за кількістю слів і перевіряє ці частини через API сервісу Grammarly для аналізу текстів і виводить результати.

На початку текст розбивається на частини залежно від заданої мінімальної та максимальної кількості слів. Це здійснюється методом SplitTextIntoParts, який працює з масивом слів і створює рядки заданої довжини, об'єднуючи їх назад у частини тексту.

Потім програма створює HTTP-запит для доступу до API Grammarly. Вона починає з відправки GET-запиту для отримання початкової сторінки, де зберігаються необхідні cookie-файли. Ці файли потрібні для авторизації подальших запитів.

Далі кожна частина тексту, отримана після розділення, відправляється через POST-запити до API Grammarly, використовуючи збережені cookie-файли. Результати обробки кожної частини тексту виводяться на екран. У разі виникнення помилок при відправленні запитів програма виводить повідомлення з описом проблеми.

Таким чином, на рисунку 3.13 можна побачити результат відповіді на запит для тестової фрази.

. по результатам можна побачити що до нас повертаються значення аналізу на ШІ у JSON форматі. Значення cooоо=0 говорить про те, що ші не був використаний. =

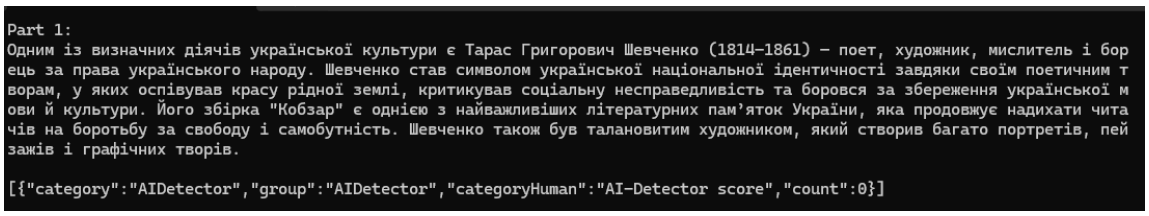


Рисунок 3.13 – Відповідь сайту

3.6 Висновки до третього розділу

В цьому розділі було продемонстровано програмну реалізацію продукту який повинен демонструвати поєднання технологій штучного інтелекту з існуючими методами виявлення плагіату та допомогти в створенні потужного інструмента для оцінки оригінальності текстів.

Було показано та коротко описано яким чином було реалізовано такі аспекти програми як:

- Підключили базу даних, що дозволить нам зберігати роботи студентів призначені для пошуку плагіату. Кожна робота в цій базі покращує результати аналізу для наступних перевірок;
- Зчитування docX файлів, що надасть можливість спростити використання цього продукту користувачами та зекономи час завдяки зручному та інтуїтивно зрозумілому способу додавання робіт в програму;
- Робота з JSON – конвертація, запис та зчитування JSON файлів дасть можливість краще працювати з базою даних та полегшить впровадження технологій API які використовують даний формат для надання результатів роботи.
- Реалізація методу косинусної подібності створить надійний інструмент виявлення недоброчесного копіювання змісту чужих робіт і є важливим аспектом цієї роботи;
- Використання API замінить дороге та ресурсне навчання власної нейронної мережі для пошуку тексту написаним штучним інтелектом.

ВИСНОВКИ

Дипломна робота Розробка програмного забезпечення для технологій обробки тексту представляє собою комплексне дослідження, спрямоване на розроблення програмного продукту, здатного ефективно виявляти плагіат і аналізувати тексти, створені за допомогою штучного інтелекту. Основні результати роботи можна розділити на три взаємопов’язані аспекти: аналітичний, проектувальний і практичний, кожен з яких зробив вагомий внесок у досягнення поставленої мети.

На першому етапі було здійснено ґрунтовний аналіз предметної області. Розглянуто поняття штучного інтелекту, зокрема його архітектуру, функціонування нейронних мереж та їх застосування для аналізу текстів. Увага приділялася технічним характеристикам і можливостям сучасних моделей ШІ, а також аспектам їхньої інтеграції у процес перевірки текстів. Також висвітлено питання академічної доброчесності, включаючи визначення плагіату, його види та культурні особливості сприйняття цього явища. Аналіз було проведено з акцентом на важливість збереження академічних стандартів і підтримки чесності у процесах навчання й наукової діяльності. Таким чином, створено теоретичний фундамент для розроблення інструмента, який здатен враховувати як технічні, так і етичні аспекти.

У другому розділі було зосереджено увагу на проектуванні програмної реалізації. Вибір мови C# та середовища Visual Studio обґрунтовано їхньою придатністю для створення надійних додатків із розширеними можливостями інтеграції та оптимізацією процесів розроблення. Докладно розглянуто структуру програми, її алгоритми, а також методи інтеграції з API для аналізу текстів. Значна увага приділена формату зберігання даних: вибір JSON-файлів забезпечує зручність і швидкість обробки, а також полегшує використання програми для майбутніх потреб. Розроблено алгоритм косинусної подібності, який став основою для виявлення схожості текстів. Усе це дозволило створити чітку та логічну архітектуру продукту, яка забезпечує його функціональність та ефективність.

Практичний етап роботи включав реалізацію функціонального прототипу програми. Основна увага була спрямована на інтеграцію таких функцій, як зчитування текстових файлів, автоматичне видалення зображень, обробка даних у форматі JSON, а також робота з базою даних SQLite для зберігання текстів. Було впроваджено метод косинусної подібності для аналізу схожості текстів, а також використано API для розпізнавання фрагментів, створених штучним інтелектом. Результатом стало створення потужного інструмента, який допомагає не лише виявляти плагіат, але й сприяє збереженню академічної доброчесності.

Таким чином, проведене дослідження та практична реалізація довели, що інтеграція технологій штучного інтелекту у процес перевірки текстів на оригінальність є перспективним напрямом. Розроблений інструмент відкриває нові можливості для боротьби з академічною не доброчесністю, сприяє автоматизації рутинних процесів і підтримує етичні стандарти у сфері освіти та науки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Greg Deckler, Learn Power BI - Second Edition: A comprehensive, step-by-step guide for beginners to learn real-world business intelligence: Model – 8 с.
2. Greg Deckler, Learn Power BI - Second Edition: A comprehensive, step-by-step guide for beginners to learn real-world business intelligence: Analysis – 10-11 с.
3. David Foster Generative Deep Learning: Teaching Machines To Paint, Write, Compose, and Play: Generative Modeling and AI – 8 с.
4. Jakub Langr, Vladimir Bok, GANs in Action: Deep learning with Generative Adversarial Networks: Introduction to GANs – 3-16 с.
5. IBM [Електронний ресурс] — Режим доступу URL: https://www.ibm.com/think/topics/variational-autoencoder
6. Noureddine Aboutabit, Mohamed Lazaar, Imad Hafidi, Advances in Machine Intelligence and Computer Science Applications: Proceedings of the International Conference ICMICSA’2022 (Lecture Notes in Networks and Systems): Unsupervising Denoising Model Based Generative Adversarial Networks – 36 с.
7. James Phoenix, Mike Taylor, Prompt Engineering for Generative AI: Future-Proof Inputs for Reliable AI Outputs: OpenAI`s Generative Pretrained Transformers – 48 с.
8. Medium [Електронний ресурс] — Режим доступу URL: https://medium.com/@aarunsoman/hybrid-ai-architectures-integrating-agents-and-pipelines-for-advanced-llm-applications-59a69acd90f6
9. Simon J.D, Prince, Understanding Deep Learning: Unsupervised learning – 7-10 с.
10. Simon J.D, Prince, Understanding Deep Learning: Reinforcement learning – 11 с.
11. AWS [Електронний ресурс] — Режим доступу URL: https://aws.amazon.com/what-is/neural-network/
12. Medium [Електронний ресурс] — Режим доступу URL: https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec

13. MarketMuse [Електронний ресурс] — Режим доступу URL: https://blog.marketmuse.com/glossary/multilayer-percpetron-definition/
14. Pabloinsente [Електронний ресурс] — Режим доступу URL: https://pabloinsente.github.io/the-adaline
15. Tutorialspoint [Електронний ресурс] — Режим доступу URL: https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_adaptive_resonance_theory.htm
16. Philadelphia [Електронний ресурс] — Режим доступу URL: https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/Lecture%2015_Self-Organizing%20Maps%20(Kohonen%20Maps).pdf
17. Geeksforgeeks [Електронний ресурс] — Режим доступу URL: https://www.geeksforgeeks.org/hopfield-neural-network/
18. Geeksforgeeks [Електронний ресурс] — Режим доступу URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
19. University College Oxford [Електронний ресурс] — Режим доступу URL: https://www.univ.ox.ac.uk/wp-content/uploads/2017/10/Plagiarism-and-Academic-Integrity.pdf
20. Bryan H. Smalls, Investigating William Branham: The Unfolding Story of Plagiarisms and Errors: The Cloud`s Origin: Natural or Supernatural? – 16-18 c.
21. Beetroot [Електронний ресурс] — Режим доступу URL: https://beetroot.academy/blog/shcho-take-c-chi-pidhodit-meni-cya-mova-programuvannya-chomu-vona-kruta
22. Steven F Lott, Dusty Phillips, Python Object-Oriented Programming - Fourth Edition: Build robust and maintainable object-oriented Python applications and libraries: Introducing object-oriented – 2 c.
23. Visual Studio [Електронний ресурс] — Режим доступу URL: https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLandingPage&cid=2030&passive=false

24. Medium [Електронний ресурс] — Режим доступу URL: https://medium.com/@arjunprakash027/understanding-cosine-similarity-a-key-concept-in-data-science-72a0fcc57599
25. Medium [Електронний ресурс] — Режим доступу URL: https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db
26. Tom Marrs, JSON at Work: Practical Data Integration for the Web: JSON Is a Standart? – 3 с.
27. Tom Marrs, JSON at Work: Practical Data Integration for the Web: Why JSON? – 3 с.
28. Tom Marrs, JSON at Work: Practical Data Integration for the Web: Core JSON – 8с
29. James Gough, Daniel Bryant, Matthew Auburn, Mastering API Architecture: Design, Operate, and Evolve API-Based Systems: Case Study: Designing the Attendee API – 12 с.
30. James Gough, Daniel Bryant, Matthew Auburn, Mastering API Architecture: Design, Operate, and Evolve API-Based Systems: ADR Guideline: Choosing an API Standard – 12 с.

ДОДАТОК А. КОДИ ФАЙЛІВ ПРОГРАМИ

1.Код файлу основної форми:

```
using System.Data;
using Xceed.Words.NET;
using Newtonsoft.Json;
namespace WinFormsApp1
{
    public partial class Form1 : Form
    {
        public static string DocumentContent;
        public static string UserDocumentName;
        public static string UserDocumentAutor;
        public static string UserDocumentInstitution;
        public Form1()
        {
            InitializeComponent();
        }
        public string CleanText(string text)
        {
            // Видаляємо табуляцію та зайві нові рядки
            string cleanedText = text.Replace("\t", " ") // Заміна
табуляції на пробіл
                                     .Replace("\r", " ") // Видалення
символів перенесення каретки
                                     .Replace("\n", " ") // Видалення
нових рядків
                                     .Replace("\f", " ") // Видалення
символів розриву сторінок
                                     .Trim();              // Видалення
зайвих пробілів на початку і в кінці
            // Видаляємо всі символи, які не є друкованими
            cleanedText                                              =
System.Text.RegularExpressions.Regex.Replace(text, @"[^\u0020-\u007E]+",
" ").Trim();
```

```
            // Заміна множинних пробілів одним пробілом
            cleanedText                                                    =
System.Text.RegularExpressions.Regex.Replace(cleanedText, @"\s+", " ");
            return cleanedText;
        }
        private void ReadDocFile(string filePath)
        {
            try
            {
                // Відкриваємо документ
                using (var document = DocX.Load(filePath))
                {
                    // Отримуємо весь текст документа
                    DocumentContent = document.Text;
                    DocumentContent = CleanText(DocumentContent);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Помилка: {ex.Message}");
            }
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            DataTable dataTable = BD.LoadData();
            if (dataTable == null)
            {
                return;
            }
            dataGridView1.DataSource = dataTable;
        }
        private string ConvertToJSON(string input)
        {
            // Перевірка на пустий рядок
```

```
            if (string.IsNullOrWhiteSpace(input))
                return "Помилка: Рядок пустий!";
            // Створення об'єкта JSON
            var jsonObject = new { Text = input };
            // Серіалізація у формат JSON
            string   json   =   JsonConvert.SerializeObject(jsonObject,
Formatting.Indented);
            return json;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Відкрити діалогове вікно для вибору файлу
            OpenFileDialog openFileDialog = new OpenFileDialog
            {
                Filter = "Word Documents (*.doc;*.docx)|*.doc;*.docx"
            };
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                ReadDocFile(openFileDialog.FileName);
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            UserDocumentName = DocName.Text;
            UserDocumentAutor = Autor.Text;
            UserDocumentInstitution = Institution.Text;

            if  (DocumentContent!=null  &&  UserDocumentName  !=  null  &&
UserDocumentAutor != null && UserDocumentInstitution != null)
            {
                using (Form2 dialog = new Form2())
                {
                    if (dialog.ShowDialog() == DialogResult.OK)
                    {
```

```
                        BD.AddToBD();
                        DataTable dataTable = BD.LoadData();
                        if (dataTable == null)
                        {
                            return;
                        }
                        dataGridView1.DataSource = dataTable;
                    }
                }
            }
            else
            {
                MessageBox.Show("Існують незаповнені поля або не обраний
документ");
            }
        }
        private void label2_Click(object sender, EventArgs e)
        {
        }
        private void label3_Click(object sender, EventArgs e)
        {
        }
        private void button3_Click(object sender, EventArgs e)
        {
            UserDocumentName = DocName.Text;
            UserDocumentAutor = Autor.Text;
            UserDocumentInstitution = Institution.Text;
            BD.AddToBD();
            DataTable dataTable = BD.LoadData();
            if (dataTable == null)
            {
                return;
            }
            dataGridView1.DataSource = dataTable;
```

```
        }
    }
}
```

2. Код файлу модального вікна з результатами аналізу:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WinFormsApp1
{
    public partial class Form2 : Form
    {
        public static string SecondPlagiatText;
        public static List<BadResult> BadResults;
        public static double pidlaKraga = -100;
        public static string pidluyAI;
        public static string Perevirka;
        public Form2()
        {
            InitializeComponent();
            BadResults = new List<BadResult>();
        }
        void ResultPlagiat()
        {
            BD.ReadUsersText();
            richTextBox1.Text = "";
            foreach (BadResult item in BadResults)
```

```
            {
                richTextBox1.Text         +=         $"{item.PlagiatName}
{item.PlagiatAutor} {item.PlagoatProcent} {item.AIprocent} \n";
            }
            foreach (BadResult item in BadResults)
            {
                if (item.PlagoatProcent > pidlaKraga)
                {
                    pidlaKraga = item.PlagoatProcent;
                }
            }
            label2.Text = $"{pidlaKraga}%";
        }
        private async void Form2_Load(object sender, EventArgs e)
        {
            await AI_Detect.PoshucAI();
            ResultPlagiat();
            label4.Text = pidluyAI + "%";
        }
        private void label4_Click(object sender, EventArgs e)
        {
        }
    }
}
```

3. Код окремого класу, що відповідає за зв'язок з базою даних:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SQLite;
using System.Data;
using System.Windows.Forms;
```

```
using Newtonsoft.Json;
using System.Data.SqlClient;
namespace WinFormsApp1
{
    public class BD
    {
        // Шлях до вашої бази даних
        static string connectionString = "Data Source = Models.db;
Version=3;";
        public static DataTable LoadData()
        {
            using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
            {
                try
                {
                    connection.Open();
                    string query = "SELECT id, name, autor, Institution,
ResultPlagiat, ResultAI FROM Models"; // SQL-запит для вибірки даних
                    SQLiteDataAdapter adapter = new
SQLiteDataAdapter(query, connection);
                    DataTable dataTable = new DataTable();
                    adapter.Fill(dataTable);
                    // Заповнення DataGridView
                    return dataTable;
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                    return null;
                }
            }
            static void JSONread()
            {
```

```
                string connectionString = "YourConnectionStringHere";
                Dictionary<string, string> dictionary = new
Dictionary<string, string>();

                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    string query = "SELECT data FROM JsonDataStore";
                    using (SqlCommand command = new SqlCommand(query,
connection))
                    {
                        using (SqlDataReader reader =
command.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                string jsonData =
reader["data"].ToString();
                                dictionary =
JsonConvert.DeserializeObject<Dictionary<string, string>>(jsonData);
                            }
                        }
                    }
                }
                // Output the retrieved dictionary
                foreach (var kvp in dictionary)
                {
                    Console.WriteLine($"{kvp.Key}: {kvp.Value}");
                }
            }
        }
        public static void AddToBD()
        {
```

```
            string addData = "INSERT INTO Models (name, autor, JSON, Institution, ResultPlagiat, ResultAI) VALUES ( @param1, @param2, @param3, @param4, @param5, @param6 )"; // SQL-запит для вибірки даних
            try
            {
                using (var con = new SQLiteConnection(connectionString))
                {
                    con.Open();
                    using (var comand = new SQLiteCommand(addData, con))
                    {
                        comand.Parameters.Add(new SQLiteParameter("@param1", Form1.UserDocumentName));
                        comand.Parameters.Add(new SQLiteParameter("@param2", Form1.UserDocumentAutor));
                        comand.Parameters.Add(new SQLiteParameter("@param3", Form1.DocumentContent));
                        comand.Parameters.Add(new SQLiteParameter("@param4", Form1.UserDocumentInstitution));
                        comand.Parameters.Add(new SQLiteParameter("@param5", Form2.pidlaKraga));
                        comand.Parameters.Add(new SQLiteParameter("@param6", Form2.pidluyAI));
                        comand.ExecuteNonQuery();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
        public static void ReadUsersText()
        {
```

```
            using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
            {
                connection.Open();
                string query = "SELECT name, JSON, autor FROM Models";
                using (SQLiteCommand command = new SQLiteCommand(query,
connection))
                {
                    using (SQLiteDataReader reader =
command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            string jsonData = reader["JSON"].ToString();
                            string name = reader["name"].ToString();
                            string autor = reader["autor"].ToString();
                            Main_app_code.CosMetod(jsonData, name,
autor);
                        }
                    }
                }
            }
        }
    }
}
```

4. Код окремого класу, що відповідає за розрахунок косинусної подібності:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.RegularExpressions;
namespace WinFormsApp1
```

```
{
    internal class Main_app_code
    {
        public static double ComputeCosineSimilarity(Dictionary<string, int> vector1, Dictionary<string, int> vector2)
        {
            IEnumerable<string> commonTerms = vector1.Keys.Intersect(vector2.Keys);
            double dotProduct = commonTerms.Sum(term => vector1[term] * vector2[term]);
            double magnitude1 = Math.Sqrt(vector1.Values.Sum(val => val * val));
            double magnitude2 = Math.Sqrt(vector2.Values.Sum(val => val * val));
            return dotProduct / (magnitude1 * magnitude2);
        }
        public static Dictionary<string, int> TextToVector(string text)
        {
            string[] terms = Regex.Split(text.ToLower(), @"\W+");
            Dictionary<string, int> termFrequency = new Dictionary<string, int>();
            foreach (string term in terms)
            {
                if (termFrequency.ContainsKey(term))
                    termFrequency[term]++;
                else
                    termFrequency[term] = 1;
            }
            return termFrequency;
        }
        public static void CosMetod(string newText, string name, string autor)
        {
            string text1 = Form1.DocumentContent;
```

```
            Dictionary<string, int> vector1 = TextToVector(text1);
            Dictionary<string, int> vector2 = TextToVector(newText);
            double    similarity    =    ComputeCosineSimilarity(vector1,
vector2);
            similarity -= 0.3;
            if (similarity < 0)
                similarity = 0;
            similarity = (similarity / 0.7) * 100;
            similarity = double.Round(similarity, 2);
            {
                Form2.BadResults.Add(new      BadResult(name,      autor,
similarity, 0));
            }
        }
    }
}
```

5. Код окремого класу, що відповідає за аналіз тексту на наявність фрагментів створених ШІ:

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;
namespace WinFormsApp1
{
    public class Root
    {
        public string category { get; set; }
        public string group { get; set; }
        public string categoryHuman { get; set; }
        public int count { get; set; }
    }
```

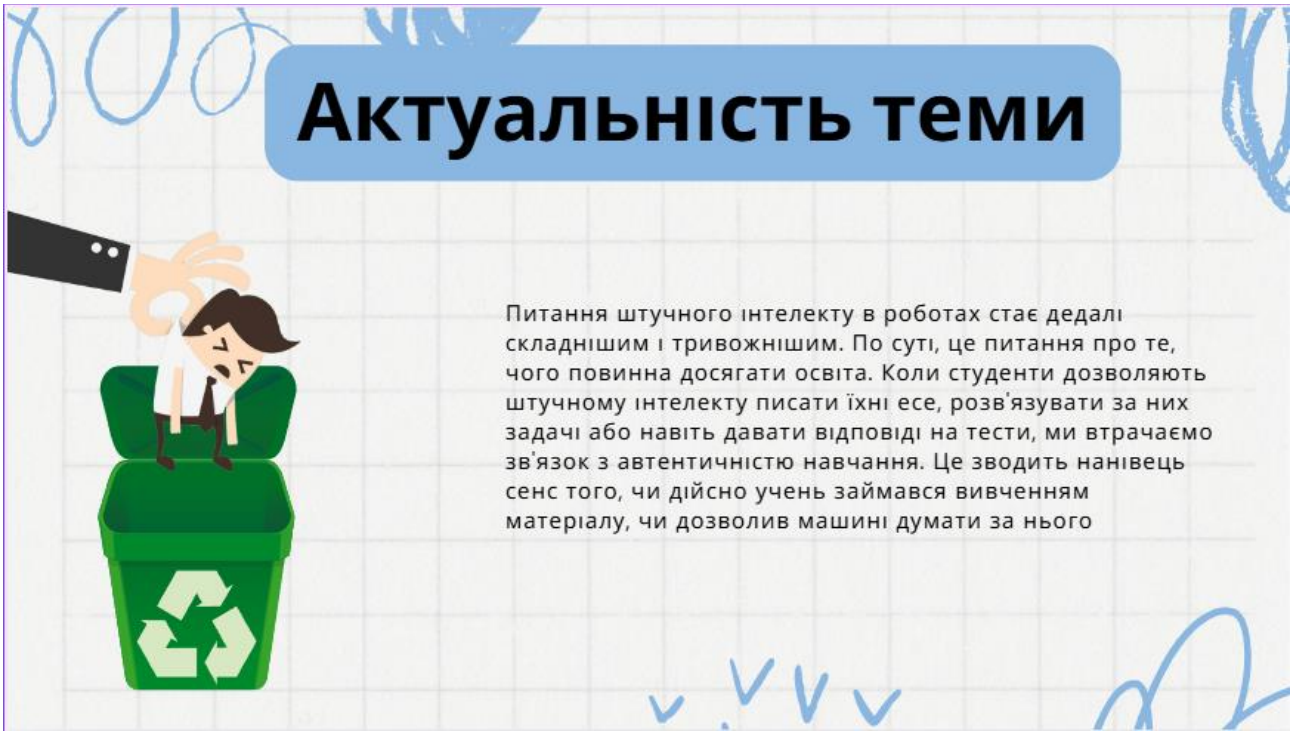
```
    public class AI_Detect
    {
        public static async Task PoshucAI()
        {
            string text = Form1.DocumentContent;
            int maxWords = 1500;
            int minWords = 100;
            List<string> parts = SplitTextIntoParts(text, minWords,
maxWords);
            for (int i = 0; i < parts.Count; i++)
            {
                Console.WriteLine($"Part {i + 1}:\n{parts[i]}\n");
            }
            string                          postUrl                          =
"https://capi.grammarly.com/api/check?aidetector=true";
            var cookieContainer = new CookieContainer();
            var handler = new HttpClientHandler { CookieContainer =
cookieContainer };
            var httpClient = new HttpClient(handler);
            try
            {
                string initialUrl = "https://www.grammarly.com/ai-
detector";
                HttpResponseMessage       getResponse       =       await
httpClient.GetAsync(initialUrl);
                getResponse.EnsureSuccessStatusCode();
                var     cookies     =     cookieContainer.GetCookies(new
Uri(initialUrl));
                foreach (var part in parts)
                {
                    var stringContent = new StringContent(part,
Encoding.UTF8, "text/plain");
                    foreach (Cookie cookie in cookies)
                    {
```

```
                    stringContent.Headers.Add("Cookie",
$"{cookie.Name}={cookie.Value}");
                }

                HttpResponseMessage postResponse = await
httpClient.PostAsync(postUrl, stringContent);
                postResponse.EnsureSuccessStatusCode();
                string postResult = await
postResponse.Content.ReadAsStringAsync();
                Console.WriteLine(postResult);
                // MessageBox.Show(postResult);
                List<Root> Result;
                Result = JsonConvert.DeserializeObject<List
<Root>>(postResult);
                Form2.pidluyAI = (Result.Last().count.ToString());
            }
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
    static List<string> SplitTextIntoParts(string text, int minWords,
int maxWords)
    {
        List<string> parts = new List<string>();
        string[] words = text.Split(' ');
        for (int i = 0; i < words.Length; i += maxWords)
        {
            int length = Math.Min(maxWords, words.Length - i);
            string part = string.Join(" ", words, i, length);
            while (length < minWords && (i + length) < words.Length)
            {
                part += " " + words[i + length];
```

```
                    length++;
                }
                parts.Add(part);
            }
            return parts;
        }
    }
}
```

ДОДАТОК Б СЛАЙДИ ПРЕЗЕНТАЦІЇ



Мета та завдання роботи

Метою даної роботи є:
Створення програмного забезпечення що зможе поєднати в собі як стандартні методи виявлення плагіату так й інтегроване вирішення проблеми використання ШІ в різного роду наукових роботах. Дослідження питань плагіату та штучного інтелекту. Реалізація алгоритму пошуку плагіату.

Завданням цієї роботи являється розробка програми, що буде поєднувати систему виявлення плагіту з системою пошуку тексту написаного ШІ.

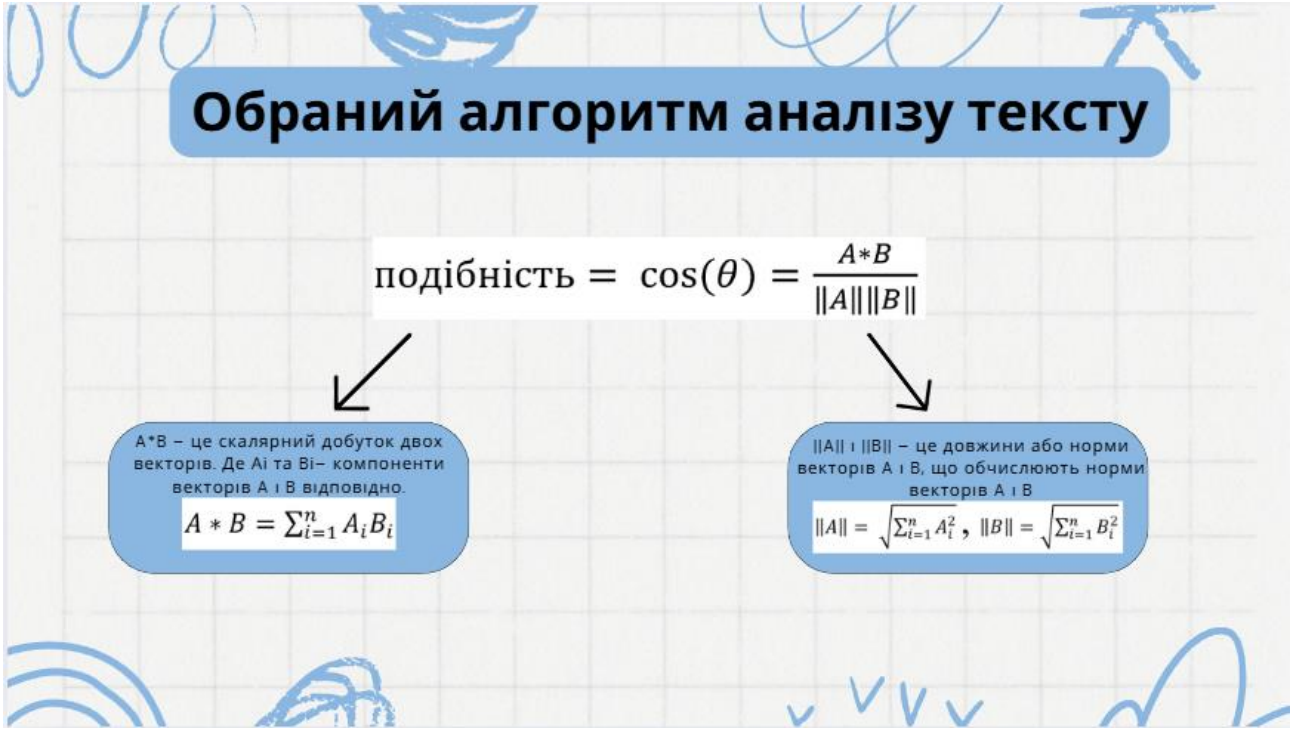
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

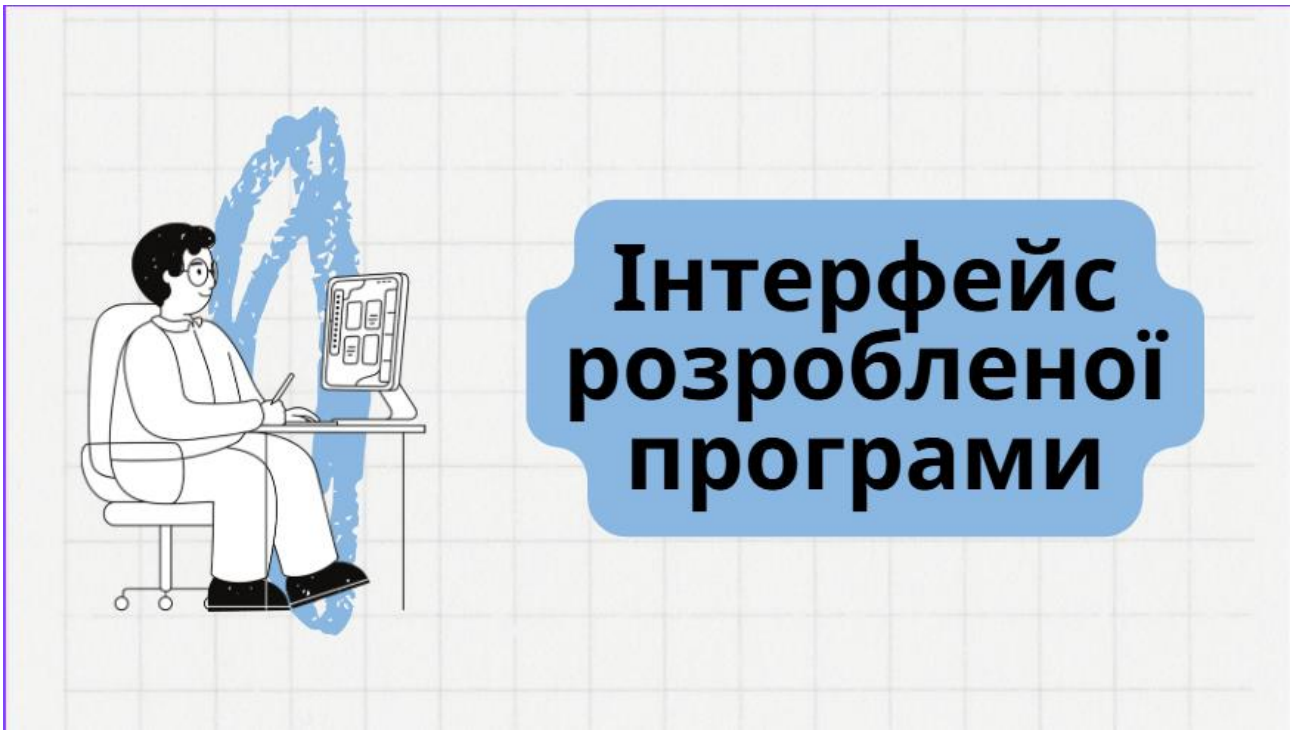
01 Штучний інтелект (ШІ) – це галузь комп'ютерних наук і технологій, присвячена створенню систем або машин, які можуть виконувати завдання, що зазвичай вимагають людського інтелекту. Ці завдання включають навчання, міркування, вирішення проблем, сприйняття, розуміння мови та прийняття рішень.

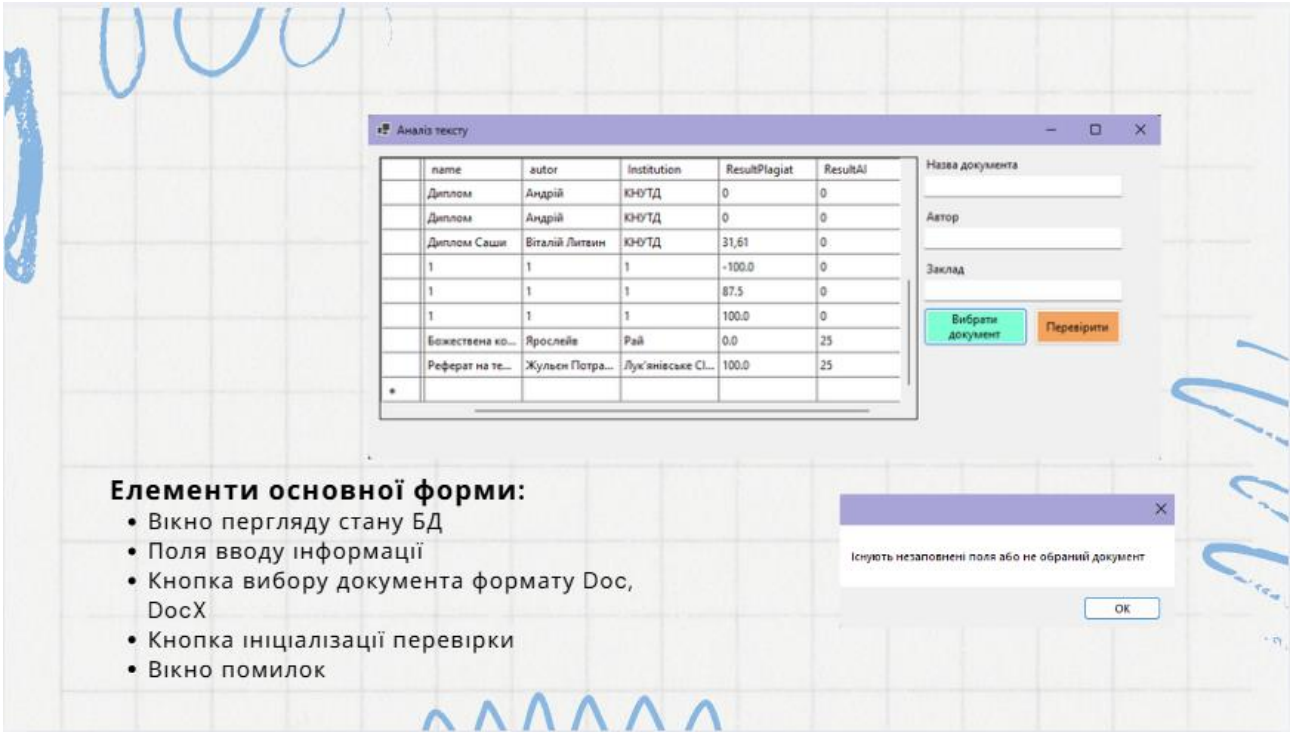
02 Штучна нейронна мережа (ШНМ) – це система обробки інформації або сигналів, що складається з великої кількості простих елементів, з'єднаних між собою прямими зв'язками, які співпрацюють для виконання паралельної розподіленої обробки з метою вирішення бажаної обчислювальної задачі.

03 Плагіат – представлення роботи або ідей з іншого джерела як власних, за згодою або без згоди оригінального автора, шляхом включення їх у свою роботу без повного визнання. Під це визначення підпадає весь опублікований і неопублікований матеріал у рукописній, друкованій або електронній формі, а також використання матеріалів, створених повністю або частково за допомогою штучного інтелекту


ПРОЕКТУВАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ
Вибір технологій і середовища розробки
Розгляд алгоритму для аналізу тексту та виявлення плагіту
Інтеграція API для аналізу текстів, створених ШІ
API

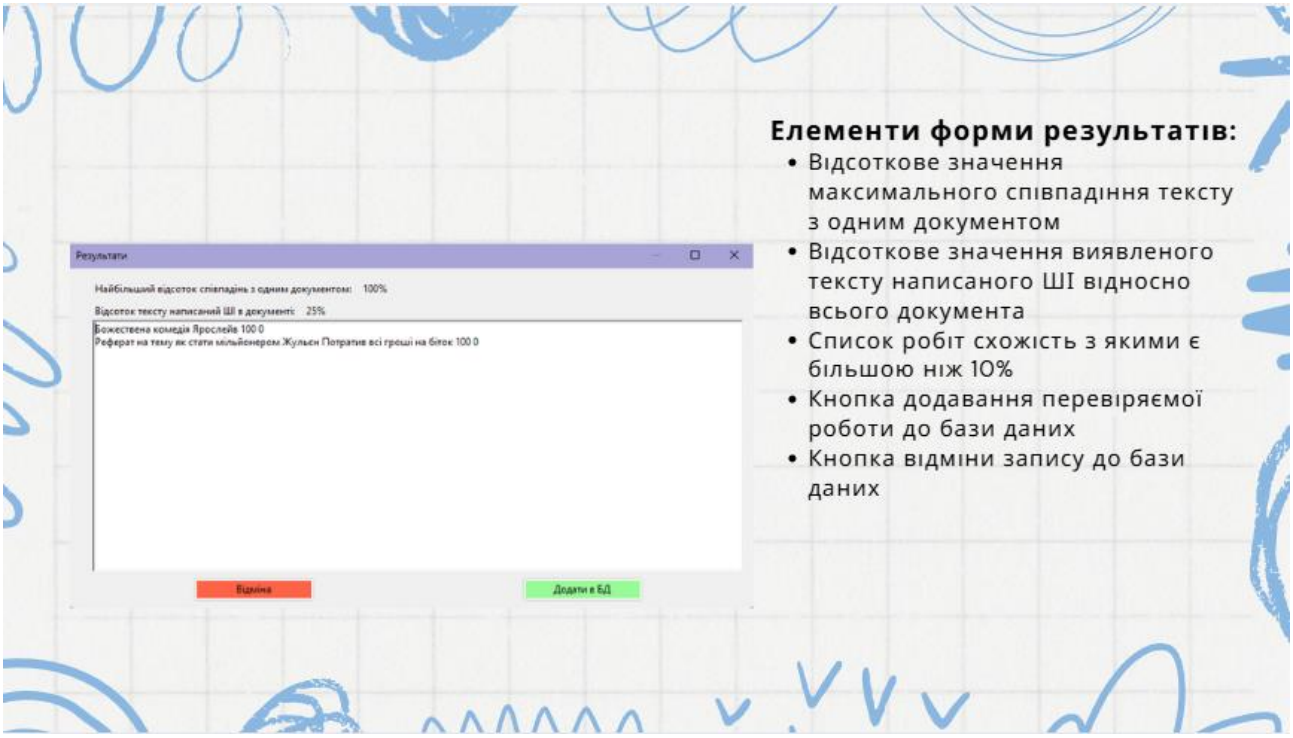

Обраний алгоритм аналізу тексту
подібність = $\cos(\theta) = \frac{A*B}{\|A\|\|B\|}$
A*B – це скалярний добуток двох векторів. Де Ai та Bi– компоненти векторів A і B відповідно.
$A * B = \sum_{i=1}^{n} A_i B_i$
||A|| і ||B|| – це довжини або норми векторів A і B, що обчислюють норми векторів A і B
$\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2}$, $\|B\| = \sqrt{\sum_{i=1}^{n} B_i^2}$





Елементи основної форми:

- Вікно перегляду стану БД
- Поля вводу інформації
- Кнопка вибору документа формату Doc, DocX
- Кнопка ініціалізації перевірки
- Вікно помилок



Елементи форми результатів:

- Відсоткове значення максимального співпадіння тексту з одним документом
- Відсоткове значення виявленого тексту написаного ШІ відносно всього документа
- Список робіт схожість з якими є більшою ніж 10%
- Кнопка додавання перевіряємої роботи до бази даних
- Кнопка відміни запису до бази даних

Висновки

Висновок

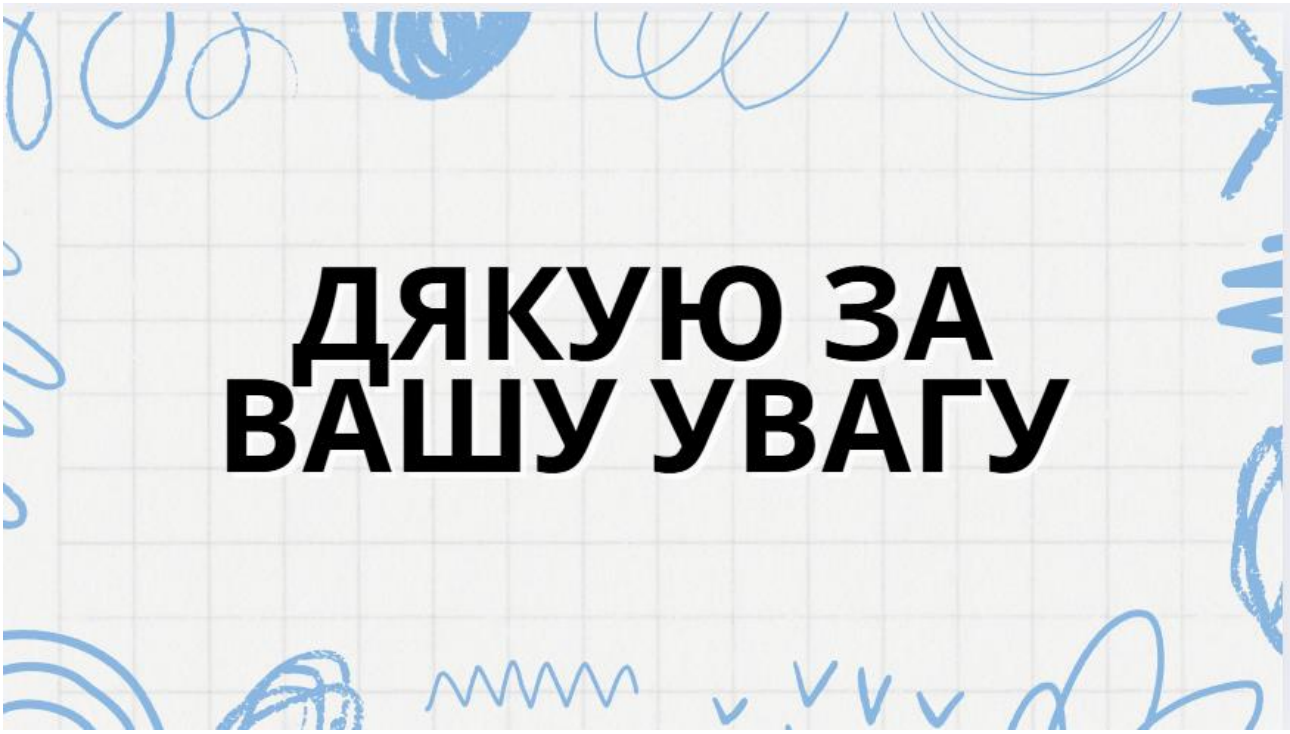
При закінчені цієї роботи була досягнута мета роботи та виконане поставлене завдяння. А саме на даний момент розроблений приклад програмного рішення для аналізу тексту на плагіат та ШІ

Перспективи подальшого дослідження

В подальшому цю роботу можна розвивати дослідивши або розробиши більше алгоритмів аналізу тексту, навчити власну нейронну мережу аналізу тексту, реалізувати більш зручний інтерфейс програми.

Розроблений функціонал

В створеному програмному рішенні було реалізовано алгоритм аналізу тексту методом косинусної подібності, систему пошуку тексту написаного ШІ, інтерфейс для зручного використання програми.


ДЯКУЮ ЗА
ВАШУ УВАГУ