МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки та інформаційні технології в бізнесі**

Група: АК-41б **денна форма навчання**

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОБОТИ З ФАЙЛАМИ

ЗА НАКАЗОМ № 4601-5/335 ВІД 07 ЛЮТОГО 2025 РОКУ

здобувача вищої освіти **Єременка Павла Юрійовича**

Робота допущена до захисту в ЕК

протокол кафедри ІТММ №__ від _____2025р.

Завідувач кафедри ІТММ
к.п.н., доцент
_____ Н.І. Стяглик

Науковий керівник
Д.т.н, професор
_____Гороховатський В. О.

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки та інформаційні технології в бізнесі

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Н. І. Стяглик
Підпис ініціали, прізвище

"08" лютого 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)
Єременко Павло Юрійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **Застосування штучного інтелекту для роботи з файлами**

керівник роботи **Гороховатський Володимир Олексійович, д.т.н, проф.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "08" лютого 2025 року № 4601-5/535

2. Строк подання студентом роботи 15 травня 2025 року

3. Перелік питань, які потрібно розробити:
У розділі 1: **Аналіз існуючих рішень та теоретичні аспекти**
У розділі 2: **Розробка системи для автоматизації роботи з файлами**
У розділі 3: **Тестування та оцінка ефективності**

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної бакалаврської роботи
2	Затвердження плану і завдання кваліфікаційної бакалаврської роботи
3	Здача кваліфікаційної бакалаврської роботи керівнику
4	Підпис кваліфікаційної бакалаврської роботи керівника
5	Підпис кваліфікаційної бакалаврської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної бакалаврської роботи
7	Захист кваліфікаційної бакалаврської роботи

5. Дата видачі завдання 08 лютого 2025 року

Студент _____ **П.Ю. Єременко** _____
підпис ініціали, прізвище

Керівник роботи _____ **В.О.Гороховатський**_____
підпис ініціали, прізвище

РЕФЕРАТ

НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

«Застосування штучного інтелекту для роботи з файлами»

Єременко Павло Юрійович

Кваліфікаційна бакалаврська робота містить 79 сторінок, 1 таблицю, 20 рисунків, список літератури з 43 найменувань.

Об’єктом дослідження є програмні засоби штучного інтелекту.

Предметом дослідження є функціонали штучного інтелекту для роботи з файлами.

Мета кваліфікаційної бакалаврської роботи полягає у вивченні можливостей ШІ для взаємодії системи з файлами.

Завданнями кваліфікаційної бакалаврської роботи є:

пободува застосунку з ШІ для аналізу інформації у файлах.

Актуальність дослідження:

підтверджується широким застосуванням засобів штучного інтелекту.

За результатами дослідження:

проаналізовано сучасні методи та інструменти ШІ для роботи з файлами, зокрема технології машинного навчання, обробки природної мови (NLP) та комп’ютерного зору.

Практична новизна:

розроблено аналог застосунку, який можна впровадити для розпізнавання файлів в існуючій системі.

Одержані результати можуть бути використані у прикладних задачах штучного інтелекту.

КЛЮЧОВІ СЛОВА: штучний інтелект, робота з файлами, неструктуровані дані.

ABSTRACT

ON THE QUALIFYING BACHELOR'S WORK

“Application of artificial intelligence for working with files”

Yeremenko Pavlo Yuriiovych

The qualifying bachelor's thesis contains 79 pages, 1 tables, 20 figures, a list of references of 43 titles.

The object of research is artificial intelligence software.

The subject of research is the functionality of artificial intelligence for working with files.

The purpose of the bachelor's thesis is to study the capabilities of AI for system interaction with files.

The tasks of the bachelor's thesis are:

To build an application with AI for analyzing information in files.

Relevance of the study:

Confirmed by the widespread use of artificial intelligence tools.

Results of the study:

Modern AI methods and tools for working with files, including machine learning, natural language processing (NLP) and computer vision technologies, are analyzed.

Practical novelty:

An application analog that can be implemented for file recognition in an existing system has been developed.

The obtained results can be used in applied tasks of artificial intelligence.

KEYWORDS: artificial intelligence, file management, unstructured data.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

AI (Artificial Intelligence) – штучний інтелект; галузь комп’ютерних наук, яка займається розробкою систем, здатних виконувати завдання, що зазвичай вимагають людського інтелекту.

ML (Machine Learning) – машинне навчання; підгалузь штучного інтелекту, що ґрунтується на використанні алгоритмів, які дозволяють системам самостійно навчатися на основі даних.

NLP (Natural Language Processing) – обробка природної мови; напрям штучного інтелекту, що займається взаємодією комп’ютерів і людської мови.

OCR (Optical Character Recognition) – оптичне розпізнавання символів; технологія, яка дозволяє розпізнавати текст у зображеннях або сканованих документах.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, що часто використовується для зберігання або передавання структурованої інформації.

API (Application Programming Interface) – інтерфейс прикладного програмування; набір інструментів і протоколів, які дозволяють програмам взаємодіяти між собою.

PDF (Portable Document Format) – формат електронних документів, що зберігає шрифт, зображення та форматування документа незалежно від пристрою.

AI-модель – алгоритм або набір алгоритмів штучного інтелекту, що пройшли процес навчання для виконання конкретного завдання, наприклад, класифікації, аналізу тексту, генерації тощо.

ШІ – штучний інтелект.

Файл – одиниця збереження даних у файловій системі, яка містить текст, зображення, звук або іншу інформацію.

Інтерпретація тексту – процес аналізу та осмислення текстової інформації з метою її структуризації або подальшого використання.

Класифікація документів – процес автоматичного розподілу файлів за категоріями або темами.

Модель глибокого навчання – тип штучного інтелекту, який використовує багатошарові нейронні мережі для аналізу складних структурованих і неструктурованих даних.

ІО -Це будь-які сутності, які взаємодіють зі своїм середовищем:

ІП - Механізми або учасники, які забезпечують доступ до ресурсів:

ІЕ - Інформаційно-Еволюційні Сутності

ЗМІСТ

ВСТУП

Актуальність. Штучний інтелект (ШІ) продемонстрував багатообіцяючі результати за останні десятиліття, але з нещодавнім відродженням інтересу та покращенням результатів у вирішенні реальних завдань ця галузь переживає вибуховий ріст. Багато з покращених результатів було отримано завдяки більшим і складнішим нейронним мережам, що складаються з багатьох шарів (так звані глибокі нейронні мережі).

Але значна частина прогресу також може бути віднесена до більших наборів даних і широкомасштабного навчання/тренування на графічних процесорах. Відновлення інтересу і збільшення кількості ресурсів також призвело до проривів у суміжних технологіях ШІ, наприклад, у статистиці, генеративних моделях та імовірнісному програмуванні/індукції [1-5].

Проблематика. Однак останнім часом лунає критика, що багато з цих підходів до створення більш інтелектуального програмного забезпечення занадто далекі від людського інтелекту і, таким чином, навряд чи будуть достатніми. Натомість критики стверджують, що нам насправді потрібні алгоритми, які будують і розширюють причинно-наслідкові моделі, можуть навчатися на дуже невеликій кількості прикладів (одно- або кількаразове навчання) і можуть символічно міркувати про закономірності і знання, які вони витягують з датчиків.

Незалежно від того, чи вистачить нинішнього набору технологій штучного інтелекту для досягнення людського рівня інтелекту чи ні, очевидно, що програмні системи все частіше включатимуть їх як компоненти і підсистеми.

Форма рішень, створених на основі цих технологій ШІ/ММ, часто виглядає зовсім інакше, ніж програмне забезпечення, яке зазвичай розробляється і розгортається. Таким чином, не тільки сама технологія ШІ швидко змінюється, але й рішення, які вона пропонує, зазвичай дуже

відрізняються від того, до чого звикли організації та інженери-розробники програмного забезпечення. Це створює новий унікальний набір ризиків і можливостей для організацій, що займаються розробкою програмного забезпечення, і вони повинні розуміти і аналізувати ці ризики, щоб вибрати відповідні стратегії [6-12].

Така гібридизація AI/ML та програмної інженерії неминуча і в іншому сенсі. З'являться широкі можливості для застосування моделей ШІ та ML для вдосконалення самої розробки програмного забезпечення. Інженери-програмісти близькі до цих технологій і, ймовірно, будуть першими, хто застосує їх до своїх проблем, методів та інструментів. Цьому також сприяє тенденція до того, що технології ШІ/ML стають все більш компонентними і їх легше використовувати і повторно застосовувати навіть неспеціалістам. Досягнення в галузі програмної інженерії дозволяють пакувати технології ШІ і легко використовувати їх повторно через RESTful API1 як автоматизовані хмарні рішення, які можуть використовувати кілька технологій, перш ніж вибрати і автоматично налаштувати одну/кілька .

Предметом дослідження є розробка та оптимізація алгоритмів штучного інтелекту для автоматизації роботи з файлами, включаючи їх класифікацію, обробку, аналіз даних і керування структурою.

Об’єктом дослідження є процес застосування штучного інтелекту для роботи з файлами різних форматів текстові, графічні, мультимедійні, структуровані у контексті автоматизації, збереження та оптимізації даних.

Завдання дослідження:

− Проаналізувати сучасні методи та інструменти ШІ для роботи з файлами, зокрема технології машинного навчання, обробки природної мови (NLP) та комп’ютерного зору.

− Визначити ключові проблеми в автоматизації роботи з файлами (наприклад, розпізнавання формату, обробка великих даних, кібербезпека).

− Запропонувати підходи на основі ШІ для оптимізації процесів сортування, пошуку, стиснення та аналізу файлів.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНІ АСПЕКТИ

1.1 Огляд методів обробки файлів за допомогою ШІ

З появою нових засобів комунікації та різноманітних додатків, таких як соціальні мережі, мобільні додатки та цифровий маркетинг, дані, що генеруються, не мають типового формату чи заздалегідь визначеної схеми, як стандартні дані, і не можуть управлятися за допомогою моделей реляційних баз даних. Дані генеруються в різних формах, таких як текст, аудіо, відео, електронні листи та зображення. Це приклади неструктурованих даних. Такі дані не мають структури і не є стандартизованими [1], що ускладнює їх редагування, пошук та аналіз для більшості організацій [2].

Статистика Forbes [3] свідчить, що аналіз неструктурованих даних є проблемою для 95% бізнес-організацій, оскільки вони не мають необхідної експертизи для роботи з неструктурованими даними. До 2025 року потрібно буде проаналізувати понад 150 трильйонів гігабайт (150 зеттабайт) неструктурованих даних. Організації можуть використовувати інструменти аналізу даних, щоб краще розуміти потреби клієнтів і прогнозувати зміни на ринку.

Огляд і майбутні напрямки - це незліченна кількість застосувань неструктурованих даних. До 2025 року річний прибуток ринку «глобальних неструктурованих даних і бізнес-аналітики» оцінюється в 274,3 мільярда доларів США [4].

Майже 80% даних, що генеруються в організації, є неструктурованими. Ці дані необхідні організації для прийняття рішень, прогнозного аналізу та пошуку закономірностей . Ефективна обробка таких неструктурованих даних завжди є складним, трудомістким і дорогим завданням для будь-якої організації.

Оскільки дані генеруються з винятковою швидкістю, цінна інформація, що міститься в них, не може бути використана, якщо не існує певної форми автоматизованого аналізу. У таблиці 1. показані різні сфери застосування автоматичного вилучення інформації з неструктурованих документів.

Автоматизація допомагає організаціям організувати та отримати доступ до корисної інформації у структурований спосіб . Автоматизація неструктурованих даних, що зберігаються в цифровому форматі, дозволить організаціям швидко отримати уявлення про свій бізнес, підвищити свою конкурентну перевагу, покращити продуктивність та впроваджувати інновації. Таким чином, організації адаптуються до рішень з автоматизації, розуміючи важливість технологій на основі штучного інтелекту (ШІ), таких як комп'ютерний зір (КЗ) та обробка природної мови (ОМП). Технології штучного інтелекту можуть розуміти і класифікувати неструктуровані дані, такі як текст, зображення і відскановані документи, краще, ніж традиційні методи вилучення інформації.

Збільшення обсягів і необхідність ефективного використання неструктурованих даних зумовлюють потребу в розробці системи обробки неструктурованих документів на основі ШІ, яка б допомогла організаціям автоматично отримувати інформацію з неструктурованих даних. Таким чином, це розглядається як важливий і перспективний напрямок досліджень.

Неструктуровані дані є невід'ємною частиною багатьох організацій. Такі форми, як рахунки-фактури, дані про клієнтів, страхові заяви, слугують доказом і записом транзакцій та інших важливих подій в організації. Належна обробка цих форм є дуже важливою. Обробка вручну може сповільнити процес і призвести до помилок і затримок. Автоматичне вилучення даних вирішує ці проблеми, надаючи автоматизоване та оцифроване рішення для обробки документів. Автоматизація трудомістких і повторюваних завдань допомагає підвищити продуктивність і розвиток організації. Штучний інтелект дає змогу ефективно й автоматично витягувати корисну інформацію з неструктурованих документів. ШІ також

допомагає створити більш зрозумілий аналіз неструктурованих документів, який організації можуть використовувати в процесі прийняття важливих рішень.

На рисунку 1.1 показано еволюцію методів, що використовуються для автоматичного вилучення інформації з неструктурованих документів. Раніше організації використовували ручну працю для введення даних, обробки паперових документів і передачі необхідної інформації на наступний ланцюжок бізнес-процесів.

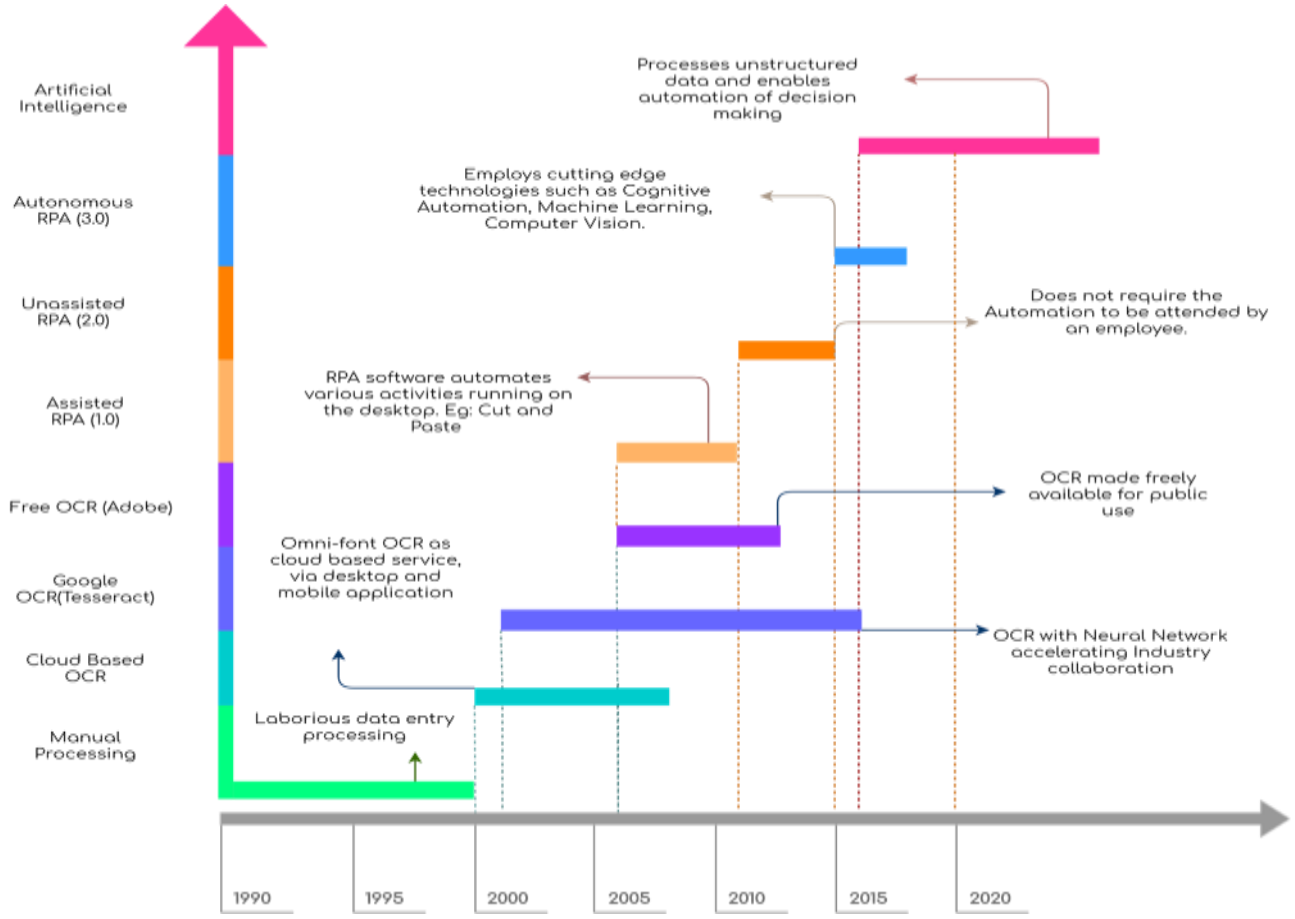

Рис. 1.1 – Еволюція методів автоматичного вилучення інформації з неструктурованих документів

Як перший крок до оцифрування, організації почали використовувати оптичне розпізнавання символів (OCR). OCR використовувався для

перетворення вмісту відсканованих документів у цифровий формат. Попередні версії OCR вимагали надання зображення кожного символу і обмежувалися розпізнаванням лише одного шрифту за один раз. На початку 2000-х років було запропоновано «Omni-font OCR», який міг обробляти текст, надрукований практично будь-яким шрифтом [5]. Пізніше він став хмарним сервісом, до якого можна було отримати доступ через десктопні та мобільні додатки. Сьогодні багато постачальників послуг розпізнавання пропонують технологію розпізнавання через API і вміють розпізнавати майже всі символи та шрифти з достатнім рівнем точності.

Просуваючись далі в автоматизації, організації почали використовувати робототехнічну автоматизацію процесів (RPA), щоб замінити засновані на правилах, структуровані та повторювані процеси програмним ботом [6]. Логіка, заснована на правилах, лежить в основі більшості автоматизованих процесів.

Assisted RPA або RPA 1.0 автоматизує кілька дій користувача і додатків, які виконуються на комп'ютері користувача. Прикладом RPA 1.0 є автоматизація такої простої роботи, як вирізання, копіювання та вставка інформації з однієї комп'ютерної системи в іншу. Однак, RPA 1.0 необхідно застосовувати, коли спілкування між людиною та комп'ютером є важливим [6].

RPA без допомоги людини або RPA 2.0 можна встановити на кількох машинах, щоб автоматизувати завдання без допомоги людини. Це може значно зменшити взаємодію людини з бізнес-процесами [7]. Співробітник, увійшовши в систему, активує запуск процесів, відстежує їх виконання і вимикає систему, коли вони завершуються. Цей процес може бути повністю автоматизований за допомогою RPA 2.0 без будь-якого втручання людини.

Автономний RPA або RPA 3.0 - це остання версія RPA. Вона використовує переваги ШІ та CV. З розвитком технологій RPA також зазнав більш корисних удосконалень [8]. Прикладом RPA 3.0 може бути сегрегація

та автоматична відповідь на кілька електронних листів. Таку RPA з використанням штучного інтелекту іноді називають когнітивною RPA [9].

Сьогодні автоматизація на основі штучного інтелекту використовує кілька перспективних підходів, таких як NLP, машинне навчання (ML) та текстова аналітика, що формулюють однакову корисність як структурованих, так і неструктурованих даних. За допомогою ШІ неструктуровані дані можна аналізувати, керувати ними та обробляти, щоб отримати цінні висновки з меншими людськими зусиллями та втручаннями .

1.2. Використання нейронних мереж та алгоритмів машинного навчання для роботи з файлами

Ручне вилучення тексту з неструктурованих документів, таких як відсканові PDF-файли, не є масштабованим і схильним до помилок, оскільки люди схильні втомлюватися і робити помилки. Останнім часом організації намагаються використовувати шаблонні підходи, такі як розпізнавання тексту, для автоматизації обробки документів. OCR використовується для розпізнавання тексту на зображенні; зазвичай це відсканований друкований або рукописний документ. OCR також може автоматично сортувати різні типи документів і організовувати їх відповідно до певних правил. Наприклад, класифікація та керування рахунками-фактурами на основі типу товару або постачальника.

На рисунку 1.2 показано основні етапи розпізнавання.

Набір даних [9]: Здебільшого дослідники використовували загальнодоступні набори даних для розпізнавання рукописних або друкованих символів. Самостійно створений набір даних також може бути використаний для вилучення тексту за допомогою OCR.

Попередня обробка [10]. Етап попередньої обробки необхідний для відокремлення символу/слова від фону на зображенні. Він включає в себе такі моменти.

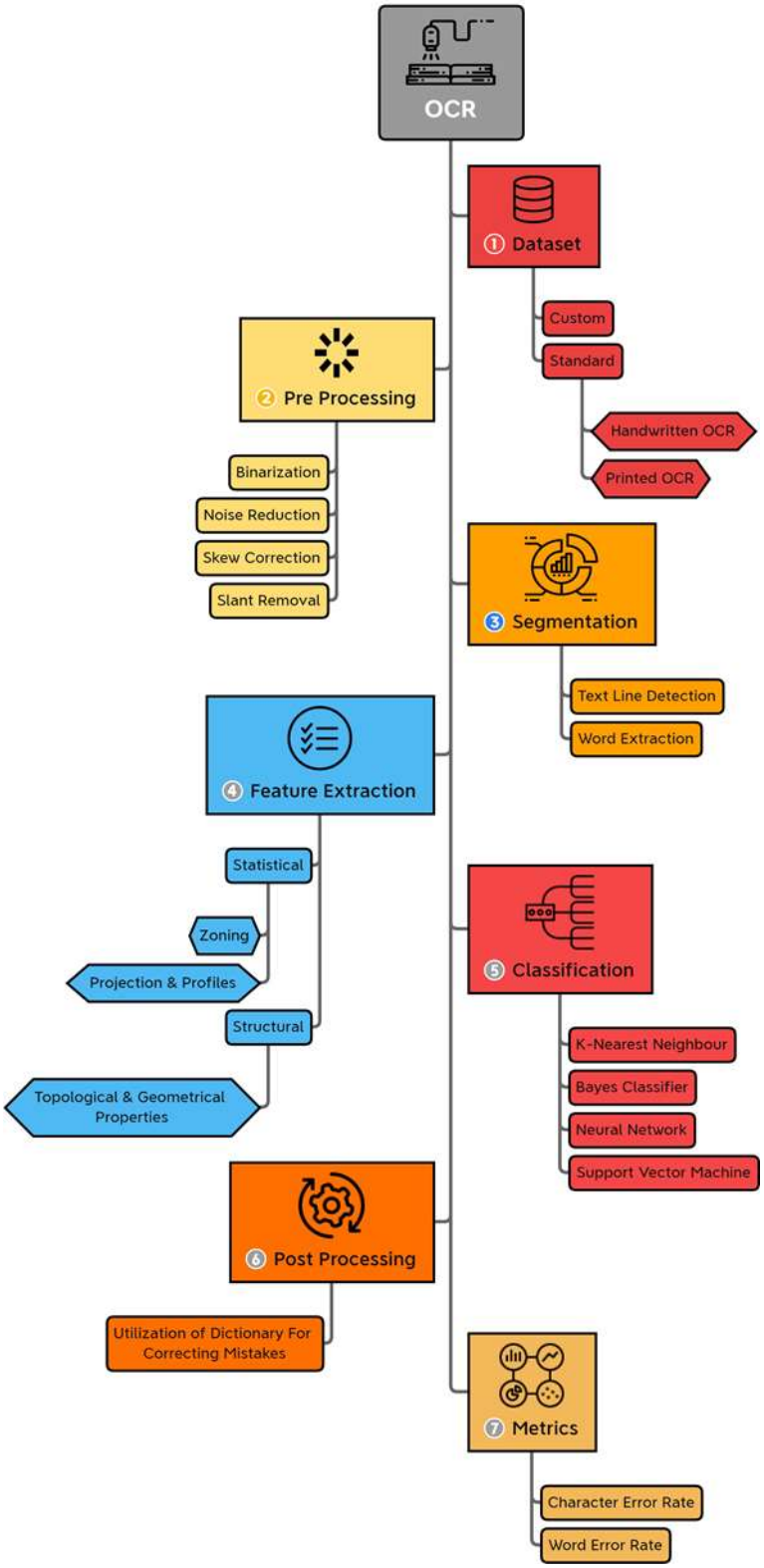

Рис. 1.2 - Основні кроки в розпізнаванні тексту.

Бінаризація. Перетворення зображення на чорно-білий піксель. Це перетворення відбувається шляхом фіксації порогового значення. Якщо значення більше, це вважається білим пікселем, інакше - чорним.

Шумозаглушення. Очищає зображення, видаляючи всі небажані точки та плями.

Виправлення перекосу. Деякий текст може бути вирівняний неправильно. Виправлення перекосу допомагає вирівняти цей текст.

Сегментація [10]. Розбиває зображення на частини для подальшої обробки, які можна сегментувати у вигляді тексту або слів. Для сегментації використовуються методи виявлення текстових рядків і виділення слів. Правильне розпізнавання символів залежить від точної сегментації.

Виділення ознак [11]. Виділення вихідних даних у керовані дані або необхідні дані. В OCR широко використовуються два методи вилучення ознак: статистичний (визначає статистичні ознаки символів) і структурний (визначає структурні ознаки, такі як горизонтальні та вертикальні лінії, кінцеві точки).

Виділення статистичних ознак можна здійснити шляхом зонування, коли зображення поділяють на зони, а потім з кожної зони виділяють ознаки, щоб сформувати вектор ознак. Проекція та профіль також можуть бути використані як метод статистичного вилучення ознак. Проекційні гістограми обчислюють кількість пікселів у різних напрямках зображення символів. Це може бути використано для розділення таких символів, як «m» і «n». Профіль використовується для підрахунку кількості пікселів від обмежувальної рамки до зовнішнього краю. Використовується для визначення зовнішніх форм символів. Це дозволяє розрізняти такі літери, як «p» і «q».

При вилученні структурних ознак вилучаються геометричні властивості символу або знака. Геометричними властивостями або структурними ознаками символу є: штрихи символу, горизонтальні лінії, вертикальні лінії, кінцеві точки, перетини між лініями та петлі. Це дає уявлення про тип компонентів, з яких складається символ. У більшості

методів розпізнавання текстів алгоритм навчається точно класифікувати набір символів і нумерацію, оскільки він тренується на відомому наборі даних. Найпопулярніші методи, що використовуються для класифікації в розпізнаванні текстів згадано нижче.

K – найближчий сусід: класифікує об'єкти зі схожими ознаками в безпосередній близькості. Використовується для сегментації та розпізнавання латинських алфавітів у верхньому та нижньому регістрах, а також приголосних та голосних деванагарі для індійських відсканованих зображень документів, написаних латиницею та деванагарі, у дослідженні [11].

Наївний класифікатор Байєса: це імовірнісний метод класифікації. Він використовує теорему Байєса про ймовірність для обчислення класу нових або невідомих даних. Використовується для розпізнавання бізнес-інвойсів та класифікації таких полів, як invoice_number та invoice_date в [12].

Нейронна мережа: показав сильні здібності до автоматичного розпізнавання тексту та вилучення даних шляхом розпізнавання основних зв'язків між символами/словами.

Згорткові нейронні мережі на основі регресії (R-CNN) використовуються для виявлення об'єктів у реальному наборі даних китайських паспортів та медичних квитанцій в.

Машина опорних векторів: Це найпоширеніший алгоритм класифікації в розпізнаванні текстів. Він працює краще, ніж будь-який інший метод класифікації. Він не базується на жодних припущеннях про незалежність, як у методі наївного Байєса. Використовується для категоризації або розпізнавання тексту в [13], [14].

Постобробка [15] – виявляє та виправляє помилки у вихідному тексті після обробки зображення за допомогою методу оптичного розпізнавання.

Метрики м-етрики використовуються для обчислення частоти помилок у словах і символах та для оцінювання ефективності методів розпізнавання.

Іншим методом автоматизації обробки неструктурованих документів є використання методу, заснованого на правилах, який називається роботизованою автоматизацією процесів (RPA). Цей метод вимагає від людини написання простих правил для виконання повторюваних дій програмним ботом.

На рисунку 1.3. показано тематичну діаграму RPA.

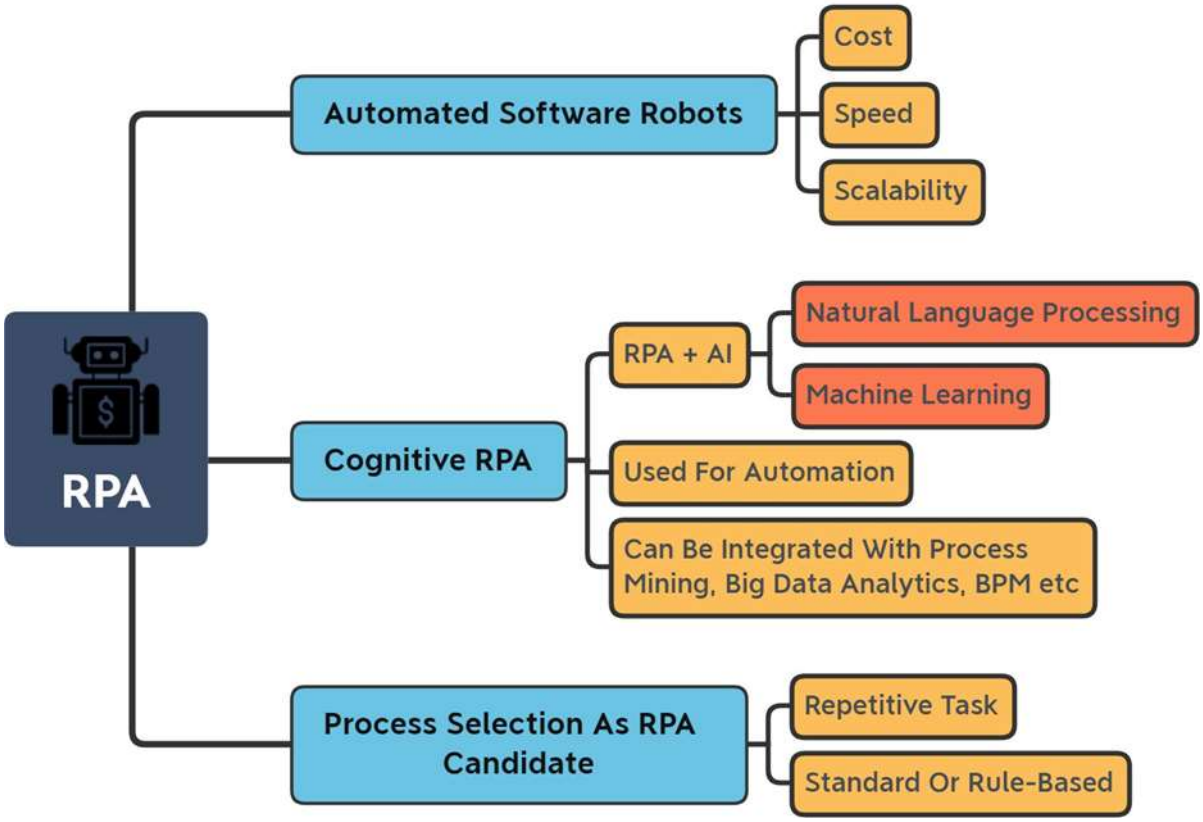

Рис. 1.3 - Діаграма для автоматизації процесів робототехніки

Автоматизовані програмні роботи RPA дозволяє організаціям автоматизувати дуже надлишкові та засновані на правилах завдання за невелику суму раніше понесених витрат і часу. Він може виконувати багато завдань, таких як вхід в систему або додаток, розміщення файлів і папок у вибраних місцях, копіювання і вставка даних, заповнення форм, вилучення структурованих даних з документів, скрапінг веб-браузерів тощо. Ці програмні роботи підвищують масштабованість, швидкість і знижують операційні витрати.

Когнітивний RPA використовує переваги NLP і ML для виконання великих обсягів повторюваних завдань, які раніше виконувала людина. Традиційного RPA може бути недостатньо для автоматизації систем, щоб самостійно приймати прості рішення. Тут важливу роль відіграє когнітивний RPA. Це поєднання RPA та штучного інтелекту. Він може керувати величезними даними, знаходити в них приховані закономірності та прогнозувати майбутні тенденції. Він покращує продуктивність з часом. Він може імітувати спосіб мислення людини та приймати розумні рішення. Він може обробляти неструктуровані документи і може бути інтегрований з аналітичними інструментами організації або додатками для управління бізнес-процесами (BPM).

Одним з важливих фактів, який може вплинути на успіх впровадження RPA, є придатність процесу-кандидата для автоматизації. Організації повинні знати критерії придатності процесу для впровадження RPA. Обраний процес може бути спочатку визначений у вигляді чітких або визначених правил, оскільки RPA підходить лише для завдань, що базуються на правилах. Стандартизація процесу перед автоматизацією також має важливе значення, оскільки більш стандартизований процес спричиняє менше винятків.

На рисунку 1.4 показані основні етапи RPA.

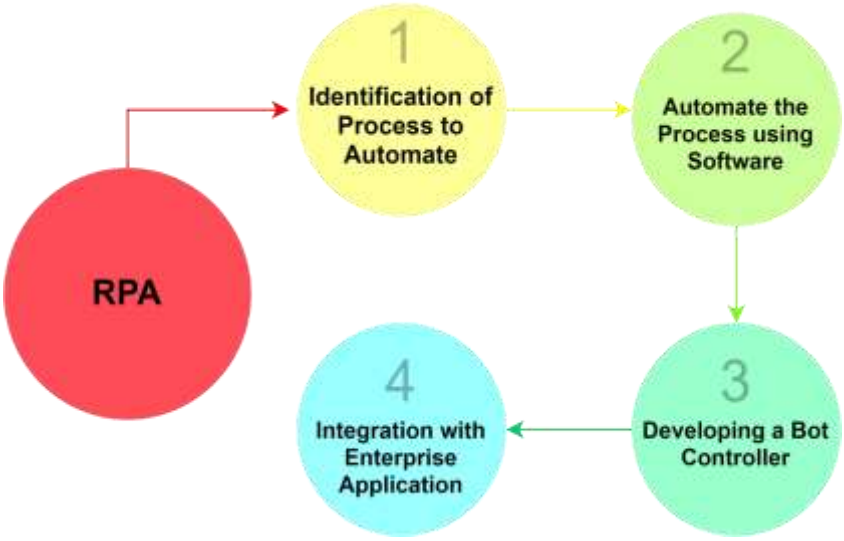

Рис. 1.4 - Основні кроки в RPA.

Визначення процесів для автоматизації - організаціям спочатку потрібно визначити процес, який підходить для автоматизації.

Автоматизація процесів за допомогою програмного забезпечення : Завдання, що виконуються ідентифікованим процесом, визначаються у вигляді серії інструкцій. На цьому кроці визначається робочий процес автоматизації процесу.

Автоматизовані процеси передаються бот-контролеру. Він використовується для контролю та визначення пріоритетів виконання процесу. Він дозволяє користувачам планувати, керувати та контролювати різні види діяльності. Бот-контролер також контролює статус виконання процесу, перевіряючи журнали виконання.

Автоматизовані програмні боти можуть бути інтегровані з аналітичними інструментами організації або програмами управління бізнес-процесами (BPM).

Підходи на основі штучного інтелекту мають значний потенціал для автоматичного вилучення корисної інформації з неструктурованих документів. На рисунку 1.5 показано діаграму для підходів на основі ШІ.

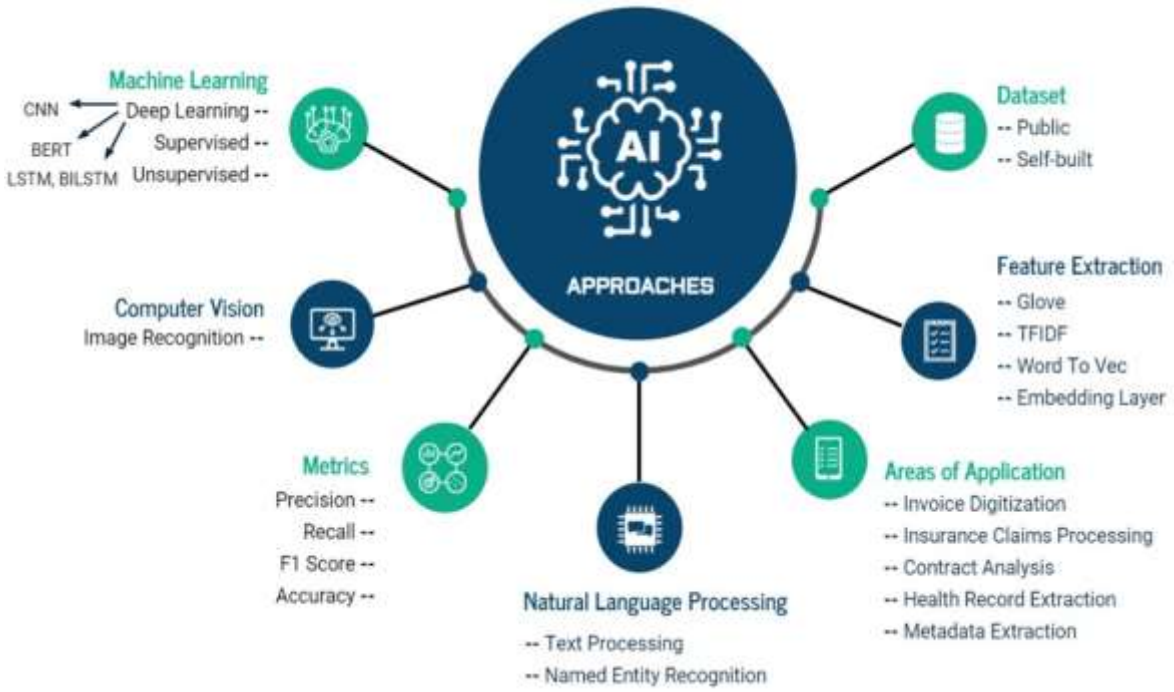

Рис. 1.5 - Діаграма для підходів на основі ШІ.

Різні дослідники широко використовують загальнодоступні набори даних для завдань аналізу документів. Щоб отримати інформацію зі сканованих документів, таких як квитанції, рахунки-фактури, дослідники підготували свої набори даних або працювали з наборами даних, наданими конкретними організаціями.

Для класифікації та розпізнавання тексту використовуються різні методи виділення ознак: GloVe, TF-IDF та Word2Vec. GlobalVectors (GloVe) використовується для отримання векторного або числового представлення слів. TF-IDF є популярним підходом для призначення ваг словам, які вказують на їх важливість у документах. Word2Vec - це найбільш стандартний метод для вивчення вбудовувань слів зі значних наборів даних за допомогою нейронних мереж (NN). Word2Vec може бути виконаний з використанням безперервного мішка слів (CBOW) або безперервного пропускання графів (Continuous Skip Gram).

Підходи на основі ШІ використовуються в різних додатках, таких як оцифрування рахунків-фактур, вилучення медичних записів, вилучення метаданих, обробка страхових претензій, аналіз контрактів і багато іншого.

Він аналізує граматичну структуру на рівні речення, а потім створює граматичні правила для отримання корисної інформації про структуру речення. Серед усіх методів, методи NER є найбільш базовими та важливими методами в НЛП. НЛП використовує синтаксичні правила на рівні речення, такі як призначення граматичних правил і шаблонів на рівні слів або лексем, таких як регулярні вирази для вилучення інформації з заданого тексту, використовуючи NER. Він автоматично сканує неструктурований текст, щоб знайти в ньому «іменовані сутності», такі як ім'я (ім'я, прізвище), місцезнаходження (наприклад, країни, міста), організація, дата та номери рахунків [17].

1.3. Аналіз існуючих програмних рішень та їх обмеження

Об’єкти обробки інформації взаємодіють зі своїм середовищем, тому їх називають взаємодіючими об’єктами (наприклад, ІО – люди, тварини, організації, частини організацій, політичні партії та комп’ютерні системи). Завдяки взаємодії ІП отримують доступ до таких ресурсів, як гроші, їжа, напої або голоси за себе чи пов’язаних ІП. За допомогою низки процесів і механізмів зворотного зв’язку похідні ІЕ (наприклад, діти, нові версії продукту, змінена організація) створюються з ІЕ. Працездатність ІЕ – його здатність продовжувати взаємодіяти та досягати сприятливих результатів і характер будь-якого похідного ІЕ залежать від ресурсів, до яких ІЕ має доступ (прямо або через пов’язані ІЕ), і результатів, яких він досягає. Доступні взаємодії та результати разом із конкуренцією за досягнення результатів визначають тиск відбору для будь-якого ІЕ. Тиск відбору впливає на характеристики похідних ІЕ. Відбір у цьому сенсі є лише результатом взаємодії. Приклади тиску відбору включають ринок, природний відбір, вибори, особистий вибір, культурні норми в суспільствах і статевий відбір, а для будь-якого ІЕ можуть застосовуватися різні комбінації тиску відбору.

Здатність ІЕ досягти сприятливого результату від стану середовища вимагає обробки інформації. Для будь-якого стану середовища ІЕ має знати, як реагувати, тому йому потрібно пов’язати стани середовища з потенційними результатами та діями, необхідними для створення результатів. Таким чином, ІЕ відчувають значення властивостей у середовищі, інтерпретують їх, роблять висновки та створюють інструкції для дій. Ця обробка інформації призводить до того, що іноді називають описовою, прогностичною та прескриптивною інформацією , що відповідає категоризації у Флориді [18].

Ступінь, до якого ІЕ може досягти сприятливих результатів, ми називаємо придатністю, ґрунтуючись на розширенні ідеї Дарвіна [19] у розвитку сучасних технологій . Є три рівні фізичної підготовки:

- вузька придатність - здатність досягати сприятливих результатів в одній взаємодії;
- широка придатність - здатність досягати сприятливих результатів у численних взаємодіях, потенційно різних типів;
- адаптивність - здатність досягати сприятливих результатів, коли характер взаємодій, доступних у середовищі, змінюється.

Широка придатність враховує фактори, які залежать від кількох взаємодій. Наприклад, є багато прикладів машинного навчання, у яких людські упередження стають очевидними з часом . Це приклади, коли широка придатність може включати етичні чи соціальні фактори, які не завжди враховуються у вузькій придатності або неочевидні в невеликій кількості взаємодій.

Ступінь придатності залежить від компонентного шаблону ІЕ. Тут використовуємо термінологію, яка використовується в ІТ-архітектурі [20]. Компонент – це роздільний елемент ІЕ – щось, що обробляє інформацію певним чином. У цьому сенсі різні програми та ІТ-інфраструктура є компонентами організації; компоненти для людей описані в [21] (автори кажуть, що «висновок і пізнання в цілому досягаються коаліцією відносно автономних модулів, які еволюціонували для вирішення проблем і використання можливостей», а «відносно автономний модуль» відповідає компоненту).

На рисунку 1.6 показано, як ці елементи співвідносяться. На малюнку верхні індекси 1, 2 і 3 позначають вузьку придатність, широку придатність і адаптивність відповідно.

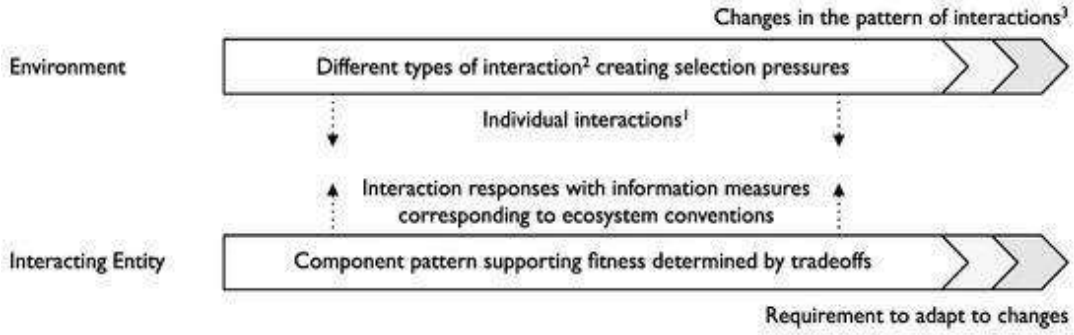

Рис. 1.6 - Рівні взаємодії та підготовленості.

Тиск відбору призводить до формування інформаційних екосистем . Приклади включають англомовних спеціалістів, комп'ютерні системи, які обмінюються певними типами банківської інформації, математиків, спеціалістів з фінансів та багатьох інших. Кожна екосистема має власні правила обміну та обробки інформації. У різних екосистемах інструменти моделювання , такі як мови, математика та комп'ютерні протоколи, еволюціонували, щоб структурувати та маніпулювати інформацією всередині екосистеми. ІЕ поза екосистемою може бути не в змозі інтерпретувати інформацію – подумайте про вченого класичних мов, який намагається зрозуміти квантову механіку.

Інформація стосується фізичного світу. Назвіть зріз безперервною підмножиною простору-часу. Зріз може відповідати сутності в певний момент часу (або, точніше, протягом дуже короткого інтервалу часу), фіксованому шматку простору протягом фіксованого періоду часу або, набагато ширше, події, яка рухається в просторі та часі. Це визначення забезпечує велику гнучкість в обговоренні інформації. Наприклад, фрагменти є достатньо загальними, щоб підтримувати загальне обговорення іменників, прикметників і дієслів, минулого та майбутнього.

Фрагменти, що відповідають умовам екосистеми для представлення інформації, яку називають вмістом щодо екосистеми. Вміст структурований у вигляді фрагментів і тверджень. Фрагмент визначає обмеження на набори фрагментів (наприклад, «Джон», «живе в Римі», «чотири кольори»).

Твердження передбачає зв’язок між обмеженнями (наприклад, «Іван живе в Римі»). В екосистемах та ІЕ фрагменти інформації пов’язані в асоціативну модель (наприклад, «силове поле» Куайна, граничними умовами якого є досвід» [20], Всесвітня павутина або «асоціативна пам’ять» Канемана [4]) з характером зв’язків, визначеним умовностями екосистеми.

Вплив конкуренції та тиску відбору з часом полягає у покращенні здатності ІЕ та екосистем обробляти інформацію, що відповідає різним показникам інформації [5]. Якість інформації може покращитися в тому сенсі, що вона зможе краще сприяти досягненню сприятливих результатів; він може бути виготовлений із меншим тертям [21] або може бути виготовлений швидше. Або можуть бути більш загальні компроміси, в яких баланс між якістю, тертям і темпом змінюється.

Тиск відбору гарантує, що інформація загалом є достатньо надійною для цілей екосистеми всередині оболонки, до якої застосовується тиск відбору. Однак проблеми з якістю та обмеження, які обговорюються нижче, означають, що за межами цієї оболонки ми не повинні очікувати, що конвенції екосистеми забезпечать надійні результати [8]. Це особливо важливо в епоху швидких змін, таку як поточна цифрова революція, коли ІП не можуть встигати за змінами, наприклад, створюючи «цифровий розрив» для людей [22 , 23] і менший ринковий успіх для бізнесу [11]. Для людей екосистеми можуть бути пов’язані з віком – наприклад, «цифрові туземці», «цифрові іммігранти» та «цифрові іноземці» [24] відрізняються за своїм підходом до використання цифрової інформації.

Зараз ШІ викликає багато дискусій. З одного боку, це обіцяє революцію в бізнесі [25], а з іншого – може допомогти викликати сингулярність [26]. Основними нещодавніми досягненнями в області штучного інтелекту є машинне навчання – Домінгос надає огляд у [27].

Як показано [25], ШІ може впливати на багато елементів обробки інформації в організаціях. Важливо, що він може значно покращити всі рівні фізичної підготовки, але щоб перетворити це на переваги, впровадження для

організації має пов’язати детальне розуміння трьох рівнів фізичної підготовки, їх зв’язок і те, як кожна можливість ШІ може їх покращити. У свою чергу, це вимагає розуміння мір інформації, таких як тертя, темп і якість [5 , 6]. Ці пункти розгорнуті нижче.

Щоб допомогти зрозуміти, як ІЕ можуть забезпечити рівні придатності, необхідні для процвітання, ми можемо намалювати модель можливостей, використовуючи техніку з архітектури підприємства [12]. Цей підхід є удосконаленням підходу, використаного в [5 , 6 , 7 , 8]. Здатність – це здатність щось робити, і ми можемо намалювати модель можливостей для інформаційних можливостей, як показано на Рисунку 1.6 , використовуючи три рівні придатності, визначені в попередньому розділі. Зверніть увагу, що це загальна модель можливостей, яка застосовується до всіх ІЕ, і ступінь наявності можливостей у будь-якому ІЕ може значно відрізнятися. Існує багато інших подібних моделей що висвітлюють різні точки зору, але рисунок 1.7 зосереджується на питаннях, які стосуються фітнесу.

ІЕ потребує здатності до взаємодії та, у свою чергу, для цього потрібна здатність відчувати навколишнє середовище та реагувати на нього (наприклад, розуміти мову та говорити). Щоб керувати різними рівнями фізичної підготовки, необхідно:

- керувати кожною взаємодією та вирішувати, як реагувати (якщо взагалі) – кожен тип взаємодії може вимагати різних конкретних можливостей;
- визначити пріоритетність реакції на різні взаємодії одного або різних типів;
- реагувати на зміни навколишнього середовища – це пріоритет для компаній, оскільки світ стає все більш цифровим і впровадження штучного інтелекту та машинного навчання набуває популярності [11].

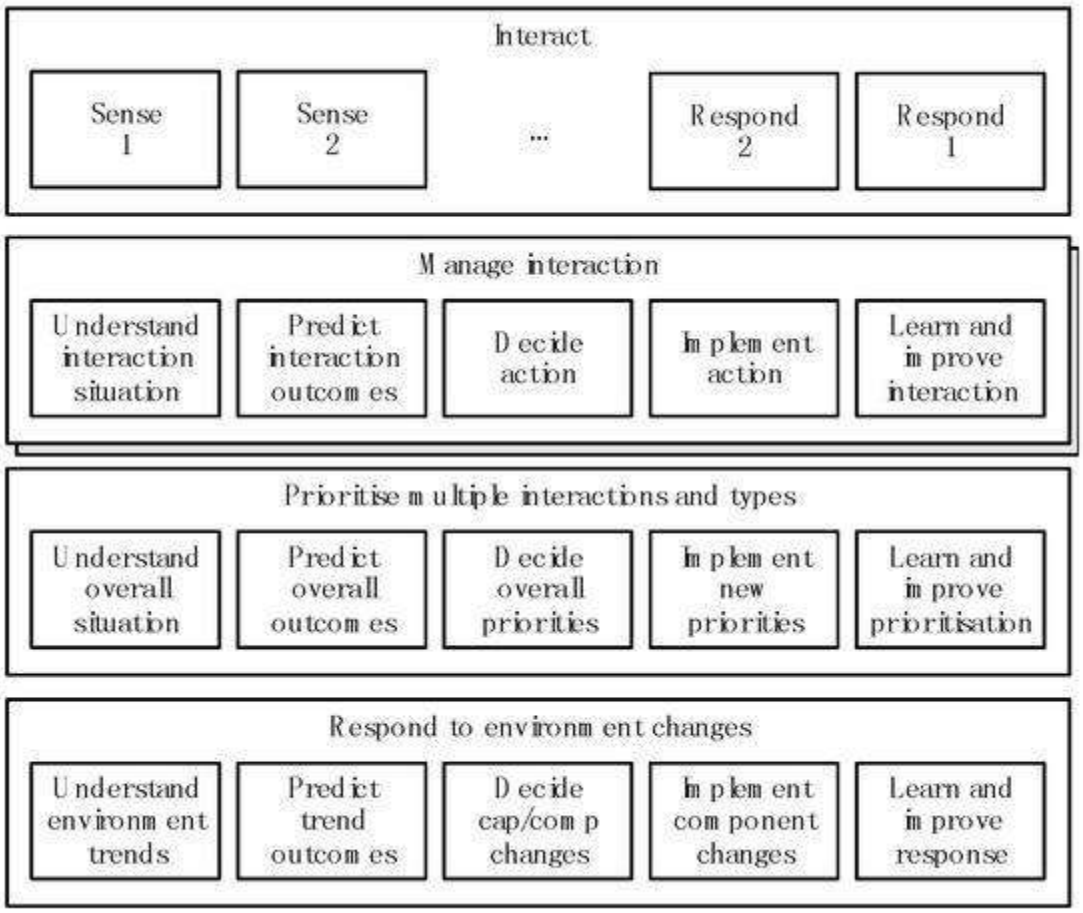

Рис. 1.7. Модель інформаційних можливостей.

У кожному випадку існує п’ять етапів процесу, які застосовуються на відповідному рівні, включаючи:

- зрозуміти ситуацію – визначити, що є релевантним (відрізнити сигнал від шуму), інтерпретувати релевантну інформацію в навколишньому середовищі, зв’язавши її з інформацією в пам’яті, і дистилювати її до відповідного рівня для аналізу та прийняття рішень;
- пов’язати ситуацію з потенційними результатами та зрозуміти їх відносну сприятливість;
- вирішити, як відповісти;
- впровадити зміни в інформаційному плані це означає перетворення рішення в інструктивну інформацію;
- вчитися і вдосконалюватися.

Інформація та її обробка підлягає багатьом обмеженням – вони детально обговорюються в [5 , 6 , 7 , 8]. Ці обмеження виникають через те, що під впливом тиску відбору важко пов’язати стан середовища з майбутніми результатами та необхідними діями для досягнення сприятливих результатів. Вплив завжди полягає в тому, що досконалість неможлива, і під різними тисками відбору існують компроміси щодо різних компонентів придатності.

Перша проблема комбінаторна. Кількість можливих станів середовища, результатів і взаємозв’язків між ними величезна, і в кожній взаємодії вони повинні бути зведені ІЕ до однієї дії (включно з можливістю бездіяльності). Основою для подолання цієї проблеми є фундаментальні характеристики інформації – символи та способи їх асоціювання різними способами.

Друга проблема полягає в тому, як знайти компроміс між показниками інформації (такими як швидкість, тертя та якість [5 , 6]), необхідними для підтримки сприятливих результатів. Ця проблема розбивається на кілька підпроблем:

- які властивості вимірювати в середовищі і яку якість?
- як подолати складності, пов’язані з інтерпретацією, висновками та інструкціями – як розробити короткі шляхи для забезпечення необхідної якості?
- як подолати суперечності на різних рівнях – між різними інформаційними показниками, між потребами сьогодення та потребами майбутнього, між потребами різних типів взаємодії, між різними екосистемами (особливо на кордонах екосистем) і як підтримувати ІЕ у відповідності з придатністю, навіть коли вимоги середовища змінюються?
- як поставити під сумнів досягнутий рівень якості – як екосистема може застосувати власний тиск відбору для забезпечення якості?

Останньою проблемою є архітектурна (в сенсі [12]) – який шаблон компонентів найкращий і як він має змінюватися з часом?

Шляхи, якими ці проблеми вирішуються в різних екосистемах, визначають умовності екосистеми та докладні механізми відбору, які застосовуються.

Інформаційне перевантаження багато обговорювалося [28], але це лише один із симптомів глибшої проблеми існує неймовірно велика кількість вимірних потенційних станів середовища, потенційних результатів і зв'язків між ними.

Стан середовища або, власне, будь-який зріз, навіть якщо виміряно з відносно низькою якістю, непросто маніпулювати та обробити – існує (потенційно велика) кількість властивостей та їхніх значень, які слід враховувати. Таким чином, існує значна економія обробки (і зменшення тертя та темпу), якщо замість цього використовувати простий ідентифікатор, пов'язаний із фрагментом. Якщо ідентифікатор певним чином пов'язаний із зрізом і зрозуміло, що означає ідентифікатор (за потреби, посилаючись на властивості зрізу), тоді обробка буде значно спрощена. Тому не дивно, що ідентифікатори широко поширені в зберіганні та обробці інформації у вигляді символів або наборів символів. Природа символу не має значення (а той факт, що символи можуть бути довільними, є фундаментальним принципом семіотики [29]). Важливо те, що символ можна приєднати, якщо потрібно, до фрагмента, до якого він підключений, і що його можна відрізнити від інших символів. (Будь ласка, зауважте, що вимога, щоб символи можна було легко розрізнити, передвіщає одну з переваг цифрового світу – див. обговорення в [30].)

Це допомагає вирішити проблему обробки, але якщо нам потрібен символ для кожного можливого зрізу, то ми не уникнули комбінаторної проблеми повністю. Було б також корисно, щоб символ застосовувався до набору фрагментів, які мають щось спільне – які відповідають певному набору обмежень. Ось, наприклад, те, як працює мова: дієслова пов'язані з наборами фрагментів подій зі спільними властивостями; прикметники стосуються наборів фрагментів із деякими спільними властивостями тощо.

Включення набору є двійковим вхід або вихід. Тому, вибравши цей шлях для вирішення комбінаторної проблеми, використання символів створило фундаментальну проблему з інформацією, яка лежить в основі багатьох обмежень, п. Екстремальна форма проблеми дискримінації виникає внаслідок хаотичних ефектів , у яких довільно малі зміни можуть призвести до великих результатів. Як показано в , більшість рутинної обробки інформації повністю ігнорує це питання. У термінах машинного навчання проблема дискримінації перетворюється на рівні ризику та толерантності, пов’язані з помилковими позитивними та помилковими негативними результатами [32].

Використання символів дає змогу застосувати інший трюк: символи можуть стосуватися інших символів, а не лише наборів фрагментів (це тому, що символи відповідають наборам фрагментів, які відповідають обмеженням певної екосистеми [5]). Тому, як описано в [6 , 7], ми повинні бути обережними, щоб розрізнити фрагменти вмісту – ті, що інтерпретуються як символи в екосистемі (ІЕ в екосистемі) – і сегменти подій – ті, які цього не роблять.

Комбінаторний виклик посилюється, коли ми розглядаємо різні типи взаємодії та зміни середовища. Для кількох типів взаємодії може знадобитися більше властивостей фрагментів, більше символів і, можливо, різні екосистеми та умовності екосистем. Крім того, розпізнавання змін у середовищі вимагає здатності зберігати та обробляти історичні дані, які дозволять ідентифікувати тенденції (доступ до цих історичних «великих даних» був одним із рушійних сил машинного навчання).

Це призводить до іншого аспекту комбінаторного виклику: як мають бути структуровані інформація та компоненти, щоб забезпечити придатність на різних рівнях (включаючи адаптивність). Пам’ятаючи, що інформація пов’язує стани, результати та дії, тут існує ключовий принцип структурування (зазвичай використовується в індустрії технологій [9]). Відокремлення двох компонентів дає змогу змінити один, не змінюючи

інший (роз'єднання обговорюється докладніше в обговоренні шаблонів компонентів нижче), і це вимагає, щоб вони були роздільними в певному сенсі.

Однією зі стратегій вирішення комбінаторної проблеми є збільшення обчислювальної потужності, і це саме те, що закон Мура [33] надав для машинного навчання (у поєднанні з доступом до доступу до великих обсягів даних – так звані «великі дані»). Це збільшення потужності та доступу до даних стало одним із рушійних сил нинішнього буму штучного інтелекту, але поки ще далеко від вирішення комбінаторної проблеми, навіть якщо не враховувати інші труднощі, описані нижче.

Вплив комбінаторної проблеми полягає в тому, що обробка інформації використовує сувору підмножину наявних властивостей станів середовища, робить компроміси щодо якості та може бути пов'язана із строгою підмножиною можливих результатів. Іншими словами, уся обробка інформації має точку зору. Це рутина в повсякденному житті, наприклад:

- маючи однакові докази, різні політичні партії діходять дуже різних висновків щодо правильного курсу дій у будь-якому випадку;
- у судових справах обвинувачення та захист представляють різні точки зору;
- навіть у науці існують дискусії щодо достоїнств окремих гіпотез (це представлено, наприклад, у філософії науки Куна [34]).

Вимірювання полягає у перетворенні станів навколишнього середовища у властивості та значення або більш абстрактний вміст (звичайно, відповідно до переважаючих умов екосистеми). Як це пов'язано з фітнес-заходами?

Коли розглядаються кілька типів взаємодії, виникає додатковий вимір – якою мірою вимірювання, необхідні для одного типу взаємодії, можна використовувати для іншого – якщо різні взаємодії використовують різні конвенції екосистеми, чи можна вимірювати й обробляти властивості однаково, і які наслідки це матиме, якщо ні? Це поширена проблема в

організаціях – якість інформації, необхідної для успішного завершення процесу, може бути набагато нижчою, ніж потрібна для точного звітування.

Машинне навчання може бути одним із рушійних сил покращення вимірювань для організацій, оскільки розпізнавання шаблонів і його автоматизація є фундаментальними принципами в цій дисципліні [32]. Машинне навчання може покращити темп, зменшити тертя та, у деяких випадках, покращити якість також завдяки автоматизації навчання на основі даних високої якості (хоча були деякі значні труднощі).

З використанням цих закономірностей можуть виникнути труднощі як для людей, так і для машин. Як зазначає Канеман [4] щодо наших вроджених підсвідомих реакцій (те, що він називає Системою 1): «Система 1 абсолютно нечутлива як до якості, так і до кількості інформації, яка породжує враження та інтуїцію». Як каже Даффі в [35], «чим більш поширеною є проблема, тим більша ймовірність, що ми сприймемо її як норму».

Машинне навчання [27] знаходить і використовує деякі з цих закономірностей, але було предметом деяких широко розголошених проблем, пов’язаних із упередженнями [17 , 18] (хоча виявлені упередження в деяких випадках були меншими, ніж люди демонструють [18]).

Машинне навчання базується на подібності, тому ця категоризація викликає запитання. Для яких типів обробки інформації машинне навчання є найбільш прийнятною технікою, а коли підходять інші методи? Зокрема, коли симуляція (пов’язана з моделюванням причинно-наслідкових зв’язків) більш доцільна?

Існують невизначеності щодо інтерпретації, висновків та інструкцій, викликані обмеженнями якості інформації. Однак взаємодія призводить до одноразової дії ІЕ (де це включає можливість узагалі не діяти), а вивчення діапазону потенційних результатів і дій збільшує тертя. Тому в багатьох випадках інтерпретація та логічний висновок призначені для отримання єдиної відповіді, а потенційно складний розподіл можливостей згортається

до єдиного результату. Якщо це згортання відбувається в кінці обробки, це може не вплинути на якість. Однак, якщо це відбувається на кількох етапах під час обробки, то, швидше за все, так і буде.

Організації регулярно використовують правила (так звані бізнес-правила). Бізнес-процеси втілюють ці бізнес-правила в двох значеннях. У великому масштабі процес визначає правила, за якими бізнес має намір здійснювати діяльність (наприклад, як управляти страховим відшкодуванням). Крім того, у більш детальному сенсі бізнес-правила фіксують, як виконати певні кроки (наприклад, питання, які потрібно поставити про характер претензії). Машинне навчання може покращити обидва ці аспекти. У першому випадку контекст процесу (наприклад, інформація про претензію) може змінити відповідний наступний крок (наприклад, відповідний рівень оцінки ризику для застосування). Таким чином, замість фіксованого набору кроків, зафіксованих у карті процесу, процес може стати сумішшю фіксованих кроків і чогось схожого на кінцевий автомат [39] або, в деяких випадках, просто кінцевий автомат. Ця зміна залежить від постійного усвідомлення ситуації, яке може використовувати машинне навчання як інструмент вимірювання. Крім того, машинне навчання може з часом уточнювати бізнес-правила на основі розвитку зв’язку між правилами та фітнес-цілями (наприклад, компроміс між якістю та тертям або темпом). Може бути доцільно змінити правила (змінивши питання, які потрібно поставити в цьому прикладі), коли буде отримано більше інформації про ефективність правил або стане можливим більш конкретно налаштувати правила для окремих прикладів.

У більш загальному плані може існувати те, що ми можемо назвати конфліктом інтересів між вузькою та широкою відповідністю, особливо коли природа якості, пов’язаної з вузькою відповідністю, не відповідає якості, пов’язаній із широкою відповідністю. У [42] автор наводить приклади впливу конфлікту інтересів на науку. Було кілька широко розголошених прикладів машинного навчання [18]. У цих випадках вузька придатність

визначається з точки зору даних, які використовуються для створення навченої поведінки, але самі дані можуть включати упередження людини. Як наслідок, вузька придатність (пов’язана з даними про тренування) не враховує етичні та соціальні питання, а широка придатність знижується.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ З ФАЙЛАМИ

2.1. Визначення вимог до системи

Сама по собі технологія не може замінити ефективні процеси управління даними, такі як проактивна атака на якість даних, переконання, що кожен розуміє свої ролі та обов'язки, побудова організаційних структур, таких як ланцюги постачання даних, і встановлення спільних визначень ключових термінів. Але ШІ є цінним ресурсом, який може значно підвищити як продуктивність, так і користь, яку компанії отримують від своїх даних.

Протягом багатьох років примітивні форми штучного інтелекту використовувалися для оптичного розпізнавання символів (OCR) для вилучення важливих даних із таких елементів, як банківські чеки чи адресовані конверти. OCR стало настільки поширеним, що ми більше не думаємо про такі можливості, як ШІ. Новіші системи штучного інтелекту розширили OCR за допомогою моделей глибокого навчання, які тепер стають здатними точно читати людський почерк.

Десятиліттями компаніям не вистачало точних вказівок щодо розміщення ключових даних у їхніх системах і записах. На щастя, каталогізація даних з'явилася за останні кілька років як важлива допомога для відстеження цього матеріалу. Створення та підтримка таких каталогів в актуальному стані, однак, було трудомістким.

ШІ може автоматизувати пошук у різних сховищах даних і автоматично створювати каталоги. Системи штучного інтелекту можуть отримувати будь-які метадані, які існують у системній документації. ШІ також може описати родовід даних – звідки вони походять, хто їх створив, як вони були змінені та де вони зараз знаходяться.

Але хоча створювати каталоги та інформацію про походження даних легше за допомогою штучного інтелекту, компаніям все одно доводиться боротися з безладдям у своїх існуючих середовищах даних. Багато компаній чинили опір створенню каталогів за допомогою традиційних трудомістких методів, тому що вони не хотіли розкривати масштаби архітектурного безладу або тому, що вони хотіли зачекати, доки дані стануть краще організованими та якіснішими, перш ніж докладати великих зусиль. Однак простота створення та оновлення каталогів за допомогою штучного інтелекту означає, що компанії можуть поєднувати легший доступ до інформації з процесами постійного вдосконалення даних.

Збереження безпеки та конфіденційності даних є критичними питаннями для будь-якої організації сьогодні. Запобігання хакерським атакам, вторгненням і відмові в обслуговуванні були в основному людською діяльністю з моменту зародження професії захисту даних.

AI може допомогти з багатьма з цих функцій. Це корисно, наприклад, для розвідки загроз спостереження за зовнішнім світом; синтез сигналів загрози, акторів і мови; і передбачити, хто що кому може робити. Розвідка загроз на основі штучного інтелекту є відповіддю на численні виклики, з якими стикаються професіонали з кібербезпеки, включаючи велику кількість суб’єктів загрози, величезну кількість, здавалося б, безглуздої інформації та брак кваліфікованих фахівців.

Провідні рішення використовують машинне навчання для автоматизації збору даних безпеки в багатьох внутрішніх і зовнішніх системах, створення структурованих даних із неструктурованих форматів і визначення загроз, які є найбільш надійними. Системи штучного інтелекту можуть передбачати ймовірні шляхи атак на основі попередніх шаблонів атак і визначати, чи надходять нові загрози від раніше відомих акторів чи від нових. Враховуючи кількість хибно-позитивних загроз кібербезпеці в кількох непов’язаних системах безпеки, комбінація правил прийняття рішень і

моделей машинного навчання може визначити пріоритетність або сортувати загрози для розслідування людьми.

ШІ також можна використовувати для виявлення внутрішніх загроз шахрайства або недотримання правил. Ця можливість становить особливий інтерес для суворо регульованих галузей, таких як банківська справа та інвестиції. Програмне забезпечення штучного інтелекту відстежує цифрові комунікації в організації та виявляє підозрілу мову або моделі поведінки. Звичайно, необхідне людське розслідування, щоб підтвердити зловживання співробітників або клієнтів.

Системні вимоги

Windows

Microsoft Windows 10/8/7/Vista/2003/XP (в т.ч.64-розрядна версія)

Мінімум 1 ГБ оперативної пам'яті

Рекомендовано 2 ГБ оперативної пам'яті

Мінімальна роздільна здатність екрана 1024x768

Python 2.4 або вище, Jython, PyPy або IronPython

Linux

Мінімум оперативної пам'яті 512 МБ, рекомендовано 1 ГБ оперативної пам'яті

Мінімальна роздільна здатність екрана 1024x768

Python 2.4 або вище, Jython, PyPy або IronPython

OC X

Mac OS X 10.8 або новішої версії

Мінімум 1 ГБ оперативної пам'яті

Рекомендовано 2 ГБ оперативної пам'яті

Python 2.4 або вище, Jython, PyPy або IronPython

2.2. Архітектура програмного забезпечення

Програма створена як єдине ціле в монолітній архітектурі, що використовує однакові ресурси та пам’ять. Підтримувати та оновлювати кодову базу може бути складно, оскільки вона часто величезна та складна. Його можна запускати на одній системі або групі машин і його легко розгорнути.

Програми малого та середнього розміру, які мають просту функціональність і не потребують великої масштабованості чи модульності, часто використовують монолітну архітектуру. Однак із зростанням складності та масштабу програми стає складніше адмініструвати й підтримувати її, що може призвести до подовження часу розробки, великих витрат і обмеженої масштабованості.

Монолітний дизайн втратив прихильність, оскільки архітектура мікросервісів набула популярності, оскільки вона обмежує можливість масштабування, оновлення та модифікації окремих компонентів програми. Тим не менш, монолітна конструкція все ще використовується багатьма застарілими системами та програмами.

Шаблон проектування програмного забезпечення, відомий як монолітна архітектура, передбачає створення цілісної автономної програми. У цій архітектурі є три основні рівні.

Рівень презентації - цей рівень обробляє інтерфейс користувача та взаємодію з ним. Він включає всі аспекти, які може бачити користувач, такі як графічні елементи, веб-сторінки та форми. Через API та інші інтерфейси рівень презентації з'єднується з іншими рівнями програми.

Прикладний рівень - бізнес-логіка програми міститься на цьому рівні. Він охоплює всі елементи, які допомагають функціоналу програми добре працювати, включаючи обробку даних, робочі процеси та алгоритми. Щоб отримати доступ до даних і змінити їх, прикладний рівень взаємодіє з рівнем даних.

Рівень доступу до даних - управління зберіганням і пошуком даних підпадає під компетенцію цього рівня. Об’єкти доступу до даних, моделі даних і з’єднувачі даних – це лише кілька прикладів компонентів, які взаємодіють з базою даних.

Усі ці рівні тісно пов’язані та розгорнуті як єдиний виконуваний файл у монолітній архітектурі.

У цієї стратегії є переваги та недоліки.

Переваги монолітної архітектури:

− Оскільки існує лише одна кодова база для обробки, розробка та розгортання спрощуються.

− Оскільки всі компоненти об’єднані в єдиний блок, тестування та налагодження є простими.

− Низька затримка, оскільки кожен компонент знаходиться на одному сервері.

Недоліки:

− Обслуговування є складним завданням, оскільки будь-яке оновлення потребує перебудови та повторного розгортання всієї програми.

− Обмежене масштабування, оскільки всю програму потрібно масштабувати одночасно.

− Оновлення небезпечне, оскільки навіть невеликі зміни можуть мати великий вплив на всю програму.

Монолітна архітектура, незважаючи на свої недоліки, все ж використовується в деяких ситуаціях, таких як системи малого та середнього розміру з обмеженою функціональністю або програми, які не потребують значної масштабованості чи модульності. Однак архітектура мікросервісів стала більш кращим рішенням, оскільки додатки зростають у складності та розмірі.

Типова програма з монолітною архітектурою має зовнішній інтерфейс користувача, інтерфейс на стороні сервера та кодову базу (база даних, що

підтримує програмне забезпечення). Якщо ваші потреби прості та вам потрібен швидкий поворот, монолітний – очевидний вибір.

Прикладом може бути, якби бізнес був стартапом із великою кількістю чудових ідей, але мало ресурсів. Щоб відкрити свій бізнес, почати масштабуватись і привернути увагу інвесторів, вам потрібно якомога швидше вивести свій продукт на ринок.

Монолітна архітектура також є хорошим вибором, якщо ви знаєте, що ваш продукт не потребуватиме великого масштабування в майбутньому. У цьому випадку ви можете зробити все простим і економічно ефективним без шкоди для якості.

Альтернативним підходом до розробки додатків є архітектура мікросервісів. Також відомий просто як мікросервіси, цей підхід створює більший додаток із кількох модульних сервісів, які спілкуються через API. Ці слабко пов’язані служби, які можна розгортати незалежно, використовують незалежні бази даних і кодування, а також певну бізнес-ціль.

За кожну послугу, яка використовується в архітектурі Microservices, зазвичай відповідає окрема команда. Це дозволяє оновлювати, тестувати, розгортати та масштабувати індивідуально для кожної служби та створювати керовану складність. Мікросервіси мають багато застосувань, але загальним є реструктуризація застарілих систем.

Мікросервіси також корисні, якщо потрібно виконувати значну кількість потокових даних і обробки в реальному часі, як, наприклад, відомі служби потокового передавання, онлайн-банкінг або програми електронної комерції. Мікросервіси набагато краще справляються з таким навантаженням даних, ніж монолітні програми.

Гарне монолітне застосування має масу переваг.

1. Проста розробка: монолітна архітектура є перевагою, якщо ви хочете розробити цілу програму та швидко вивести її на ринок. Невелика команда може швидко об’єднатися та створити виконувану програму за допомогою

монолітної системи. Це робить монолітну архітектуру ідеальною для стартапів без великих бюджетів на розробку програмного забезпечення.

2. Легке розгортання: монолітна архітектура не така складна, як мікросервіси. У ньому менше рухомих частин, тому менше компонентів для керування та ремонту. Автономна природа монолітної програми полегшує її розгортання, керування та підтримку, ніж рішення мікросервісів.

3. Нескладне тестування та налагодження: оскільки додаток встановлено як єдине ціле та працює разом як єдине ціле, ви можете виконувати тестування монолітної архітектури та налагодження швидко та легко з центральної системи реєстрації.

З мікросервісами всі частини додатків потрібно тестувати окремо, від архітектури програмного забезпечення до таких речей, як кешування, залежності та доступ до даних. Ви також повинні перевірити, чи всі служби працюють належним чином. Це може зробити тестування мікросервісів дорогим і трудомістким ще до початку самого налагодження.

Хоча не варто ігнорувати всі переваги монолітної архітектури, важливо також враховувати недоліки. Якщо наведені нижче недоліки монолітної архітектури створюють проблему для вашого бізнесу, тоді монолітні програми вам не підходять.

1. Менша масштабованість: оскільки програмне забезпечення монолітної архітектури тісно пов’язане, його може бути важко масштабувати. Якщо ви хочете додати нові функції або ваша кодова база розшириться, вам потрібно буде взяти з собою всю архітектуру.

Окрім витрат часу та ресурсів, це також може призвести до перебоїв у безперервній доставці.

2. Нездатність адаптуватися до нових технологій: як згадувалося раніше, монолітні програми тісно пов’язані між собою. Візьмемо, наприклад, музичний додаток. Каталог тісно пов'язаний з сервісами покупки та гри.

Хоча в деяких випадках це корисно, це також означає, що важко запроваджувати нові технології чи веб-сервіси без демонтажу всієї програми.

3. Висока залежність між функціями: Монолітні програми також можуть зіткнутися з розробкою програмного забезпечення та проблемами простою через їх тісну залежність.

	Monolithic	Microservices
Deployment	Simple and fast deployment of the entire system	Requires distinct resources, making orchestrating the deployment complicated
Scalability	It is hard to maintain and handle new changes; the whole system needs to be redeployed	Each element can be scaled independently without downtime
Agility	Not flexible and impossible to adopt new tech, languages, or frameworks	Integrate with new technologies to solve business purposes
Resiliency	One bug or issue can affect the whole system	A failure in one microservice does not affect other services
Testing	End-to-end testing	Independent components need to be tested individually
Security	Communication within a single unit makes data processing secure	Interprocess communication requires API getaways raising security issues
Development	Impossible to distribute the team's efforts due to the huge indivisible database	A team of developers can work independently on each component

Рис. 2.1 – Порівняння монолітної архітектури та мікросервісної архітектури

Розглянемо приклад музичної програми. Оскільки функції каталогу, гри та покупки настільки залежні одна від одної, якщо одна вийде з ладу, інші вийде з ладу разом з нею.

Хоча деякі люди можуть лише захотіти прослухати треки, які вони вже мають у своєму каталозі, тобто вони не використовують функцію покупки, якщо ця функція зламається, уся програма стане функціонально непридатною, доки не буде вирішено проблему з оплатою.

Мікросервіси мають багато можливостей. Ось кілька причин, чому багато компаній обирають їх для підвищення своїх бізнес-можливостей.

1. Незалежні служби: в архітектурі мікросервісів кожна служба розробляється незалежно від інших, а бізнес-логіка розподіляється на різних платформах. Це означає, що робочі процеси для однієї служби не впливають на розробку інших. Крім того, такі ресурси, як інструменти розробки, не залежать від нерелевантних функцій. Натомість кожна служба отримує повну увагу своєї команди. Це забезпечує швидку та ефективну незалежну розробку з подальшим простим об'єднанням служб.

2. Забезпечує гнучку розробку: оскільки кожна служба розробляється незалежно, ви можете вибрати стек технологій і мову програмування, які найкраще відповідають кожній функції. Це означає, що для кожної окремої послуги можна використовувати найкращі інструменти. Навпаки, монолітна архітектура програмного забезпечення іноді змушує служби використовувати універсальний підхід. Така гнучкість дозволяє швидше та ефективніше розробляти програми.

3. Масштабована та надійна: монолітну технологію важко оновлювати та масштабувати. Це не стосується гібридних хмарних мікросервісів. Через слабкий зв'язок кожної функції можна легко оптимізувати, тестувати, налагоджувати та виправляти функції незалежно одна від одної.

Цей слабкий зв'язок робить служби архітектури мікросервісів більш надійними. Один збій служби не виводить із ладу весь додаток. Якщо зі шлюзом API щось піде не так, інші служби можуть працювати незалежно.

Якщо функції покупки та завантаження вийшли з ладу, клієнти все одно зможуть отримати доступ до інтерфейсу користувача та відтворювати музику, якою вони вже володіли.

Однак, як і монолітна архітектура, також повинні ретельно розглянути недоліки мікросервісів.

1. Витрата часу та ресурсів: хоча індивідуальна автономія кожного мікросервісу може пришвидшити розробку, вона також може її сповільнити.

Все залежить від того, наскільки складний додаток ви хочете створити. Це пояснюється тим, що хоча розробка окремого мікросервісу зазвичай є швидшою та ефективнішою, ніж розробка того самого сервісу в монолітному контексті, реалізація багатьох різних мікросервісів і об'єднання їх усіх може зайняти значну кількість часу. Крім того, хоча мікросервіси дозволяють використовувати оптимальні інструменти для кожної служби, об'єднання індивідуальних стеків технологій може вимагати багато ресурсів і досвіду.

2. Складне розгортання: після того, як ви розробили кожен мікросервіс, його потрібно інтегрувати у функціональну програму, перш ніж їх можна буде розгорнути. Це може бути складним процесом. Для початку наскрізний (у якому кожна мікросервіс потребує перевірки або авторизації для продовження) може бути складним. З монолітною архітектурою тісний зв'язок означає, що авторизація для однієї служби зазвичай працює як авторизація в усіх напрямках. Це не стосується мікросервісів, і щоб зробити це таким, вам потрібно виконати складну роботу зі зв'язування. Загалом, розгорнути архітектуру мікросервісу не завжди легко, і це стає важче, чим складнішим є ваш додаток.

3. Комплексне тестування: кожну службу в мікросервісній програмі потрібно тестувати окремо. Після завершення всіх тестів окремих послуг вам потрібно перевірити, як вони працюють у цілому. Важливо провести тестування перед випуском будь-якого продукту на ринок (а потім проводити регулярні тести функціональності, поки продукт залишається в обігу). Якщо ви хочете, щоб ці тести проходили якомога швидше, мікросервіси можуть вам не підійти.

2.3. Використані технології та інструменти

Python – це універсальна та потужна мова програмування, яка містить програми в різних областях. Ось деякі з найвідоміших застосувань. Такі реймворки, як Django, Flask і FastAPI, дозволяють розробляти динамічні веб-

сайти та веб-додатки. Розробка бекенда для API та обробки на стороні сервера.

Інтеграція з базами даних за допомогою таких бібліотек, як SQLAlchemy та Django ORM. Такі бібліотеки, як Pandas, NumPy і SciPy, для обробки даних і чисельних обчислень. Інструменти візуалізації, такі як Matplotlib, Seaborn і Plotly. Використовується в дослідницькому аналізі даних (EDA) і створенні прогнозних моделей. Такі фреймворки, як TensorFlow, PyTorch і Scikit-learn для розробки моделей ШІ.

Обробка природної мови (NLP) з використанням таких бібліотек, як NLTK і spaCy.

Застосування в комп'ютерному зорі, навчанні з підкріпленням і системах рекомендацій.

Автоматизація повторюваних завдань за допомогою таких бібліотек, як PyAutoGUI і Selenium.

Написання сценаріїв для адміністрування системи, роботи з файлами та обробки даних. Веб-збирання за допомогою BeautifulSoup і Scrapy. Такі бібліотеки, як Pygame, для розробки 2D-ігор. Серверна логіка для ігрових серверів і багатокористувацьких функцій.

Використовується в програмуванні мікроконтролерів за допомогою MicroPython і CircuitPython. Інтеграція з апаратними датчиками та пристроями IoT.

Обчислювальна біологія, фізичне моделювання та інші наукові галузі з використанням SciPy та SymPy.

Такі фреймворки, як Jupyter, для інтерактивних обчислень і візуалізації.

Алгоритмічна торгівля з використанням Quantlib і backtrader.

Аналіз даних для фінансового моделювання та управління ризиками.

Виявлення шахрайства та прогнозна аналітика.

Розробка GUI за допомогою Tkinter, PyQt і Kivy.

Створення легких міжплатформних додатків для настільних ПК.

Написання інструментів для тестування на проникнення та аналізу вразливостей за допомогою таких бібліотек, як Scapy і Paramiko.

Автоматизація завдань безпеки та розробка власних експлойтів.

Зручна для початківців мова, яка часто використовується для навчання концепціям програмування.

Допоміжні інструменти та ресурси, такі як Jupyter Notebooks, для інтерактивного навчання.

Написання скриптів та інструментів розгортання для хмарних сервісів, таких як AWS, Azure та Google Cloud.

Автоматизація інфраструктури за допомогою таких фреймворків, як Ansible і Terraform.

Python став однією з найпопулярніших мов програмування за останні роки, і це не дарма. Його простота, універсальність і розгалужена екосистема роблять його найкращим вибором для розробників у різних галузях. У цьому есе досліджуються переваги Python порівняно з іншими мовами програмування, зосереджуючись на його функціях, зручності використання та застосуванні.

Однією з найважливіших переваг Python є його простота. Розроблений з урахуванням зручності читання, Python використовує простий синтаксис, який дуже нагадує природну мову. Ця функція скорочує криву навчання для початківців і дозволяє досвідченим розробникам писати чіткий і стислий код. Наприклад, завдання, які можуть вимагати кількох рядків коду в таких мовах, як Java або C++, часто можна виконати за допомогою одного рядка в Python. Ця простота сприяє більш ефективному процесу розробки та мінімізує ймовірність помилок.

Python є мовою загального призначення, тобто її можна використовувати для широкого кола програм. Незалежно від того, створюєте ви веб-додатки, аналізуєте дані, розробляєте моделі машинного навчання чи автоматизуєте завдання, Python надає інструменти, необхідні для виконання роботи. Його широка стандартна бібліотека та доступність пакетів сторонніх

розробників через індекс пакетів Python (PyPI) роблять його універсальним рішенням для багатьох програмних потреб.

Наука про дані та машинне навчання : такі бібліотеки, як NumPy, pandas і scikit-learn, є основними для аналізу даних і машинного навчання.

Веб-розробка - фреймворки, такі як Django та Flask, дозволяють швидко розробляти веб-додатки.

Автоматизація - такі інструменти, як Selenium і PyAutoGUI, допомагають автоматизувати повторювані завдання.

Наукові обчислення - SciPy і Matplotlib підтримують наукові дослідження та візуалізацію.

Штучний інтелект - TensorFlow і PyTorch є провідними фреймворками для розробки ШІ.

Ця екосистема зменшує потребу писати код з нуля, дозволяючи розробникам зосередитися на вирішенні проблем, а не винаходити колесо.

Python є кросплатформною мовою, тобто вона може працювати в різних операційних системах, включаючи Windows, macOS і Linux, без змін. Ця гнучкість спрощує процеси розробки та розгортання, особливо для програм, призначених для різних користувачів.

Простота Python і широка підтримка бібліотек роблять його ідеальним для швидкої розробки та створення прототипів. Розробники можуть швидко створювати та тестувати ідеї, що дозволяє повторювати вдосконалення та швидше виводити їх на ринок. Ця гнучкість особливо цінна в таких галузях, як технології та стартапи, де інновації та швидкість мають вирішальне значення.

Python відмінно підходить для інтеграції з іншими технологіями та мовами. Його сумісність дозволяє розробникам викликати бібліотеки та функції, написані такими мовами, як C, C++ і Java. Ця функція особливо корисна для використання існуючих кодових баз або оптимізації важливих для продуктивності компонентів програми.

Python часто є мовою вибору для навчання програмуванню через його простий синтаксис і легкість читання. Навчальні заклади та онлайн-платформи для навчання часто використовують Python для ознайомлення з концепціями кодування. Його доступність знижує бар'єри для початківців, заохочуючи більше людей входити у сферу програмування.

Можливість адаптації Python зробила його провідною мовою для нових технологій, таких як штучний інтелект, машинне навчання та наука про дані. Його фреймворки та бібліотеки розроблені для виконання обчислювальних і аналітичних вимог у цих сферах, дозволяючи розробникам створювати інноваційні рішення. Наприклад, TensorFlow і Keras спрощують розробку нейронних мереж, а pandas і NumPy полегшують маніпулювання даними та аналіз.

Python є відкритим вихідним кодом, тобто його можна вільно використовувати, поширювати та змінювати. Така економічна ефективність робить його привабливим варіантом для стартапів, навчальних закладів і незалежних розробників. Крім того, широка бібліотечна підтримка скорочує час і витрати на розробку, зводячи до мінімуму потребу в індивідуальних рішеннях.

2.4. Реалізація основних функцій

Керування файлами на комп'ютері подібне до зберігання документів у картотеці. Картотечна шафа використовується для зберігання паперових файлів у картонних папках. Таким же чином можемо зберігати файли та папки на комп'ютері.

Операційна система Windows організовує свої диски, папки та файли в ієрархічній структурі папок. Файли зберігаються на комп'ютері в папках. Папки використовуються для організації файлів на комп'ютері, щоб їх було легше знайти. Папка міститиме вкладені папки, а потім файли.

У наступному прикладі пояснимо папку, порівнюючи її з деревом. Структура має форму піраміди, де кожен ряд елементів пов’язаний з елементами під ним. Через цю пірамідальну структуру ця ієрархічна структура також відома як «перевернуте дерево».

На рисунку 2.2 показано приклад перевернутого дерева.

Різні диски, як-от різні жорсткі диски, диски CD/DVD, USB-накопичувачі, а також мережеві диски, знаходяться в рядку під коренем. У наступному рядку відображаються папки, пов’язані з певними дисками.

Будь-які вкладені папки та файли, знайдені в папці вище, відображаються в наступному рядку. Цей шаблон продовжується, доки останній рядок не міститиме лише файли.

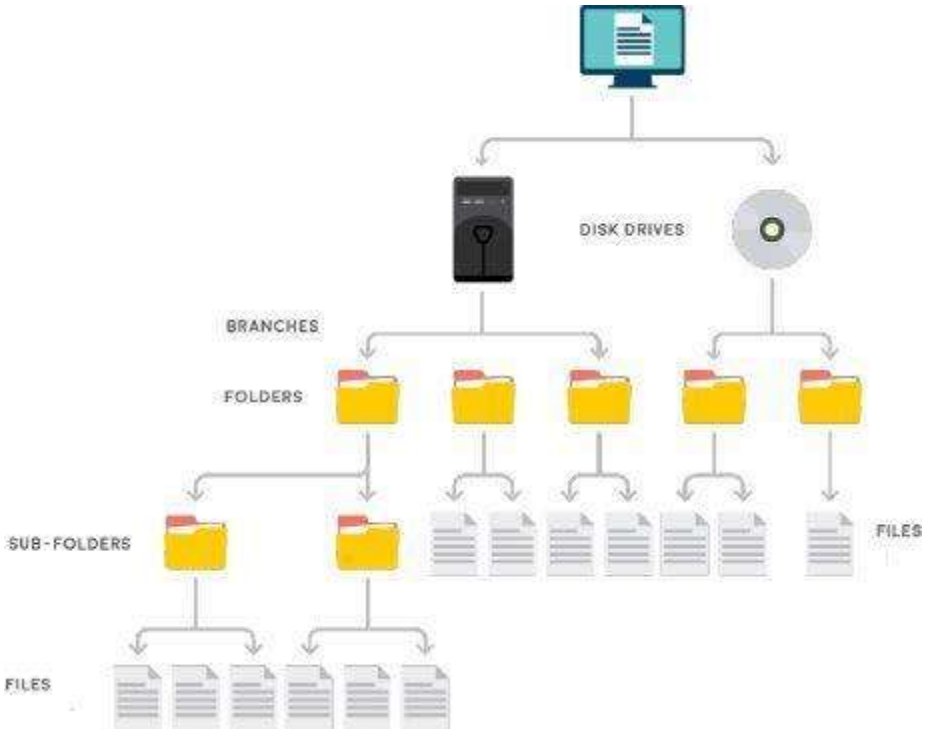

Рис. 2.2 - Ієрархічне дерево

Потім документи утворюють підпапку кореневого каталогу диска C, яка записується як C:\.

Головні функції для реалізації включають:

– Визначення типу документа

- Витягнення ключових слів з файлу та визначення до якої тематики відноситься файл(технічна література,резюме і т.д
- Навчання моделі нових типів файлів і т.д
- Можливість визначення графічних файлів(OCR)

Основні папки, такі як робочий стіл, документ або завантаження в операційній системі Windows, здаються сотнями файлів. Оскільки кількість файлів у певній папці збільшується, керувати цими файлами стає складніше. Запропонована система вирішує всі ці проблеми і є першою спробою створення автоматичної системи керування файлами для операційних систем. Запропонована архітектура складається з двох різних модулів, які називаються глобальним і локалізованим модулями, як показано на рис 2.3.

Глобальний модуль відповідає за загальне вивчення категорій і міток категорій. У той же час локальний організатор класифікує та зберігає файли у відповідних папках на основі схожості файлів. Глобальний модуль передає навчену модель нейронної мережі та налаштування параметрів для попередньої обробки даних, вилучення ознак і розкладів усічених сингулярних значень. Підхід кластеризації K-Means використовується для початкової кластеризації файлів, яка налаштована за допомогою оптимізатора Adam для прогнозування кінцевих класів. Коли кількість документів у будь-якій папці збільшується до певного порогу, локальний організатор викликає глобальний організатор, який може реструктуризувати папки та перегрупувати файли у відповідні папки.

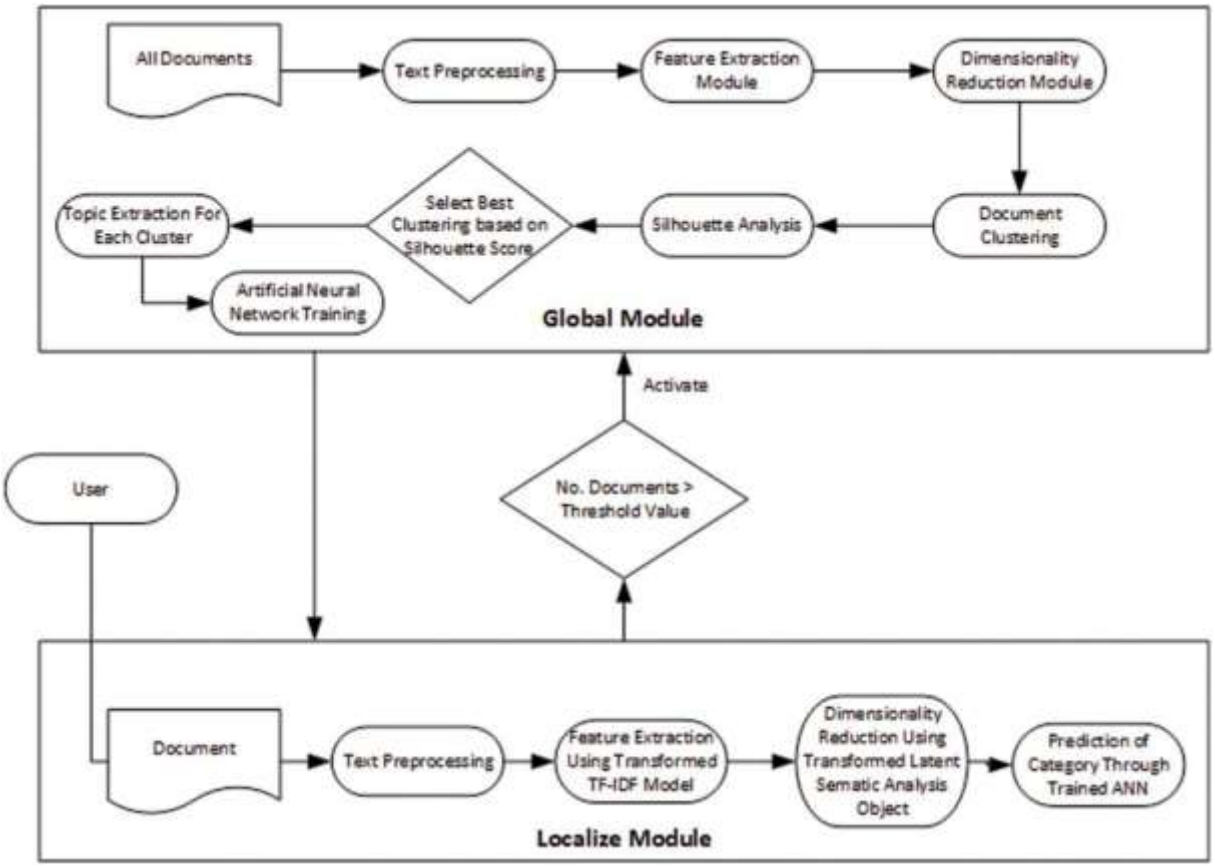

Рис. 2.3 - Архітектура системи автоматичної організації файлів

Глобальний модуль підрозділяється на модуль попередньої обробки, модуль вилучення функцій, модуль зменшення розмірності, модуль кластеризації, модуль аналізу, модуль вилучення тем і навчальний модуль нейронної мережі.

Більшість файлів у комп'ютерній системі зазвичай класифікуються як неструктуровані документи; отже, майже всі файли мають бути попередньо оброблені перед початком фактичної кластеризації. Попередня обробка файлів включає кілька етапів, таких як фільтрація, видалення стоп-слова, токенізація та створення коренів слів документа.

Фільтрація: завдання фільтрації полягає в очищенні даних для всіх посилань, знаків пунктуації, цифр і непотрібних пробілів.

Вилучення стоп-слова: завдання цього підмодуля полягає у видаленні стоп-слів, які не мають жодних цінних значень і збільшують довжину документа без потреби з точки зору аналізу тексту. Бібліотека Natural

Language ToolKit (NLTK) [16] була використана в цьому дослідженні для усунення всіх стоп-слів.

Токенізація - коли стоп-слова видаляються з документа, маркери призначаються всім словам, що залишилися, щоб краще аналізувати текст. Завдання токенізації також виконується за допомогою бібліотеки NLTK.

Корінь -Основа надає корінь кожного слова, щоб знайти важливі терміни, використані в документі. Алгоритм визначення основи Snowball [26] є одним із найпопулярніших підходів створення основи для документів англійською мовою, які використовуються в цьому дослідженні для визначення основи всіх лексем. Після завершення стемінінгу для модулів виділення тем підтримується Словник основних токенів (STD). Кожен запис у цьому словнику містить основу з усіма відповідними лексемами.

Після завершення попередньої обробки файлів основні токени витягуються з файлів. Для цього необхідно представити всі файли у відповідному вигляді. Модель векторного простору є одним із найпоширеніших підходів до представлення документів як векторного простору [20]. Щоб знайти подібність між різними файлами, використовується частота слів. У цьому дослідженні кожен файл Fi розташований у n-вимірному векторному просторі, де всі n вимірів витягуються за допомогою векторизатора термінової частоти, зворотної частоти документа (TFIDF). Це означає, що всі компоненти вектора відображають термін у даному файлі. Значення кожного компонента визначається ступенем асоціації між відповідним файлом і пов’язаним терміном.

Вихід векторизатора TFIDF із пороговими значеннями для mindf і maxdf є матрицею TFIDF форми (18846, 32329), де 18846 є номером. документів 32329 – це функції, витягнуті векторизатором TFIDF.

Файли в комп’ютерній системі користувача можуть містити дані дуже великої розмірності, і це стає серйозною проблемою під час застосування статистичних або машинних методів міркування до таких даних. Наприклад,

модуль вилучення ознак у цьому дослідженні визначив 32329 ознак із заданого набору даних, що є дуже великою кількістю. Складно навчити алгоритм кластеризації або класифікації для такої кількості вхідних змінних. Таким чином, необхідно зменшити витягнуті ознаки, перш ніж алгоритм кластеризації або класифікації можна буде успішно застосувати до набору даних [27]. Зменшення розмірності може бути двома можливими способами. Один із способів – зберегти найбільш відповідні змінні з вихідного набору даних. Другий спосіб полягає в тому, щоб використати надмірність функцій різними способами та знайти менший набір нових змінних. У другому підході визначається менший набір нових змінних. Кожен набір є комбінацією вхідних змінних, що містить ту саму інформацію, що й вхідні змінні.

Важлива матриця V × f документа-терму D, де V не дорівнює f, і неправдоподібно, що D є симетричним. Першим завданням зменшення розмірності є пошук сингулярного розкладання (SVD). Для розкладання D нехай X буде матрицею V × V, де стовпці матриці є ортогональними власними векторами DDT, а Y буде матрицею f × f, де стовпці є ортогональними власними векторами DTD. Ці X і Y представляють лівий сингулярний вектор і правий сингулярний вектор відповідно.

Алгоритм кластеризації k-середніх [21] широко використовується в задачах кластеризації документів і визнаний ефективним підходом до кластеризації великих наборів даних [19]. K у цьому алгоритмі є попередньо визначеною або визначеною користувачем константою, і метою алгоритму є створення розділу даних на основі ознак у k кластерів. Алгоритм K-means добре працює з лінійною часовою складністю для різної кількості документів і особливо забезпечує кращу продуктивність для великої кількості документів [28]. Як зрозуміло з попереднього обговорення, тисячі створених користувачами файлів зберігаються в будь-якій комп'ютерній системі; отже, алгоритм K-Means може забезпечити кращі та ефективні початкові кластери [29]. Основна ідея K-Means полягає у визначенні

центроїдів для всіх кластерів. Ці центроїди представлені k і сформовані таким чином, що всі елементи в кластері тісно пов’язані з центроїдом з точки зору функції подібності. Цю подібність характеристик можна виміряти різними методами, включаючи косинусну подібність і евклідову відстань, для всіх об’єктів у цьому кластері [30]. Результатом алгоритму є набір із K центроїдів кластера, де кожен k має відповідну мітку L. Кластери оновлюються післяμkотримано набір центроїдів, щоб утримувати точки, найближчі до кожного центроїда. Центроїди перераховуються як середнє значення всіх точок. Цей процес повторюється, якщо не спостерігається жодних змін у центроїдах і призначеннях кластерів.

Розпізнавання символів – це техніка виявлення, ідентифікації та сегментації символів, присутніх на зображеннях. Ця техніка розпізнавання символів із зображень перетворює їх у машинозчитувану форму. Процес розпізнавання використовується в області автоматизації та взаємодії між людиною та машиною. Оскільки ця техніка є перевагою в області розпізнавання образів, досліджується протягом багатьох років завдяки її широкому спектру застосувань.

OCR виконує певні етапи обробки, які включають сканування вхідного зображення, попередню обробку, сегментацію, виділення ознак, класифікацію, розпізнавання та постобробку. Відскановані зображення містять шум через додаткові деталі, тому етап попередньої обробки є вимогою для видалення шуму за допомогою різноманітних фільтрів, таких як фільтр Вінера [3], фільтр Габора [4] тощо. Попередня обробка також включає перетворення зображення RGB у двійкове або зображення у градаціях сірого для кращих результатів. Сегментація зображення виконується просто для зображення, щоб можна було отримати значущу інформацію для подальшого аналізу. Виділення ознак – це крок для вилучення відповідної та необхідної інформації або тексту із зображень, які далі класифікуються за допомогою класифікації в аналізі зображень із попередньо визначеним набором, і таким

чином виконується розпізнавання тексту або символів, за яким у подальшому слідує постобробка результату.

Для ідентифікації символів на зображенні використовується оптичний сканер. Процес сканування виконується для перетворення текстів із зображення у текстовий файл. Зображення з документів, що підлягають обробці, скануються за допомогою оптичного сканера. Відскановане зображення перетворюється на кольорове зображення в градаціях сірого для подальшої обробки, щоб заощадити пам’ять і обчислювальну роботу.

Етап попередньої обробки передбачає перетворення кольорового зображення у двійкове зображення або зображення в градаціях сірого за допомогою двійкового перетворення. Цей крок виконується, лише якщо вхідне зображення не перетворено оптичним сканером. Якщо сканер перетворює зображення, тоді попередня обробка включає видалення шуму, який може виникнути через сканування зображення, або розбіжності в зображенні, спричинені пороговими значеннями зображення. Попередня обробка є важливим кроком для усунення всіх видів розбіжностей у зображенні для отримання кращого результату. Попередня обробка зображень виконує завдання усунення шуму за допомогою таких фільтрів, як фільтр Гауса, розпізнавання перекосів за допомогою кластера перетворення Хафа та бінаризація за допомогою порогового значення.

Сегментація – це крок у процесі аналізу зображення, на якому розташовуються області з корисною інформацією, щоб відокремити їх від нерелевантних даних, цифр і шуму. Більш точне використання сегментації [2] полягає у призначенні міток кожному пікселю зображення, де пікселі з однаковою міткою мають певні характеристики. Виявлення країв можна використовувати для виділення або сегментування зображення, покриваючи все зображення. Виявлення країв застосовується до зображень, щоб виділити межі об’єктів, присутніх на зображеннях. Це робиться шляхом виявлення різниці в яскравості на зображеннях, тобто розривів яскравості, і надання значень пікселів, де яскравість різко збільшується або зменшується.

Вилучення ознак – це процес зменшення розмірності, за допомогою якого початковий набір необроблених даних скорочується до більш керованих груп для обробки. Характерною рисою великих наборів даних є велика кількість змінних, для обробки яких потрібно багато обчислювальних ресурсів. Зображення після попередньої обробки використовується як вхідні дані для етапу виділення ознак. Відповідні дані витягуються із зображень за допомогою різних методів, таких як розподіл точок, розширення рядів і структурний аналіз. Подальше використання зіставлення шаблонів [5] витягнутих даних зіставляється з попередньо завантаженими даними в системі. Максимальна відповідність шаблону визначає символ, присутній на зображенні.

Класифікація – це етап сортування тексту, виділеного із зображень, на основі певних класів, який супроводжується розпізнаванням. Таким чином, термін класифікація використовується для класифікації зображення за його візуальним вмістом. Розпізнавання зображень [6] – це комбінація виявлення та класифікації зображень, яка окремо має здатність виявляти текст і класифікувати його далі, закінчуючи розпізнаванням. Розпізнавання використовується в багатьох програмах, включаючи автоматизацію, безпеку, спостереження тощо.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

3.1. Засоби тестування розробленої системи

Тестування програмного забезпечення – це процес оцінювання та перевірки того, що програмний продукт або програма виконує те, що він повинен робити. Переваги якісного тестування включають запобігання помилкам і покращення продуктивності.

Тестування програмного забезпечення сьогодні є найефективнішим, коли воно безперервне, вказуючи на те, що тестування починається під час проектування, продовжується під час створення програмного забезпечення та навіть відбувається під час розгортання у виробництві. Безперервне тестування означає, що організаціям не потрібно чекати, поки всі компоненти будуть розгорнуті, перш ніж розпочати тестування. Shift-left, який наближає тестування до дизайну, і Shift-right, де кінцеві користувачі виконують перевірку, також є філософією тестування, яка нещодавно набула популярності в спільноті програмного забезпечення. Коли ваша стратегія тестування та плани управління зрозумілі, автоматизація всіх аспектів тестування стає необхідною для підтримки необхідної швидкості доставки.

Існує багато різних типів тестування програмного забезпечення, кожен з яких має певні цілі та стратегії:

Приймальне тестування - перевірка того, чи вся система працює належним чином.

Перевірка коду - підтвердження того, що нове та модифіковане програмне забезпечення відповідає стандартам кодування організації та дотримується її найкращих практик.

Інтеграційне тестування - забезпечення спільної роботи програмних компонентів або функцій.

Модульне тестування - перевірка того, що кожен програмний блок працює належним чином. Одиниця це найменший тестований компонент програми.

Функціональне тестування: перевірка функцій шляхом імітації бізнес-сценаріїв на основі функціональних вимог. Тестування чорної скриньки є поширеним способом перевірки функцій.

Тестування продуктивності - тестування роботи програмного забезпечення під різними навантаженнями. Навантажувальне тестування, наприклад, використовується для оцінки продуктивності в умовах реального навантаження.

Регресійне тестування – перевірка того, чи нові функції порушують чи погіршують функціональність. Тестування працездатності можна використовувати для перевірки меню, функцій і команд на поверхневому рівні, коли немає часу для повного регресійного тесту.

Тестування безпеки: перевірка того, що ваше програмне забезпечення не відкрите для хакерів або інших зловмисних типів уразливостей, які можуть бути використані для заборони доступу до ваших служб або спричинення їх некоректної роботи.

Стрес-тестування перевірка того, яке навантаження може витримати система, перш ніж вона вийде з ладу. Стрес-тестування вважається різновидом нефункціонального тестування.

Тестування зручності використання перевірка того, наскільки добре клієнт може використовувати систему або веб-додаток для виконання завдання.

У кожному разі перевірка базових вимог є критичною оцінкою. Не менш важливо, що дослідницьке тестування допомагає тестувальнику або команді тестувальників виявити важкопередбачувані сценарії та ситуації, які можуть призвести до програмних помилок.

План керування тестуванням допомагає визначити пріоритетність типів тестування, які забезпечують найбільшу цінність, враховуючи доступний час і ресурси. Ефективність тестування оптимізується шляхом виконання найменшої кількості тестів для виявлення найбільшої кількості дефектів.

Тестування програмного забезпечення з'явилося разом із розробкою програмного забезпечення, яка почалася одразу після Другої світової війни.

Налагодження було основним методом тестування в той час і залишалося ним протягом наступних двох десятиліть. До 1980-х років команди розробників не лише виявляли та виправляли помилки програмного забезпечення, а тестували програми в реальних умовах. Це заклало основу для ширшого погляду на тестування, яке охоплювало процес забезпечення якості, який був частиною життєвого циклу розробки програмного забезпечення.

Хоча саме тестування коштує грошей, компанії можуть заощаджувати мільйони на рік на розробці та підтримці, якщо вони мають хорошу техніку тестування та процеси контролю якості. Раннє тестування програмного забезпечення виявляє проблеми ще до виходу продукту на ринок. Чим швидше команди розробників отримають відгук про тестування, тим швидше вони зможуть вирішити такі проблеми, як:

- архітектурні недоліки;
- невдалі дизайнерські рішення;
- недійсна або неправильна функція;
- вразливі місця безпеки;
- проблеми масштабованості.

Коли розробка залишає достатньо місця для тестування, це підвищує надійність програмного забезпечення, а високоякісні програми постачаються з невеликою кількістю помилок. Система, яка відповідає або навіть перевершує очікування клієнтів, потенційно призводить до збільшення продажів і збільшення частки ринку.

Тестування може зайняти багато часу. Для невеликих збірок може бути достатньо ручного або спеціального тестування. Однак для великих систем інструменти часто використовуються для автоматизації завдань. Автоматизоване тестування допомагає командам реалізовувати різні сценарії, тестувати відмінності (наприклад, переміщення компонентів у хмарне середовище) і швидко отримувати відгуки про те, що працює, а що ні.

3.2. Аналіз продуктивності алгоритмів

Якщо до цього на системі було натреновано модель то програма запуститься відповідне повідомлення буде відображатися в консолі (рис 3.1).

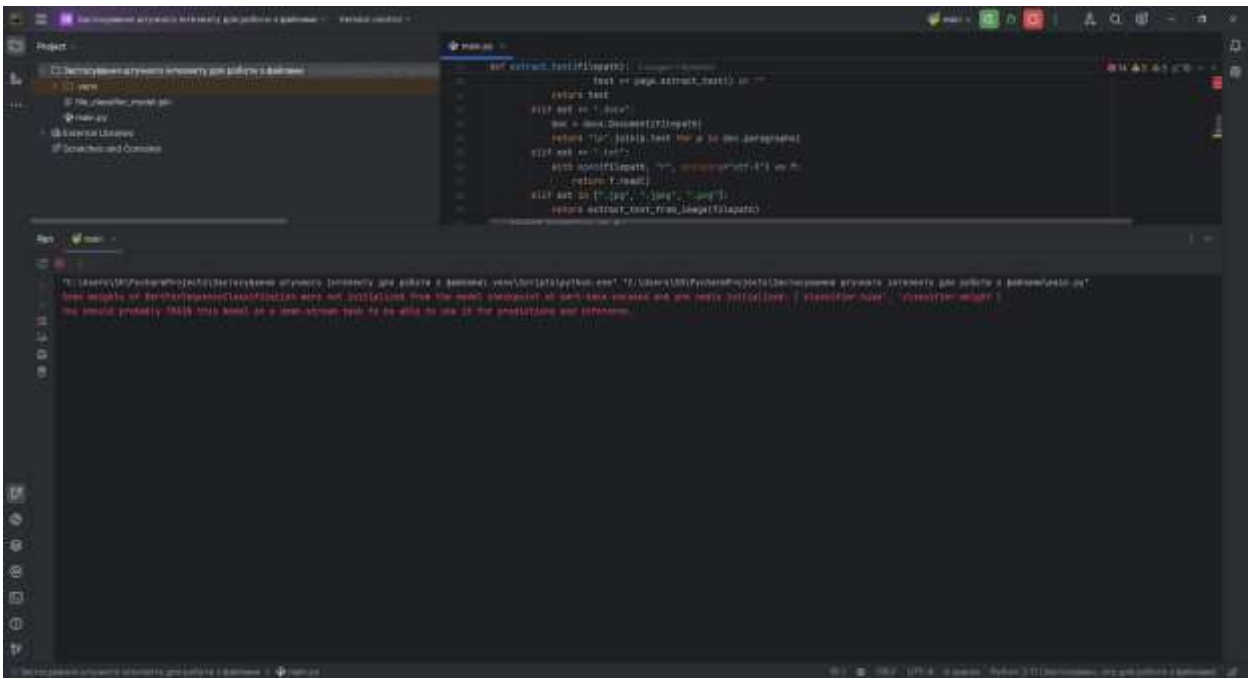

Рис. 3.1 – Повідомлення про завантажену модель

Після запуску або тренування моделі перед користувачем з’явиться робоча форма ПЗ(рис 3.2)

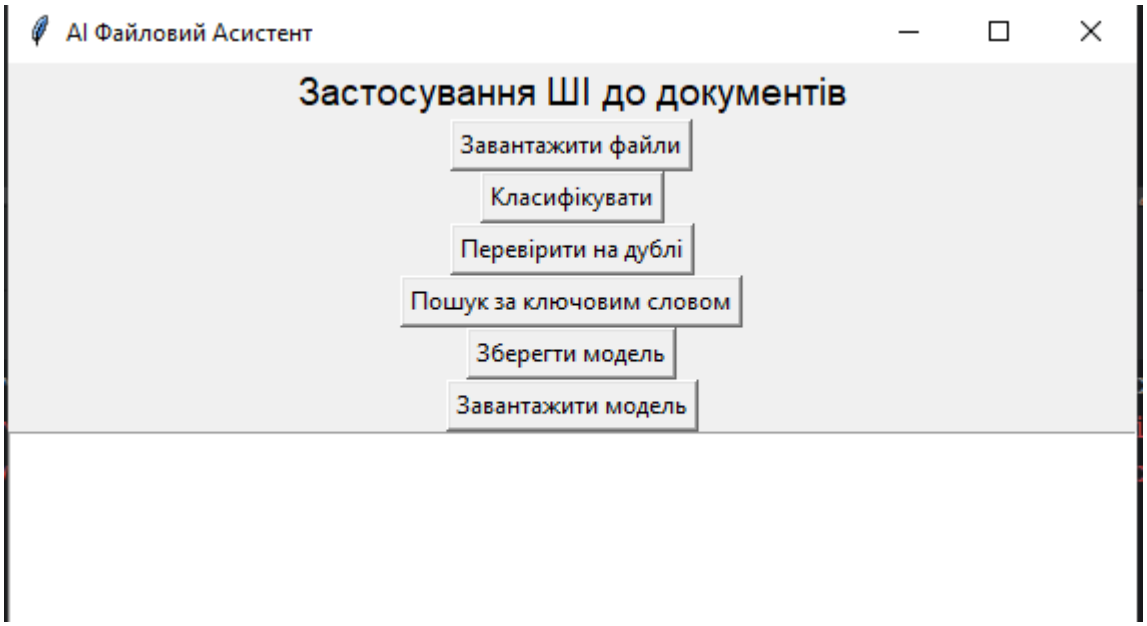

Рис. 3.2 – Робоча форма застосунку

Програма має такий функціонал :

- Завантаження файлу або декількох файлів для аналізу
- Класифікація файлу
- Перевірка на копії чи є в системі
- Пошук за ключовим словом або словами
- Також користувач має змогу навчання існуючої моделі якщо програма некоректно відображає результат

Коли користувач натискає на завантаження то з’являється можливість обрати файли для перевірки (рис 3.3)

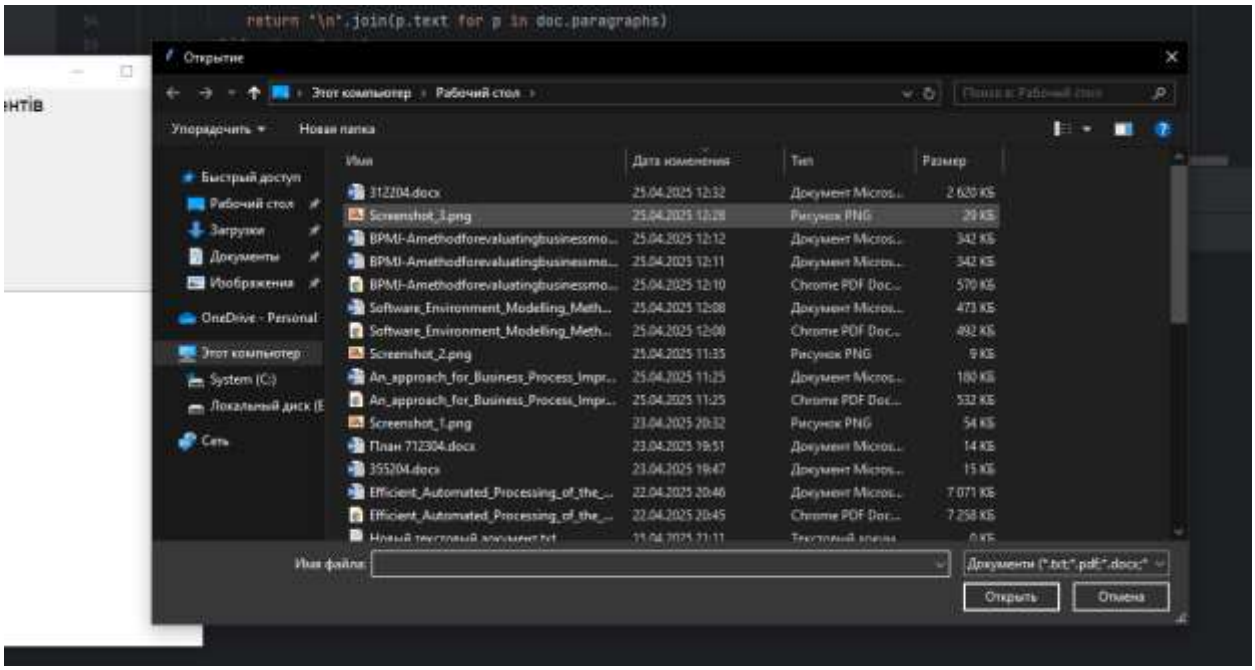

Рис. 3.3 – Завантаження файлу

Після успішного файлу для аналізу з’явиться повідомлення про успішність дії.(рис 3.4)

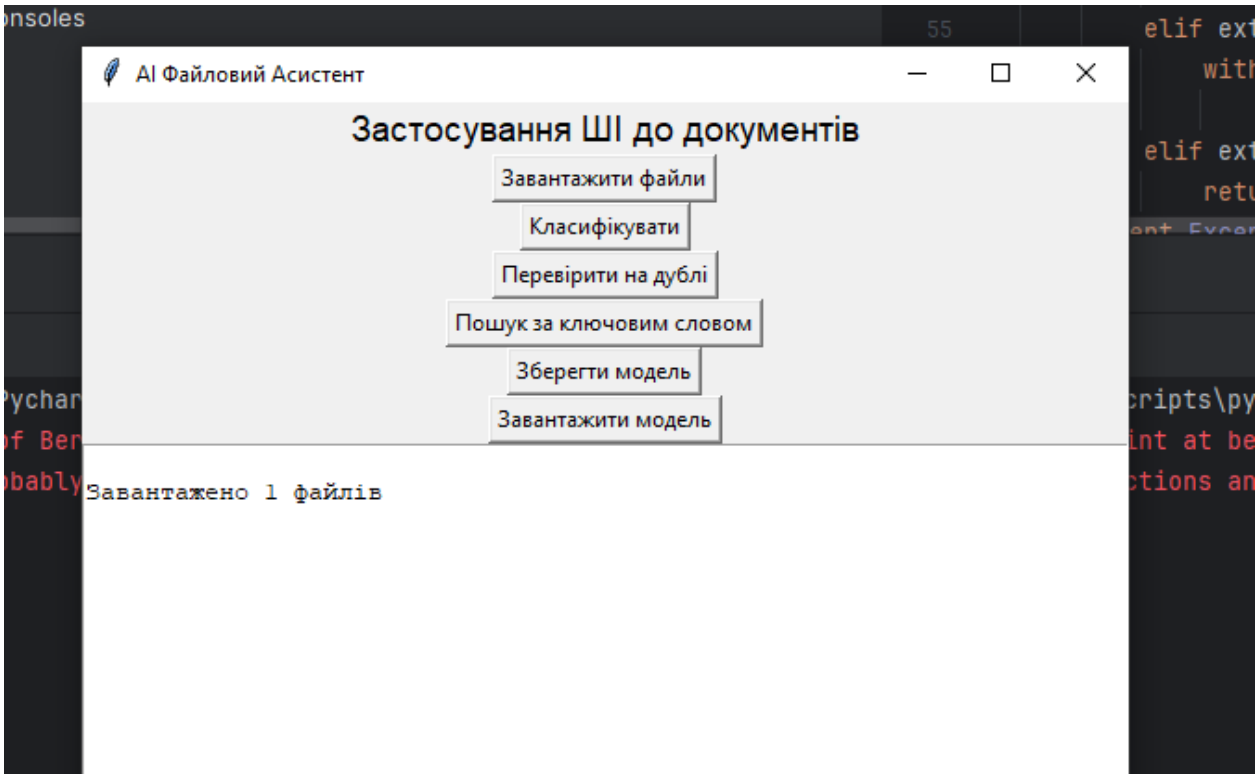

Рис. 3.4 – Успішність завантаження

Також перевірено функцію класифікації файлів(рис 3.5).

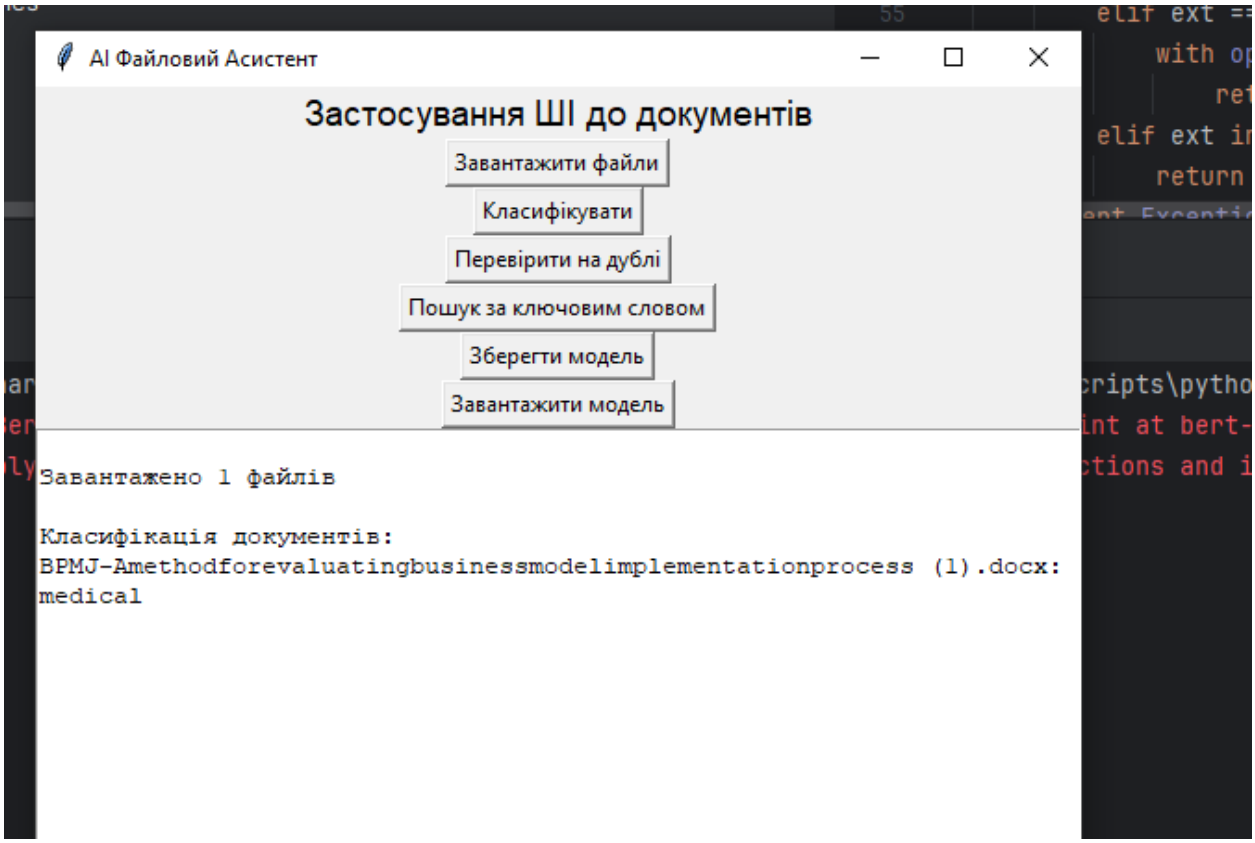

Рис. 3.5 – Перевірка функції класифікації файлів

Також додатково перевіримо на копії (рис 3.6)

Рис. 3.6 – Перевірка на дублікат

Особливість застосунку це пошук у файлі по ключовим словам(Рис 3.7-3.8)

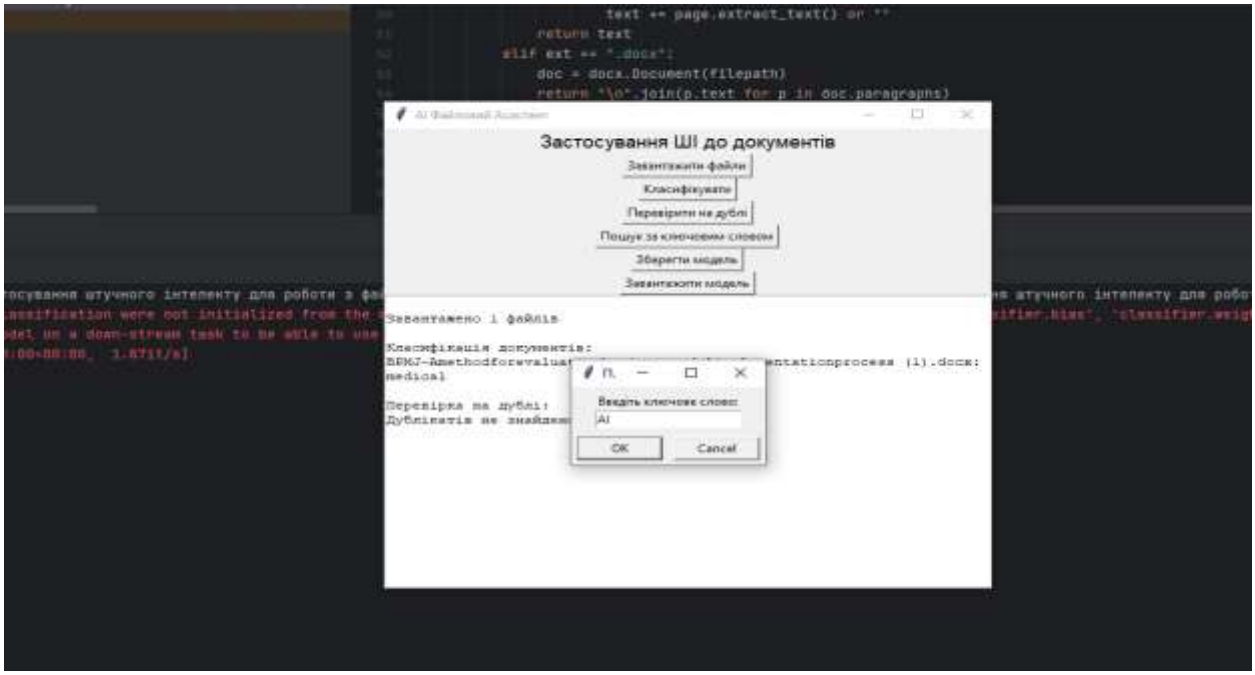

Рис. 3.7 – Уведення ключового слова

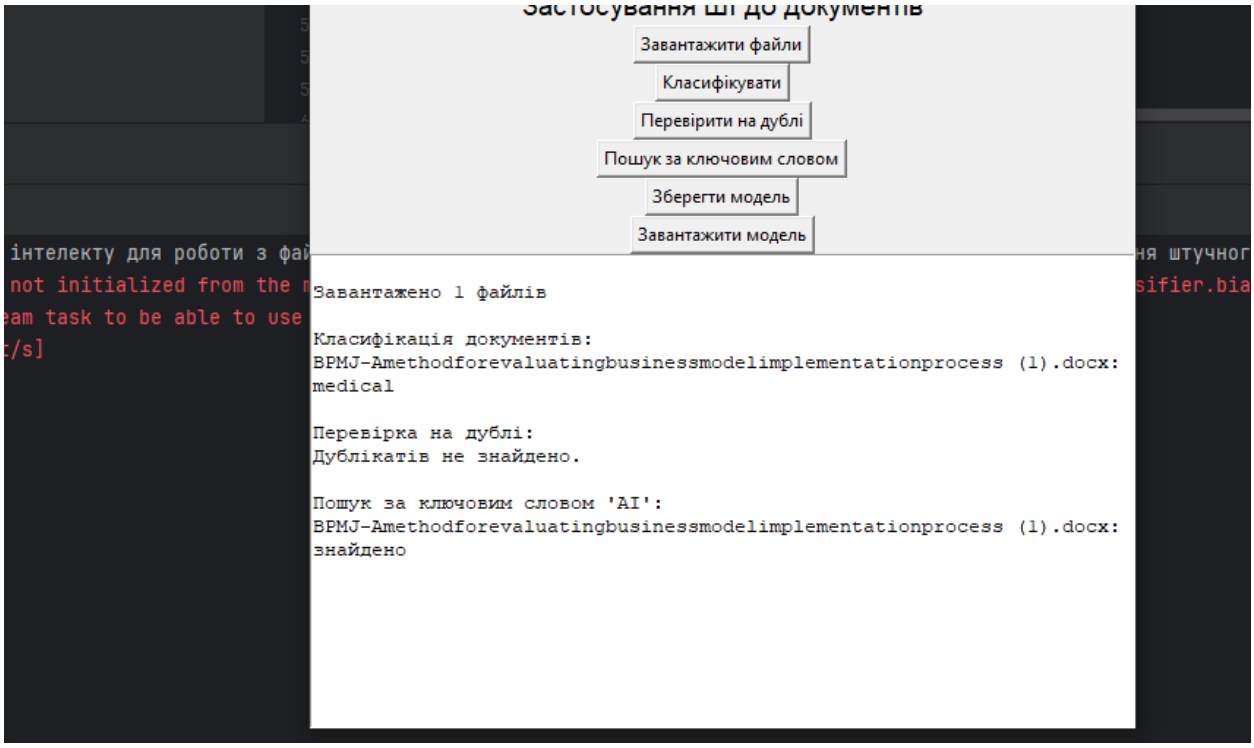

Рис. 3.8 – Результат

На завершення роботи користувач має змогу донавчити модель для більшої точності у подальшому.(рис 3.9 – 3.10).

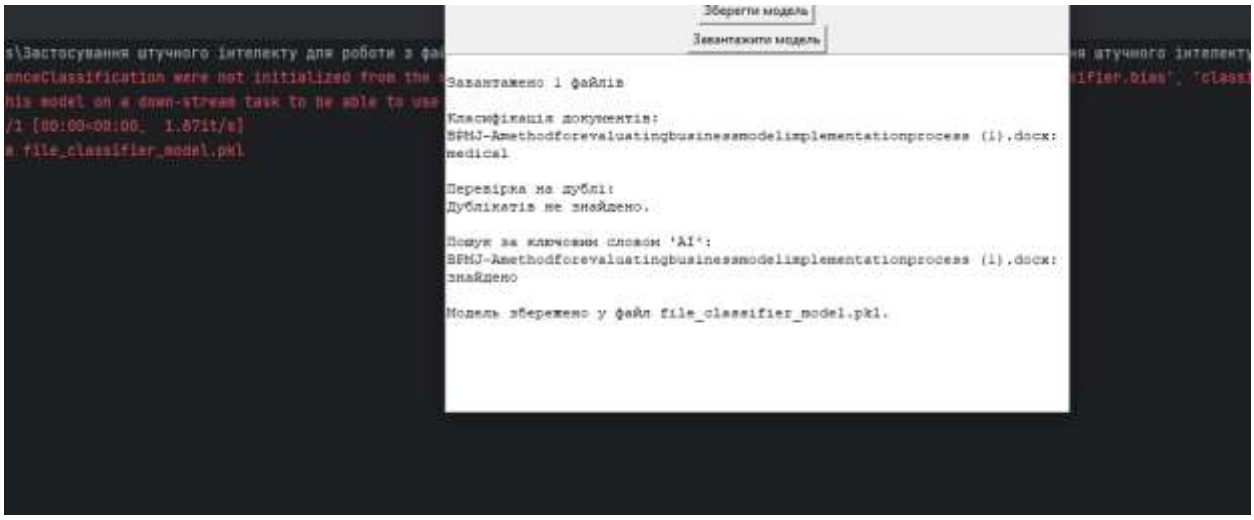

Рис. 3.9 – Збереження моделі

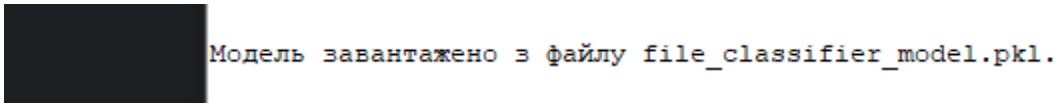

Рис. 3.10 – Повідомлення про завантаження

В загальному програма працює швидко та ніяких помилок не виникало.

В подальшому можна буде модернізувати або замінити на інші моделі для більшої точності та коректності роботи програми.

3.3. Порівняння з існуючими аналогами

«ШІ» (штучний інтелект) це знання та методи, які дозволяють машинам імітувати когнітивні функції людини (навчання, вирішення проблем, сприйняття та прийняття рішень). Але коли впроваджуєте концепції штучного інтелекту в програму чи додаток, це стає програмним забезпеченням штучного інтелекту. Наприклад, такий додаток, як чат-бот, який використовує NLP, є програмним забезпеченням штучного інтелекту,

яке можна запускати та з яким можна взаємодіяти. ШІ схожий на теорію, а програмне забезпечення ШІ є практичним застосуванням цієї теорії.

Таблиця 3.1 – Порівняння ПЗ та аналогів

Критерій / Програма	**Розроблений застосунок (на Python)**	**Adobe Acrobat Pro**	**Docugami AI**	**Smallpdf AI**	**ChatGPT (через API для аналізу текстів)**
Платформа	Десктоп	Десктоп	Веб	Веб	Веб/API
Графічний інтерфейс	☐ Tkinter	☐	☐	☐	☐ (CLI/API)
Класифікація документів	☐ (з BERT)	☐	☐	☐	☐ (з промтами)
Навчання на власних категоріях	☐	☐	Обмежено	☐	☐ (непрямо, через fine-tuning або context)
Виявлення дублікатів документів	☐	☐	☐	☐	☐
Пошук за ключовими словами	☐	☐	☐	☐	☐
Підтримка PDF, DOCX, TXT	☐	☐	Обмежено	☐	☐ (з конвертацією)
Можливість розширення та кастомізації	☐ (відкрите ПЗ)	☐ (пропрієтарне)	☐	☐	☐
Потребує підключення до Інтернету	☐ (локально працює)	☐	☐	☐	☐
Вартість	Безкоштовно	Платно	Платно	Частково безкоштовно	Платно (в залежності від API)

Навчена модель штучного інтелекту більше схожа на файл, що містить структуровані дані, ваги та інші дані у форматі .pt (PyTorch), який програма (програмне забезпечення ШІ) завантажує та виконує. Процес навчання (який створює модель) виконується до того, як вона стане частиною програмного забезпечення ШІ.

Після навчання модель є просто структурованим набором даних, який не «запускається» сам по собі без програмного забезпечення ШІ, яке

завантажує та виконує його. ШІ може існувати як послуга (AIaaS), яка все ще є програмним забезпеченням, що доставляється через хмару. Але знову ж таки, це програмна реалізація ШІ. Порівняння між традиційним програмуванням і машинним навчанням (ML) Машинне навчання (ML) відрізняється від традиційного програмування. У той час як традиційне програмування покладається на програмістів, які визначають чітку логіку та інструкції, що стосуються певного сценарію, ML дозволяє машинам самостійно навчатися та приймати рішення без детальних інструкцій для кожного завдання.

ВИСНОВКИ

Збільшення обсягів і необхідність ефективного використання даних зумовлюють потребу в розробці системи обробки документів на основі ШІ, яка б допомогла автоматично отримувати інформацію з даних.

Неструктуровані дані є невід'ємною частиною функціонування багатьох організацій. Такі форми, як рахунки-фактури, дані про клієнтів, страхові заяви, слугують доказом і записом транзакцій та інших важливих подій в організації. Належна обробка цих форм є дуже важливою. Обробка вручну може сповільнити процес і призвести до помилок і затримок. Автоматичне вилучення даних вирішує ці проблеми, надаючи автоматизоване та оцифроване рішення для обробки документів. Автоматизація трудомістких і повторюваних завдань допомагає підвищити продуктивність і розвиток організації. Штучний інтелект дає змогу ефективно й автоматично витягувати корисну інформацію з неструктурованих документів. ШІ також допомагає створити більш зрозумілий аналіз неструктурованих документів, який організації можуть використовувати в процесі прийняття важливих рішень.

В результаті роботи було розроблено застосунок, який має такий функціонал:

- завантаження файлу або декількох файлів для аналізу;
- класифікація файлу;
- перевірка на копії в системі;
- пошук за ключовим словом або словами;
- користувач має змогу навчання існуючої моделі, якщо програма некоректно відображає результат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. He, L. Wang, L. Liu, J. Feng, and H. Wu, «Long Document Classification from Local Word Glimpses via Recurrent Attention Learning», IEEE Access, vol. 7, pp. 40 707- 40 718, 2019. [Онлайн]. Доступний: 10.1109/access.2019.2907992;https://dx.doi.org/10.1109/access.2019.2907992

2. J. Yang, Y. Liu, M. Qian, C. Guan, and X. Yuan, «Вилучення інформації з електронних медичних записів з використанням багатозадачної рекурентної нейронної мережі з контекстним вбудовуванням слів», Applied Sciences, vol. 9, no. 18, pp. 3658-3658, 2019. [Онлайн]. Доступно: 10.3390/app9183658;https://dx.doi.org/10.3390/app9183658

3. D. Tkaczyk, P. Szostek, M. Fedoryszak, P. J. Dendek, and Ł. Bolikowski, «CERMINE: автоматичне вилучення структурованих метаданих з наукової літератури», International Journal on Document Analysis and Recognition (IJDAR), vol. 18, no. 4, pp. 317-335, 2015. [Онлайн]. Доступний за посиланням: 10.1007/s10032-015-0249-8;https://dx.doi.org/10.1007/s10032-015-0249-8

4. 2020. [Онлайн]. Режим доступу: https://kommandotech.com/statistics/big- data-statistics/

5. G. Zaman, H. Mahdin, K. Hussain, and Atta-Ur-Rahman, "Information extraction from semi- and unstructured data sources: Систематичний огляд літератури", ICIC Express Letters, vol. 14, no. 6, pp. 593-603, 2020.

6. Я. Сідерська, «Роботизована автоматизація процесів - рушій цифрової трансформації?» Інжиніринговий менеджмент у сфері виробництва та послуг, vol. 12, no. 2, pp. 21-31, 2020. [Онлайн]. Доступний: 10.2478/emj-2020-0009;https://dx.doi.org/10.2478/emj-2020-0009

7. A. W. Harley, A. Ufkes, and K. G. Derpanis, «Evaluation of deep convolutional nets for document image classification and retrieval,» Proc. Int. Conf. Doc. Anal. Recognition, ICDAR, vol. 2015, pp. 991-995, 2015.

8. B. Bataineh, «База даних друкованих зображень PAW арабської мови для аналізу та розпізнавання документів», Journal of ICT Research and Applications, vol. 11, no. 2, pp. 200-200, 2017. [Онлайн]. Доступно: 10.5614/itbj.ict.res.appl.2017.11.2.6;https://dx.doi.org/10.5614/itbj.ict.res.appl.2017.11.2.6

9. J. Memon, M. Sami, R. A. Khan, and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)", IEEE Access, vol. 8, pp. 142 642- 142 668, 2020. [Онлайн]. Доступний за посиланням: 10.1109/access.2020.3012542;https://dx.doi.org/10.1109/access.2020.3012542

10. N. Perera, M. Dehmer, and F. Emmert-Streib, «Named Entity Recognition and Relation Detection for Biomedical Information Extraction», Frontiers in Cell and Developmental Biology, vol. 8,pp. 673-673, 2020. [Онлайн]. Доступно за посиланням: 10.3389/fcell.2020.00673;https://dx.doi.org/10.3389/fcell.2020.00673

11. X. Ma та E. Hovy, «Наскрізне маркування послідовностей за допомогою двонаправленого LSTM-CNNs-CRF,» 54th Annu," Meet. Assoc. Comput. Linguist. ACL 2016 - Long Pap, vol. 2, pp. 1064-1074, 2016.

12. F. Yi, «Розпізнавання медичних рахунків за подвійною моделлю», Sensors (Швейцарія), vol. 19, no. 20, 2019.

13. X. Хан і Л. Ван, «Новий метод вилучення відношень на рівні документів на основі BERT та інформації про сутність», IEEE Access, vol. 8, pp. 96 912- 96 919, 2020.

14. C. V. A. C. V. A. Naidu and K. Vedavathi, "Cognitive modeling: Роль штучного інтелекту", Int. J. Innov. Technol. Explor. Eng, vol. 8, no. 6, pp. 1624-1629, 2019.

15. Y. Li, K. He, and J. Sun, "R-fcn: Виявлення об'єктів за допомогою регіональних повністю згорткових мереж", Nips, pp. 379-387, 2016.

16. B. Moysset, C. Kermorvant, and C. Wolf, «Навчання виявленню, локалізації та розпізнаванню багатьох текстових об'єктів на зображеннях документів на кількох прикладах», International Journal on Document Analysis and Recognition (IJDAR), vol. 21, no. 3, pp. 161-175, 2018. [Онлайн]. Доступний за посиланням: 10.1007/s10032-018-0305-2;https://dx.doi.org/10.1007/s10032-018-0305-2

17. П. Хофманн, К. Самп та Н. Урбах, «Автоматизація роботизованих процесів», Електронні ринки, т. 30, № 1, с. 99-106, 2020. [Онлайн]. Доступно за посиланням: 10.1007/s12525-019-00365-8;https://dx.doi.org/10.1007/s12525-019-00365-8

18. Л. Тодоран, М. Воррінг та А. В. М. Смолдерс, «Набір даних кольорових документів США», Міжнародний журнал аналізу та розпізнавання документів (IJDAR), т. 7, ні. 4, с. 228-240, 2005.

19. [Онлайн]. Доступно за посиланням: 10.1007/s10032-004-0135-2;https://dx.doi.org/10.1007/s10032-004-0135-2

20. «Вилучення значущої інформації з неструктурованих клінічних документів», Proc. Asia-Pacific Adv. Netw, vol. 48, pp. 42-47, 2019.

21. Ю. Є., «Уніфікована схема локалізації тексту та вилучення структурованих даних для спільного розпізнавання тексту та інтелектуального аналізу даних», Proc. 2018 IEEE Int. Conf. Big Data, Big Data, с. 2373-2382, 2018.

22. Р. Шарма, Н. Гоел, Н. Аггарвал, П. Каур і К. Пракаш, с. 55-60, 2020.

23. Р. Саїд, "Роботизована автоматизація процесів: Сучасні теми та виклики", Comput. Ind, vol. 115, pp. 103 162-103 162, 2020.

24. Ф. Брауер, Р. Рігер, А. Мокан та В. М. Барчинський, «Уможливлення вилучення інформації шляхом виведення регулярних виразів

із зразків сутностей», Міжнародна конференція з управління інформацією та знаннями, Матеріали, с. 1285-1294, 2011.

25. У. Мунір та М. Озтюрк, «Автоматичне вилучення символів з рукописного тексту - десять відсканованих документів для створення великомасштабної бази даних», Sci. Meet. Electr. Biomed. Eng. Comput. Sci. EBBT, vol. 2019, no. 1, pp. 1-4, 2019.

26. Н. Нобіле та К. Й. Суен, «Сегментація тексту для розпізнавання документів», в Довідник з обробки та розпізнавання зображень документів. Springer, 2014, с. 257-290.

27. І. Супріана та А. Насуція, «Розробка системи розпізнавання арабських символів», Procedia Technology, т. 11, с. 334-341, 2013. [Онлайн]. Доступно за посиланням: 10.1016/j.protcy.2013.12.199;https://dx.doi.org/10.1016/j.protcy.2013.12.199

28. D. D. A. Bui, G. D. Fiol, and S. Jonnalagadda, «Класифікація PDF-тексту для вилучення інформації зі звітів про публікації», Journal of Biomedical Informatics, vol. 61, pp. 141-148, 2016. [Онлайн]. Доступно за посиланням: 10.1016/j.jbi.2016.03.026;https://dx.doi.org/10.1016/j.jbi.2016.03.026

29. I. Chalkidis, I. Androutsopoulos, and A. Michos, «Extracting contract elements,» Proc. Int. Conf. Artif. Intell. Law, с. 19-28, 2017.

30. П. Сахаре та С. Б. Дхок, «Багатомовні схеми сегментації та розпізнавання символів для індійських зображень документів», IEEE Access, vol. 6, no. ii, с. 10 603-10 617, 2018. [Онлайн]. Доступно за посиланням: 10.1109/ access.2018.2795104;https://dx.doi.org/10.1109/access.2018.2795104

31. L. Arras, F. Horn, G. Montavon, K.-R. Мюллер і В. Самек, "Що є релевантним у текстовому документі?": Інтерпретований підхід до машинного навчання", PLOS ONE, vol. 12, no. 8, pp. e0 181 142-e0 181 142, 2017. [Онлайн]. Доступно за посиланням: 10.1371/journal.pone.0181142;https://dx.doi. org/10.1371/journal.pone.0181142

32. Y. Sun, X. Mao, S. Hong, W. Xu, and G. Gui, «Template Matching-Based Method for Intelligent Invoice Information Identification», IEEE Access, vol. 7, pp. 28 392-28 401, 2019. [Онлайн]. Доступно за посиланням: 10.1109/access. 2019.2901943;https://dx.doi.org/10.1109/access.2019.2901943

33. Д. Чакрабарті, «Використання штучного інтелекту для аналізу ризику в юридичних документах для кращої підтримки прийняття рішень», IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON, с. 683-688, 2019.

34. Р. Б. Р. Б. Палм, О. Вінтер і Ф. Лоуз, «CloudScan - система аналізу рахунків-фактур, що не потребує конфігурації, з використанням рекурентних нейронних мереж», Матеріали Міжнародної конференції з аналізу та розпізнавання документів, ICDAR, т. 1, с. 406-413, 2017.

35. H. T. H. T. Ha, «Розпізнавання рахунків-фактур зі сканованих документів», Recent Adv. Slavon. Nat. Lang. Process, с. 71-78, 2017.

36. Y. Wang, "Програми вилучення клінічної інформації: Огляд літератури", J. Biomed. Inform, vol. 77, pp. 34-49, 2017.

37. P. Shah, S. Joshi, and A. K. Pandey, «Legal clause extraction from contract using machine learning with heuristics improvement,» Conf. Comput. Commun. Autom. ICCCA, с. 1-3, 2018.

38. J. Huang, J. Chai, and S. Cho, "Глибоке навчання у фінансах та банківській справі: Огляд літератури та класифікація", Frontiers of Business Research in China, vol. 14, no. 1, pp. 13-13, 2020. [Онлайн]. Доступно за посиланням: 10.1186/s11782-020-00082-6;https://dx.doi.org/10.1186/s11782-020-00082-6

39. C.-A. Boiangiu, O.-A. Dinu, C. Popescu, N. Constantin, and C. Petrescu, «Voting-Based Document Image Skew Detection», Прикладні науки, vol. 10, no. 7, pp. 2236-2236, 2020. [Онлайн]. Доступно за посиланням: 10.3390/app10072236;https://dx.doi.org/10.3390/app10072236

40. Y. Chen, J. Wang, P. Li, and P. Guo, «Вилучення ключових слів з одного документа шляхом кількісної оцінки структурних особливостей

вищого порядку графа входження слів», Comput. Speech Lang, vol. 57, pp. 98-107, 2019.

41. М. П. Бах, Живко Крстич, С. Селян та Л. Туруля, "Інтелектуальний аналіз тексту для аналізу великих даних у фінансовому секторі: Огляд літератури", Сталий розвиток, vol. 11, no. 5, pp. 1277-1277, 2019. [Онлайн]. Доступно за посиланням: 10.3390/su11051277;https://dx.doi.org/10.3390/su11051277

42. С, «OCR4all - інструмент з відкритим вихідним кодом, що забезпечує (напівавтоматичний) робочий процес розпізнавання історичних друкованих видань», Appl. Sci, vol. 9, no. 22, 2019.

43. K. Jung, K. I. Kim, and A. K. Jain, «Видобування текстової інформації на зображеннях та відео: огляд», Pattern Recognition, vol. 37, no. 5, pp. 977-997, 2004. [Онлайн]. Доступно за посиланням: 10.1016/j.patcog.2003.10.012;https://dx.doi.org/10.1016/j.patcog.2003.10.012

ДОДАТКИ

Додаток А.Лістинг

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import pdfplumber
import docx
import os
import torch
from transformers import BertTokenizer, BertForSequenceClassification,
pipeline
from sentence_transformers import SentenceTransformer
import numpy as np
import pytesseract
from PIL import Image
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import joblib  # Для збереження та завантаження моделей
import logging

# Моделі
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=4)  # 4 категорії
similarity_model = SentenceTransformer("all-MiniLM-L6-v2")

# Розширення категорій
categories = {
    "finance": 0,
    "medical": 1,
    "technology": 2,
    "law": 3
}
reverse_categories = {v: k for k, v in categories.items()}

# Логування
logging.basicConfig(level=logging.INFO)


# OCR для зчитування тексту з зображень
def extract_text_from_image(image_path):
    image = Image.open(image_path)
    text = pytesseract.image_to_string(image)
    return text


# Функція для зчитування тексту з файлів
def extract_text(filepath):
    ext = os.path.splitext(filepath)[1].lower()
    try:
        if ext == ".pdf":
            text = ""
            with pdfplumber.open(filepath) as pdf:
                for page in pdf.pages:
                    text += page.extract_text() or ""
            return text
        elif ext == ".docx":
            doc = docx.Document(filepath)
            return "\n".join(p.text for p in doc.paragraphs)
        elif ext == ".txt":
```

```
            with open(filepath, "r", encoding="utf-8") as f:
                return f.read()
        elif ext in [".jpg", ".jpeg", ".png"]:
            return extract_text_from_image(filepath)
    except Exception as e:
        logging.error(f"Помилка читання файлу {filepath}: {e}")
        return f"[Помилка читання: {e}]"
    return ""


# Функція для класифікації тексту
def classify_text(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True,
truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    predicted_class = torch.argmax(outputs.logits).item()
    return reverse_categories.get(predicted_class, "Unknown")


# Функція для пошуку дублікатів
def find_duplicates(texts):
    vectors = similarity_model.encode(texts)
    sim_matrix = cosine_similarity(vectors)
    duplicates = []
    for i in range(len(texts)):
        for j in range(i + 1, len(texts)):
            if sim_matrix[i][j] > 0.9:
                duplicates.append((i, j, sim_matrix[i][j]))
    return duplicates


# Функція для пошуку за ключовим словом
def keyword_search(text, keyword):
    return keyword.lower() in text.lower()


# Функція для збереження та завантаження моделі
def save_model(model, filename):
    joblib.dump(model, filename)
    logging.info(f"Модель збережено в {filename}")


def load_model(filename):
    return joblib.load(filename)


# Клас для графічного інтерфейсу
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("AI Файловий Асистент")
        self.texts = []
        self.filepaths = []
        self.model_filename = "file_classifier_model.pkl"

        tk.Label(root, text="Застосування ШІ до документів", font=("Arial",
14)).pack()
        tk.Button(root, text="Завантажити файли",
command=self.load_files).pack()
        tk.Button(root, text="Класифікувати",
command=self.classify_files).pack()
        tk.Button(root, text="Перевірити на дублі",
```

```
command=self.check_duplicates).pack()
        tk.Button(root, text="Пошук за ключовим словом",
command=self.search_keyword).pack()
        tk.Button(root, text="Зберегти модель",
command=self.save_model).pack()
        tk.Button(root, text="Завантажити модель",
command=self.load_saved_model).pack()

        self.result_text = tk.Text(root, height=20, width=70)
        self.result_text.pack()

    def load_files(self):
        self.filepaths = filedialog.askopenfilenames(filetypes=[("Документи",
"*.txt *.pdf *.docx *.jpg *.jpeg *.png")])
        self.texts = [extract_text(fp) for fp in self.filepaths]
        self.result_text.insert(tk.END, f"\nЗавантажено {len(self.filepaths)}
файлів\n")

    def classify_files(self):
        self.result_text.insert(tk.END, "\nКласифікація документів:\n")
        for path, text in zip(self.filepaths, self.texts):
            category = classify_text(text)
            self.result_text.insert(tk.END, f"{os.path.basename(path)}:
{category}\n")

    def check_duplicates(self):
        self.result_text.insert(tk.END, "\nПеревірка на дублі:\n")
        dups = find_duplicates(self.texts)
        if dups:
            for i, j, score in dups:
                self.result_text.insert(tk.END,

f"{os.path.basename(self.filepaths[i])} ↔
{os.path.basename(self.filepaths[j])} (схожість: {score:.2f})\n")
        else:
            self.result_text.insert(tk.END, "Дублікатів не знайдено.\n")

    def search_keyword(self):
        keyword = simpledialog.askstring("Пошук", "Введіть ключове слово:")
        self.result_text.insert(tk.END, f"\nПошук за ключовим словом
'{keyword}':\n")
        for path, text in zip(self.filepaths, self.texts):
            if keyword_search(text, keyword):
                self.result_text.insert(tk.END, f"{os.path.basename(path)}:
знайдено\n")

    def save_model(self):
        save_model(model, self.model_filename)
        self.result_text.insert(tk.END, f"\nМодель збережено у файл
{self.model_filename}.\n")

    def load_saved_model(self):
        try:
            loaded_model = load_model(self.model_filename)
            self.result_text.insert(tk.END, f"\nМодель завантажено з файлу
{self.model_filename}.\n")
        except FileNotFoundError:
            self.result_text.insert(tk.END, "\nМодель не знайдено.\n")


if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = App(root)
    root.mainloop()
```