Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету(відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія парсингу резюме»

Виконав: студент 2-го курсу, групи 1КН-22м
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
_____ Олійник Н. Ю.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
_____ Колесницький О. К.
(прізвище та ініціали)
«07» 12 2023 р.

Опонент: к.т.н., професор каф.КСУ
_____ Биков М. М.
(прізвище та ініціали)
«07» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А. А.
(прізвище та ініціали)
«08» 12 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
29. 08 2023 р.

ЗАВДАННЯ
НА МАГІСТЕРЬСКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Олійнику Нікіті Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Інформаційна технологія парсингу резюме.
керівник роботи: к.т.н., професор кафедри КН Колесницький О. К.,
затверджені наказом вищого навчального закладу "13" 09 2023 року №247
2. Строк подання студентом роботи 13 11. 2023 р.
3. Вихідні дані до роботи :
Мова програмування Python, Середовище розробки Visual Studio Code;
Формати графічних файлів – PDF;
Кількість навчальних даних– 7000 екземплярів.
Кількість тестових даних – 1000 екземплярів.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
Вступ, обґрунтування доцільності розробки інформаційної технології парсингу резюме, розробка інформаційної технології парсингу резюме, вибір засобів реалізації інформаційної технології парсингу резюме, тестування та аналіз результатів роботи, економічна частина, висновки, перелік використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Архітектура мережі SpaCy
Процеси обробки інформації при інтелектуальному парсингу резюме, Архітектура SVM класифікатора, Архітектура моделі GPT-3, Приклад методу опорних векторів, Приклад роботи програми

Консультанти розділів роботи

Розділи	Прізвище ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Виконання прийняв
1-4	Колесницький О.К., к.т.н., проф. каф. КН	29.08.23	20.11.23
5	Адлер О. О., к. т. н., доц. каф. ЕПВМ	24.10.23	23.10.23

1. Дата видачі завдання 29.08.2023 року

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ МКР

№ з/п	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз сучасного рівня інформаційних технологій парсингу резюме, порівняння даної технології з аналогами	1.09.23	07.09.23	Розділ 1
2	Моделювання інформаційної технології парсингу резюме	08.09.23	24.09.23.	Розділ 2
3	Структурна організація та особливості програмної реалізації розробки інформаційної технології парсингу резюме та тестування	25.09.23	20.10.23	Розділ 3 та Розділ 4
4	Підготовка економічної частини	21.10.23	29.10.23	Розділ 5
5	Апробація та/або впровадження результатів дослідження	30.10.23	05.11.23	Тези
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	06.11.23	10.11.23	Пояснювальна записка, графічний матеріал, презентація

Студент _____ (підпис) Олійник Н. Ю.

Керівник роботи _____ (підпис) Колесницький О. К.

АНОТАЦІЯ

УДК 004.8

Олійник Н. Ю. Інформаційна технологія парсингу резюме. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма – системи штучного інтелекту. Вінниця: ВНТУ, 2023. 123 с.

На укр. Мові. Бібліогр.: 21 назв; рис.: 21; табл. 13

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для парсингу резюме, класифікації тексту, а також human-like спілкуванню. В даній роботі було досліджено відомі класифікації тексту, виокремленню сутностей та загальні концепції обробки природної мови. Було обрано нейро-мережевий метод та відповідні механізми навчання нейронних мереж, а також використання великих мовних моделей для аналізу тексту та генерації відповідей. Для парсингу резюме були використані відомі методи класифікації тексту за допомогою підтримуючих векторів, розпізнавання сутностей та аналіз тексту за допомогою вбудувань та великих мовних моделей. Архітектура обраної нейронної мережі базується на архітектурі мережі SVM з використанням згорткових нейронних мереж, з доповненими шарами класифікації та згортки. Вхідними даними до мережі є текст, який спершу обробляється та очищається від шумів, після чого передається нейронним мережам, вихідним значенням json файл, що містить класифікаований та розбитий на групи текст, а також оброблений текст, який передається великій мовній моделі. Для програмної реалізації було обрано мову програмування Python. Достовірність парсингу на 5% краще за аналоги. Також не було виявлено аналогів, що мають інтерфейс спілкування. У економічному розділі розраховано суму витрат на розробку та виготовлення нового технологічного рішення, яка склала 4,169,509 гривень, прогнозована орієнтована величина витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0.75 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: LLM, парсинг, детектинг, розпізнавання сутностей, нейронна мережа, SVM, NLP.

ABSTRACT

Oliinyk N. Y. Information Technology of Resume Parsing. Master's Qualification Thesis in the field of 122 – Computer Science, educational program – Artificial intelligence systems. Vinnytsia: VNTU, 2023. 123 p.

In Ukrainian. Bibliography: 21 titles; figures: 21; tables: 13.

This master's qualification thesis is dedicated to the development of software for resume parsing, text classification, and human-like communication. The study explores known text classifications, entity recognition, and general concepts of natural language processing. A neural network approach and corresponding mechanisms for neural network training were selected, along with the use of large language models for text analysis and response generation.

For resume parsing, established methods of text classification using support vectors, entity recognition, and text analysis through embeddings and large language models were employed. The architecture of the chosen neural network is based on the SVM network architecture using convolutional neural networks, supplemented with classification and convolution layers. The input data to the network is text, initially processed and cleaned from noise, then passed through neural networks. The output is a JSON file containing classified and grouped text, as well as processed text passed to the large language model.

Python was chosen as the programming language for implementation. The parsing accuracy is 5% better than analogs, and no counterparts with communication interfaces were identified. In the economic section, the cost amount for the development and production of the new technological solution is calculated at 4,169,509 hryvnias. The forecasted estimated cost for each cost item is calculated, net profit is determined, and the payback period for the manufacturer is 0.75 years. Additionally, the economic effect for the consumer when using this development is assessed.

Keywords: LLM, parsing, detecting, entity recognition, neural network, SVM, NLP.

ЗМІСТ

ВСТУП

Актуальність. В сучасному світі Інтернет-рекрутинг стає все більш популярним серед систем набору персоналу. Це призводить до того, що в мережі з’являється безліч резюме різних кандидатів. Багато з них намагаються виділитися серед інших, використовуючи різні формати оформлення, такі як розмір і колір шрифту, таблиці та інше. Проте ця розмаїтість форматів ускладнює обробку даних, таких як видобування інформації з резюме, автоматичний добір вакансій та ранжування претендентів. Існують методи, які базуються на правилах або навчанні з наглядом, для видобування фактів з резюме, але вони потребують інформації про ієрархічну структуру або великої кількості розмічених даних, які не завжди доступні. Це знижує ефективність рекомендацій рекрутерам, які шукають кандидатів, що відповідають бажанням роботодавця, і збільшує часові затрати на підбор персоналу.

У останнє десятиліття було розроблено чимало інструментів електронного найму для отримання інформації з резюме. Однак, хоча існують загальні теоретичні основи та методи обробки веб-даних, багато інструментів електронного найму все ще мають проблеми з обробкою тексту та визначенням ступеня відповідності кандидатів до вимог роботи. Штучний інтелект (ШІ), який ґрунтується на машинному навчанні та глибокому навчанні, може допомогти у вирішенні цих проблем. Використовуючи ШІ, можна оцінити резюме людини та навіть створити рейтинг враховуючи певну посаду у компанії.

Парсер резюме — програмний продукт, який перетворює неструктуровану інформацыю у структуровану форму. Це компонент, який автоматично розподіляє інформацію за різними категоріями та параметрами, такими як контактна інформація, освітній рівень, досвід роботи, навички, досягнення, професійні сертифікати, щоб швидко допомогти вам знайти найбільш відповідні резюме за вашими критеріями.

У цій роботі буде описана програмна реалізація глибокої нейронної мережі та SVM (Support Vector Machine) класифікатора, які навчаються за допомогою технології глибокого навчання, і мають за мету інтелектуальний парсинг резюме.

З додавання LLM – даний продукт зможе максимально якісно взаємодіяти з користувачем через зручний спосіб спілкування, а також з використання UI для більшої ефективності роботи користувача з продуктом. Ця програма може бути корисною для рекрутерів, для полегшення та прискорення пошуку кандидатів на різні посади.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп'ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета і завдання досліджень. Метою магістерської кваліфікаційної роботи є підвищення результативності інтелектуального парсингу резюме за рахунок використання глибокої нейронної мережі LSTM із зворотніми зв'язками, класифікаційного алгоритму Support Vector Machine (SVM), використанням великих мовних моделей (LLM) а також створення зручного інтерфейсу користувача.

Задачі подальшого дослідження:

- проаналізувати відомі методи інтелектуального парсингу резюме та обрати напрямок досліджень;
- розробити структуру процесів обробки інформації технології парсингу резюме;
- обґрунтувати вибір архітектури глибокої нейромережі;
- розробити алгоритми роботи інформаційної технології парсингу резюме;
- розробити програмне забезпечення інформаційної технології парсингу резюме на основі глибокої нейронної мережі LSTM, SVM класифікатора та з використанням LLM у складі LangChain;
- провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об’єкт дослідження. Об’єктом дослідження є процес комп’ютеризованого аналізу макету резюме, його інтелектуального парсингу та обробки результатів.

Предмет дослідження. Інформаційна технологія та програмні засоби інтелектуального парсингу резюме на основі глибокої нейронної мережі LSTM із зворотніми зв’язками, класифікаційного алгоритму Support Vector Machine (SVM) та результативності їх роботи.

Методи дослідження. У роботі використані наступні методи наукових досліджень: системного аналізу, теорії штучних нейронних мереж для реалізації інформаційної технології, методи математичної статистики для розробки процесу розв'язання задачі нейромережевого парсингу документів та обрахунків результатів експериментів із програмним засобом, великі мовні моделі.

Наукова новизна одержаних результатів.

Удосконалено інформаційну технологію інтелектуального парсингу резюме, яка відрізняється від відомих використанням глибокої нейронної мережі та SVM класифікатора, що дозволило підвищити результативність програмних засобів видобування інформації з резюме та визначення ступеня відповідності кандидатів до вимог роботи.

Практичне значення одержаних результатів полягає в тому, що на основі проведених досліджень розроблено програмне забезпечення парсингу резюме.

Запропонована інформаційна технологія сприяє підвищенню достовірності програмних засобів парсингу документів, зокрема:

- розроблено алгоритм роботи програмного забезпечення парсингу резюме;
- розроблено програмні засоби для парсингу резюме.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології парсингу резюме. Адекватність розроблених

математичних моделей підтверджується результатами експериментальних досліджень.

Апробація результатів роботи. Результати роботи були апробовані на конференції «LII науково-технічної конференції підрозділів ВНТУ» (м. Вінниця, Україна, 2023р.). [1]

Публікації. За результатами досліджень опубліковано одні тези доповіді на науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації. [1]

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПАРСИНГУ РЕЗЮМЕ

1.1 Постановка задачі

Вхідними даними в межах даної роботи є PDF-зображення, що містить текст. Ціль роботи полягає в тому, щоб використовуючи різні моделі для розпізнавання, сегментування та класифікації тексту в документі, здійснити аналіз макету документу та вивести «вгадані» навички кандидата. Також це завдання включає методи обробки природної мови (NLP).

Метою роботи є огляд існуючих підходів та алгоритмів з метою підвищення ефективності інтелектуального аналізу резюме. Розбір резюме передбачає автоматизацію процесу оперативного збору та систематизації інформації про кандидатів і претендентів на різні професії.

У цій роботі планується вирішити такі завдання:

- Підготовка наборів даних для інтелектуального аналізу резюме.
- Визначення впливу якості навчального набору на ефективність навчання нейронної мережі та алгоритму SVM.
- Порівняння результатів розробленої програми з існуючими альтернативами.

Завдання можна сформулювати як отримання вихідних векторів, які залежать від текстових ознак, знайдених у тестовому документі. Для вирішення цієї проблеми використовується класифікація, в якій встановлюється кінцевий набір класів. Для кінцевої підмножини об’єктів відомо, до якого класу вони належать, і ця підмножина називається навчальною вибіркою. Мета полягає в тому, щоб побудувати алгоритм, здатний класифікувати будь-який об'єкт з початкової множини, приналежність до класу якого невідома [2]. Також є окрема задача, в межах якої потрібно реалізувати модель, що на основі контексту резюме кандидата могла б вказати (передбачити) скілли, які людина потенційно може мати. Для демонстрації використання даного сервісу буде використана модель аналізу тону повідомлення, в якій буде проведено симуляцію діалогу кандидата

з рекрутером. На основі спілкування, можна буде виділити «настрій» у якому відбувалось спілкування, це допоможе краще зрозуміти ефективність сервісу.

Класифікація об'єкта означає присвоєння йому певного номера або імені класу [3]. У цьому випадку класифікаційне завдання складається з шести класів: ім’я кандидата, електронна адреса, номер телефону, бажана посада, мови кандидата та навички.

У контексті магістерської кваліфікаційної роботи буде розроблено модель аналізатора тону разом із компонентом розпізнавання іменованих сутностей (NER). NER, підполе класифікації, зосереджується на ідентифікації та категоризації об’єктів у тексті, включаючи осіб, місця, організації та інші типи, які можуть мати відношення до конкретної програми. Використання NER має вирішальне значення для різних завдань обробки природної мови (NLP), таких як вирішення займенникової анафори або побудова систем запитання-відповідь[4]. Розділення анафори займенника допомагає визначити референт займенника в даному тексті, покращуючи загальне розуміння змісту.

Для моделі аналізатора тону, вхідні дані складатимуться з текстового набору даних, що містить понад 6000 екземплярів із позначеними сутностями. Кожен екземпляр буде представлено як словник ключ-значення, де мітка відповідає одному з шести класів. Цей набір даних слугуватиме основою для навчання та оцінки продуктивності аналізатора тону.

Крім того, окремий набір даних, що складається з понад 900 екземплярів, буде використовуватися спеціально для завдання класифікації «навичок». Подібно до основного набору даних, кожен екземпляр у цьому додатковому наборі даних буде структуровано як словник ключ-значення, що містить мітку, яка вказує, чи екземпляр представляє «навички» чи «ненавички».

Шляхом інтеграції моделі тонального аналізатора з компонентом NER і використання цих позначених наборів даних магістерська кваліфікаційна робота спрямована на розробку комплексної системи, здатної точно класифікувати тони та розпізнавати іменовані сутності, що в кінцевому підсумку сприяє прогресу в аналізі тексту та полегшує різноманітні програми NLP.

Отже, буде розроблено програмний додаток для виконання інтелектуального аналізу резюме. Програма витягує текст із резюме, сегментує його на окремі блоки та класифікує ці блоки за відповідними категоріями, такими як ім’я кандидата, електронна адреса, номер телефону, бажана посада, мови та навички. Результати будуть представлені в структурі ключ-значення, що забезпечує зручний аналіз, при цьому ключ представлятиме одну з класифікованих категорій.

1.2 Огляд відомих методів класифікації для парсингу резюме

Класифікація тексту — це важлива техніка машинного навчання, яка призначає звичайному тексту попередньо визначені категорії. Це дозволяє організовувати, структурувати та класифікувати різні типи тексту, такі як опитування, коментарі, електронні листи тощо [5]. Класифікація тексту є фундаментальним завданням обробки природної мови з широким застосуванням, включаючи аналіз настроїв, додавання тегів до тем, виявлення спаму та виявлення намірів.

Класифікація тексту на основі машинного навчання має кілька переваг. По-перше, він демонструє масштабованість, автоматично аналізуючи великі обсяги тексту, наприклад мільйони документів, швидко та економічно ефективно. Інструменти класифікації тексту можуть адаптуватися до потреб бізнесу, незалежно від розміру чи обсягу текстових даних.

Ще одна перевага класифікації тексту машинного навчання полягає в узгодженості критеріїв, які вона застосовує. На відміну від людей-анотаторів, чия суб’єктивність може внести невідповідності, машинне навчання використовує ту саму стратегію та критерії для всіх даних і дає надійні результати. Коли модель класифікації тексту відповідним чином навчена, вона досягає надзвичайної точності своїх прогнозів.

Автоматична класифікація тексту охоплює три основні типи систем: системи на основі правил, системи на основі машинного навчання, і гібридні

системи. Системи на основі правил класифікують текст, застосовуючи створені вручну лінгвістичні правила, які визначають відповідні категорії на основі семантично відповідних елементів у тексті. Однак системи, засновані на правилах, мають недоліки, такі як необхідність глибокого знання предметної області, значні витрати часу на створення правил,
і труднощі в обслуговуванні та масштабованості через вплив додавання нових правил на існуючі.

З іншого боку, системи, засновані на машинному навчанні, навчають класифікаторів на попередньо позначених прикладах. Ці алгоритми вивчають асоціації між частинами тексту та очікуваним результатом, таким як теги або категорії.

Виділення ознак є важливим кроком у навчанні класифікатора машинного навчання NLP, який часто використовує такі методи, як представлення сумки слів, де частоти слів у попередньо визначеному словнику утворюють числове векторне представлення. Класифікація тексту машинного навчання перевершує системи на основі правил, особливо в складних завданнях NLP. Його також легше підтримувати, і він може постійно вчитися на нових прикладах для додаткових завдань.

Гібридні системи поєднують класифікатор на основі машинного навчання з системою на основі правил для підвищення точності класифікації.
Можна додати власні правила для вирішення конфліктів або покращення результатів, коли базовий класифікатор може неадекватно змоделювати певні теги.

Таким чином, класифікація тексту з використанням методів машинного навчання забезпечує масштабовану та послідовну категоризацію тексту, перевершуючи системи на основі правил за точністю та простотою обслуговування. Гібридні системи пропонують гнучкість для точного налаштування результатів класифікації шляхом включення спеціальних правил.

1.3 Огляд відомих методів аналізу тону тексту

Аналіз тону відіграє вирішальну роль у класифікації тексту, особливо коли йдеться про розуміння глибинних емоцій, ставлень і настроїв, виражених у текстових даних. Аналізатори тону — це інструменти, розроблені для виявлення та класифікації емоційного тону чи настрою певного тексту, що дозволяє глибше зрозуміти зміст. Було розроблено кілька відомих методів для ефективного аналізу тонів. Ці методи використовують різні техніки, зокрема лінгвістичний аналіз, машинне навчання та статистичне моделювання. Ось деякі з поширених підходів:

Методи на основі лексикону: засновані на тональних аналізаторах використовуються попередньо визначені словники або лексикони, що містять слова та фрази, пов’язані з певними емоціями чи почуттями. Ці лексикони зазвичай створюються за допомогою ручної анотації або краудсорсингу. Аналізуючи текст, аналізатор порівнює слова, присутні в тексті, з лексиконом і відповідно призначає бали настрою. Загальний настрій тексту можна визначити шляхом агрегування цих балів. Хоча цей метод є відносно простим, він може боротися з нюансами та залежними від контексту вираженнями емоцій.

Методи базовані на машинному навчанні: алгоритми машинного навчання можна навчити класифікувати текст на основі тону, вивчаючи шаблони з позначених наборів даних. Ці моделі витягують із тексту релевантні характеристики, такі як частоти слів, синтаксичні структури чи семантичні уявлення, і використовують їх для прогнозування настрою чи емоційного тону. Для цієї мети зазвичай використовуються методи керованого навчання, такі як опорні векторні машини (SVM), дерева рішень або моделі глибокого навчання, такі як рекурентні нейронні мережі (RNN).

Основні підходи можуть працювати зі складнішими лінгвістичними нюансами, але потребують значної кількості позначених навчальних даних і обчислювальних ресурсів.

Гібридні методи: гібридні підходи поєднують кілька методів для підвищення точності аналізу тонів. Наприклад, гібридна модель може включати як лексикон, так і машинне навчання.

Лексикони можуть надавати початкові оцінки настроїв, які потім уточнюються та коригуються за допомогою алгоритмів машинного навчання. Ця комбінація використовує сильні сторони обох методів, компенсуючи їхні індивідуальні обмеження. Гібридні методи можуть надати більш нюансовані та контекстно-залежні результати тонального аналізу.

Варто зазначити, що ефективність тональних аналізаторів значною мірою залежить від якості та охоплення базових ресурсів, таких як лексикони або навчальні набори даних. Постійне оновлення та обслуговування цих ресурсів є важливими для забезпечення точного та актуального тонального аналізу.

У міру розвитку технологій і проведення нових досліджень у сфері обробки природної мови продовжують з’являтися нові методи та підходи до тонального аналізу. Дослідники та практики постійно прагнуть покращити точність, ефективність та можливість інтерпретації аналізаторів тону,
дозволяє глибше зрозуміти емоційні аспекти, які передаються в текстовому вмісті.

1.4 Обгрунтування вибору аналогу до інформаційної технології парсингу резюме

На даний момент існує кілька відомих реалізацій програмного забезпечення для інтелектуального аналізу резюме. Одним із таких сервісів є Affinda: The Enterprise Grade Resume Parser [6], яке є рішенням на основі Python (див. рис. 1.1). Affinda пропонує продукти штучного інтелекту та індивідуальні рішення штучного інтелекту для підприємств. Він використовує обробку PDF, спрощує процес перегляду резюме та стандартизує його. Архітектура глибокого навчання Affinda витягує попередньо визначені текстові поля з резюме та CV, представляючи дані у форматі Excel. Продукт автоматично визначає та виділяє

ключові компоненти резюме на основі критеріїв, визначених клієнтом, зосереджуючись на найбільш релевантних словах і фразах.

Однак ця послуга має певні недоліки, зокрема низьку продуктивність і те, що це платна послуга з відносно високою вартістю.

Іншим продуктом, який варто розглянути, є ALEX Resume Parser [7]. Він використовує різні стратегії штучного інтелекту, включаючи обробку природної мови та методи розпізнавання образів, щоб витягти релевантну інформацію з резюме. Діючи як граматичний аналізатор резюме, ALEX призначає значення термінам (словам і реченням) на основі їх контексту.

Подібно до попереднього сервісу, ALEX також має недоліки, такі як низька надійність у деяких випадках, залежність від конкретних шаблонів документів і є платною послугою.

Skills detail

Document ID	Name	Skill
XYLopcUN	Roman Holovatyi	Test
XYLopcUN	Roman Holovatyi	Agile Software
XYLopcUN	Roman Holovatyi	HyperText Markup
XYLopcUN	Roman Holovatyi	Skype
XYLopcUN	Roman Holovatyi	Automation
XYLopcUN	Roman Holovatyi	JIRA
XYLopcUN	Roman Holovatyi	C (Programming Language)
XYLopcUN	Roman Holovatyi	Selenium Webdriver
XYLopcUN	Roman Holovatyi	Software Versioning
XYLopcUN	Roman Holovatyi	Application Programming
XYLopcUN	Roman Holovatyi	SQL (Programming
XYLopcUN	Roman Holovatyi	Cascading Style Sheets (CSS)
XYLopcUN	Roman Holovatyi	Apple Automator (OS X)
XYLopcUN	Roman Holovatyi	Testcomplete
XYLopcUN	Roman Holovatyi	Android (Operating System)

Рисунок 1.1 – Результат парсингу сервісу Afinda у вигляді Excel таблиці

Інші реалізації програмного забезпечення також страждають від недоліку низької ефективності інтелектуального аналізу резюме. Враховуючи ці обмеження, виникає потреба розробити новий програмний інструмент для інтелектуального аналізу резюме, який зможе подолати ці проблеми та досягти більшої ефективності.

Існують різні системи та технології, які можуть аналізувати резюме або резюме людини, щоб визначити, зробити висновок про їхні навички та передбачити якими ще навичками володіє людина. У цих системах використовується обробка природної мови (NLP), машинне навчання (ML) і методи інтелектуального аналізу даних, щоб отримати відповідну інформацію та зробити прогноз щодо навичок. Однією з таких систем є Skillate. Skillate — це платформа відбору резюме на основі штучного інтелекту, яка спрямована на оптимізацію та автоматизацію процесу відбору кандидатів. Він використовує передові технології, такі як обробка природної мови (NLP) і машинне навчання (ML), щоб аналізувати та витягувати інформацію з резюме, особливо зосереджуючись на навичках і кваліфікації. Ось деякі ключові функції та можливості Skillate:

- Вилучення навичок. Алгоритми штучного інтелекту Skillate розроблені для вилучення навичок, згаданих у резюме кандидатів. Він може ідентифікувати як технічні навички (навички, пов’язані з певною областю), так і навички комунікації, лідерства, роботи в команді тощо. Платформа використовує графік навичок, щоб представити їх в структурованому та зручному для читання форматі.

- Аналіз навичок. Skillate може оцінити навички, згадані в резюме, і порівняти їх із необхідними для конкретної роботи чи ролі. Ця функція допомагає рекрутерам визначити прогалини і оцінити придатність кандидата для певної посади.

- Розуміння контексту. Платформа виходить за рамки відповідності ключових слів, використовуючи техніки НЛП, щоб зрозуміти контекст. Це дозволяє точніше проаналізувати навички кандидата, враховуючи варіанти, синоніми та пов’язані терміни.

- Розбір резюме та витяг даних. Skillate надає ефективні можливості аналізу резюме, вилучаючи з резюме різні дані, такі як контактна інформація, досвід роботи, освіта та сертифікати. Ці структуровані дані можна легко інтегрувати в системи відстеження кандидатів (ATS) або інші робочі процеси найму.

- Налаштування та інтеграція. Skillate пропонує варіанти налаштування для адаптації до конкретних організаційних вимог і профілів посад. Його можна інтегрувати з існуючими системами ATS або рекрутингу, що забезпечує безперебійну передачу даних та інтеграцію робочого процесу.

- Аналітика та статистичні дані. Skillate надає аналітичні дані та аналіз навичок кандидатів, що дозволяє рекрутерам отримати глибше розуміння навичок потенційних кандидатів. Цей підхід на основі даних допомагає приймати обґрунтовані рішення під час перевірки та відбору кандидатів.

Використовуючи технології штучного інтелекту та НЛП, Skillate дозволяє рекрутерам приймати більш обґрунтовані рішення та складати список кандидатів, які точно відповідають необхідним профілям навичок, заощаджуючи час і зусилля в процесі найму.

1.5 Висновок до розділу 1

У розділі було розглянуто постановку задачі інтелектуального парсингу резюме, проведено огляд відомих методів класифікації тексту, які можна використовувати для поставленої задачі. Обґрунтовано доцільність використання нейронних мереж та класифікаторів, для інтелектуального парсингу резюме. Було проаналізовано різні сервіси інтелектуального парсингу резюме та обрано аналоги, головними недоліками яких є невисока результативність та висока ціна інтелектуального парсингу резюме, що ставить мету дослідження – підвищення результативності інтелектуального парсингу

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПАРСИНГУ РЕЗЮМЕ

2.1 Розробка структури процесів обробки інформаційної технології парсингу резюме

Завданням цієї роботи є покращення попередньо створеного інтелектуального модуля, який би із PDF документу, що містить резюме людини довільного формату, екстрактив та класифікував текст. Тому вхідними даними програмного продукту є резюме людини в довільному форматі. Подальші кроки обробки цього зображення представлені на рис. 2.1.

Рисунок 2.1 – Процеси обробки інформації при інтелектуальному парсингу резюме

Вхідний документ, що надходить у систему, повинен бути попередньо оброблений, щоб він був придатний для навчання нейронної мережі та SVM класифікатора. Ця обробка включає в себе два етапи:

- Екстракція необробленого тексту;
- Приведення тексту до формата, який сприймає нейронна мережа та SVM класифікатор.

На першому етапі з документа видаляються всі нетекстові елементи, такі як зображення, таблиці та код. На другому етапі текст приводиться до певного формату, наприклад, до набора токенів.

Після попередньої обробки вхідний документ використовується для навчання нейронної мережі та SVM класифікатора. Обидва ці алгоритми навчаються виконувати класифікацію тексту, тобто визначати, до якої категорії належить текст.

Нейронная мережа LSTM є перспективним алгоритмом для класифікації тексту, оскільки вона може ефективно обробляти довгі послідовності символів. SVM класифікатор є іншим популярним алгоритмом для класифікації тексту, який може бути ефективним для невеликих наборів даних.

Після навчання нейронної мережі та SVM класифікатора їх необхідно зберегти. Збережена мережа може бути використана для класифікації тексту в інших документах. Класифікатор може бути використаний для екстракції текстових блоків на вхідному документі довільної форми. Для цього документ розбивається на блоки, а потім кожному блоку присвоюється категорія.

Результат класифікації може бути збережений у форматі JSON. Цей формат є зручним для зберігання та обробки даних. Для поліпшення якості класифікації можна використовувати додаткову інформацію, наприклад, теги, метадані та інформацію про автора документа.

Також можна використовувати різні алгоритми для класифікації тексту. Це може допомогти підвищити точність класифікації.

2.2 Вдосконалення архітектури глибокої нейронної мережі та SVM класифікатора

Модель має архітектуру, яка є функцією, що створює екземпляр моделі для використання в компоненті конвеєра або як шар більш складної мережі. Людське мислення не починається кожного разу з нуля. Ми не забуваємо все і не думаємо з чистого аркуша. Наші думки зберігають послідовність.

Звичайні нейронні мережі не мають цього властивості, і це їх слабке місце. Наприклад, якщо ми хочемо класифікувати події у фільмі, то не зрозуміло, як звичайна нейронна мережа може враховувати попередні події фільму для аналізу наступних. Цю проблему можна вирішити за допомогою рекурентних нейронних мереж (Recurrent Neural Networks, RNN) [8]. Це мережі, які мають зворотній зв’язок і можуть зберігати інформацію. Зворотній зв’язок робить рекурентні нейронні мережі трохи таємничими. Але насправді вони не дуже відмінні від простих нейронних мереж. Рекурентну мережу можна уявити як багато копій однакової мережі, кожна з яких передає інформацію наступному екземпляру. На рисунку 2.2 показано розгорнуту схему мережі.

Одна з переваг RNN полягає в тому, що вони потенційно можуть використовувати попередню інформацію для поточного завдання, наприклад, знання про попереднього кадра відео можуть сприяти розумінню поточного кадра. Якщо RNN мали таку можливість, вони були б дуже корисними. Але на жаль, коли відстань між інформацією на нейронах стає більшою – RNN втрачають можливостя пов’язувати інформацыю.

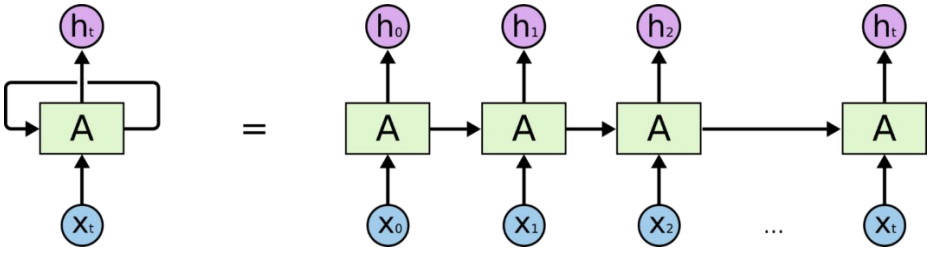


Рисунок 2.2 – Архітектура рекурентної нейронної мережі в розгортці

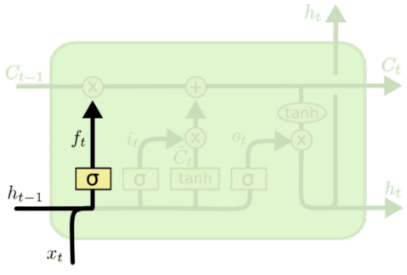
Довга короткострокова пам’ять (Long short-term memory; LSTM) - це спеціальний вид архітектури рекурентних нейронних мереж, який може навчатися довгостроковим залежностям. Вони були запропоновані Зеппом Хохрайтером і Юргеном Шмідгубером у 1997 році, а потім удосконалені і популяризовані у роботах багатьох інших науковців. Вони добре справляються з багатьма різними завданнями і зараз широко застосовуються.

LSTM створено спеціально, щоб подолати проблему довготривалої залежності [9]. Традиційні рекурентні нейронні мережі (РНМ) мають труднощі з навчанням довгостроковим залежностям, оскільки їхні стани осередків схильні до згасання з часом. LSTM вирішує цю проблему за допомогою трьох спеціальних шарів: шару фільтра забування, шару входу і шару виходу.

Шар фільтра забування визначає, яку інформацію можна викинути зі стану комірки. Він дивиться на поточний стан комірки і вихідні значення РНМ і повертає число від 0 до 1 для кожного числа зі стану осередку. 1 означає "повністю зберегти", а 0 - "повністю викинути".

Шар входу визначає, яку нову інформацію можна додати до стану комірки. Він дивиться на поточний стан комірки, вихідні значення РНМ і вихідні значення шару фільтра забування і повертає число від 0 до 1 для кожного числа зі стану осередку. 1 означає "повністю додати", а 0 - "повністю не додавати".

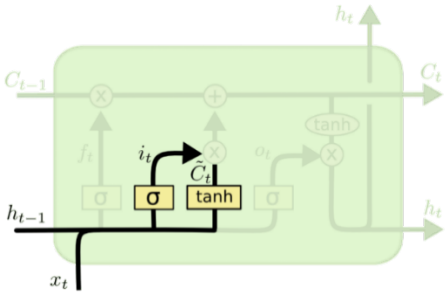
Шар виходу визначає, яку інформацію можна вивести з стану комірки. Він дивиться на поточний стан комірки і вихідні значення РНМ і повертає число від 0 до 1 для кожного числа зі стану осередку. 1 означає "повністю вивести", а 0 - "повністю не вивести".



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Рисунок 2.3 – Перший крок архітектури LSTM

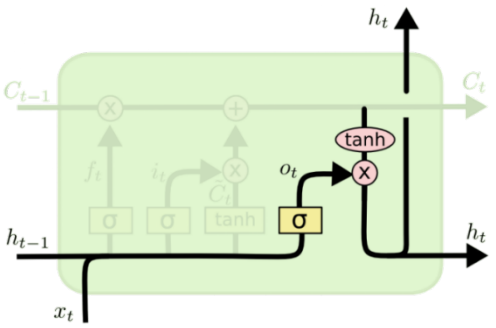
Наступний крок – вирішити, яка нова інформація зберігатиметься у стані осередку. Цей етап складається із двох частин. Спочатку сигмоїдальний шар визначає які значення слід оновити. Потім tanh-шар будує вектор нових значень-кандидатів, які можна додати до стану осередку, даний крок зображено на рисунку 2.4.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Рисунок 2.4 – Другий крок архітектури LSTM

Зрештою, потрібно вирішити, яку інформацію ми хочемо отримувати на виході. Вихідні дані будуть засновані на нашому стані осередку, до них будуть використані деякі фільтри. Спочатку ми застосовуємо сигмоїдальний шар, який вирішує, яку інформацію зі стану осередку ми виводитимемо. Потім значення стану осередку проходять через tanh-шар, щоб отримати на виході значення діапазону від -1 до 1, і перемножуються з вихідними значеннями сигмоїдального шару, що дозволяє виводити тільки необхідну інформацію. Останній крок зображено на рисунку 2.5



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Рисунок 2.5 – Останній крок архітектури LSTM

Цей вид мережі має великий потенціал, але в нашому випадку кожен клас містить багато даних, які складно семантично пов’язати між собою, бо такі дані як ім’я кандидата чи потрібна посада - не мають прямого семантичного зв’язку між елементами класу, також елементів дуже багато, а як було зазначено раніше про архітектуру рекурентних нейронних мереж - з підвищенням відстані між даними на нейронах - RNN втрачають здатність пов’язувати дані. Тому такий вид нейронної мережі не здатний точно визначити до якого класу належить текст. Система Spacy NER використовує стратегію вбудовування слів з функціями підслів, а також глибоку нейронну мережу згортки з резидуальними зв’язками. Система створена для досягнення хорошого балансу ефективності, точності та адаптивності, її архітектура показана на рисунку 2.6. Нейронна мережа, яка застосовується у цій бібліотеці, може використовуватися для NER або аналізу залежностей. Синтаксичний аналіз на основі переходів - це метод структурованого прогнозування, де задача передбачення структури відображається на послідовність переходів станів.

Модель прогнозування стану нейронної мережі складається з двох або трьох підмереж [10]:

- tok2vec: переводить кожен маркер у векторне представлення. Ця підмережа запускається один раз для кожного пакету навчальних даних.
- lower: будує специфічний вектор для кожної пари (токен, ознака). Це також виконується один раз для кожної партії. Тоді побудова представлення стану є просто питанням підсумовування ознак компонентів і застосування нелінійності.
- upper (необов’язково): мережа з прямим зв’язком, яка прогнозує результати з представлення стану. Якщо його немає, вихідні дані нижчої моделі використовуються безпосередньо як оцінки дії.

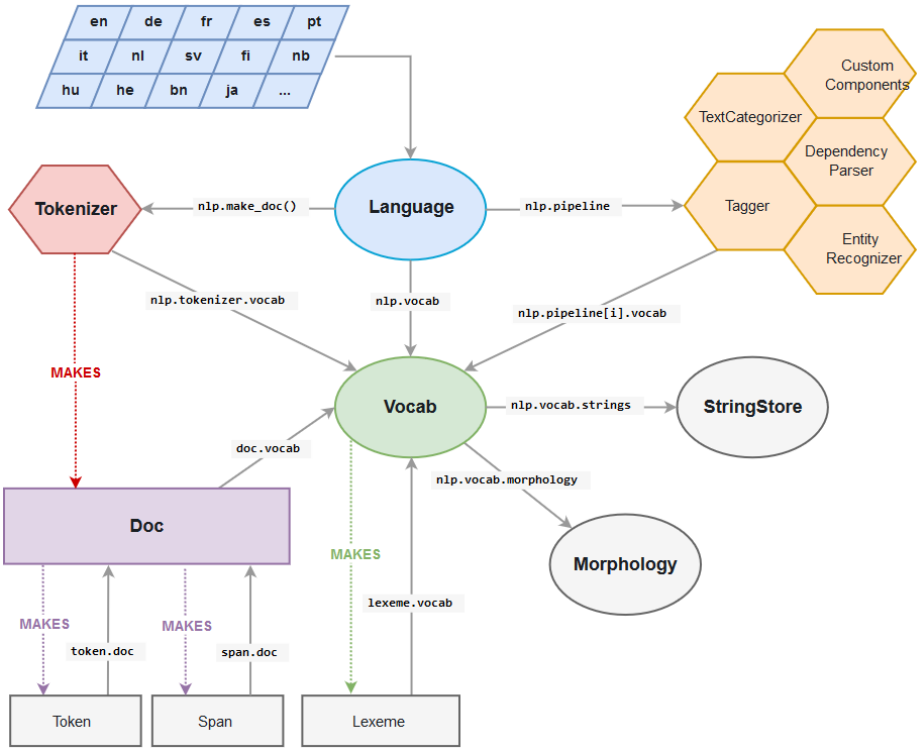


Рисунок 2.6 – Архітектура мережі SpaCy

У SpaCy застосовується глибока нейронна мережа, яка є покращеною версією мережі LSTM, а саме LSTM з резидуальними зв'язками. Основна ідея резидуальної мережі полягає в тому, що вона надає швидкий доступ між шарами, який може бути використаний для додаткового шляху градієнта. Резидуальний LSTM забезпечує додатковий просторовий шлях швидкого доступу з нижчих шарів для ефективного навчання глибоких мереж з багатьма шарами LSTM. Резидуальний LSTM розділяє просторовий шлях швидкого доступу за допомогою вихідних шарів, що може допомогти уникнути конфлікту між потоками градієнта просторової та часової області [11]. Крім того, резидуальний LSTM повторно використовує вихідну матрицю для керування потоком просторової інформації замість додаткових мереж, що ефективно зменшує понад 10% параметрів мережі. Експеримент з віддаленого розпізнавання мовлення на корпусі AMI SDM показав, що 10-шарові звичайні мережі LSTM показали збільшення точності на 13,7%. А 10-шарові резидуальні мережі LSTM забезпечили найбільше покращення точності - на 41,0%.

Також розглянемо архітектуру SVM. Машина опорних векторів (SVM) - це алгоритм машинного навчання з наглядом, який простий у використанні та призначений для класифікації та/або регресії. Вона більше підходить для класифікації, але інколи також може бути корисною для регресії. Суть SVM полягає в тому, що вона шукає гіперплощину, яка створює кордон між типами даних. У двовимірному просторі ця гіперплощина є просто лінією [12]. У SVM ми представляємо кожен елемент даних у наборі даних у N-вимірному просторі, де N - кількість функцій/атрибутів у даних. Потім знаходимо оптимальну гіперплощину для розділення даних. Тому SVM за своєю природою може виконувати тільки двокласову класифікацію (тобто вибирати між двома класами). Однак існують різні технології, якими можна скористатися для вирішення багатокласових задач. Архітектура цього класифікатора показана на рисунку 2.7.

Отже, у цій роботі буде використана мережа LSTM з резидуальними зв’язками, яка є частиною Spacy, а також власна мережа SVM класифікатора.

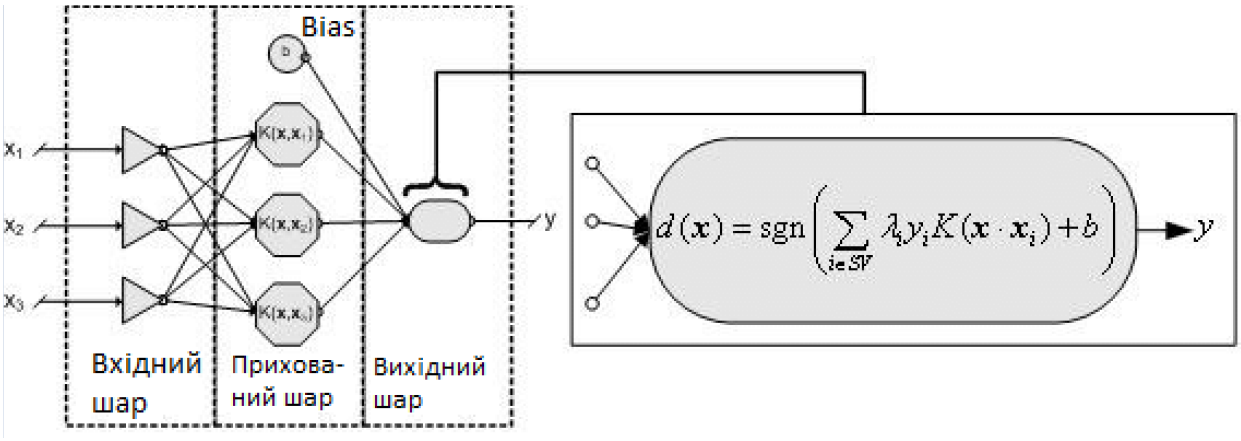


Рисунок 2.7 – Архітектура SVM класифікатора

Для того щоб розробити власну модель на основі глибоких нейронних мереж та SVM класифікатора, необхідно розглянути принцип роботи та навчання глибоких нейронних мереж та мережі SVM класифікатора.

2.3 Розробка математичних моделей нейронних мереж та SVM класифікатора

Глибинне навчання виявляється як метод навчання машин, що передбачає автономне вивчення загальних правил з наданих даних, намагаючись симулювати процес навчання штучної нейронної мережі. При навчанні нейронних мереж, особливо для систем обробки природної мови, використовується навчання з учителем, тобто на конкретних прикладах даних з передбачуваними результатами. В той час як термін "глибоке навчання" вказує на навчання в "глибоких" штучних нейронних мережах, що включають сотні "прихованих" шарів між вхідним і вихідним шарами для обробки введення та виведення. Матриця прихованого шару служить вхідними даними для наступного шару. У цьому випадку лише матриця вихідних даних останнього шару містить результат.

Основне завдання в навчанні нейронних мереж - мінімізація функції помилки [13], зазвичай, оптимізуючи апостеріорну ймовірність. Для навчання мережі необхідно подати підготовлені дані і порівняти згенеровані нею вихідні результати з результатами нашого тестового набору даних. Оскільки мережа ще не навчена, результати будуть неточними. Після пропуску всіх даних можна визначити функцію, яка вказуватиме на відмінність результатів роботи алгоритму від реальних даних. Ця функція називається функцією втрат. В ідеалі ми хотіли б, щоб функція втрат дорівнювала нулю. Тоді вихідні результати роботи мережі повністю збігаються з результатами тестового набору даних.

Для налаштування мережі потрібно коригувати ваги між нейронами. Це можна робити випадковим чином, поки функція втрат не буде дорівнювати нулю, але цей метод не є дуже ефективним. Натомість ми будемо скористовуватися методом градієнтного спуску. Градієнтний спуск - це техніка, що дозволяє знайти мінімум функції. У нашому випадку ми шукаємо мінімум функції втрат. Суть методу полягає в невеликих змінах параметрів після кожної

ітерації. Обчислюючи похідну (або градієнт) функції втрат для певного набору параметрів, можна визначити, у якому напрямку знаходиться мінімум.

Розглянемо один із найвідоміших методів навчання штучних нейронних мереж з використанням градієнтного спуску - алгоритм зворотнього поширення, або backpropagation [14]. Це ітеративний градієнтний метод, що використовується для мінімізації помилки роботи багатошарового перцептрону та отримання бажаного виходу. Основна ідея цього методу полягає в передачі сигналів помилки від виходів мережі до її входів, в напрямку, протилежному прямому розповсюдженню сигналів у звичайному режимі. Щоб можна було застосовувати метод зворотного поширення помилки, функція активації нейронів повинна бути диференційованою.

Для кращого розуміння спершу розглянемо принцип навчання одношарового перцептрону, який зображений на рисунку 2.8.

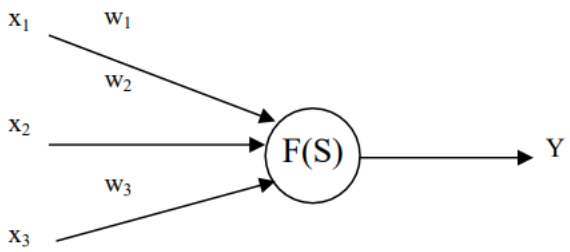


Рисунок 2.8 – Штучний нейрон

Зважена сума та функція активації даного нейрона (формула 2.1):

$$S_i = x_{i-3} \cdot w_1 + x_{i-2} \cdot w_2 + x_{i-1} \cdot w_3,$$
$$Y_i = 1/(1+\exp(-S_i))*10, \quad (2.1)$$

де w_1, w_2, w_3 – синаптичні ваги;

x_{i-3}, x_{i-2}, x_{i-1} – вхідні сигнали – відомі попередні значення часового ряду (і-й набор вхідних даних);

S_i – зважена сума і-го набору вхідних даних;

Y_i – прогнозоване значення і-го члена часового ряду x_i ;

10 – масштабний множник

Навчання нейронної мережі полягає в знаходженні таких значень ваг *w*, при яких нейромережа буде здатна видавати на основі вхідних даних вірні вихідні дані з певною наперед заданою точністю. Дана задача задовільно вирішується за допомогою алгоритму зворотного поширення (back propagation) [15], що полягає в наступному:

1) Спочатку всі вагові коефіцієнти нейронної мережі встановлюються довільно.

2) Через мережу пропускаються тренувальні дані (перший набір вхідних даних), і обчислюється сумарна функція помилки (сума квадратів помилки), яка зображена на формулі 2.2:

$$E = \sum_{i=1}^{N} (Y_i - y_i)^2 , \tag{2.2}$$

де Y_i – обчислене значення виходу нейрона;

N – кількість нейронів вихідного шару;

y_i – правильне значення наступного члену часового ряду.

3) Обчислюється значення похідної функції помилки E't для кожного вагового коефіцієнта (формула 2.3):

$$\boldsymbol{E_i' = (Y_i - y_i) \cdot \left(\frac{\exp(-s_i)}{(1 + \exp(-s_i))^2} \right) \cdot x_i,} \tag{2.3}$$

4) На основі E'_{i1}, E'_{i2}, та E'_{i3}, здійснюється розрахунок виправлень Δw_{i1}, Δw_{i2}, та Δw_{i3} до відповідних вагомвих коефіцієнтів за формулою 2.4:

$$\Delta w_i = -\nu \cdot E'_i, \tag{2.4}$$

де v – коефіцієнт швидкості навчання.

Виправлення необхідно знайти для кожного і-го набору вихідних даних, і обчислити середні значення $\Delta w_{\text{середнє1}}$, $\Delta w_{\text{середнє2}}$, та $\Delta w_{\text{середнє3}}$ для всього набору:

$$\Delta w_{середнє} = \frac{1}{N}\sum_{i=1}^{N}\Delta w_i \tag{2.5}$$

де N – кількість наборів вхідних даних для навчання.

5) Вагові коефіцієнти коректуються на величину обчислених виправлень (формула 2.6):

$$w = w + \Delta w_{середнє}, \tag{2.6}$$

6) Поточне значення сумарної функції помилки E зберігається в іншій змінній: $E_0 = E$.

Кроки 2–5 алгоритму зворотного поширення повторюються для кожного і-того набору вхідних даних (назвемо це цикли навчання), поки функція помилки не знизиться до заданого рівня, наприклад $|E - E_0| < 0{,}0001$.

Згадаймо про поняття та основні різновиди функцій активації, які використовуються найчастіше у алгоритмах зворотного навчання.

Функція активації F(S) – визначає залежність сигналу на виході нейрона від зваженої суми сигналів на його входах. Використання різних функцій активації дозволяє вносити нелінійність в роботу нейрона і в цілому нейронної мережі. Приклади функцій активації:

Лінійна передавальна функція (формула 2.7):

$$F(S) = \begin{cases} 0 \ if \ S \le 0 \\ 1 \ if \ S \ge 1 \\ S \ else \end{cases}, \tag{2.7}$$

Сигмоїдальна передавальна функція (формула 2.8):

$$F(S) = \frac{1}{(1 + \exp(-S))}, \tag{2.8}$$

Багатошарові мережі дозволяють створювати більш складні, нелінійні зв'язки між вхідними даними й результатами на виході. На рис. 9. багатошарова мережа складається із вхідного, проміжного (або схованого) і вихідного шарів.

Типова мережа BackPropagation має вхідний прошарок, вихідний прошарок та принаймні один прихований прошарок. Теоретично, обмежень відносно числа прихованих прошарків не існує, але практично застосовують один або два. Нейрони організовано в пошарову структуру з прямою передачею (вперед) сигналу. Кожний нейрон мережі продукує зважену суму своїх входів, пропускає цю величину через передатну функцію і видає вихідне значення. Мережа може моделювати функцію практично будь якої складності, причому число прошарків і число нейронів у кожному прошарку визначають складність функції. Важливим при моделюванні мережі є визначення числа проміжних прошарків і числа нейронів в них. Більшість дослідників та інженерів використовують загальні правила, зокрема: Кількість входів та виходів мережі визначаються кількістю вхідних та вихідних параметрів досліджуваного об'єкту, явища, процесу, тощо.

На відміну від зовнішніх прошарків, число нейронів прихованого прошарку n при x обирається емпіричним шляхом. Якщо складність у відношенні між отриманими та бажаними даними на виході збільшується, кількість нейронів прихованого прошарку повинна також збільшитись. Після

того, як визначено число прошарків і число нейронів в кожному з них, потрібно знайти значення для синаптичних ваг і порогів мережі, які спроможні мінімізувати похибку спродукованого результату. Саме для цього існують алгоритми навчання, де відбувається підгонка моделі мережі до наявних навчальних даних. Похибка для конкретної моделі мережі визначається шляхом проходження через мережу всіх навчальних прикладів і порівняння спродукованих вихідних значень з бажаними значеннями. Множина похибок створює функцію похибок, значення якої можна розглядати, як похибку мережі.

Мета навчання нейромережі полягає в знаходженні найнижчої точки. Поверхня станів має складну будову і досить неприємні властивості, зокрема, наявність локальних мінімумів (точки, найнижчі в своєму певному околі, але вищі від глобального мінімуму), пласкі ділянки, сідлові точки і довгі вузькі яри. Аналітичними засобами неможливо визначити розташування глобального мінімуму на поверхні станів, тому навчання нейромережі по суті полягає в дослідженні цієї поверхні. Відштовхуючись від початкової конфігурації ваг і порогів (від випадково обраної точки на поверхні), алгоритм навчання поступово відшукує глобальний мінімум. Обчислюється вектор градієнту поверхні похибок, який вказує напрямок найкоротшого спуску по поверхні з заданої точки. Якщо трошки просунутись по ньому, похибка зменшиться. Зрештою алгоритм зупиняється в нижній точці, що може виявитись лише локальним мінімумом (в ідеальному випадку - глобальним мінімумом). Складність полягає у виборі довжини кроків. При великій довжині кроку збіжність буде швидшою, але є небезпека перестрибнути рішення, або піти в неправильному напрямку. При маленькому кроці, правильний напрямок буде виявлено, але зростає кількість ітерацій. На практиці розмір кроку береться пропорційним крутизні схилу з деякою константою – швидкістю навчання. Процес навчання припиняється або коли пройдена визначена кількість епох, або коли похибка досягає визначеного рівня малості, або коли похибка перестає зменшуватись (користувач переважно сам вибирає потрібний критерій зупинки).

Популярність алгоритму зворотного поширення нещодавно відродилася, враховуючи широке поширення глибоких нейронних мереж для розпізнавання зображень і мовлення. Він вважається ефективним алгоритмом, і сучасні реалізації використовують переваги спеціалізованих графічних процесорів для подальшого підвищення продуктивності [16].

Далі розглянемо навчання SVM класифікатора, або ж методу опорних векторів. Даний метод спочатку відносився до бінарних класифікаторів, хоча існують способи змусити його працювати і для задач мультикласифікації. Ідею методу зручно проілюструвати на наступному простому прикладі: точки на площині, розбиті на два класи. Проведемо лінію, яка розділяє ці два класи. Далі всі нові точки (не з навчальної вибірки) автоматично класифікуються наступним чином: точка вище прямої потрапляє до класу A, точка нижче прямої - до класу B, приклад зображено на рисунку 2.9.

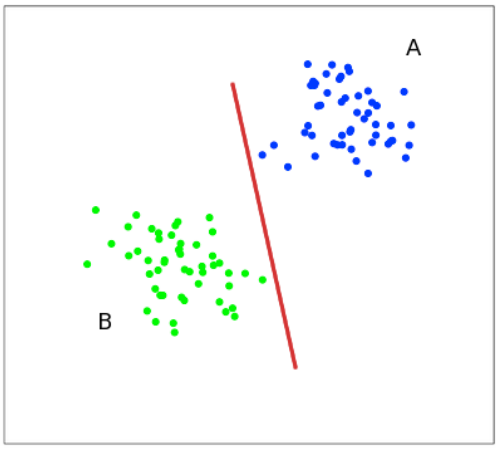


Рисунок 2.9 – Приклад методу опорних векторів

Таку пряму назвемо прямою, що розділяє. Проте, у просторах високих розмірностей пряма не буде розділяти наші класи, оскільки поняття «нижче прямої» чи «вище прямої» втрачає всякий сенс. Тому замість прямих необхідно розглядати гіперплощини - простори, розмірність яких на одиницю менша, ніж розмірність вихідного простору. В, наприклад, гіперплощина - це звичайна

двовимірна площина [17]. Але може існувати кілька прямих, які розділяють два класи, як показано на рисунку 2.10.

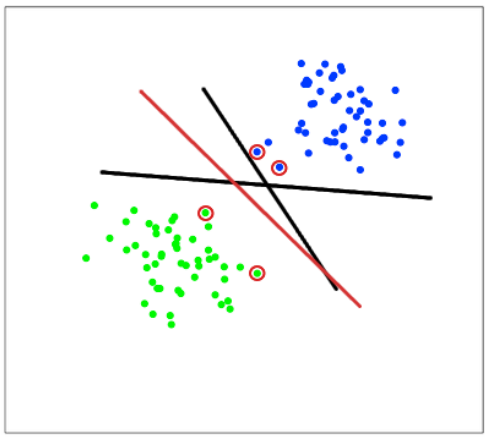


Рисунок 2.10 – приклад методу опорних векторів з декількома розділювальними прямими

З погляду точності класифікації краще вибрати пряму, відстань від якої до кожного класу максимально. Іншими словами, виберемо ту пряму, яка розділяє класи якнайкраще (червона пряма на рисунку 2.9). Така пряма, а в загальному випадку - гіперплощина, називається оптимальною роздільною гіперплощиною. Вектори, що лежать найближче до роздільної гіперплощини, називаються опорними векторами (support vectors).

Метод опорних векторів будує класифікуючу функцію F у вигляді $F(x) = sign(\langle w, x\rangle + b)$, де $\langle w, x\rangle$- скалярний добуток, w - нормальний вектор до роздільної гіперплощини, b - допоміжний параметр. Ті об'єкти, для яких $F(x) = 1$ потрапляють до одного класу, а об'єкти з $F(x) = -1$ — до іншого. Вибір саме такої функції невипадковий: будь-яка гіперплощина може бути задана у вигляді $\langle w, x\rangle + b = 0$ для деяких w та b. Далі, ми хочемо вибрати такі w і b, які максимізують відстань до кожного класу. Можна підрахувати, що ця відстань дорівнює $\frac{1}{\|w\|}$. Проблема знаходження максимуму $\frac{1}{\|w\|}$ еквівалентна проблемі знаходження мінімуму $\|w\|^2$. Запишемо все це як завдання оптимізації (формула 2.9):

$$\begin{cases} \arg\min_{w,b}\|w\|^2, \\ y_i(\langle w, x\rangle + b) \geq 1, i = 1, \dots, m. \end{cases} \quad (2.9)$$

яка є стандартним завданням квадратичного програмування та вирішується за допомогою множників Лагранжа.

Насправді випадки, коли дані можна розділити гіперплощиною, або, як ще кажуть, лінійно, досить рідкісні. Приклад лінійної нероздільності можна побачити рисунку 2.11. В такому випадку всі елементи навчальної вибірки вкладаються у простір X вищої розмірності з допомогою спеціального відображення $\varphi: R^n \to x$. При цьому відображення φ вибирається так, щоб у новому просторі X вибірка була лінійно розділена.

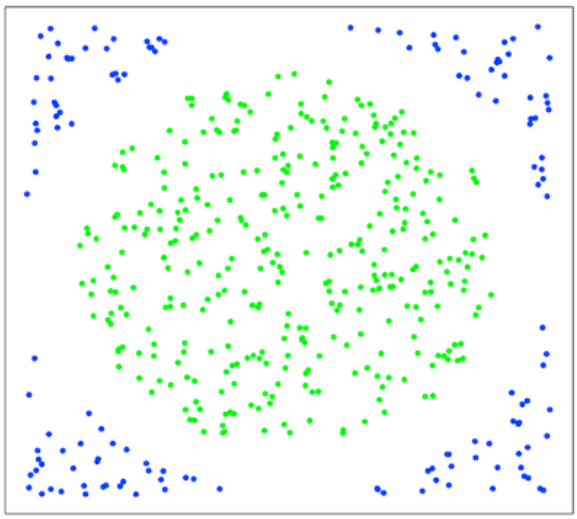


Рисунок 2.11 – Приклад лінійної нероздільності

Класифікуюча функція F набуває вигляду $F(x) = sign(\langle w, \varphi(x)\rangle + b)$. Вираз $k(x, x') = \langle \varphi(x), \varphi(x')\rangle$ називається ядром класифікатора. З математичної точки зору ядром може бути будь-яка позитивно визначена симетрична функція двох змінних. Позитивна визначеність необхідна у тому, щоб відповідна функція Лагранжа завдання оптимізації була обмежена знизу, тобто завдання оптимізації було б коректно визначено. Частіше всього на практиці зустрічаються такі функції ядра як: поліном, радіальна базисна функція та сигмоїда.

2.4 Модифікація архітектури LangChain у контексті технології парсингу резюме

У даному продукті важлива ефективна комунікації програми з користувачем. Програмний модель парсингу резюме, що був створений для бакалаврської дипломної роботи не мав при собі жодних механізмів, які можна було б назвати «зручними» для обробки результатів роботи. Для того, щоб отримувати «людську» відповідь на результати роботи парсеру можна використовувати різноманітні мапінги, якщо-то оператори і т.п. Але це все одне не зможе повністю покрити усе різноманіття відповідей. Також покращений парсинг резюме націлений на надання рекомендацій щодо отриманих даних, то ж для такого стеку задач – прості логічні чи математичні оператори не будуть вигідним рішенням.

LangChain – фреймворк, що спрямований на розробку застосунків із використанням великих мовних моделей. LangChain слідує загальному пайплайну, коли користувач задає питання мовній моделі, де векторне представлення питання використовується для пошуку подібності у векторній базі даних, а відповідна інформація витягується з векторної бази даних, а відповідь пізніше передається до модель мови. далі, мовна модель генерує відповідь або виконує дію. [19]

Технологія парсингу резюме націлена на створення відповідей, що схожі на спілкування з людиною, а також, у наданні рекомендації, виходячи з бази даних резюме, яку користувач буде створювати самостійно.

Однією з найголовніших особливостей фреймворку є концепція ланцюгів. Ланцюги дозволяють розробникам поєднувати кілька компонентів, таких як шаблони запитів, мовні моделі та інші спеціалізовані операції, в єдиний, послідовний процес. Окрім цього, LangChain може використовувати зовнішні джерела даних для генерації тексту, що надає можливості резюмування довгих текстів та формування відповіді на запитання з використанням заданого користувачем контексту.

Це потужний інструмент, який можна використовувати для створення широкого спектру додатків на базі LLM. Він простий у використанні та має велику спільноту користувачів і учасників.

- Аналіз та сумаризація документів.
- Чат-боти: LangChain можна використовувати для створення чат-ботів, які природно взаємодіють з користувачами. Наприклад, LangChain можна використовувати для створення чат-бота, який може відповідати на запитання клієнтів, надавати допомогу клієнтам і навіть призначати зустрічі.
- Аналіз коду.
- Відповіді на запитання за допомогою джерел: LangChain можна використовувати для відповідей на запитання за допомогою різних джерел, включаючи текст, код і дані. Наприклад, LangChain можна використовувати для відповідей на запитання щодо певної теми шляхом пошуку в різних джерелах, таких як Вікіпедія, новинні статті та сховища коду.
- Збільшення даних: збільшення даних шляхом створення нових даних, подібних до існуючих. Наприклад, створення нових текстових даних, схожих на існуючі текстові дані. Це може бути корисним для навчання моделей машинного навчання або для створення нових наборів даних.
- Класифікація тексту та резюмування тексту.
- Машинний переклад: LangChain можна використовувати для перекладу вхідних текстових даних різними мовами.

Основні властивості LangChain Framework:

- Компоненти: Компоненти — це модульні будівельні блоки, готові та прості у використанні для створення потужних програм. Компоненти включають LLM Wrappers, шаблон запиту та індекси для пошуку відповідної інформації.
- Ланцюжки: ланцюги дозволяють нам об’єднувати декілька компонентів разом для вирішення конкретного завдання. Ланцюги полегшують

реалізацію складних програм, роблячи їх більш модульними та простими для налагодження та обслуговування.

- Агенти: Агенти дозволяють LLM взаємодіяти зі своїм середовищем. Наприклад, використання зовнішнього API для виконання певної дії.

На додаток до генерації тексту, LangChain може витягувати векторні вбудування (надалі ембедінги) з мовних моделей. Ембедінги представляють внутрішнє семантичне значення слів і сутностей LLM. Основні використання ембедінгів:

- Семантичний пошук: вбудовані індекси для пошуку документів, схожих на запит за значенням, а не лише за ключовими словами.
- Системи рекомендацій: знайдіть пов’язані продукти/вміст на основі подібності вбудовування.
- Аналіз настроїв: кластеризуйте позитивні та негативні вкладення для класифікації настроїв.
- Zero-Shot Classification: класифікуйте нові документи на основі близькості до прикладів вбудовування кожного класу.

LangChain надає такі інструменти, як EmbeddingsDatabase, для індексування та запиту вбудованих компонентів, отриманих із базового LLM [20]. Це дозволяє створювати передові програми, використовуючи неявні знання моделі.

У нашому випадкудля інформаційної технології буде використоано API GPT 3.5 turbo, це технологія, що була створена компанією OpenAI. Архітектура GPT-3 базується на моделі трансформатора та працює з послідовностями слів, також відомих як токени1. Ось спрощене пояснення того, як це працює:

Введення/виведення: введенням для GPT-3 є послідовність із N слів (токенів). Вихід – це припущення слова, яке, швидше за все, буде розміщено в кінці вхідної послідовності. Наприклад, якщо вхідна послідовність: «Не всі герої носять», результатом може бути «накидки». Якщо нам потрібно більше одного слова, ми можемо додати вихідне слово до послідовності та отримати наступне слово [21].

Кодування: Оскільки GPT-3 працює з векторами чисел, йому потрібно перетворювати слова на вектори. Це робиться шляхом присвоєння кожному слову значення зі словника всіх слів (GPT-3 має словник із 50257 слів). Потім кожне слово перетворюється на одноразовий вектор кодування розміром 50257, де лише розмірність за індексом і (значення слова) дорівнює 1, а всі інші – 01.

Механізм уваги: механізм уваги в GPT-3 дозволяє зосереджуватися на різних частинах вхідної послідовності під час прогнозування кожного вихідного токена. На практиці моделі уваги можуть використовувати один великий об’єднаний тензор ваги для всіх голів, виконати множення матриці один раз, а потім розділити його на матриці a, k, v.

Послідовність виводу. Вихідні дані GPT — це не просто одне припущення, це послідовність (довжиною 2048) припущень (ймовірність для кожного вірогідного слова). По одному для кожної «наступної» позиції в послідовності. Але під час генерування тексту ми зазвичай дивимось лише на припущення для останнього слова послідовності.

Математичне представлення цих процесів включає складні формули та матричні операції. Наприклад, механізм уваги можна представити у вигляді:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V,$$

Позиційне кодування: оскільки трансформаторна модель не має жодного внутрішнього відчуття позиції (вона взаємозамінно розглядає вхідні позиції), нам потрібно додати деяку інформацію про відносну чи абсолютну позицію слів у послідовності [21]. Це робиться шляхом додавання позиційних кодувань до вбудованих вхідних даних. Позиційні кодування мають ту саму розмірність, що й вкладення, тому їх можна підсумувати. Є багато варіантів позиційного кодування, як-от навчене позиційне кодування та фіксоване позиційне кодування.

Рівні транформерів: модель трансформатора складається з шарів, кожен з яких має два підрівні: механізм самоконтролю з кількома головками та просту, позиційно повністю зв'язану мережу прямого зв'язку. Результатом кожного підрівня є

LayerNorm(x + Sublayer(x)), де Sublayer(x) — це функція, реалізована підрівнем. Щоб полегшити ці залишкові зв'язки, усі підрівні в моделі, а також шари вбудовування створюють вихідні дані розмірності d_model=768.

Декодування: Останнім кроком у GPT-3 є перетворення вихідних векторів назад у слова. Це робиться шляхом застосування лінійного перетворення, а потім функції softmax, щоб отримати ймовірності для кожного слова в словнику. Слово з найбільшою ймовірністю вибирається як вихідне слово.

Приклад архітектури можна побачити на рисунку 2.12.

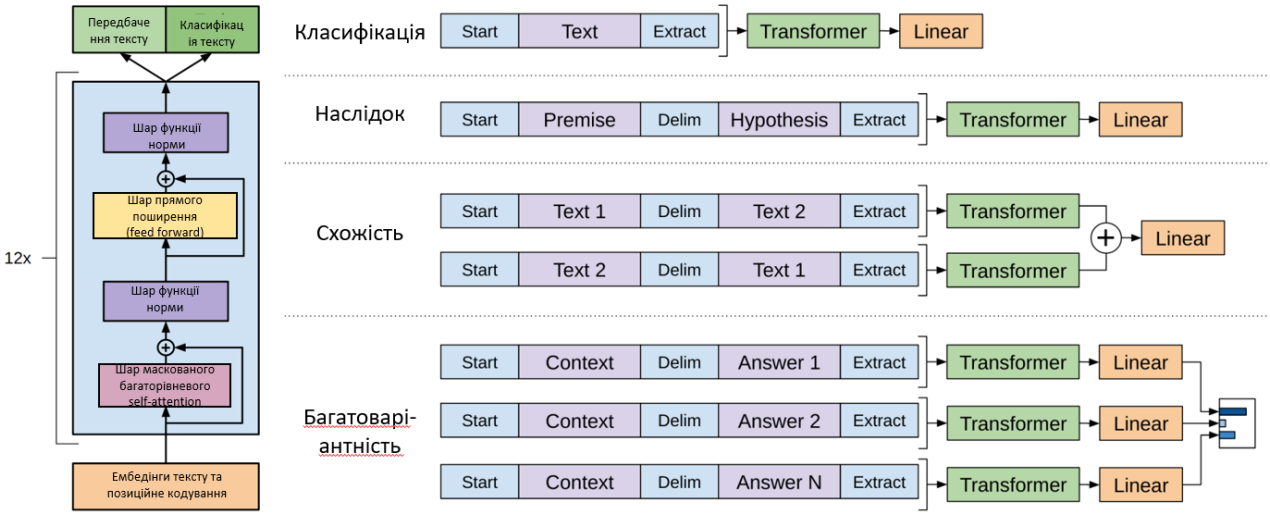


Рисунок 2.12 – Архітектура моделі GPT-3

Далі ми розглянемо можливості машинного навчання LangChain.

Основна концепція LangChain полягає в його здатності об'єднувати «ланцюжок думок» навколо LLM, як видно з його назви. Однак LangChain не обмежується кількома LLM — він надає широкий спектр компонентів, які працюють разом як будівельні блоки для складних випадків використання LLM.

обмежується кількома LLM — він надає широкий спектр компонентів, які працюють разом як будівельні блоки для складних випадків використання LLM.

LLM попередньо навчаються на масивних текстових наборах даних, насамперед на наборах даних загального призначення, які не є предметно-спеціальними. Це дозволяє моделі навчатися з широкого кола джерел, що полегшує розуміння моделлю основних мовних завдань.

Деякі LLM попередньо натреновані з використанням загальних і специфічних для домену наборів даних, щоб забезпечити високу продуктивність моделі в певному домені, а також виконувати загальні завдання. Під час навчання LLM вивчають слова, їх значення, структуру, зв’язки та текстові шаблони в мові, що дозволяє їм розуміти контекст і генерувати відповідний текст.

Типова архітектура LLM складається з кількох рівнів нейронних мереж, рівнів прямого зв’язку, рівнів вбудовування та рівнів уваги, які працюють разом, щоб обробляти вхідні дані та генерувати відповіді.

Хоча LangChain моделі можуть генерувати дивовижний вміст, вони також відомі тим, що генерують неточну інформацію, яка може вводити в оману. Під час інтеграції LLM у програмний продукт парсеру резюме, я вроваджував перевірки у вигляді промтів та програмних стоперів, щоб гарантувати, що модель не вигадує вміст, коли вона не має чіткої відповіді.

То ж для задач, які поставлені для технології парсингу резюме – LangChain та його структури є підходящими по концепції. Для реалізації зв’язку контенту з моделями LangChain потрібні знання у сфері машинного навчання, парсингу даних, попередньої обробки даних, prompt engeneering та у деяких випадках – fine-tuning

2.5 Розробка алгоритму роботи інформаційної технології парсингу резюме

Алгоритм роботи інформаційної технології парсингу резюме, розроблений на основі всіх описаних у цьому розділі процедур, представлено на рис. 2.13 і він складається з 4 етапів:

1) Навчання LSTM із залишковими зв’язками, що включає в себе завантаження датасету, обробку тексту, генерування та тренування LSTM із залишковими зв’язками за допомогою бібліотке та збергання моделі.

2) Навчання SVM класифікатора, що влючає в себе завантаження датасету множин речень зі скілами та без них, обробку тексу, згенерування та тренування SVM класифікатора та збереження.

3) Поєднання, що включає в себе завантаження класифікаторів з диску, обрання тестового документу, екстрактинг тексту із документу та його попередню обробку, застосування завантажених моделей та функцій.

4) Виведення, обробка отриманих результатів окремими алгоритмами на основі великої мовної моделелі, вибір засобу для обробки результатів, відповідно до запиту користувача, та виведення результатів у користувацький інтерфейс.

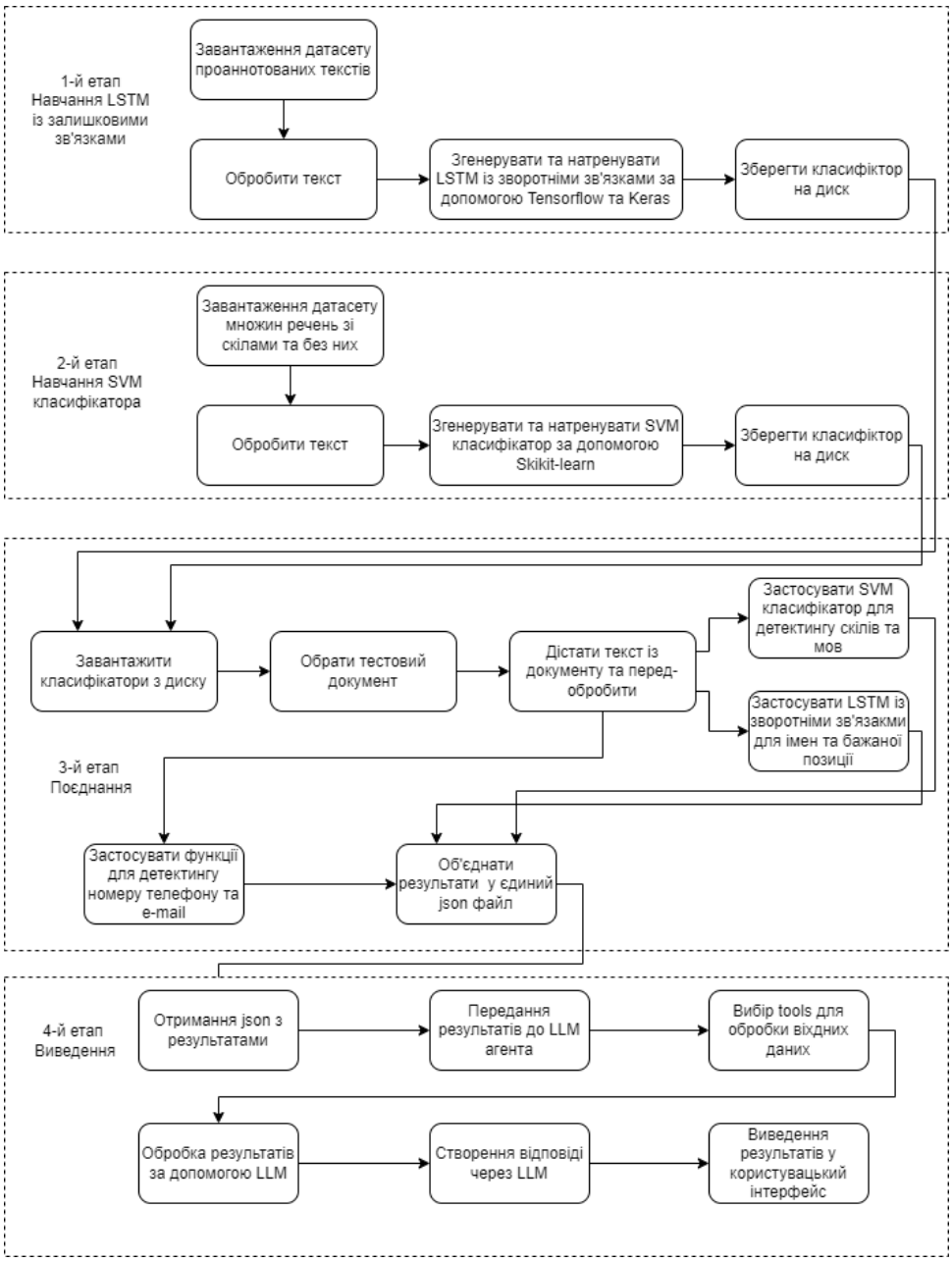


Рисунок 2.13 - Алгоритм роботи інформаційної технології парсингу резюме

2.6 Висновок до розділу 2

У цьому розділі було проаналізовано різні структури нейронних мереж та обґрунтовано вибір нейронної мережі для даної задачі. У даному розділі було виділено основні компоненти інформаційної технології парсингу резюме та побудовано алгоритм роботи інформаційної технології парсингу резюме, що спрощує процес створення програмного забезпечення.

3 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПАРСИНГУ РЕЗЮМЕ

3.1 Обґрунтування вибору мови програмування

Python - це потужна та універсальна мова програмування, яка підкреслює читабельність та простоту. Вона динамічна та інтерпретована, що означає, що код виконується безпосередньо, а не скомпільований в машинний код. Python підтримує кілька парадигм програмування, включаючи структурне, процедурне, об'єктно-орієнтоване та функціональне. Він також має потужні бібліотеки для обробки даних, машинного навчання та штучного інтелекту.

Python широко використовується для розробки веб-додатків, машинного навчання, штучного інтелекту та науки про дані.

Ось деякі з ключових особливостей Python:

- Простота: Python має простий і інтуїтивно зрозумілий синтаксис, що робить його ідеальним для початківців.
- Читабельність: Python підкреслює читабельність коду, використовуючи значні відступи.
- Універсальність: Python підтримує широкий спектр завдань програмування, включаючи веб-розробку, машинне навчання та науку про дані.
- Модульність: Python підтримує модулі, які дозволяють повторно використовувати код.
- Бібліотеки: Python має велику бібліотеку стандартних бібліотек, які забезпечують доступ до широкого спектру функцій.

Python - це багатопарадигмальна мова програмування, яка підтримує об’єктно-орієнтоване, структурне, функціональне та аспектно-орієнтоване програмування. Він має динамічну типізацію, автоматичне управління пам’яттю та динамічне прив’язування імен. Він також дозволяє розширювати свою функціональність за допомогою модулів, які можуть бути написані на Python або

інших мовах, таких як C. Python надає велику стандартну бібліотеку, яка містить різноманітні інструменти для роботи з текстом, файлами, мережами, базами даних, графікою та іншими областями.

Python славиться своєю читабельністю та елегантністю коду. Він базується на філософії дизайну, яка висловлюється в документі Zen of Python (PEP 20), де сказано, що “простота краще за складність”, “ясність краще за неясність” та “повинен бути один очевидний спосіб зробити щось”. Python намагається зменшити когнітивне навантаження на програміста та сприяти ефективності та виразності коду.

Одна з проблем Python - це його безпека. У всіх версіях Python виявлено критичні уразливості, які можуть дозволити зловмисникам виконувати віддалений код або отруювати веб-кеш. Це призвело до швидкого випуску оновлень Python 3.9 та 3.10, які виправляють ці проблеми. Користувачам рекомендується оновити свої версії Python до останньої доступної версії.

Python призначений для надання певної підтримки функціонального програмування на основі Lisp. За допомогою функцій фільтрування, відображення та скорочення; перелічено розуміння генераторів, словників, наборів та виразів. Стандартна бібліотека має два модулі (itertools та functools) для реалізації функціональних інструментів, запозичених у Haskell та Standard ML. Основні принципи мови викладені в документі Zen of Python (PEP 20), який включає такі сентенції:

- Красиво-краще, ніж потворно;
- Явне краще, ніж неявне;
- Просте - краще, ніж складне;
- Комплекс краще, ніж комплекс;
- Розрахунок читабельності.

Python не має всіх своїх функцій, вбудованих у своє ядро, але розроблений таким чином, щоб бути дуже розширюваним (з модулями).

Ця компактна модульність робить його особливо популярним як спосіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван

Россума щодо малих основних мов, великих стандартних бібліотек та легко розширюваних перекладачів випливає з його невдоволення ABC, який підтримує протилежний підхід. Python прагне спростити та зменшити плутанину граматики та граматики, дозволяючи розробникам обирати методи кодування. На відміну від девізу Perl "Існує кілька способів зробити це", Python висвітлює філософію дизайну "Для цього повинен бути один - бажано, лише один очевидний спосіб" [3]. Алекс Мартеллі, співробітник Фонду програмного забезпечення Python і автор книги про Python, писав: "Опис чогось як "розумного " не вважається компліментом до культури Python".

Розробники Python прагнуть до простоти та читабельності коду. Вони уникають передчасної оптимізації, яка може ускладнити код і зробити його менш зрозумілим. Коли швидкість є критичним фактором, програмісти Python можуть використовувати розширення, написані мовами, такими як C, або компілятори, такі як PyPy або Cython.

Python має ряд особливостей, які сприяють його простоті та читабельності. Ключові слова Python є англійськими, а не спеціальними символами, як у багатьох інших мовах. Блоки коду виділяються за допомогою відступів, а не фігурних дужок. Це робить код більш візуально зрозумілим.

Розробники Python вважають, що краще спочатку написати простий і зрозумілий код, а потім оптимізувати його, якщо це необхідно. Передчасна оптимізація може призвести до коду, який важко зрозуміти і підтримувати.

Розширення Python дозволяють програмістам додавати до мови нові функції та можливості. Компілятори Python можуть стати в нагоді, коли швидкість є критичним фактором. Відступи є важливим аспектом синтаксису Python, оскільки вони використовуються для розділення блоків коду. Рекомендований розмір відступу - чотири пробіли. Оператори Python включають (серед іншого):

- Використовуйте знак рівності = оператор присвоєння;
- Оператор if, який використовується разом з else та elif (скорочення від else-if) для умовного виконання блоку коду;

- Оператор for, який фіксує кожен елемент у локальній змінній для використання вкладеним блоком для перегляду об'єкта, що ітерацію;
- Поки оператор, доки умова є істинним, виконується блок коду;
- Оператор try, який дозволяє захоплювати та обробляти витяги, що містяться у вкладених блоках коду (крім речень); він також гарантує, що незалежно від того, як блок виходить, код очищення в блоці завжди буде виконаний в кінцевому підсумку;
- Оператор підвищення застосовується для отримання зазначеного винятку або повторного підняття спійманого винятку;
- Оператори класів, які виконують блоки коду та приєднують свій локальний простір імен до класу для використання в об'єктно-орієнтованому програмуванні;
- Оператор Def, який визначає функцію або метод;
- Оператор with з Python 2.5, випущений у вересні 2006 р. [4], який охоплює блок коду в диспетчері контексту (наприклад, отримання блокування перед запуском кодового блоку, а потім звільнення блокування або відкриття файлу. Потім закрийте його), дозволяють поведінку подібних ресурсів ініціалізувати (RAII) та замінюють загальне використання try / final;
- Оператор Break, вихід із циклу;
- Оператор continue пропускає цю ітерацію та переходить до наступного пункту;
- Оператор del видалить змінну, що означає, що посилання на ім'я значення буде видалено, а спроби використання цієї змінної призведуть до помилки. Ви можете перепризначити видалені змінні;
- Виконувати функції NOP через операторів. Синтаксично це необхідно для створення порожнього блоку коду;
- Заява про затвердження, що використовується для перевірки умов, що застосовуються під час налагодження;

- Оператор Yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану;
- Оператор return використовується для повернення значень із функцій;

Перехід від мовних конструкцій до машинних команд здійснює транслятор. Одна мова програмування може бути представлена різними реалізаціями, тобто зберігаються основні принципи й ідеї мови, розробляються зовсім різні транслятори, у синтаксисі допускаються деякі відмінності. Вибір мови програмування залежить не тільки від її можливостей і сфери застосування, але і від великої кількості інших факторів, таких як: тип ліцензії, наявність зручних середовищ програмування, об’єм спільноти, кількість та якість відповідних бібліотек та фреймворків.

Синтаксис Python чіткий і стислий. Мова розроблена так, щоб вона була читабельною і була близькою до фактичної англійської, що дозволяє легко її розшифрувати. Python також вимагає менше рядків коду для досягнення результатів порівняно з мовами, такими як C або Java.

Простота Python дуже допомагає, коли вам доведеться прочитати написаний вами код або код від іншого розробника. Перегляд коду проходить набагато швидше, коли у вас є менше рядків, і код читається як англійська.

C# це об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. C# створювалася як мова компонентного програмування, і в цьому одне з головних переваг мови, спрямоване на можливість повторного використання створених компонентів. З інших об'єктивних факторів відзначимо наступні; C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами; C# є потужною об'єктною мовою з можливостями спадкування й універсалізації. Разом з тим C# передусім розроблялась як мова програмування прикладного рівня і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#.

Java — об’єктно-орієнтована мова, разом з ООП вивчаються концепції наслідування, абстракції, поліморфізму і так далі. Java навчить концепціям, які можна застосовувати в більшості інших мов, наприклад, в Python Некомерційна організація Cloud Foundry Foundation (CFF) опублікувала рейтинг найбільш затребуваних мов програмування для корпоративних хмарних розробок. Згідно з їхнім дослідженням, лідером стала Java. Java має гарно пророблений API, великий вибір інструментарію, велику кількість фреймворків.

Порівняльна характеристика мов програмування Java, C# і Python з їх кросплатфоменними рішеннями наведена у таблиці 3.1.

Таблиця 3.1 – Порівняння різних мов програмування

	Kotlin(Java)	Xamarin(C#)	Python
Imperative	+	+	+
ООП	+	+	+
Functional	-	+	-
Libraries for ML	-	-	+
Good perfomance	+	+	+
Big community	+	-	+
Cheap development	+	-	+

Усі мови мають свої переваги та недоліки, але найкращою мовою для реалізації системи є Python, адже ця мова є більш зрозумілою та легшою у використанні, має кращу підтримку зі сторони розробників та спільноти, а також має у своєму арсеналі багато засобів для машинного навчання та аналізу.

Тепер порівняємо деякі IDE для реалізації Python коду. PyCharm — серія продуктів фірми Jet Brains, розроблена на основі платформи IntelliJ IDEA, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як

консольні програми, так і програми з графічним інтерфейсом, а також веб-сайти, веб-застосунки, веб-служби.

Середовище PyCharm містить зручні засоби розробки Python- застосувань, що виконують замість програміста рутинну роботу - створення шаблонів застосувань і форм, заготовок обробників подій, організацію циклу обробки повідомлень і т. д [19].

Sublime Text — це вільне програмне забезпечення з широкою підтримкою спільноти. Редактор працює з кількома програмними мовами. Найчастіше можна використовувати незареєстрований Sublime Text, але іноді ви будете отримувати спливаюче вікно з проханням зареєструвати товар і придбати ліцензію. Sublime Text тонко налаштовується та доповнюється пакетами для налагодження, автозавершення коду, лінтингу тощо.

Таблиця 3.3 – Порівняння середовищ розробки на Python

Середовище розробки	Плюси	Мінуси
PyCharm	Офіційна. Функціональна; є багато плагінів, з допомогою яких можна розшити функціональність; вбудований менеджер версій.	Платне поширення, потребує багато ресурсів системи.
Spider	Не потребує великої кількості ресурсів; безкоштовна; є деякі плагіни.	Низька функціональність; примітивність рівня текстового редактору.
Visual Studio Code	Офіційна; розроблена Microsoft; безкоштовна; є багато плагінів для полегшення розробки; не вимагає багато ресурсів.	Не виявлено

Visual Studio Code – безкоштовний редактор коду від Microsoft для Windows, Linux та MacOS. Його можливості - налагодження, підсвічування синтаксису, інтелектуальне завершення коду, певні фрагменти коду, рефакторинг та інтеграція з Git. Підтримуються різні мови програмування. Для початку роботи з Python може знадобитись кілька додаткових пакетів, але встановити їх досить просто. Редактор постійно оновлюється. Visual Studio Code — один із найкращих редакторів не лише для Python, а й для інших мов програмування.

Порівняння середовищ розробки наведено у таблиці 3.3.

Для програмної реалізації нейронної мережі та основних модулей API було обрано мову програмування Python.

В якості середовища програмування для Python було обрано IDE Visual Studio Code, що має ряд зручностей, таких як: підсвічування синтаксису, автодоповнення, менеджер пакетів, інтеграція з системами контроля версій і т. д.

3.2 Використання фреймворків та бібліотек

У роботі з машинним навчанням часто використовуються бібліотеки, які надають набори інструментів та алгоритмів для реалізації різних завдань. У даній статті розглянуто використання бібліотек Keras, Tensorflow, Scikit-learn, NumPy, Pandas, PDFMiner, Gensim, SpaCy, LangChain та Streamlit для розробки глибокої нейронної мережі, SVM класифікатора, алгоритму обробки природнього тексту та створення простої UI.

Keras — це відкрита нейромережна бібліотека, написана мовою Python. Вона здатна працювати поверх TensorFlow, Microsoft Cognitive Toolkit, R, Theano та PlaidML. Керас призначена для швидких експериментів з мережами глибинного навчання, її зосереджено на тому, щоби вона була зручною в користуванні, модульною та розширюваною.

TensorFlow — це відкрита програмна бібліотека для машинного навчання, розроблена компанією Google. TensorFlow використовується для вирішення

завдань побудови та тренування нейронної мережі. Основний API для роботи з бібліотекою реалізований для Python, також є реалізації для R, C Sharp, C++, Haskell, Java, Go та Swift.

Scikit-learn — це один з найбільш широко використовуваних пакетів Python для Data Science та Machine Learning. Він дозволяє виконувати безліч операцій та надає безліч алгоритмів. Scikit-learn також пропонує відмінну документацію про свої класи, методи та функції, а також опис алгоритмів, що використовуються. Scikit-Learn підтримує:

- Попередню обробку даних
- Зменшення розмірності
- Вибір моделі
- Регресії
- Класифікацію
- Кластерний аналіз

Pandas — це високорівнева бібліотека Python для аналізу даних. Вона побудована поверх нижчого рівня бібліотеки NumPy, що є великим плюсом у продуктивності. Pandas є найбільш просунутою бібліотекою, що швидко розвивається, для обробки та аналізу даних.

NumPy — це бібліотека мови Python, яка додає підтримку великих багатовимірних масивів і матриць, а також велику бібліотеку високорівневих (і дуже швидких) математичних функцій для роботи з цими масивами.

PDFMiner — це інструмент для вилучення тексту для документів PDF. PDFMiner витягує всі тексти, які відображаються програмно. Він також витягує відповідні місця розташування, назви шрифтів, розміри шрифтів, напрямок написання (горизонтальний або вертикальний) для кожного сегмента тексту. Він не розпізнає текст на зображеннях.

Gensim — це бібліотека з відкритим вихідним кодом для неконтрольованого моделювання тем і обробки природної мови з використанням сучасного статистичного машинного навчання. Gensim розроблено для роботи з

великими текстовими колекціями за допомогою потокової передачі даних та інкрементальних онлайн-алгоритмів.

SpaCy — це бібліотека програмного забезпечення з відкритим вихідним кодом для розширеної обробки природних мов, написана мовами програмування Python і Cython. Бібліотека видається за ліцензією MIT, а її головними розробниками є Метью Хоннібал та Інес Монтані, засновники компанії Explosion, що розробляє програмне забезпечення.

LangChain — це бібліотека машинного навчання для розробки моделей обробки природної мови, написана мовою Python. Бібліотека заснована на архітектурі трансформерів, які є ефективним підходом для обробки текстових даних. LangChain надає широкий спектр функцій для розробки моделей обробки природної мови, таких як:

- Токенізація
- Видалення стоп-слів
- Стемінг
- Лемматизація
- Відповідність слів
- Параметризація
- Тренування

LangChain є потужним інструментом для розробки моделей обробки природної мови. Бібліотека легка у використанні та забезпечує високу продуктивність.

Streamlit — це фреймворк для створення веб-додатків на основі Python. Фреймворк дозволяє швидко та легко створювати інтерактивні веб-додатки з використанням Python.Streamlit надає широкий спектр функцій для створення веб-додатків, таких як: Візуалізація даних, додавання форм та таблиць, створення кнопок, списків, відео і т.д.

Streamlit є хорошим вибором для створення веб-додатків, які використовують машинне навчання. Фреймворк дозволяє легко інтегрувати моделі машинного навчання у веб-додатки.

LangChain та Streamlit можна використовувати разом для створення веб-додатків, які використовують моделі машинного навчання для обробки природної мови. Наприклад, можна використовувати LangChain для розробки моделі класифікації текстів, а Streamlit для створення веб-додатку, який дозволяє користувачам вводити текст і отримувати результат класифікації.

Для розробки глибокої нейронної мережі було обрано бібліотеку Keras. Keras є простим у використанні та дозволяє швидко експериментувати з різними архітектурами нейронних мереж. Для попередньої обробки тексту було обрано бібліотеки NumPy та Pandas. NumPy забезпечує швидке та ефективне обробку даних, а Pandas дозволяє легко працювати з таблицями.

Для розробки SVM класифікатора було обрано бібліотеку Scikit-learn. Scikit-learn є однією з найпопулярніших бібліотек машинного навчання для Python. Вона надає широкий спектр алгоритмів класифікації, у тому числі SVM.

3.3 Програмна реалізація інформаційної технології парсингу резюме

Розробка програмного продукту буде розділена на два основні сегменти. Спочатку навчання мережі SVM, а потім її інтеграція з попередньо навченою глибокою мережею Spacy NER. SVM буде використовуватися для виявлення та класифікації навичок кандидатів. Спочатку важливо зазначити, що навчання моделі та визначення класу зазвичай відбуватимуться з простим текстом. Проте тестування проводитиметься на PDF-файлах, що потребуватиме вилучення повного тексту за допомогою бібліотеки PDFminer.

```
text_pdf = extract_text(os.path.join(cv_dir, cv_name))
```

Цей код запрацює на етапі тестування, оскільки його результатом буде необроблений текст із документа. Текст буде введено в моделі, а модель видасть свій результат. Ми підготуємо текст, перетворивши його на векторне представлення після очищення.

```
        string = re.sub(r'https?://\S+|www\.\S+', '', string)
            string = re.sub(r'<.*?>', '', string)
            for x in string.lower():
                if x in punctuations:
                    string = string.replace(x, "")
            string = string.lower()
            string = ' '.join([word for word in string.split() if word not
in stopwords])
            string = re.sub(r'\s+', ' ', string).strip()
```

Цей код виконує кілька перетворень тексту, усуваючи непотрібні знаки пунктуації, перетворюючи текст на малі літери, обрізаючи зайві пробіли та видаляючи слова, знайдені в корпусі stopwords. stopwords стосуються слів, які не мають контекстуального значення. Якщо модель ігнорує ці слова, ми можемо зменшити шум і зосередитися на контекстуально релевантних термінах. При створенні навчальних матриць для вбудовування слів одним із гіперпараметрів є розмір контекстного вікна (w). Його мінімальне значення встановлено на 1, оскільки для роботи алгоритму потрібен контекст. Давайте використаємо w = 2. Перше слово є цільовим словом, а слова праворуч вважаються контекстними словами. Цей процес передбачає визначення частини мови та можливі комбінації цих частин, де одне слово описує інше. Згодом з усіх текстів ми формуємо список унікальних слів.

```
for i, word in enumerate(text):
        for w in range(window):
            if i + 1 + w < len(text):
                word_lists.append([word] + [text[(i + 1 + w)]])
            if i - w - 1 >= 0:
              word_lists.append([word] + [text[(i - w - 1)]])
words = list(set(text))
words.sort()
unique_word_dict = {}
for i, word in enumerate(words):
     unique_word_dict.update({word: i})
```

До цього моменту те, що ми згенерували, ще не підходить для нейронної мережі, оскільки наші дані існують як пари (цільове слово, контекстне слово).

Щоб увімкнути обчислення, нам потрібен розумний спосіб перетворення цих точок даних у числові точки даних. Одним із таких інтелектуальних підходів є техніка, що називається одноразовим кодуванням. Цей метод перетворює слово на вектор, який складається переважно з нулів, за винятком однієї координати, що дорівнює 1, що представляє відповідне слово. Розмір вектора відповідає кількості унікальних слів у документі.

```
for i, word_list in tqdm(enumerate(word_lists)):
    main_word_index = unique_word_dict.get(word_list[0])
    context_word_index = unique_word_dict.get(word_list[1])
    X_row = np.zeros(n_words)
    Y_row = np.zeros(n_words)
    # One hot encoding
    X_row[main_word_index] = 1
    Y_row[context_word_index] = 1
    X.append(X_row)
    Y.append(Y_row)
```

Тепер ми сформували матриці X і Y на основі пар слів - одне у фокусі, а інші в контексті. Наступний крок передбачає визначення розміру вбудовування. Я вибрав розмірність 2, щоб пізніше впорядковувати слова та спостерігати, чи схожі слова мають тенденцію до утворення кластерів. Розмір прихованого шару представляє наші розміри вбудовування слів. Для вихідного шару функція активації є softmax, тоді як функція активації для прихованого шару залишається лінійною. Розмір вхідних даних відповідає загальній кількості унікальних слів (наша X-матриця має розміри n x 21). Кожен вхідний вузол пов’язаний з двома ваговими коефіцієнтами, які пов’язують його з прихованим шаром. Ці ваги є вкладеними словами. Коли мережу буде навчено, ми витягнемо ці ваги, відкинувши всі інші. Щоб навчити мережу, ми будемо використовувати Keras і TensorFlow.

```
inp = Input(shape=(X.shape[1],))
x = Dense(units=2, activation='linear')(inp)
x = Dense(units=Y.shape[1], activation='softmax')(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss = 'categorical_crossentropy', optimizer =
'adam')
model.fit(
    x=X,
    y=Y,
    batch_size=256,
    epochs=1000)
```

У моделі існує один прихований шар. Вхідний шар відповідає розміру масиву X, тоді як прихований шар має розміри 2, як описано раніше, а вихідний рівень відповідає розміру масиву Y. Згодом модель компілюється за допомогою оптимізатора Адама, а функція втрат визначається за допомогою категоріальної крос-ентропії. Запуск моделі на нових прикладах дозволяє нам визначати вбудовування для невідомих текстів.

Наступним завданням є окреслення операцій моделі класифікації. Як описано в попередньому розділі, буде використано глибоку нейронну мережу, вдосконалену версію LSTM, яка спеціально включає LSTM із циклами зворотного зв’язку. Побудова аналізатора аргументів і синтаксичний аналіз аргументів відбувається наступним чином:

```
        parser.add_argument("--dataset", required=True, type=str,
help="set of M4 time series to be tested")
        parser.add_argument("--results_directory",        required=True,
type=str, help = "directory where M4 data will be downloaded")
        parser.add_argument("--gpu_id", required=False, type=int,
help="an integer that specify which GPU will be used")

help="an integer that specify which GPU will be used")
        parser.add_argument("--use_cpu", required=False, type=int,
help="1 to use CPU instead of GPU (uses GPU by default)")
        parser.add_argument("--num_obs", required=False, type=int,
help="number of M4 time series to be tested (uses all data by default)")
        parser.add_argument("--test", required=False, type=int,
help="run fast for tests (no test by default)")
        args = parser.parse_args()
```

Нам потрібно завантажити дані, зокрема набір даних речень, анотованих іменами людей та їхніми бажаними посадами. Цей набір даних було зібрано з інформації, доступної в Інтернеті:

```
    names = pd.read_csv(name_db)
    names_list = names["name"].apply(str)
    names_list_eng = names_list.tolist()
    names_male = pd.read_csv(name_db_rus_m, names=["name",
"some_numbers"])
    names_male_list = names_male["name"].apply(str)
    names_male_list = names_male_list.tolist()
    names_female = pd.read_csv(name_db_rus_f, names=["name",
"some_numbers"])
    names_female_list = names_female["name"].apply(str)
    names_female_list = names_female_list.tolist()
    names_list_rus = list(set(names_female_list + names_male_list))
    new_names_rus = prepare_names(names_list_rus)
    new_names_eng = prepare_names(names_list_eng)
languages = pd.read_csv(language_db)
    list_of_languages = [
        language.strip() for language in languages["Language name
"].tolist()
    ]
```

Далі ми розділяємо наші дані на тестові та навчальні набори.

```
    X_train_df, y_train_df, X_test_df, y_test_df = prepare_m4_data(
dataset_name = args.dataset, directory = args.results_directory, num_obs
= num_obs)
```

Розмір вхідного шару має збігатися з вихідним розміром із вкладень. Згодом додається прихований залишковий шар глибиною 8, випадковим чином виключаючи 20% нейронів, а потім включається останній щільний шар, щоб зменшити кількість функцій відповідно до кількості тегів.

```
rnn_depth=8
rnn_dropout=0.2
x = Input(shape=(X.shape))
for i in range(rnn_depth):
        return_sequences = i < rnn_depth - 1
        x_rnn    =    LSTM(rnn_width,    recurrent_dropout=rnn_dropout,
dropout=rnn_dropout, return_sequences=return_sequences)(x)
        if return_sequences:
            if i > 0 or input.shape[-1] == rnn_width:
                x = add([x, x_rnn])
            else:
                x = x_rnn
        else:
            def slice_last(x):
                return x[..., -1, :]
            x = add([Lambda(x[..., -1, :]), x_rnn])
rnn_depth.add(tf.keras.layers.Dense(32, activation='relu'))
```

Проміжні шари LSTM видають послідовності, тоді як останній шар повертає один елемент. Вхід також є послідовністю. Щоб вирівняти вхідні та вихідні форми LSTM для підсумовування, це стосується всіх шарів, крім останнього. Згодом модель компілюється та відображається її структура.

```
model = Model(inputs=input, optimizer='adam', outputs=output)
model.summary()
```

Після того, як модель буде складено, наступним кроком стане її навчання на наших навчальних даних і оцінка її точності за допомогою тестового набору.

```
model.fit(X_train_df, y_train_df, X_test_df, y_test_df)
# Predict on test set
print('\nForecasting')
y_hat_df = model.predict(X_test_df)
```

Щоб розпочати навчання мережі SVM, початковим кроком є завантаження набору речень. Деякі містять ключові фрази та мови, а інші – ні. З цієї колекції ми створюємо остаточний масив міток класів. Ми починаємо із завантаження набору речень із відповідними мітками класів.

```
df = pd.read_excel(labeled_ds1)
df2 = pd.read_excel(labeled_ds2)
frames = [df, df2]
result = pd.concat(frames)
result.reset_index(inplace=True)
result.drop(["index"], axis=1, inplace=True)
```

Попередня обробка вхідного тексту представлена нижче:

```
splited_texts = []
    for i in final_extracted_texts:
        text_procesed = i.split('\n')
        for i in range(len(text_procesed)):
            text_procesed[i] = " ".join(text_procesed[i].split())
            text_procesed[i] = re.sub(r'[^A-Za-z0-9. ,.:;#/"]\s+',
'', text_procesed[i])
        text_procesed = list(filter(None, text_procesed))
        splited_texts.append(text_procesed)
    for i in range(len(splited_texts)):
        splited_texts[i] = '. '.join(splited_texts[i])
```

Процес екстрактингу чанків починається згодом:

```
chunked = chunk_func(pos_tag(word_tokenize(text)))
        continuous_chunk = []
        current_chunk = []
        for subtree in chunked:
            if type(subtree) == Tree:
                current_chunk.append(" ".join([token for token, pos in
subtree.leaves()]))
            elif current_chunk:
                named_entity = " ".join(current_chunk)
                if named_entity not in continuous_chunk:
                    continuous_chunk.append(named_entity)
                    current_chunk = []
            else:
                continue
```

Одразу ж після цього відбувається ініціалізація модуля Tf-Idf, який переведе дані закодовані слова у векторне представлення у вигляді структури, і має назву embeddings:

```
tfidf = TfidfVectorizer(
min_df=2,
norm="l2",
encoding="latin-1",
ngram_range=(1, 2),
stop_words="english",
)
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
tfidf_transformer = TfidfTransformer()
X_train_tfidf =
tfidf_transformer.fit_transform(X_train_counts).toarray()
```

Початковий параметр керує словником термінів. Під час побудови цього переліку терміни з частотою появи документів значно нижчою за вказане порогове значення не враховуються. Цю величину також називають граничною. Якщо значення знаходиться в діапазоні [0,0, 1,0], параметр представляє частку документа.

Другий параметр відноситься до нормалізації. Кожен вихідний рядок підтримує одиничну норму: "l2" гарантує, що сума квадратів елементів вектора дорівнює 1. Подібність косинуса між двома векторами є їхнім добутком, коли застосовується норма l2.

Третій параметр керує кодуванням. При роботі з байтами або файлами для аналізу це кодування використовується для декодування.

ngram_range визначає мінімальну та максимальну кількість слів у колекції, яка найкраще характеризує певний клас. Stop_words представляють вилучення слів, які не мають контекстуального значення. Видаляючи часто вживані слова в даній мові, ми можемо зосередитися на основних словах. Згодом ми навчаємо цей Tf-Idf на наших даних.

Після попередньої обробки та видалення непотрібних слів, наступний крок включає в себе поділ цього на навчальні та тестові набори у співвідношенні 80%:20%.

```
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
```

Далі проініціалізуємо мережу SVM:

```
clf = LinearSVC(C=1.0, fit_intercept=True, loss='hinge',
max_iter=1000, penalty='l2', tol=0.0001, verbose=1).fit(X_train_tfidf,
y_train)
```

C – параметр регуляризації. Його значення має бути суто позитивним.
fit_intercept bool – визначає, чи повинна модель обчислювати перехоплення даних. Якщо встановлено значення false, перехоплення не використовуватиметься в обчисленнях (за умови, що дані вже відцентровані).
Втрата – визначає функцію втрат. "шарнір" є стандартною функцією втрати для SVM.
max_iter – максимальна кількість ітерацій для виконання.
Штраф – вказує, яку нормалізацію використовувати. "l2" є типовим для SVC.
Tol – Допуск критерію зупинки, тобто точність.
Verbose – установіть значення 1 для детального навчання.
Після ініціалізації мережі починається процес навчання. Він складається з певної кількості ітерацій. Кожна ітерація змінює значення функції прийняття рішень на основі векторних зображень слів.
Далі ми оцінюємо точність моделі за допомогою навчального набору даних, зберігаємо цю модель на диск і проводимо перевірку точності.

```
filename = 'clf_model.sav'
pickle.dump(clf, open(filename, 'wb'))
prediction = clf.predict(count_vect.transform([i for i in X_test]))
actual = np.array(y_test)
accuracy = (prediction == actual).mean()
```

Цю модель було навчено для класифікації «навички та ненавички». Виявлення мови також було виконано за допомогою класифікатора SVM, оскільки мова також є навиком. Щоб відрізнити мовні навички від інших, було

проведено порівняння між усіма визначеними навичками та глобальним набором даних мов.

```
final_lead_languages = []
    for phrase in filtered_sentence_list:
        if p_lang.search(phrase):
            matched_language = p_lang.search(phrase).group(0)
            final_lead_languages.append(matched_language[1:-1])
```

Ми не будемо використовувати окрему нейронну мережу для ідентифікації телефонних номерів і електронних адрес, оскільки більшість цих даних слідує певним шаблонам, які можна виявити за допомогою регулярних виразів. Для виявлення розділів, які потенційно містять необхідну інформацію, була використана та сама мережа LSTM із залишковими з'єднаннями.

Після навчання всіх моделей і опису функцій їх результати будуть консолідовані та збережені за категорією у файлі JSON.

3.4 Інструменти розробки чат-бота

Як відомо, раніше було описано алгоритми, які дозволяють парсити дані людей із резюме та продемонстровано уривки функцій з реалізаціями. Однак, це лише частина програмного продукту, адже крім парсингу, також наявний чат-бот та зручний інтерфейс для користувача. Згаданий бот реалізований за допомогою бібліотеки langchain та сукупності різних алгоритмів машинного навчання. Завантажуючи документ резюме кандидата або кількох кандидатів, даний чат-бот проаналізує кожен файл та допоможе обрати людину, яка найбільше підходить для певної вакансії за досвідом та навичками.

LangChain — це фреймворк, призначений для спрощення створення програм із використанням великих мовних моделей (LLM). LLM — це моделі штучного інтелекту, які можуть генерувати тексти природною мовою на основі деяких вхідних даних, таких як підказка, запитання або контекст. LLM мають

багато потенційних застосувань, таких як аналіз документів, чат-боти, генерація коду тощо. Однак ізольованого використання LLM часто недостатньо для створення справді потужної та корисної програми. Справжня сила приходить, коли ви можете поєднувати LLM з іншими джерелами обчислень або знань, такими як бази даних, API або інші моделі. Тут на допомогу приходить LangChain.

LangChain — це структура інтеграції мовної моделі, що означає, що вона допомагає вам підключати LLM до різних джерел даних і компонентів і оркеструвати їх для створення наскрізних додатків. LangChain пропонує набір інструментів, компонентів та інтерфейсів, які спрощують процес створення програм на базі LLM. Деякі з особливостей LangChain:

LangSmith: уніфікована платформа розробника для створення, тестування та моніторингу програм LLM. LangSmith надає графічний інтерфейс користувача (GUI), який дозволяє створювати проекти LangChain і керувати ними, а також інтерфейс командного рядка (CLI), який дає вам більше можливостей контролю та гнучкості. LangSmith також інтегрується з різними постачальниками LLM, такими як OpenAI, Hugging Face і Google, і дозволяє вибрати найкращий LLM для вашого випадку використання. LangSmith також допомагає вам розгортати та контролювати ваші програми LangChain, а також надає аналітику та відгуки, щоб допомогти вам покращити їх.

Варто зазначити, що компоненти LangChain - це набір повторно використовуваних і конфігурованих компонентів, які ви можете використовувати для створення своїх програм LangChain. Компоненти LangChain засновані на концепції ланцюжків, які є послідовностями LLM і джерел даних, які працюють разом для виконання конкретного завдання. LLM розшифровується як Large Language Model, яка є типом системи штучного інтелекту, та яка може генерувати тексти природною мовою на основі заданого введення або підказки. LangChain надає стандартний інтерфейс для виклику LLM, а також інтегрує їх з іншими інструментами та компонентами. LangChain також пропонує наскрізні рішення для звичайних програм LLM, таких як

підсумовування, переклад, генерація тексту тощо. Наприклад, ланцюг може бути системою відповідей на запитання, яка використовує LLM для створення запиту, базу даних для отримання відповідної інформації та інший LLM для створення відповіді. Компоненти LangChain розроблені для роботи з даними та агентами, що означає, що вони можуть адаптуватися до різних джерел даних і контекстів, а також можуть навчатися на основі їх взаємодії та зворотного зв'язку. Компоненти LangChain також можна комбінувати разом для створення складніших ланцюжків і програм.

LangChain — це фреймворк із відкритим кодом, який постійно розвивається та вдосконалюється завдяки внеску спільноти та розробників. LangChain має на меті дозволити розробникам створювати контекстно-залежні, аргументовані та персоналізовані програми.

Розглянемо застосування даної бібліотеки у моїй роботі та проаналізуємо їх.

У функції, зображеній нижче відбувається зберігання створеної векторної бази даних у проект

```
def save_faiss(db, full_folder_name):
    try:
        db.save_local(full_folder_name)
    except Exception as e:
        print('EXCEPTION ERROR:\n', e).
```

На прикладі, поданому нижче, завантажується векторна база даних, створена раніше. Також, якщо її не існує, то вона створюється. Відкриваємо резюме у форматі pdf за допомогою функції pdf_loader.load(), екстрактимо текст і додаємо текст у векторну базу даних у вигляді embeddings.

```
def load_text_from_cv(path_pdf):
    try:
        faiss_db = FAISS.load_local(
```

```
            'faiss_streamlit',
            EMBEDDINGS
            )
    except Exception as e:
        faiss_db = FAISS.from_texts('pass', EMBEDDINGS)
    pdf_loader = PyPDFLoader(path_pdf)
    documents = pdf_loader.load()
    faiss_db.add_documents(documents)
    faiss_db_one_resume = FAISS.from_documents(documents, EMBEDDINGS)
    return faiss_db, faiss_db_one_resume.
```

Наступна функція є надважливою для розробленого продукту, адже саме в ній відбувається логіка ініціалізації LangChain агента.

```
def create_agent_chain(faiss_db, prompt):
    llm = LLM
    chain = load_qa_chain(llm, chain_type="stuff")
    print('chain loaded')
    matching_docs = faiss_db.similarity_search(prompt
    print('similarity_search done')
    answer = chain.run(input_documents=matching_docs, question=prompt)
    return answer, matching_docs.
```

Розглянемо рядок функції:

```
    matching_docs = faiss_db.similarity_search(prompt).
```

Саме на цьому моменті до векторної бази даних потрапляє задане питання і використовуючи алгоритм пошуку близькості відбувається знаходяться найбільш схожі документи залежно від питання. Всі знайдені документи повертаємо в ініціалізованого бота. На даному етапі важливо розглянути існуючі алгоритми, для векторної схожості. Але спочатку сформуємо визначення даного поняття.

Векторна схожість – це спосіб вимірювання схожості двох фрагментів даних шляхом представлення їх у вигляді векторів, які є числовими масивами, які фіксують деякі особливості даних. Наприклад, слово можна представити як вектор чисел, які вказують, як часто воно зустрічається разом з іншими словами у великому корпусі тексту. Зображення можна представити у вигляді вектора чисел, які вказують на колір, форму та текстуру пікселів.

Існують різні алгоритми для обчислення подібності між двома векторами залежно від мети та типу даних. Деякі з поширених алгоритмів:

Евклідова відстань: це найбільш інтуїтивно зрозумілий спосіб вимірювання подібності, оскільки це просто відстань по прямій лінії між кінцями векторів. Чим менша відстань, тим більше схожі вектори. Цей алгоритм добре працює для числових або категоріальних даних, але він може бути чутливим до масштабу та розмірності векторів.

Косинус подібності: це косинус кута між векторами, який коливається від -1 до 1. Чим ближчий косинус до 1, тим більше схожі вектори. Цей алгоритм добре працює для текстових даних або даних зображення, оскільки він фіксує напрямок і орієнтацію векторів, а не їх величину. На нього також менше впливають масштаб і розмірність векторів.

Скалярний добуток: це сума добутків відповідних елементів векторів. Чим більший скалярний добуток, тим більше схожі вектори. Цей алгоритм добре працює для даних, які мають поняття популярності або частоти, оскільки вони пропорційні довжині векторів. Однак він також може спотворюватися викидами або екстремальними значеннями.

В даній роботі використовується алгоритм косинус подібності і тому є необхідність розповісти про плюси і мінуси його використання у програмному продукті.

Косинусна подібність є інваріантною щодо масштабу та розмірності векторів, що означає, що вона фіксує лише напрямок і орієнтацію даних, а не величину. Це може бути корисно для даних, які мають поняття популярності чи

частоти, наприклад тексту чи зображень, де довжина векторів не має такого значення, як відносна частота ознак.

Косинус-подібність також може обробляти розріджені дані, де багато функцій мають нульові значення, наприклад у текстових даних або даних зображення. Косинусна подібність не штрафує за нульові значення, оскільки враховує лише відмінні від нуля ознаки, які є спільними для векторів.

Однак, існують і мінуси. Подібність косинусів може ввести в оману, коли вектори лінійно залежні, що означає, що вони кратні один одному. Наприклад, якщо один вектор — [1, 1, 2, 4], а інший — [10, 10, 20, 40], вони мають однаковий напрямок і орієнтацію, але різні величини. Косинусна подібність дасть значення 1, що вказує на повну подібність, але це може не відображати справжньої різниці між векторами.

На косинусну подібність також можуть впливати викиди або екстремальні значення, які можуть спотворювати напрямок і орієнтацію векторів. Наприклад, якщо один вектор дорівнює [1, 1, 1, 1], а інший — [1, 1, 1, 100], вони мають дуже різні напрямки та орієнтації, але подібність косинусів дасть значення 0,97, що вказує на високу подібність, але це може не відображати справжньої відмінності між векторами.

Алгоритм «Евклідова відстань» заснований на теоремі Піфагора, яка стверджує, що квадрат гіпотенузи прямокутного трикутника дорівнює сумі квадратів двох інших сторін. Алгоритм евклідової відстані можна використовувати для різних застосувань, таких як пошук найближчих сусідів, кластеризація, класифікація та сегментація зображень.

Деякі переваги алгоритму евклідової відстані:

1. Це просто та інтуїтивно зрозуміло для розуміння та реалізації.
2. Він може обробляти будь-яку кількість вимірів і будь-який тип даних (числових, категоріальних тощо).
3. Його можна легко узагальнити для інших показників відстані, таких як Манхеттенська відстань, відстань Мінковського тощо.

Деякі недоліки алгоритму евклідової відстані:

1. На нього можуть впливати викиди та шуми в даних, які можуть спотворювати вимірювання відстані.
2. Це може бути неефективним і повільним для багатовимірних і розріджених даних, оскільки відстань стає менш дискримінантною та більш чутливою до невеликих варіацій.
3. Він може бути зміщений через масштаб і розподіл даних, що може вплинути на відносну важливість різних характеристик.

Алгоритм скалярного добутку векторної подібності має також деякі переваги та недоліки. Плюси:

1. Його легко обчислити та інтерпретувати, оскільки він вимагає лише множення та додавання компонент вектора.
2. Він інваріантний до довжини векторів, тобто залежить лише від їхнього напрямку, а не від величини.
3. Він може фіксувати лінійний зв’язок між двома векторами, наприклад, наскільки вони паралельні чи ортогональні.
4. Його можна використовувати для порівняння векторів у будь-якому вимірі, якщо вони мають однакову кількість компонентів.

Мінуси:

1. Він може вимірювати подібність векторів лише з точки зору їхнього кута, а не їх відстані чи орієнтації.
2. Він може бути чутливим до масштабування векторів, тобто множення вектора на константу може змінити його схожість з іншим вектором.
3. Він може не вловити нелінійний або складний зв’язок між двома векторами, наприклад, наскільки вони вигнуті чи скручені.
4. Його можна визначити лише для векторів у R3, що робить його обмеженим як міра подібності для інших типів об’єктів, таких як матриці чи тензори.

Врешті-решт, алгоритм косинус-подібності добре підійшов для моєї задачі. Адже стояла задачі порівняти дані, які мають поняття популярності або частоти, наприклад текст або зображення, де довжина векторів не має такого значення, як відносні частоти об’єктів.

Повернімось до аналізу реалізації продукту. Розглянемо наступну функцію:

```
def parse_cv(cv_name):
    path_pdf = os.path.join("cv_folder_streamlit", cv_name)
    faiss_db_storage = 'faiss_streamlit'
    faiss_db, faiss_db_one_resume = load_text_from_cv(path_pdf)
    f = {
        "link": path_pdf
    }
    resp = requests.post(url=url_parse, json=f)
    only_skills = resp.json()['skills']
    skills_string = ", ".join(only_skills)
    only_skills_string = f"Here are all person's skills from resume: {skills_string}"
    doc =  Document(page_content=only_skills_string,
                    metadata={
                        "source": f"{path_pdf}",
                        "page": -1
                        })
    faiss_db.add_documents([doc])
    save_faiss(faiss_db, faiss_db_storage)
    return faiss_db, only_skills, faiss_db_one_resume.
```

На цьому етапі відбувається підключення алгоритмів описаних раніше, а також підключення до API, в якому відбувається парсинг резюме. Після того, як надсилається резюме у форматі pdf, ми отримуємо відповідь парсингу і дані людини, зазначеної у резюме, а саме ім’я, прізвище, попреднє місце роботи, досвід та скіли. Для детальнішого аналізу навичок людини, викоремлюємо скіли, робимо з них стрічку і конвертуємо в формат, який дозволяє зберегти їх у векторній базі даних. Надалі зберігаємо їх туди за допомогою функції save_faiss().

Всі резюме, які ми надсилаємо під час спілкування з ботом, зберігаються в пам’яті. Таким чином, навіть після перезапуску бота усі завантажені резюме залишаються в директорії проекту.

```
        def save_uploadedfile(uploadedfile):
            folder = 'cv_folder_streamlit'
            if not os.path.exists(folder):
                os.mkdir(folder)
            with open(os.path.join(folder, uploadedfile.name),"wb") as f:
                f.write(uploadedfile.getbuffer())
return st.success("Saved File:{} to
destination_folder".format(uploadedfile.name)).
```

Таким чином, за допомогою бібліотеки LangChain було розроблено функціонал розумного чат-бота, який залежно від питання аналізує документи та підбирає найкращого кандидата з усіх завантажених резюме.

3.5. Розробка користувацького інтерфейсу

Для зручної та зрозумілої взаємодії користувача з чат-ботом, було вирішено використати фреймворк Streamlit. Streamlit — це фреймворк, який дозволяє створювати програми даних і ділитися ними на Python. Його розроблено, щоб полегшити та швидко створювати інтерактивні веб-додатки з функціями обробки даних і машинного навчання. Streamlit працює, запускаючи сценарій Python зверху вниз щоразу, коли щось змінюється на екрані, наприклад введення користувача або оновлення коду. Streamlit надає простий API для додавання віджетів, діаграм, таблиць, карт тощо до програми. Також можна безкоштовно розгорнути свою програму в хмарі спільноти Streamlit або використовувати компоненти Streamlit, щоб розширити функціональність своєї програми за допомогою власного HTML, CSS і JavaScript.

Архітектура Streamlit складається з чотирьох основних компонентів:

1 Інтерфейс командного рядка, який запускає сервер у Python. Тут ви запускаєте свою програму за допомогою команди streamlit run your_script.py.

2 Сервер, який обслуговує зв’язок між сценарієм Python і браузером. Він також керує станом програми, кешуванням і керуванням сеансами.

3 Генератор дельти, який перетворює команди Streamlit у вашому сценарії в ряд дельт (змін), які надсилаються в браузер.

4 Браузер, який рендерить додаток за допомогою React і TypeScript. Він також обробляє взаємодію користувача та надсилає події назад на сервер.

Тепер розглянемо безпосередньо приклад реалізації користувацького інтерфейсу з допомогою streamlit.

Оскільки Streamlit працює, запускаючи сценарій Python щоразу, при будь-яких змінах, необхідно ініціалізувати змінні, які будуть зберігатись і пам’яті веб-застосунку увесь часі повертати необхідні значення для відпрацювання логіки. На фрагменті коду, поданого нижче є приклад створення таких змінних.

```
if "messages" not in st.session_state:
    st.session_state.messages = []
if 'start_conv' not in st.session_state:
    st.session_state.start_conv = False
if 'new_cv' not in st.session_state:
    st.session_state.new_cv = True.
```

Наступний приклад коду визначає функцію під назвою agent_answer, яка приймає параметр main_agent_res, який є рядком, що містить відповідь основного агента.

```
def agent_answer(main_agent_res):
    with st.chat_message("assistant"):
        message_placeholder = st.empty()
        message_placeholder.markdown(main_agent_res)
    st.session_state.messages.append({"role": "assistant", "content": main_agent_res}).
```

Функція робить наступне:

- створює спливаючу підказку з повідомленням чату з міткою «assistant» за допомогою st.chat_message("assistant ").
- створює порожній наповнювач для вмісту повідомлення за допомогою st.empty().
- заповнює наповнювач основною відповіддю агента за допомогою message_placeholder.markdown(main_agent_res).
- додає головну відповідь агента до повідомлень змінної стану сеансу, яка є списком словників, що містять роль і вміст кожного повідомлення в чаті.

Цей приклад коду визначає функцію під назвою user_request, яка приймає підказку параметра, яка є рядком, що містить дані користувача.

```
def user_request(prompt):
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.markdown(prompt).
```

Функція робить наступне:

- додає дані користувача до змінної стану сеансу messages, яка є списком словників, що містять роль і вміст кожного повідомлення в чаті.
- створює спливаючу підказку з повідомленням чату з міткою « user» за допомогою st.chat_message(«user»).
- відображає введені користувачем дані як текст уцінки за допомогою st.markdown(prompt).

Перейдемо до основної імплементованої частини користувацького інтерфейсу.

```
uploaded_resume = False
if st.session_state.new_cv:
    path_cv = st.file_uploader("Upload resume", type=['pdf'])
    if path_cv is not None:
        save_uploadedfile(path_cv)
```

```
            uploaded_resume = True.
```

В даному фрагменті коду створюється логічна змінна uploaded_resume і встановлюється для неї значення False. Далі слідує перевірка змінної стану сеансу new_cv. Якщо вона має значення True, то це означає, що користувач хоче завантажити нове резюме. В такому випадку створюється віджет завантажувача файлів за допомогою st.file_uploader("Завантажити резюме", type=['pdf']) і призначається завантажений файл змінній path_cv.

```
    if uploaded_resume:
            st.write("Parsing resume and saving to vector storage...")
            st.session_state.faiss_db,                              only_skills,
st.session_state.faiss_db_one_resume = parse_cv(path_cv.name)
            st.write("Resume uploaded and parsed, now you can ask questions
based on this resume")
            uploaded_resume = False
            st.session_state.start_conv = True
            st.session_state.new_cv = False.
```

Код виконується, коли користувач завантажує файл резюме в додаток. Ось що відбувається в кожному рядку коду:

- if uploaded_resume: це умовний оператор, який перевіряє, чи змінна uploaded_resume має значення True, що означає, що користувач завантажив файл резюме.
- st.write("Аналіз резюме та збереження у векторному сховищі..."). Це проста команда, яка записує повідомлення в інтерфейс користувача програми, інформуючи користувача про те, що програма аналізує резюме та зберігає його у векторному сховищі. Синтаксичний аналіз означає вилучення з резюме відповідної інформації, такої як навички, освіта, досвід тощо.
- st.session_state.faiss_db, only_skills, st.session_state.faiss_db_one_resume = parse_cv(path_cv.name). Це рядок, який викликає спеціальну функцію під

назвою parse_cv, яка приймає назву файлу резюме як аргумент і повертає три значення: st. session_state.faiss_db, only_skills і st.session_state.faiss_db_one_resume. Перше значення — це база даних Faiss, яка є структурою даних, що забезпечує швидкий і ефективний пошук подібності між векторами. Друге значення – список навичок, витягнутий із резюме. Третє значення — ще одна база даних faiss, яка містить лише вектор завантаженого резюме. Об’єкт st.session_state — це спосіб зберігання даних під час різних запусків програми, щоб програма могла запам’ятовувати дії та налаштування користувача.

- st.write("Резюме завантажено та проаналізовано, тепер ви можете задавати запитання на основі цього резюме"). Це ще одна проста команда, яка записує повідомлення в інтерфейс користувача програми, повідомляючи користувачеві, що резюме завантажено та проаналізовано, і що тепер користувач може задавати питання на основі резюме. Наприклад, користувач може запитати про навички, освіту, досвід або досягнення власника резюме.
- uploaded_resume = False. Це рядок, який присвоює значення False змінній uploaded_resume, що означає, що файл резюме оброблено і програма готова до наступної дії.

Розглянемо цикл, який повторює список повідомлень, що зберігаються в атрибуті messages об’єкта st.session_state.

```
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"]).
```

Об’єкт st.session_state — це спосіб зберігання даних під час різних запусків програми, щоб програма могла запам’ятовувати дії та налаштування користувача. Список повідомлень містить словники, які представляють

повідомлення, якими обмінюються користувач і програма, з ключами «роль» і «вміст».

```
   if st.session_state.start_conv and prompt:
       if "Find the most suitable resume" in prompt or
st.session_state.general_target == "Find the most suitable resume":
           st.session_state.general_target = "Find the most suitable resume"
       else:
           st.session_state.general_target = "Answer questions based on current
CV"
       if prompt == 'CV':
           st.session_state.new_cv = True
       elif prompt and st.session_state.general_target == "Answer questions based
on current CV":
           user_request(prompt)
           main_agent_res, matching_docs =
create_agent_chain(st.session_state.faiss_db_one_resume, prompt)
           agent_answer(main_agent_res)
       elif prompt and st.session_state.general_target == "Find the most suitable
resume":
           user_request(prompt)
           main_agent_res, matching_docs =
create_agent_chain(st.session_state.faiss_db, prompt)
           output = f"\nLinks to CVs:\n\n"
           links = ["\n"].
```

if st.session_state.start_conv і prompt: це умовний оператор, який перевіряє, чи атрибут start_conv об’єкта st.session_state має значення True і чи змінна prompt не порожня. Атрибут start_conv вказує, чи готова програма почати розмову з користувачем на основі резюме. Змінна підказки зберігає повідомлення користувача. Якщо обидві умови виконуються, програма продовжить обробку повідомлення користувача та згенерує відповідь.

if «Знайти найбільш підходяще резюме» в підказці або st.session_state.general_target == «Знайти найбільш підходяще резюме»: це

вкладений умовний оператор, який перевіряє, чи містить повідомлення користувача фразу «Знайти найбільш підходяще резюме» або якщо атрибут general_target об’єкта st.session_state має значення «Знайти найбільш підходяще резюме». Атрибут general_target зберігає намір користувача, який може бути або «Відповісти на запитання на основі поточного CV», або «Знайти найбільш підходяще резюме». Якщо будь-яка умова виконується, програма встановить для атрибута general_target значення «Знайти найбільш підходяще резюме».

else: це альтернативна гілка вкладеного умовного оператора, яка виконується, якщо повідомлення користувача не містить фрази «Знайти найбільш підходяще резюме», і якщо атрибут general_target не має значення «Знайти найбільш підходяще резюме» . У цьому випадку програма встановить для атрибута general_target значення «Відповідати на запитання на основі поточного резюме».

if prompt == 'CV': це ще один вкладений умовний оператор, який перевіряє, чи повідомлення користувача дорівнює 'CV'. Якщо це правда, програма встановить для атрибута new_cv об’єкта st.session_state значення True. Атрибут new_cv вказує, чи потрібно програмі обробити новий файл резюме.

Підказка elif і st.session_state.general_target == «Відповісти на запитання на основі поточного CV»: це перша альтернативна гілка вкладеного умовного оператора, яка виконується, якщо повідомлення користувача не дорівнює «CV», і якщо general_target атрибут має значення «Відповісти на запитання на основі поточного CV». Якщо це правда, програма викличе спеціальну функцію під назвою user_request, яка приймає повідомлення користувача як аргумент і друкує його на консолі. Потім програма надрукує повідомлення на консолі, вказуючи, що вона відповідає лише на основі одного резюме. Далі програма викличе іншу спеціальну функцію під назвою create_agent_chain, яка приймає як аргументи базу даних Faiss, яка містить лише вектор завантаженого резюме, і повідомлення користувача, і повертає два значення: main_agent_res і matching_docs. Значення main_agent_res — це рядок, який містить відповідь програми на запитання користувача. Значення matching_docs — це список документів, які відповідають

запиту користувача. Нарешті, програма викличе іншу спеціальну функцію під назвою agent_answer, яка приймає значення main_agent_res як аргумент і записує його у віджет повідомлення чату з роллю «помічник».

Підказка elif і st.session_state.general_target == «Знайти найбільш підходяще резюме»: це друга альтернативна гілка вкладеного умовного оператора, яка виконується, якщо повідомлення користувача не дорівнює «CV» і якщо атрибут general_target має значення «Знайти найбільш підходяще резюме». Якщо це правда, програма викличе ті самі спеціальні функції, що й у попередній гілці, але з базою даних faiss, яка містить усі вектори резюме в векторному сховищі, замість тієї, яка містить лише вектор завантаженого резюме . Програма також додасть рядок, який містить посилання на резюме, які відповідають запиту користувача, до значення main_agent_res і запише його у віджет повідомлень чату з роллю «помічник»

3.6 Висновок до розділ 3

У данному розділі обґрунтовано вибір об’єктно-орієнтованої мови програмування високого рівня Python та середовища програмування Visual Studio Code для реалізації процесів створення та навчання глибокої нейронної мережі. Використання бібліотек Tensorflow та Keras дозволило ефективно реалізувати інформаційну технологію парсингу резюме, яка базується на глибокій нейронній мережі та SVM класифікаторі.

Додатково, для досягнення більшої гнучкості та інтерактивності у взаємодії з користувачем, було обґрунтовано використання LangChain та Streamlit. LangChain надає можливість здійснювати обробку природної мови, що важливо для аналізу та розуміння текстової інформації у резюме. З іншого боку, Streamlit дозволяє створювати легко налаштовувані та інтуїтивно зрозумілі інтерфейси для взаємодії з розробленою системою, спрощуючи процес вивчення результатів парсингу та подальших дій користувача.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ

4.1 Тестування програмного продукту

Для початку тестування, ми запускаємо термінал і переходимо до папки, що містить розроблений програмний продукт на основі технології глибокого навчання. Зображення запущеного середовища зображено на рисунку 4.1.

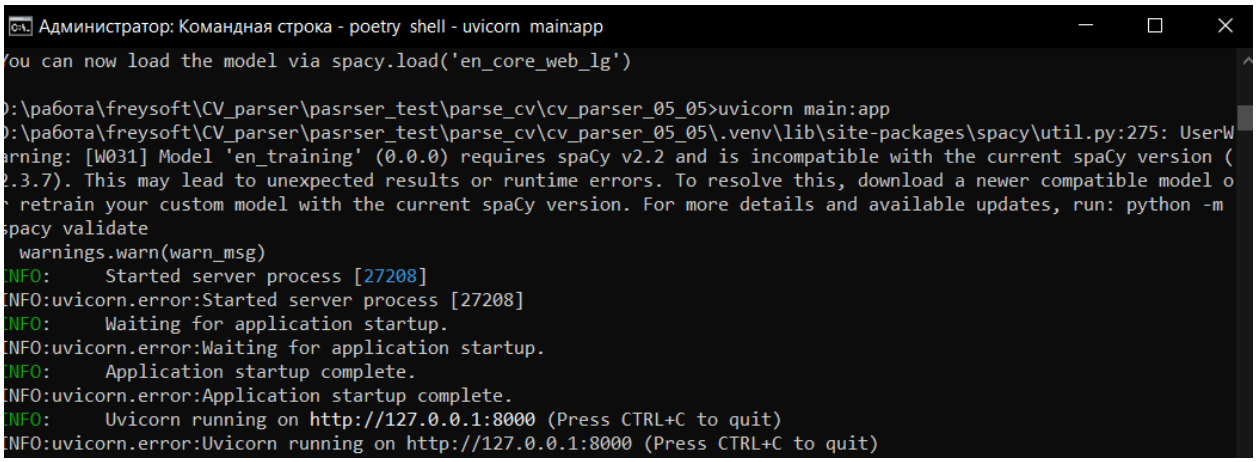


Рисунок 4.1 – Середовище програми

Щоб вибрати документ для аналізу, потрібно розмістити потрібне резюме в папці test_data, яка знаходиться в каталозі проекту. На рисунку 4.2 показано всі документи, наявні в цій папці, які за бажанням можна проаналізувати.

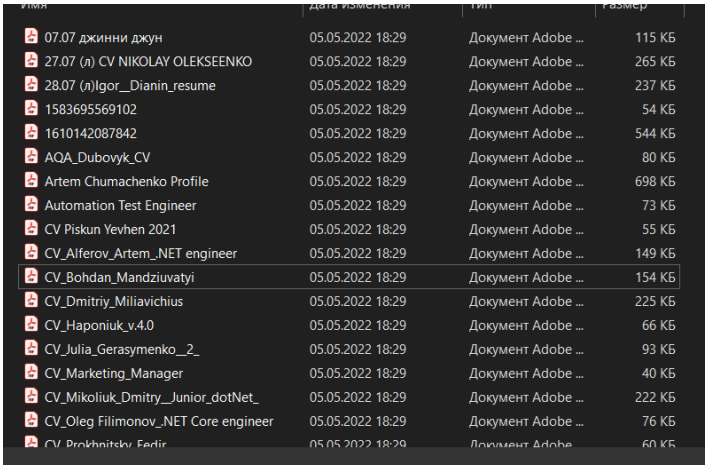
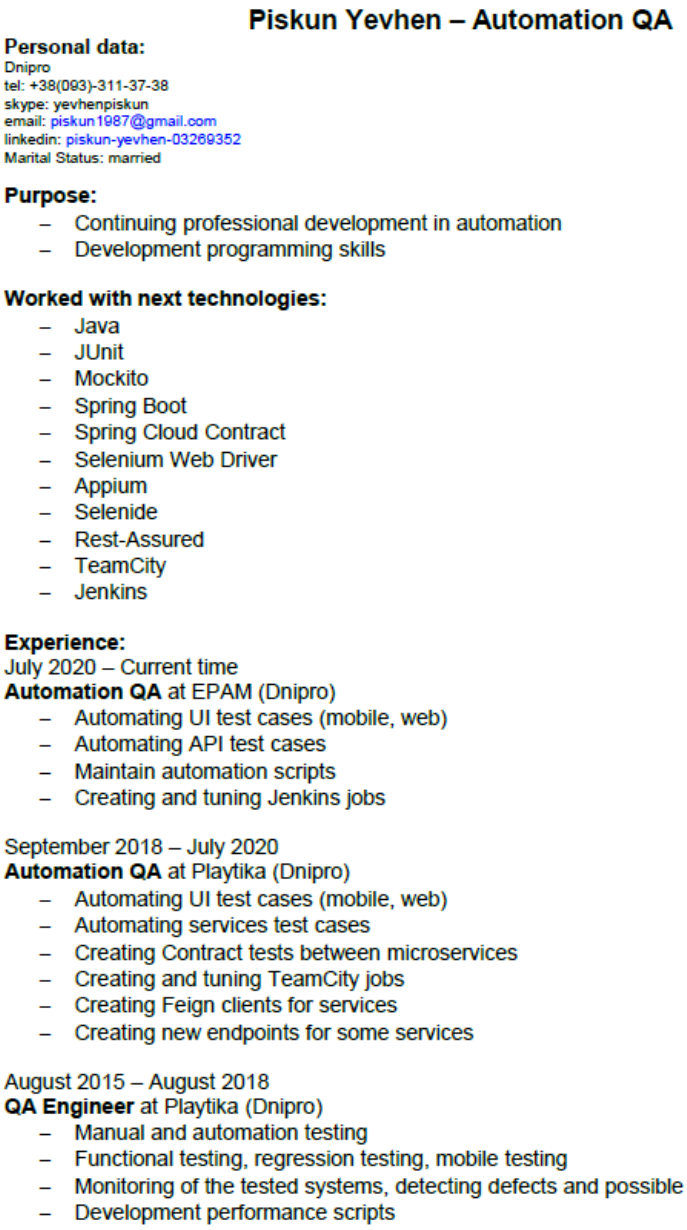


Рисунок 4.2 – Документи для аналізу

Для тестування ми виберемо резюме під назвою "CV Piskun Yevhen 2023.pdf", яке знаходиться в каталозі тестів. Частина цього резюме зображена на рисунку 4.3.



Piskun Yevhen – Automation QA

Personal data:
Dnipro
tel: +38(093)-311-37-38
skype: yevhenpiskun
email: piskun1987@gmail.com
linkedin: piskun-yevhen-03269352
Marital Status: married

Purpose:
- Continuing professional development in automation
- Development programming skills

Worked with next technologies:
- Java
- JUnit
- Mockito
- Spring Boot
- Spring Cloud Contract
- Selenium Web Driver
- Appium
- Selenide
- Rest-Assured
- TeamCity
- Jenkins

Experience:
July 2020 – Current time
Automation QA at EPAM (Dnipro)
- Automating UI test cases (mobile, web)
- Automating API test cases
- Maintain automation scripts
- Creating and tuning Jenkins jobs

September 2018 – July 2020
Automation QA at Playtika (Dnipro)
- Automating UI test cases (mobile, web)
- Automating services test cases
- Creating Contract tests between microservices
- Creating and tuning TeamCity jobs
- Creating Feign clients for services
- Creating new endpoints for some services

August 2015 – August 2018
QA Engineer at Playtika (Dnipro)
- Manual and automation testing
- Functional testing, regression testing, mobile testing
- Monitoring of the tested systems, detecting defects and possible causes;
- Development performance scripts

Рисунок 4.3 – Резюме "CV Piskun Yevhen 2023.pdf"

Для того, щоб протестувати парсинг резюме, необхідно запустити стрімліт, виконавши команду, зображену на рисунку 4.4.

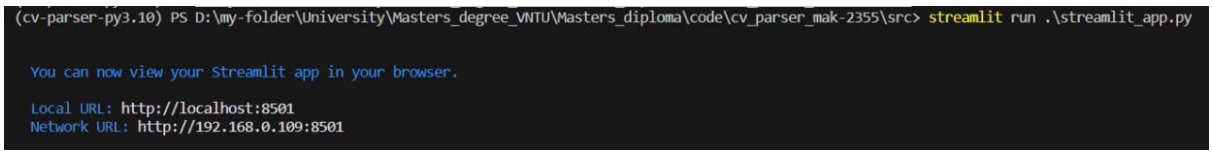


Рисунок 4.4 – Команда для запуску програмного продукту

Нам варто перейти за посиланням, яке відкрилось в терміналі або прописати в браузері шлях до локального сервера. Приклад шляху з назвою хоста та певним портом зображено на рисунку 4.5.

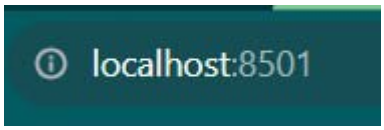


Рисунок 4.5 – Шлях до локального сервера

При переході на сторінку відкривається вікно, на якому зображено блок, який дозволяє завантажити бажаний документ із резюме. Ми можемо як перетягнути документ, так і завантажити за допомогою кнопки “Browse files”. Після завантаження резюме бот повідомляє користувача про успішне збереження файлу, парсинг резюме, збереження даних у векторну базу даних, а також про те, що відтепер є можливість розпочати із ним спілкування. Приклад успішного відпрацювання блоку можна розглянути на рисунку 4.6.

У користувача є альтернатива, яка полягає у тому, що він може або дізнатись детальніше про одне завантажене резюме, так і обрати найкраще із уже збережених в пам’ять чат-бота.

Розпочнемо із тестування першого випадку та попросимо бота розповісти детальніше про прикліплений документ. Розглянемо результат на рисунку 4.7.

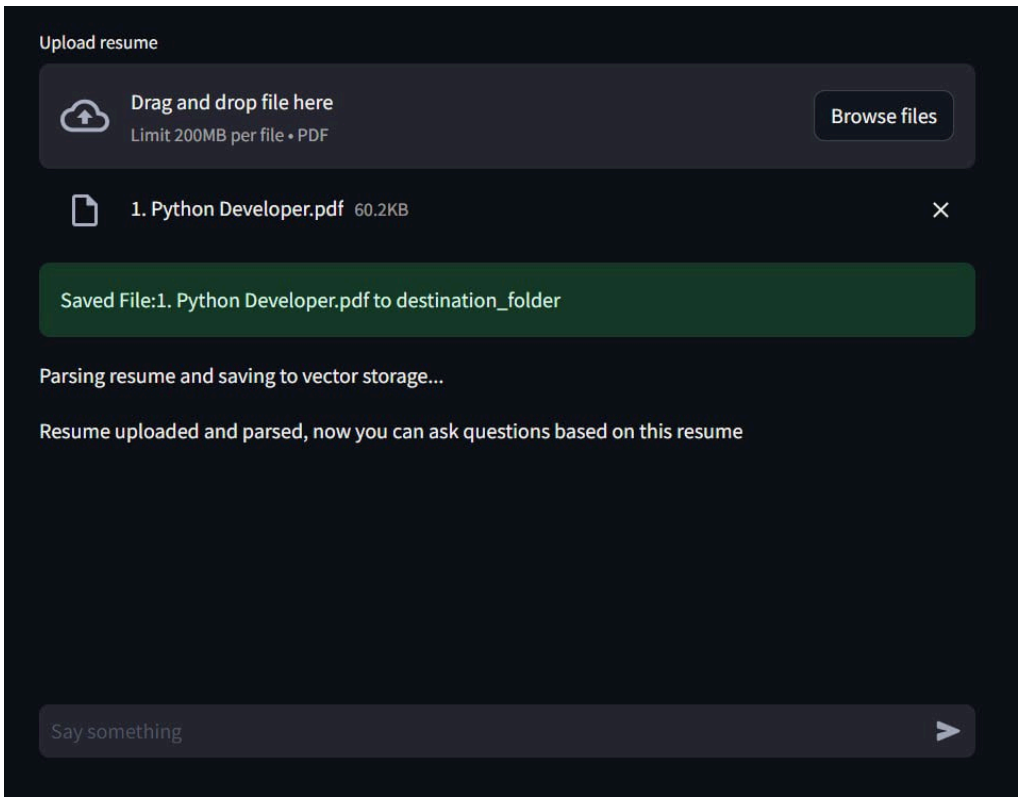


Рисунок 4.6 – Успішне завантаження резюме

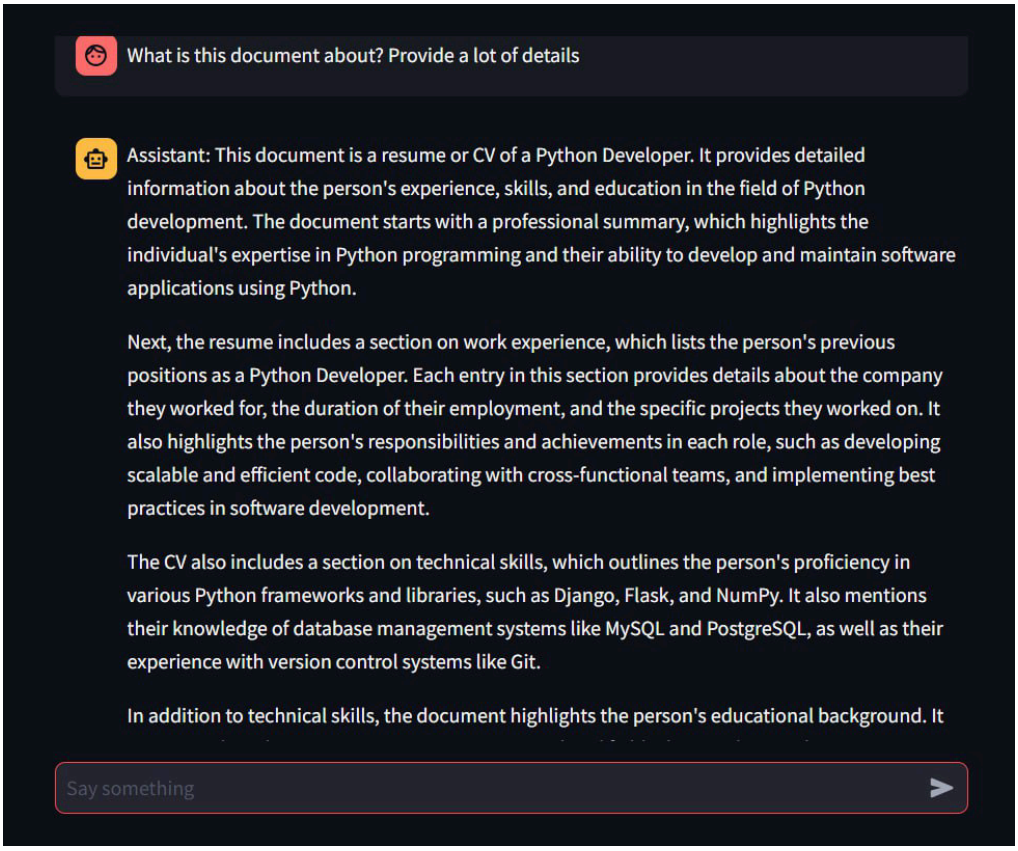


Рисунок 4.7 – Q&A-сесія стосовно резюме

Також, варто додати результати тестування парсингу. Розглянемо рисунок 4.8. Ми можемо побачити результати парсингу через алгоритми на основі нейронних мереж та SVM (ліва частина терміналу), а також результат того, як LangChain агент генерує відповідь та зберігає у пам'яті (права частина терміналу).

Рисунок 4.8 – Результат тестування парсингу резюме

Загалом, дані, які модель спарсила можна поділити на такі категорії:

- ім’я (name),
- e-mail (email),
- номер (phone),
- бажана позиція (designition),
- мови (languages),
- скіли (skills).

4.2 Аналіз результатів роботи програмного продукту

Сегмент інтелектуального аналізу резюме, зокрема LSTM із зворотним зв’язком, був навчений на 7000 анотованих текстах вручну. Для оцінки точності розробленої мережі LSTM зі зворотним зв'язком під час тестування було використано 500 текстів, що містять імена та бажані посади, і ще 500 текстів, які

не містять інформації. Результати тестування запропонованого програмного продукту детально описані в таблиці 4.1.

Таблиця.4.1 – Результати тестування LSTM із зворотніми зв'язками

	Результат класифікації	
	Ім'я та бажана позиція ПРИСУТНІ	Ім'я та бажана позиція ВІДСУТНІ
На вході ім'я та бажана позиція присутні (загальна кількість зображень з інформацією P=100)	Вірнопозитивний True positive – **TP=92**	Хибнонегативний False Negative – **FN=8**
На вході ім'я та бажана позиція відсутні (загальна кількість зображень без інформації N=100)	Хибнопозитивний False positive – **FP=7** помилка 1-го роду	Вірнонегативний True Negative– **TN=93**

Розрахуємо основні показники якості мережі LSTM із зворотніми зв'язками на основі даних табл.4.1.

Результативність:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \times 100\% = \frac{92 + 93}{200} \times 100\% = 92.5\%$$

Точність:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \times 100\% = \frac{92}{92 + 7} \times 100\% = 92.9\%$$

Повнота

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% = \frac{92}{92 + 8} \times 100\% = 92\%$$

Ми обчислили три ключові показники продуктивності для мережі LSTM із зворотним зв'язком: результативність 92,5%, точність 92,9% і повнота 92%. Далі ми протестуємо класифікатор SVM. Класифікатор SVM було навчено на 500 реченнях, анотованих вручну. Для оцінки точності розробленого класифікатора SVM під час тестування було використано 100 текстів, що містять підказки, і ще 100 текстів без підказок. Результати тестування запропонованого програмного продукту наведено в таблиці 4.2.

Таблиця.4.2 – Результати тестування LSTM із зворотніми зв'язками

	Результат класифікації	
	Скіли ПРИСУТНІ	Скіли ВІДСУТНІ
На вході скіли присутні (загальна кількість документів P=100)	Вірнопозитивний True positive – **TP=88**	Хибнонегативний False Negative – **FN=12**
На вході скіли відсутні (загальна кількість документів N=100)	Хибнопозитивний False positive – **FP=9** помилка 1-го роду	Вірнонегативний True Negative– **TN=91**

Розрахуємо основні показники якості мережі LSTM із зворотніми зв'язками на основі даних табл.4.2.

Результативність:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \times 100\% = \frac{88 + 91}{200} \times 100\% = 89.5\%$$

Точність:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \times 100\% = \frac{88}{88 + 9} \times 100\% = 90.7\%$$

Повнота:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% = \frac{88}{88 + 12} \times 100\% = 88\%$$

Ми розрахували три основні показники продуктивності для класифікатора SVM: результативність 89,5%, точність 90,7% і повнота 88%. Оскільки програмний продукт складався з двох мереж, кінцева продуктивність моделі була б середньою між цими двома моделями, зокрема: результативність 90%, точність 91% і відкликання 89%.

Щоб підтвердити досягнення мети дослідження, важливо порівняти ці метрики якості з аналогом (див. Таблицю 4.2). Таке порівняння наведено в таблиці 4.3.

Таблиця.4.3 – Результати тестування аналога (ALEX Resume Parser)

	Результат класифікації	
	Класова інформація ПРИСУТНЯ	Класова інформація ВІДСУТНЯ
На вході резюме (загальна кількість документів P=100)	Вірнопозитивний True positive – **TP=88**	Хибнонегативний False Negative – **FN=12**
На вході випадковий текст (загальна кількість документів N=100)	Хибнопозитивний False positive – **FP=9** помилка 1-го роду	Вірнонегативний True Negative– **TN=91**

З таблиці 4.3 видно, що розроблена програма показує 92,5% точності класифікації тексту для імені та бажаної посади, тоді як аналогічна програма

відображає 88,5%. Це вказує на 4% вищу точність класифікації тексту для імені та бажаної позиції в розробленій програмі. Крім того, таблиця 4.4 ілюструє вищу точність на 3,3% (92,9% порівняно з 89,6%) і на 5% більшу повноту (92% порівняно з 87%) у розробленій програмі порівняно з аналогічною. Таким чином, мета дослідження досягнута — підвищення результативності інтелектуального парсингу резюме за допомогою глибоких нейронних мереж на основі технології глибокого навчання та алгоритму класифікації Support Vector Machine (SVM) на 2.5%

Таблиця.4.4 – Порівняння показників якості запропонованого програмного продукту із показниками аналога (ALEX Resume Parser)

	Аналог (ALEX Resume Parser)	Запропонований програмний продукт
Результативність	87,5 %	90 %
Точність	89.6 %	92.9 %
Повнота	87 %	92 %

Продовжимо тестувати чат-бот. По стандарту, після того як ви закинули документ - поле для завантаження документу зникає. Якщо у поле для введення вказати слово "CV" - то чат бот дає можливість надіслати новий документ у "Upload resume", після чого можна продовжити спілкування, але вже стосовно іншого, нещодавно завантаженого документу. Приклад відповідей чат-бота, а також введення ключового слова CV у поле введення, можна розглянути на рисунку 4.9.

Варто згадати, що чат бот зберігає історію спілкування у пам'яті. Це дозволяє боту запам’ятовувати більше контексту. Але іноді є необхідність очистити історію спілкування, щоб у пам'яті бота не було зайвих повідомлень і не "забруднювався" процес мислення та визначення контексту спілкування. Якщо надіслати запит "Clear history", це дає сигнал боту, що необхідно очистити

пам'ять від усіх повідомлень, що зберігаються у пам'яті. Приклад такого запиту зображено на рисунку 4.10.

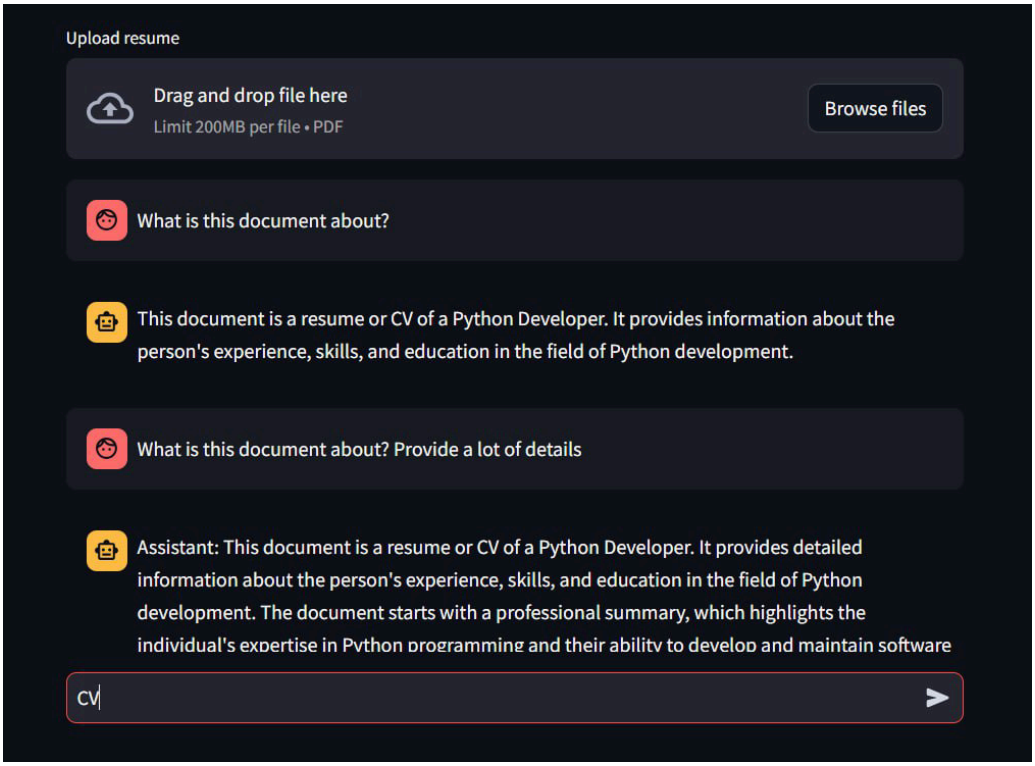


Рисунок 4.9 – Приклад роботи програми

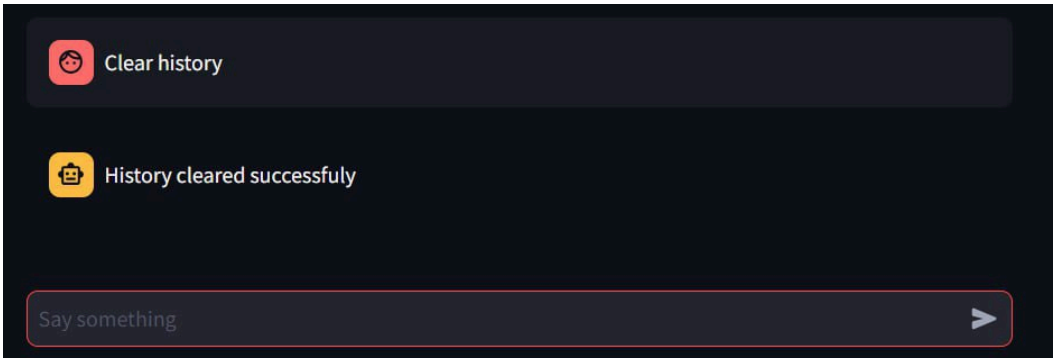


Рисунок 4.10 – Очистка історії бота

Якщо користувач вводить ключові слова "Find the most suitable resume", це дає сигнал чат боту, що потрібно не просто виконувати пошук в даному завантаженому документі, але й також потрібно шукати доцільну інформацію в усіх документах, що були завантажені у пам'ять чат-бота за увесь період його роботи. В такому випадку чат бот видає усю корисну інформацію по документам, а також наприкінці видає посилання та назви тих резюме, які являються

максимально доцільними залежно від запиту користувача. Усі ці повідомлення також зберігаються у пам'яті, але лише даної сесії спілкування. Приклад такого запиту зображено на рисунку 4.11.

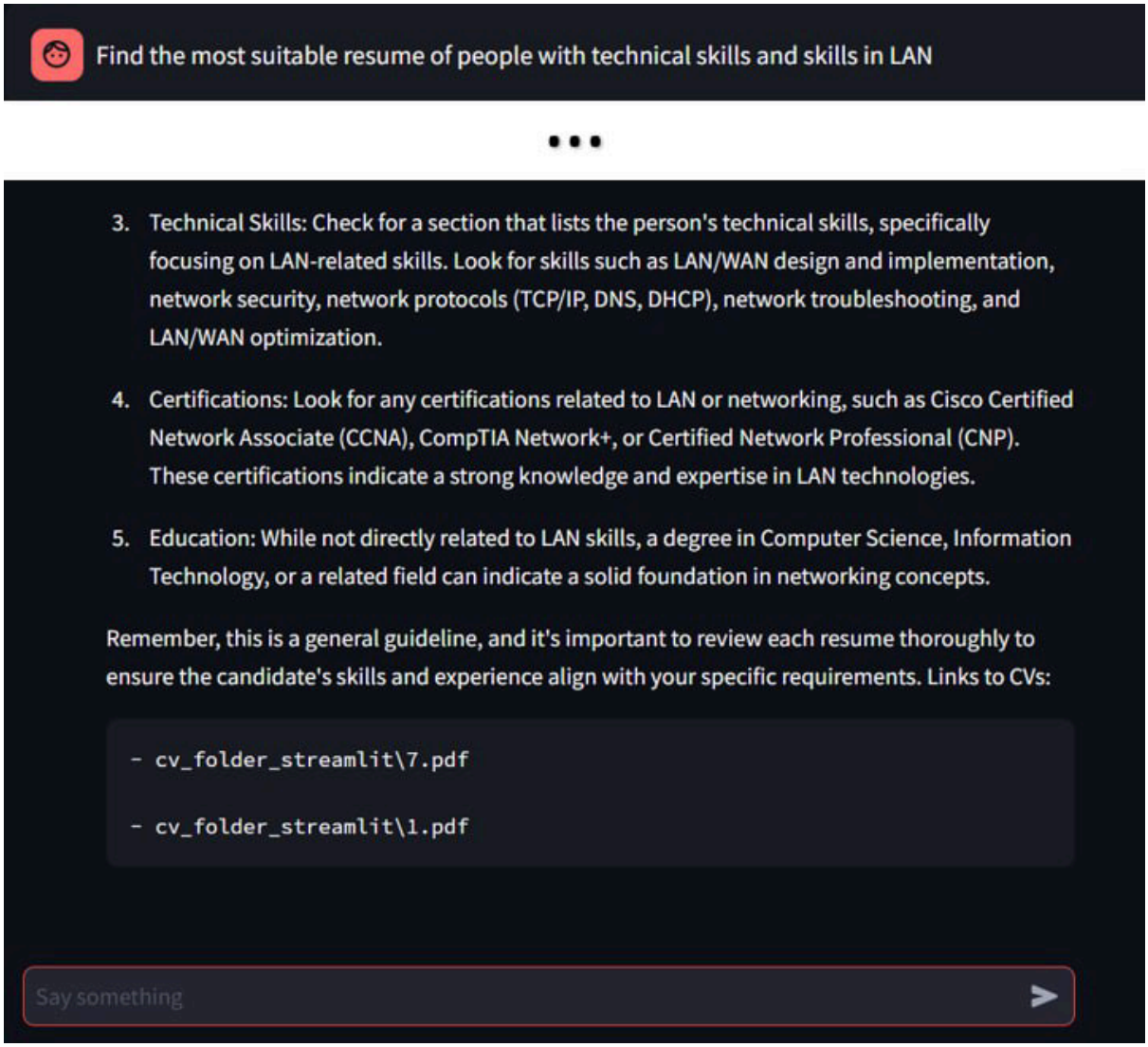


Рисунок 4.11 – Вибір найдоцільніших резюме залежно від вакансії

Інформаційна технологія розбору резюме на основі технології Deep Learning пройшов тестування. Аналіз результатів роботи програми показав результативність парсингу на 2.5% вище, ніж у аналога. Крім того, розроблена програма демонструє на 3,5% вищу точність (93% порівняно з 89,6%) і на 5% вищу повноту (92% порівняно з 87%), ніж аналог. Таким чином, мета досягнута — підвищення результативності парсингу резюме в рамках розробленого

програмного засобу на 2.5%. Крім цього, було проведено тестування функціоналу чат-бота. Було завантажено один документ, задано детальні питання стосовно резюме. Також, протестовано дуже корисний функціонал бота, а саме - вибір найбільш доцільного резюме із кількох прикріплених, залежно від вакансії. Продемонстровано збереження резюме в історію, відповіді бота та функцію очистки пам’яті бота. Розглянутий функціонал показує весь потенціал розробленого продукту, а також беззаперечну користь, яку він може принести ринку праці.

4.3 Висновок до розділу 4

Було проведення тестування інформаційної технології парсингу резюме. Аналіз результатів роботи розробленої програми показав, що вона має результативність парсингу резюме на 2.5% вищу за аналог (90% порівняно з 87.5%), на 3.3% вищу точність (92.9% порівняно з 89.6%) і на 3% вищу повноту (90% порівняно з 87%), ніж аналог. Тобто мета роботи досягнута – результативність парсингу резюме у розробленому програмному засобі підвищена на 2.5%.

5 ЕКОНОМІЧНА ЧАСТИН

5.1 Проведення комерційного та технологічного аудиту інформаційної технології парсингу резюме.

Метою проведення комерційного і технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності, тобто під час виконання магістерської кваліфікаційної роботи.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової кафедри.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу здійснюємо із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 5.1.

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу засобу поляриметричного аналізу оптично активних рідни

Критерії	Експерти		
	Арсенюк І.Р.	Озеранський В.С.	Сілагін О.В
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	2	3
Ринкові переваги (наявність аналогів)	4	3	3
Ринкові переваги (ціна продукту)	2	3	2
Ринкові переваги (технічні властивості)	3	3	3
Ринкові переваги (експлуатаційні витрати)	3	2	3
Ринкові перспективи (розмір ринку)	4	3	4
Ринкові перспективи (конкуренція)	3	3	3
Практична здійсненність (наявність фахівців)	3	4	3
Практична здійсненність (наявність фінансів)	3	3	3

Продовження табл. 5.1

Практична здійсненність (необхідність нових матеріалів)	4	4	3
Практична здійсненність (термін реалізації)	4	4	3
Практична здійсненність (розробка документів)	3	3	2
Сума балів	39	37	35
Середньоарифметична сума балів, СБ	37		

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал інформаційної технології парсингу резюме – середній.

5.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати на оплату праці. Належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Основна заробітна плата дослідників. Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р},$$

де к – кількість посад дослідників, залучених до процесу дослідження; $М_{ni}$ – місячний посадовий оклад конкретного розробника (інженера, дослідника,

науковця тощо), грн.; $Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні; $t_і$ – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 2.

Таблиця 5.2 – Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	80000	3636	180	654480
Розробник	40000	1818	140	254520
Всього:	909000			

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_і \cdot t_і,$$

де $С_і$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год; $t_і$ – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$С_і = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}},$$

де $М_м$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2023 році

$М_м$=6700 грн; $К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду; $К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати; $Т_р$ – середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні; $t_{зм}$ – тривалість зміни, год. Результати у таблиці 5.3.

Таблиця 5.3 – Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Створення web-сторінки додатку	180	4	48,3	1,27	8694
Створення ML моделей парсингу резюме	280	6	55,2	1,45	15456
Створення браузерного додатку	220	5	51,8	1,36	11396
Створення дизайну додатку	160	4	48,3	1,27	7728
Менеджмент проєкту	290	5	51,8	1,36	15022
Спілкування з інвесторами	100	5	51,8	1,36	5180
Всього					63404

Додаткова заробітна плата. Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (909000 + 63404) = 97240{,}4 \text{ грн.}$$

Відрахування на соціальні заходи. Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) =$$
$$= 0{,}22 \cdot (909000 + 63404 + 97240{,}4) = 235322 \text{ грн.}$$

де $З_о$ – основна заробітна плата розробників, грн.; $З_р$ – основна заробітна плата робітників, грн.; $З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

Програмне забезпечення. До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{1}^{к} Ц_{іпрг} \cdot С_{прг.і} \cdot К_і,$$

де $Ц_{іпрг}$ – ціна придбання програмного забезпечення і-го виду, грн.; $С_{прг.і}$ – кількість одиниць програмного забезпечення відповідного виду, шт.; $К_і$ –

коефіцієнт, що враховує інсталяцію, налагодження програмного забезпечення, $К_і$ = (1,1…1,12); к – кількість видів програмного забезпечення. Витрати продемонстровані у таблиці 5.4.

Таблиця 5.4 – Витрати на придбання програмного забезпечення

Найменування програмного забезпечення	Ціна за одиницю, грн.	Витрачено	Вартість програмного забезпечення, грн.
Сервер “AWS EC2 with 4 T4 GPU”	4810	9	43290
Хмарне сховище “AWS Bucket”	185	9	1665
Google Colab	407	6	2442
Всього, з врахуванням коефіцієнта інсталяції та налагодження			52137

Амортизація обладнання. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц_б}{Т_в} \cdot \frac{t}{12},$$

де $Ц_б$ – загальна балансова вартість всього обладнання, комп’ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.; t – термін використання основного фонду, місяці; $Т_в$ – термін корисного використання основного фонду, роки. Розрахунки представлені у таблиці 5.5.

Таблиця 5.5 – Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Ноутбук Macbook Pro 2019	35000	5	3	1750
WI-FI Роутер	1500	3	3	125
Всього	1875			

Витрати на електроенергію для науково-виробничих цілей. Витрати на силову електроенергію $В_е$, що представлені у таблиці 5.6, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

Таблиця 5.6 – Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Ноутбук Macbook Pro 2019	0,38	1350
WI-FI Роутер	0,01	1400

$$Ве = \sum \frac{W_i \cdot t_i \cdot Ц_е \cdot К_{впі}}{ККД} = \frac{0{,}38 \cdot 1350 \cdot 7{,}5 \cdot 0{,}95}{0{,}98} + \frac{0{,}01 \cdot 1400 \cdot 7{,}5 \cdot 0{,}95}{0{,}98} = 3832 \text{ грн.},$$

W_i – встановлена потужність обладнання, кВт; t_i – тривалість роботи обладнання на етапі дослідження, год.; Це – вартість 1 кВт електроенергії, грн.; $К_{впі}$ – коефіцієнт використання потужності; ККД – коефіцієнт корисної дії обладнання.

Інші витрати. До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Ів} = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100\%} = (909000 + 63404) \cdot \frac{50}{100} = 486202 \text{ грн.},$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

Накладні (загальновиробничі) витрати. До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%} = (909000 + 63404) \cdot \frac{110}{100} = 1069644 \text{ грн.},$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати на проведення науково-дослідної роботи. Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + В_{прг} + А_{обл} + В_е +$$
$$+І_в + В_{нзв} = 909000 + 63404 + 97240{,}4 + 235322 + 52137 + 1875 + 3832$$
$$+ 486202 + 1069644 = 2918656{,}4 \text{ грн.}$$

Загальні витрати. Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{2918656{,}4}{0{,}7} = 4169509 \text{ грн.},$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи. Оскільки, якщо науково-технічна розробка знаходиться на стадії розробки промислового зразка, то η=0,7.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу; N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість послуги у році до впровадження інформаційної системи; $\pm\Delta Ц_о$ – зміна вартості

послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm \Delta Ц_о \cdot N + Ц_о \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right),$$

де ±ΔЦ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки); $\pm\Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни); N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки; $Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році; $Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів; ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки); λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333; ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ =0,2...0,5; ϑ – ставка

податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році ϑ =18%.

Очікуваний термін життєвого циклу розробки 1 рік, тому:

$$\Delta П = ((250000 - 100000) \cdot 20000 - (20000 - 10000) \cdot 100000) \cdot 0{,}8333 \cdot 0{,}3 \cdot \left(1 - \frac{18}{100}\right) = 42240000 \text{ грн.}$$

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{42240000}{(1+0{,}1)^1} = 38400000 \text{ грн.},$$

де ΔП – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.; Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=1 рік); τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15; t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} \cdot ЗВ = 5 \cdot 4169509 = 20847545 \text{ грн.}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи

тощо; зазвичай кінв=2...5, але може бути і більшим; ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП - PV = 38400000 - 20847545 = 17552455 \text{ грн.},$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн.; PV – теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс}>0$, то можемо припустити про потенційну зацікавленість інвесторів у розробці.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Еабс}{PV}} = \sqrt[1]{1 + \frac{17552455}{20847545}} = 1{,}34,$$

де $Т_ж$ – життєвий цикл розробки, роки.

Визначимо бар'єрну ставку дисконтування $\tau_{мін}$, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f = 0{,}9 + 0{,}05 = 0{,}95,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,9...0,12; f – показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина f =0,05...0,5, але може бути і значно вищою.

Оскільки $E_в=1{,}34>\tau_{мін}=0{,}95$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки та виведенні її на ринок, тобто в її комерціалізації.

Далі розраховуємо період окупності інвестицій $T_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_о = \frac{1}{E_в} = \frac{1}{1{,}34} = 0{,}75 \text{ року.}$$

Оскільки $T_о=0{,}75<1…3$-х років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

5.4 Висновок до розділу 5

Економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 4169509 гривень. Було

прогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є конкурентоспроможним. Період окупності складе близько 0,75 роки.

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології парсингу резюме за допомогою SVM класифікатора та нейронних мереж, а також було запропоновано новий підхід до процесу надання спаршеної інформації користувачам.

Всі завдання поставлені перед магістерською кваліфікаційною роботою виконані в повному об’ємі, а саме:

1. Проаналізовано існуючі рішення та відомі аналоги парсингу резюме і за основу обрано метод нейронних мереж з використанням LSTM та використанням LLM GPT для генерації відповіді користувачу.
2. Розроблено структуру процесів, побудовано та описано діаграму структури процесів.
3. Обґрунтовано вибір методу розв’язання задачі, який полягає у використанні LSTM, SVM об’єднанні результатів та переданні в LLM GPT.
4. Розроблено алгоритм роботи нформаційної технології парсингу резюме, що складається з 4 основних етапів: навчання LSTM та SVM, поєднання результатів роботи даних алгоритмів, та виведення результатів обробки через LLM GPT користувачу.
5. Розроблено програмне забезпечення інформаційної технології парсингу резюме на мові Python з використанням відповідних бібліотек для машинного навчання для зручного використання користувачами.
6. Проведено тестування розробленої програми парсингу резюме, виявлено, що створений програмний продукт має результативність парсингу вищу за аналог на 2.5%, а також на 3.3% вищу точність (92.9% порівняно з 89.6%) і на 3% вищу повноту (90% порівняно з 87%), ніж аналог.

Інформаційна технологія реалізована з використанням SVM класифікатора, нейронної мережі LSTM та великих мовних моделей, що забезпечило досягнення визначеної мети дослідження. Реалізована модель

дозволяє автоматизувати процес розбору резюме кандидата у зручний для користувача спосіб, зберегти результати та порівняти резюме кандидатів під час спілкування з чат-ботом.

Розроблену інформаційна технологія можливо покращити шляхом реалізації додаткового модуля, який зможе забезпечити опрацювання вихідних даних, для розподілення задач до інших під-модулів системи. Мета магістерської кваліфікаційної роботи досягнута – результативність парсингу резюме підвищена на 2.5%, додано використання великих мовних моделей, даний метод парсингу резюме показує конкурентні результати серед аналогів. За рахунок використання великих мовних моделей останнього покоління вдалось досягти ще більшої результативності та якості відповіді моделі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Олійник Н. Інформаційна технологія парсингу резюме. LII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2023). URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/18950.
2. Da-Wen Sun. Computer vision technology for food quality evaluation: підручник. London: Academic Press, 2016. 110 p.
3. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Computer_vision.
4. Susan L. Named Entity Recognition with NLTK and SpaCy. *Towardsdatascience*. URL: https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da.
5. Text classification: what it is and why it matters. *MonkeyLearn*. URL: https://monkeylearn.com/text-classification/.
6. Toner B., Melton J. Affinda resume parsing library – python, javascript, C#, java. *Affinda*. URL: https://api.affinda.com/docs#auth.
7. HireAbility ALEX Resume Parser offers outstanding accuracy. *HireAbility Resume Parsing and Job Parsing Solutions*. URL: https://www.hireability.com/products/alex-cv-resume-parser/.
8. Нельсон Д. Що таке RNN і LSTM у Deep Learning?. unite.ai. URL: https://www.unite.ai/uk/what-are-rnns-and-lstms-in-deep-learning/.
9. Бойко Н. Побудова моделей для прогнозування часових рядів застосовуючи мережі довгострокової пам’яті. Вид-во “Говерла”. 2022. Т. 40, № 1. С. 113.
10. Trained Models & Pipelines · spaCy Models Documentation. *Trained Models & Pipelines*. URL: https://spacy.io/models.
11. Jaeyoung Kim, Jungwon Lee, Mostafa El-Khamy. Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition. *Arxiv* : конспект лекцій. 2017. С. 5. URL: https://arxiv.org/pdf/1701.03360.pdf.

12. Rohith Gandhi. Support Vector Machine – Introduction to Machine Learning Algorithms. *Towards Data Science*.
13. Як працює Deep Learning. foxminded.ua. URL: https://foxminded.ua/deep-learning/ (дата звернення: 15.10.2023).
14. Dasaradh S. K. A Gentle Introduction To Math Behind Neural Networks. *Towards Data Science*. URL: https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.
15. Nielsen M. Neural networks and deep learning. *Neural networks and deep learning*. URL: http://neuralnetworksanddeeplearning.com/chap2.html.
16. Simeon Kostadinov. Understanding Backpropagation Algorithm. *Towards Data Science*. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.
17. Філонов І. Классификация данных методом опорных векторов [Електронний ресурс] / Ігорь Філонов // Habr. – 2010. – Режим доступу до ресурсу: https://habr.com/ru/post/105220/.
18. Мізюченко О. Путівник мовою програмування Python 3.8 [Електронний ресурс] / Олександр Мізюк // Путівник мовою програмування Python. – 2020. – Режим доступу до ресурсу: https://pythonguide.rozh2sch.org.ua/#_короткий_опис.
19. What is a Large Language Model (LLM). geeksforgeeks. URL: https://www.geeksforgeeks.org/large-language-model-llm/ (дата звернення: 10.10.2023).
20. Wangui W. Understanding Large Language Models and How to Use them with LangChain. Medium. URL: https://medium.com/sciforce/what-is-gpt-3-how-does-it-work-and-what-does-it-actually-do-9f721d69e5c1 (дата звернення: 10.10.2023).
21. Improving Language Understanding by Generative Pre-Training / Alec Radford та ін. OpenAI. URL: https://openai.com/

Додаток А (обов'язковий)
Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія парсингу резюме

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 83,0% Схожість 17,0%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Олійник Н.Ю.

Керівник роботи Колесницький О.К.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
import os
import re
from typing import List, Tuple, Callable, Dict, Union

import pandas as pd
import numpy as np

from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keybert import KeyBERT
from pdfminer.high_level import extract_text

from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

from nltk import word_tokenize, pos_tag, ne_chunk
from nltk import RegexpParser
from nltk import Tree

from logger import logger
import time
import pickle

def get_skill_df(labeled_ds1, labeled_ds2) -> pd.DataFrame:
    """Return dataframe with labeled skills."""
    # Считываю 2 кастомных пролейбленных датасета
    # (лейблил по разным принципам, поэтому они и отдельно)
    df = pd.read_excel(labeled_ds1)
    df2 = pd.read_excel(labeled_ds2)

    df.drop(["Unnamed: 0"], axis=1, inplace=True)
    df.drop(df.index[[range(835, 849)]], axis=0, inplace=True)
    df.reset_index(inplace=True)
    df.drop(["index"], axis=1, inplace=True)
    df[0] = df[0].astype(str)
```

```
    df2.drop(["Unnamed: 0"], axis=1, inplace=True)
    df2.reset_index(inplace=True)
    df2.drop(["index"], axis=1, inplace=True)
    df2[0] = df2[0].astype(str)

    labels = df["labels"].to_numpy()
    for i in range(len(labels)):
        if labels[i] == "not_skill":
            labels[i] = 0
        else:
            labels[i] = 1

    frames = [df, df2]
    result = pd.concat(frames)
    result.reset_index(inplace=True)
    result.drop(["index"], axis=1, inplace=True)
    return result
def preprocess_input(dfm: pd.DataFrame) -> np.ndarray:
    docs = dfm[0].tolist()

    vocab_size = 500  # максимум слов всего
    encoded_docs = [one_hot(d, vocab_size) for d in docs]

    max_length = 37  # максимум слов в предложении
    pad_docs = pad_sequences(encoded_docs, maxlen=max_length, padding="post")

    return pad_docs


def train_model(result: pd.DataFrame) -> Tuple[object, CountVectorizer]:
    X = result[0]
    y = result["labels"].astype("int")
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=202)

    tfidf = TfidfVectorizer(
            sublinear_tf=True,
            min_df=2,
            norm="l2",
            encoding="latin-1",
            ngram_range=(1, 2),
```

```
                stop_words="english",
            )

        count_vect = CountVectorizer()
        X_train_counts = count_vect.fit_transform(X_train)
        tfidf_transformer = TfidfTransformer()
        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts).toarray()

        clf = LinearSVC().fit(X_train_tfidf, y_train)

        filename = 'clf_model.sav'
        pickle.dump(clf, open(filename, 'wb'))

        prediction = clf.predict(count_vect.transform([i for i in X_test]))
        actual = np.array(y_test)
        accuracy = (prediction == actual).mean()

        return clf, count_vect
    def first_setup(labeled_ds1, labeled_ds2):
        kw_model = KeyBERT()
        result = get_skill_df(labeled_ds1, labeled_ds2)

        labels = np.array(result["labels"].tolist())
        labels = np.asarray(labels).astype("int")
        padded_docs = preprocess_input(result)

        data = pd.DataFrame(padded_docs)

        clf, count_vect = train_model(result)

        return clf, count_vect, kw_model

    start = time.time()

    labeled_ds1 = os.path.join(
            os.path.dirname(os.path.realpath('__file__')), "..", "test_data",
"custom_algorithm", "test.xlsx"
        )
    labeled_ds2 = os.path.join(
            os.path.dirname(os.path.realpath('__file__')), "..", "test_data",
"custom_algorithm", "test_2.xlsx"
```

```
        )
    clf, count_vect, kw_model = first_setup(labeled_ds1, labeled_ds2)

    print('final first setup:', time.time() - start)

    def extract_keywords_from_cv(predictions: np.ndarray, dfm: pd.DataFrame, kw_model) -
> Tuple[Dict, Dict, Dict, Dict]:
            res_dict = {}
            phrase_dict = {}

            for i in range(len(predictions)):
                if predictions[i] == 1:
                    phrase_dict[i] = dfm[0][i]
                    res_dict[i] = kw_model.extract_keywords(
                        dfm[0][i], keyphrase_ngram_range=(1, 2), stop_words=None
                    )

            skills_dict_final_loc = {}
            for key, value in phrase_dict.items():
                if not all(c.isnumeric() or c.isspace() for c in phrase_dict[key]):
                    skills_dict_final_loc[key] = value

            res_dict_final = {}
            for key, value in res_dict.items():
                for score in res_dict[key]:
                    if len(res_dict[key]) == 1 and score[1] >= 0.7:
                        res_dict_final[key] = score[0]
                        continue
                    if score[1] >= 0.7:
                        res_dict_final[key] = score[0]
                        break

            return phrase_dict, res_dict_final, skills_dict_final_loc, res_dict
    def split_and_preprocess(final_extracted_texts: str) -> List[str]:
        splited_texts = []

        for i in final_extracted_texts:
            text_procesed = i.split("\n")
            for i in range(len(text_procesed)):
                text_procesed[i] = " ".join(text_procesed[i].split())
                text_procesed[i] = re.sub(
```

```
                    r'[^A-Za-z0-9. ,.:;#/"]\s+', "", text_procesed[i]
                )
            text_procesed = list(filter(None, text_procesed))
            splited_texts.append(text_procesed)

        for i in range(len(splited_texts)):
            splited_texts[i] = ". ".join(splited_texts[i])

        return splited_texts

    def test_model(cv_dir, cv_name: str, clf: object, count_vect: CountVectorizer) ->
Tuple[pd.DataFrame, np.ndarray]:
        text = extract_text_from_CV(os.path.join(cv_dir, cv_name))
        splitted_and_preprocessed = split_and_preprocess(text)
        chanks_extracted = extract_chanks(splitted_and_preprocessed)
        df_test = pd.DataFrame(chanks_extracted)
        # Проверка работы модели на нашем кастомном резюме
        prediction = clf.predict(count_vect.transform([i for i in df_test[0]]))
        return df_test, prediction


    def get_continuous_chunks(text: pd.Series, chunk_func: Callable = ne_chunk) ->
List[str]:
        chunked = chunk_func(pos_tag(word_tokenize(text)))
        continuous_chunk = []
        current_chunk = []

        for subtree in chunked:
            if type(subtree) == Tree:
                current_chunk.append(
                    " ".join([token for token, pos in subtree.leaves()])
                )
            elif current_chunk:
                named_entity = " ".join(current_chunk)
                if named_entity not in continuous_chunk:
                    continuous_chunk.append(named_entity)
                    current_chunk = []
            else:
                continue

        return continuous_chunk
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПАРСИНГУ РЕЗЮМЕ

Виконав: студент 2-го курсу,
групи 1КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Олійник Н. Ю.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф. КН
_____ Колесницький О.К.
(прізвище та ініціали)
«07» 12 2023 р.

Вінниця ВНТУ – 2023 рік

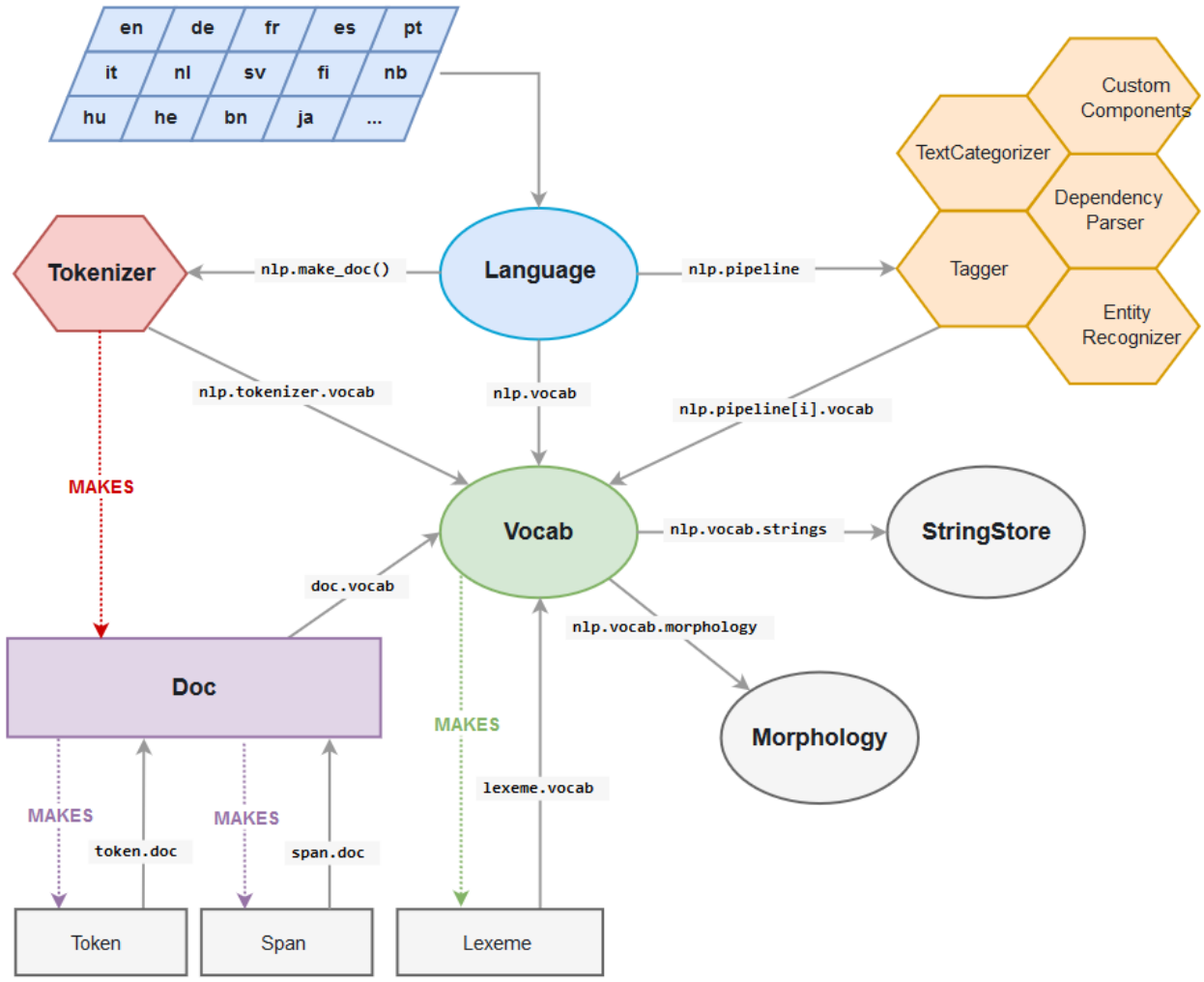


Рисунок В.1 – Архітектура мережі SpaCy.

Рисунок В.2– Процеси обробки інформації при інтелектуальному парсингу резюме

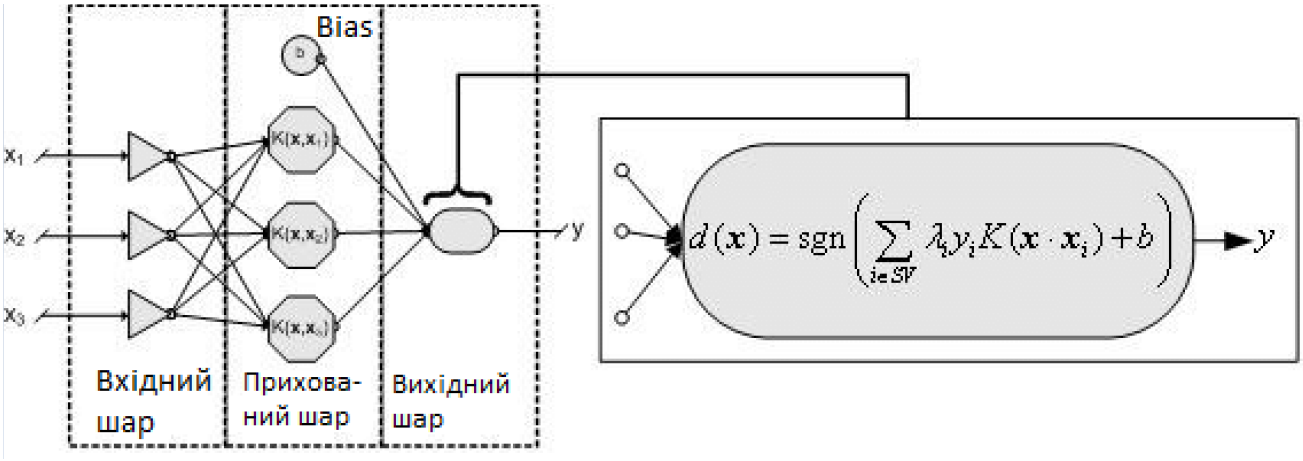


Рисунок В.3 – Архітектура SVM класифікатора.

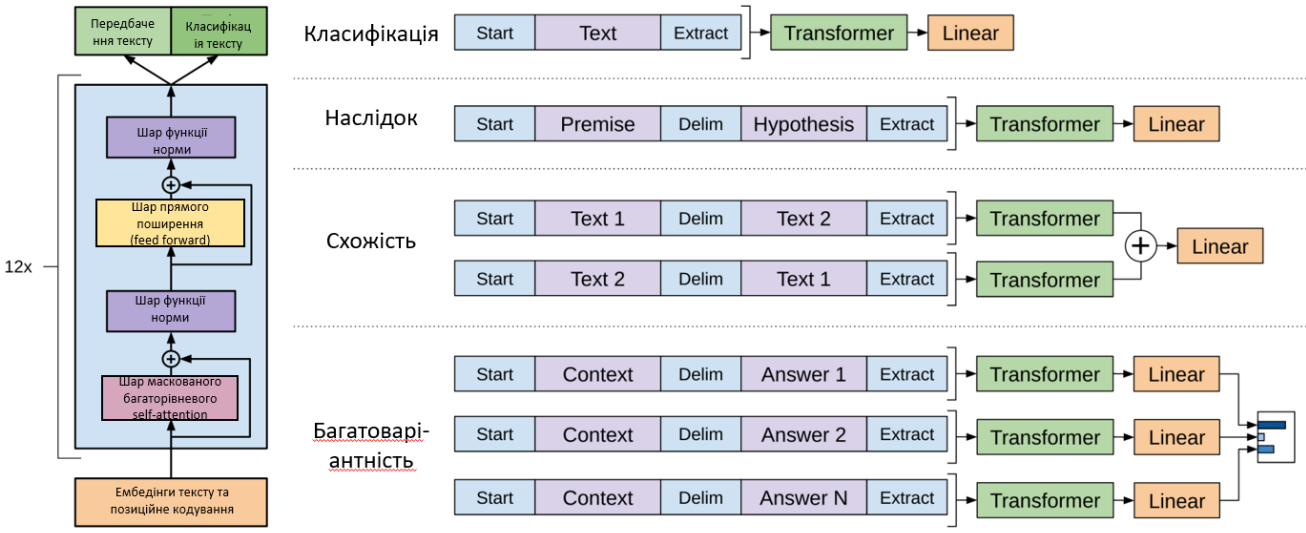


Рисунок В.4 – Архітектура моделі GPT-3

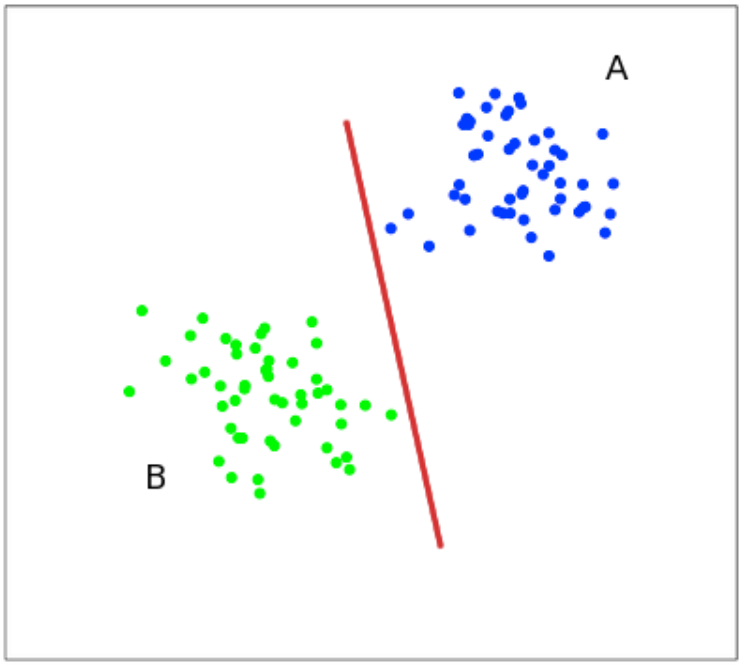


Рисунок В.5 – Приклад методу опорних векторів

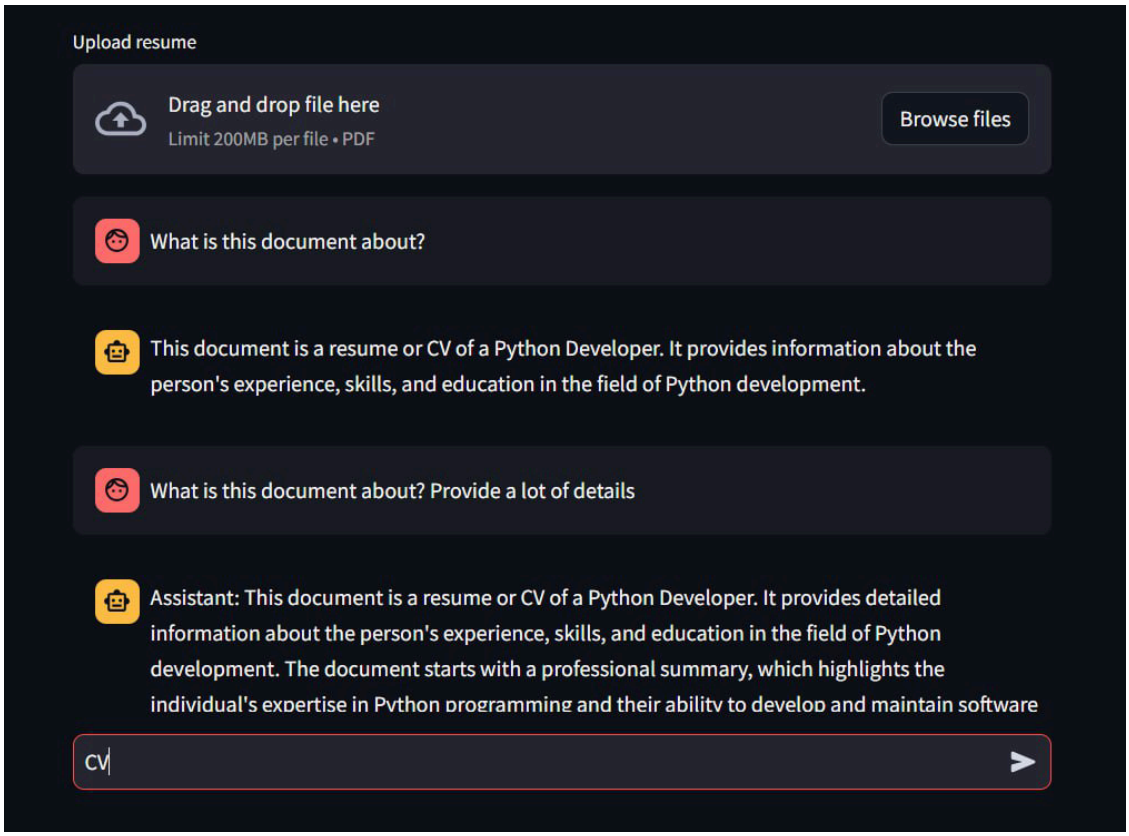


Рисунок В.6 – Приклад роботи програми

Додаток Г (довідниковий)

Інструкція користувача

Для запуску програмного продукту інформаційної технології парсингу резюме необхідно перейти у директорії з програмним продуктом, запустити віртуальне середовище за допомогою команди `poetry shell` після чого потрібно зачекати декілька секунд. Необхідно запустити віртуальне оточення командою «power shell», приклад запуску зображено на рисунку Г.1.

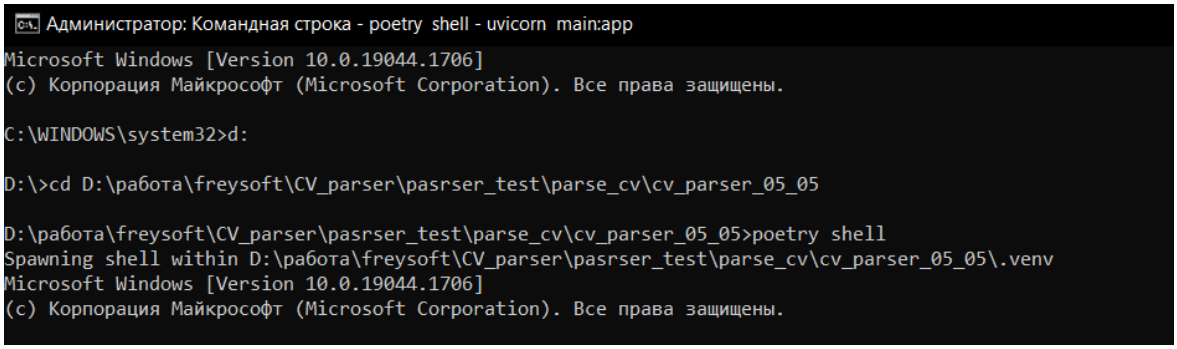


Рисунок Г.1 – Запуск віртуального оточення

Потім необхідно запустити емуляцію серверу uvicorn на локальній машині як зображено на рисунку Г.2.

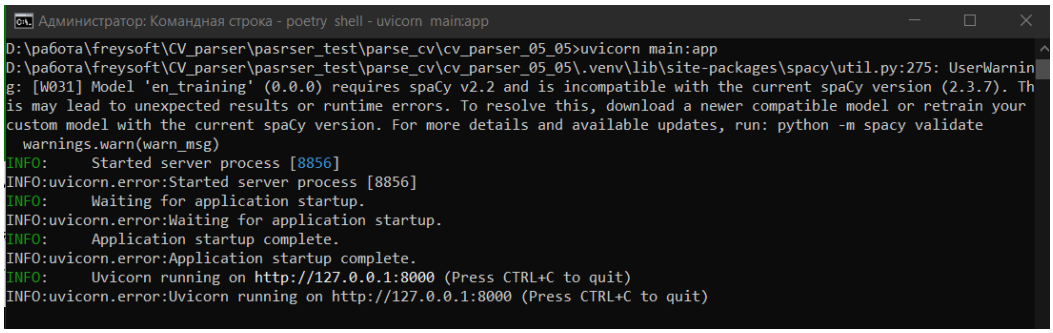


Рисунок Г.2 – Запуск серверу

Програмний продукт був створений за допомогою бібліотеки Streamlit, тож необхідно перейти у директорію /src та запустити Streamlit додаток. приклад запуску зображено на рисунку Г.3.

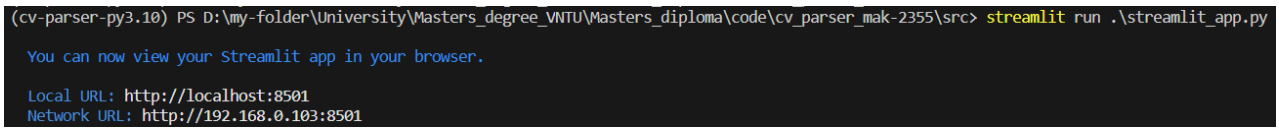


Рисунок Г.3 – Запуск Streamlit додатку

Після цього автоматично відкривається вікно браузера у якому нас просять «завантажити резюме». Опісля ми побачимо, що резюме успішно зберігається у векторне сховище і за декілька секунд нас проінформує, що резюме завантажено, спаршено і ми можемо задавати питання по цьому документу, як це зображено на рисунку Г.3.

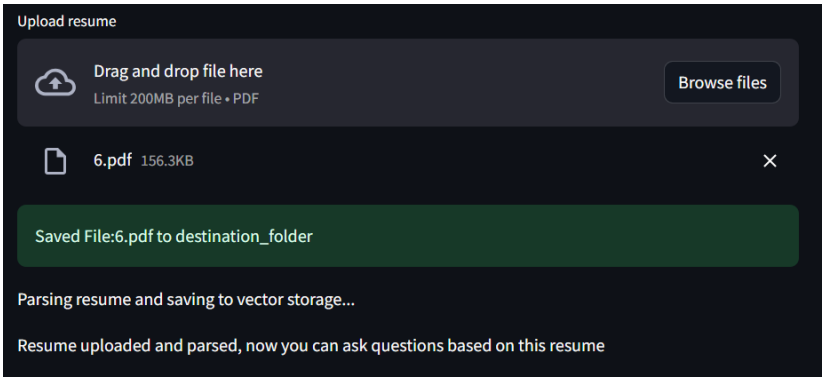


Рисунок Г.3 – Завантаження, парсинг та збереження резюме кандидата

Далі у користувача є широкий вибір можливостей, користувач може запитати, які основні скілли кандидата і, наприкладі даного резюме, отримати чітку та структуровану відповідь. При порівнянні з ориганальним резюме кандидата – можна помітити, що чат бот абсолютно точно дав відповідь. відповідь, як це зображено на рисунку Г.4 (зліва – частина оригінального документа, справа – відповідь агента).

Також у користувача є можливість завантажити нове резюме написавши ключове слово «CV». Після чого на екрані знову з’явиться вікно для завантаження нового резюме, як це зображено на рисунку Г.5.

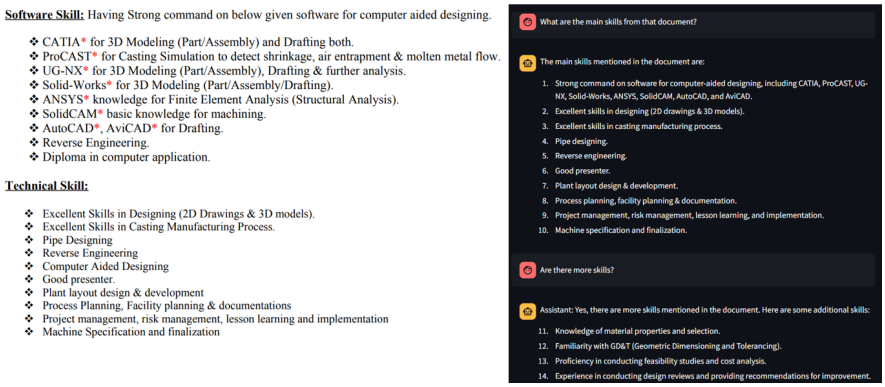


Рисунок Г.4 – Питання користувача про резюме кандидата та відповідь агента

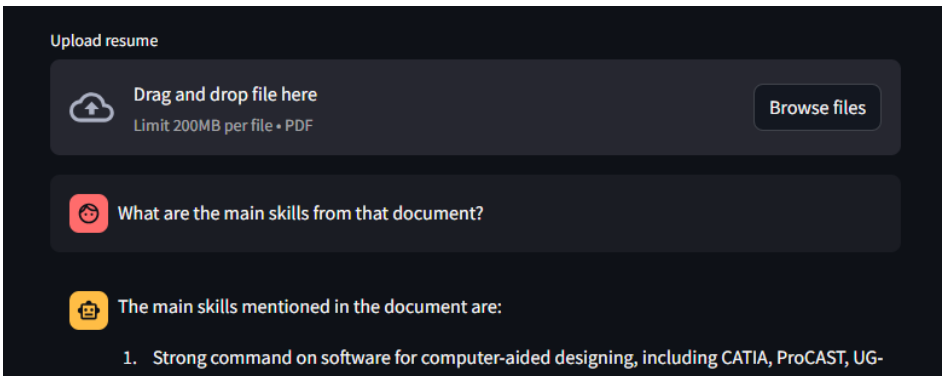


Рисунок Г.5 – Завантаження нового резюме

Після цього користувач може так само задавати необхідні питання по новому резюме та аналізувати його вміст через «людську мову» (наприклад дізнатись місця роботи, яка освіта у кандидата і т.д.). Завантаження нових резюме можна робити необмежену кількість разів. Коли користувач завершить мануальний аналіз, він може надіслати запит до агента в якому попросить надати інформацію про «найбільш релеванте резюме з усіх збережених» виходячи із запиту (наприклад, «резюме, у якому кандидат має знання Python»). Для цього користувач може використати ключову фразу «Find the most suitable resume…» і далі вказати по якому критерію шукати. Приклад роботи агента в цьому режимі наданий на рисунку Г.6.

Як ми можемо побачити з результатів такого запиту на рисунку Г.6 – агент надає детальну інформацію про людей, які мають такий скіл (або потенційно можуть мати такий скілл), а також унизу надає посилання на розташування усіх

завантажених в систему резюме, що підходять під цей запит. При детальній перевірці оригінального вмісту резюме – уся інформація співпадає. А також агент зберігає сам текст резюме, що можна побачити з рисунку Г.7.

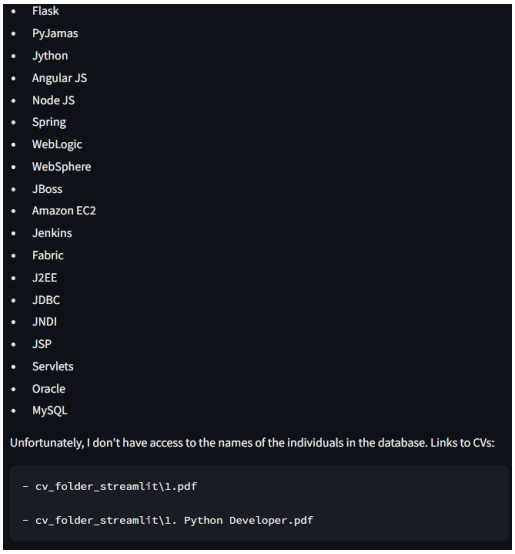


Рисунок Г.6 – Запит до агента надати інформацію згідно конкретного скіла

Рисунок Г.7 – Вміст парсингу резюме

Для завершення роботи модуля необхідно закрити обидва термінали, усі дані було збережено.