Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка засобів підбору та перевірки коректності УДК-шифрів наукових робіт»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1911»

	(підпис студента)	/Данило САФОНОВ/ (Ім'я ПРІЗВИЩЕ)
Керівник:	(підпис)	/доц. Олена КУРОП'ЯТНИК (посада. Ім'я ПРІЗВИЩЕ)
Нормоконтролер:	(підпис)	/доц. Світлана ВОЛКОВА/ (посада. Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of tools for selection and verification of UDC codes of scientific works»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1911:	/Danylo SAFONOV/
Scientific Supervisor:	/Olena KUROPIATNYK/
Normative controller:	/Svitlana VOLKOVA/

Dnipro – 2023

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: «Комп’ютерні технології і системи»

Кафедра: «Комп’ютерні інформаційні технології»

Рівень вищої освіти: бакалавр

Освітня програма: «Інженерія програмного забезпечення»

Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

_____ /Вадим ГОРЯЧКІН/

(підпис)

Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студенту Сафонову Данилу Євгеновичу

1. Тема роботи: «Розробка засобів підбору та перевірки коректності УДК-шифрів наукових робіт»

Керівник роботи: Куроп’ятник Олена Сергіївна, доцент

затверджена наказом № 1209 ст від 07.12.2022

2. Строк подання студентом роботи: 15.06.2023

3. Вихідні дані до роботи:

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

Збір та аналіз вимог

Проектування

Розробка програми

Тестування

Поширення додатку

Загальні висновки та рекомендації

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень):

Презентація

Відео роботи програми

КАЛЕНДАРНИЙ ПЛАН

Стадія	Зміст	Строки виконання
Підготовка до розробки	Збір та аналіз вимог	31.03.23
	Проектування	30.04.23
Розробка	Розробка програми	15.06.23
	Поширення додатку	13.05.23
	Тестування	15.06.23
Захист	Подання кваліфікаційної роботи до кафедри	16.06.23
	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.23

Студент _____ Данило САФОНОВ

(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олена КУРОП’ЯТНИК

(підпис) (посада, Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Предметом розробки є система підбору та перевірки коректності УДК-шифрів наукових робіт.

Мета роботи полягає у розробці програмного забезпечення для підбору та перевірки коректності УДК-шифрів наукових робіт. Основними задачами є розробка та реалізація:

- засобів попередньої обробки та токенізації текстів;
- моделі УДК, яка містить класи та ключові та слова, та засобів її наповнення шляхом тренувань;
- засобів підбору та перевірки коректності УДК-шифру на основі розробленої моделі.

Методи розв'язання задачі. Для реалізації системи застосовано об'єктно-орієнтований (ОО) підхід до проектування з використанням UML та ОО-програмування мовою Python. Для обробки текстів використано алгоритми токенізація, видалення стоп слів, розмічування частин мови та іменованих сутностей, побудовані на основі компонентів бібліотеки spaCy.

Отримані результати. Розроблено настільний додаток з CLI для підбору та перевірки коректності УДК-шифрів для англомовних текстів.

Значення роботи. Додаток може бути корисним для часткової автоматизації класифікації великих бібліотек: результати ручної класифікації застосовуються для навчання моделі, для класифікації решти використовуються рекомендації програми. У розділі висновків наведено пропозиції щодо подальшого розвитку та покращення роботи додатку.

Пояснювальна записка складається з семи розділів, переліку умовних познак, списку використаної літератури та додатків. Загальний обсяг: 80 с., 16 рис., 4 табл., 1 додаток, 101 джерел.

Ключові слова: python, spaCy, машинне навчання, обробка природної мови, УДК, інтерфейс командного рядка, ітераційна розробка, автоматизація.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

УДК — універсальна десяткова класифікація

НР — наукова робота

ПЗ — програмне забезпечення

UML — Unified Modeling Language, уніфікована мова моделювання

ВСТУП

Задача, яку розглядає дана робота — переклад та перевірка точності перекладу текстів [1] (НР) з природної мови на формальну (шифр УДК).

Точність УДК шифрів є ключовою для ефективної класифікації та пошуку наукової літератури. Однак, ручний вибір та перевірка УДК шифрів вимагає значних зусиль, займає багато часу та супроводжується високим ризиком виникнення помилок. Зовнішня перевірка може зменшити ризик помилок, але вимагає додаткових зусиль. Натомість, автоматизоване рішення може значно зменшити час та ресурси, необхідні для вибору та перевірки, а також мінімізувати ризик помилок. Тому ця робота має на меті розробити надійний та ефективний автоматизований інструмент для вибору та перевірки УДК шифрів НР.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

1.1 Універсальна десяткова класифікація

Універсальна десяткова класифікація (УДК) [2] —— бібліографічна та бібліотечна класифікація, представляє систематичне впорядкування всіх галузей людських знань, організованих як узгоджена система, у якій галузі знань заємопов'язані.

В УДК використовуються арабські цифри в десятковому порядку. Кожне число розглядається як десятковий дріб з опущеною початковою десятковою крапкою, яка визначає порядок запису. Для зручності читання УДК зазвичай ставиться розділовий знак після кожної третьої цифри.

Основні таблиці [3] містять різні дисципліни та галузі знань, розташовані в 9 основних класах, пронумерованих від 0 до 9 (при цьому 4 клас є вільним). На початку кожного класу також є серія спеціальних допоміжних слів, які виражають аспекти, що повторюються в цьому конкретному класі. Основні таблиці в УДК містять понад 60 000 підрозділів.

0. Наука і знання. Організація. Комп'ютерна наука. Інформатика. Документація. Бібліотечна справа. Заклади. Публікації
1. Філософія. Психологія
2. Релігія. Теологія
3. Суспільствознавство
4. -
5. Математика. Природничі науки
6. Прикладні науки. Медицина, Техніка
7. Мистецтво. Розваги. Спорт
8. Мовознавство. Література

9. Географія. Історія

Загальні допоміжні засоби (табл. 1) — концепції, які можна використовувати в поєднанні з будь-яким іншим кодом УДК з основних класів або з іншими загальними допоміжними засобами. Вони мають унікальні позначення, що виділяють їх у складних виразах. Звичайні допоміжні числа завжди починаються з певного символу, напр. = (знак рівності) завжди вводить поняття, що представляють мову документа; (0...) цифри в круглих дужках, починаючи з нуля, завжди представляють поняття, що позначає форму документа. Таким чином (075) підручник і =111 англійська мова може бути об'єднана, щоб виразити, наприклад, (075)=111 підручники англійською мовою, і в поєднанні з числами з основних таблиць УДК їх можна використовувати таким чином: 2(075)=111 підручники з релігії англійською мовою, 51(075)=111 Підручники з математики англійською мовою та ін.

Синтаксис	Назва
=...	Загальні допоміжні засоби мови.
(0...)	Загальні допоміжні форми форми.
(1/9)	Загальні допоміжні слова місця.
(=...)	Загальні допоміжні ознаки людського походження, етнічної групи та національності.
"..."	Загальні допоміжні слова часу, допомагає зробити хвилинний поділ часу, наприклад: "1993-1996"
-0...	Загальні допоміжні характеристики загальних характеристик: Властивості, Матеріали, Відносини/Процеси та Особи.
-02	Загальні допоміжні властивості.
-03	Загальні допоміжні матеріали.
-04	Загальні допоміжні елементи відносин, процесів і операцій.
-05	Загальні допоміжні особи та особисті характеристики.

Таблиця 1 — Загальні допоміжні засоби УДК

Доступно кілька сполучних символів для зв'язку та розширення номерів УДК (табл. 2).

Символ	Значення
+	узгодження, доповнення
/	послідовне розширення
:	відношення
[ ]	підгрупування
*	Впроваджує нотацію, відмінну від УДК
A/Z	Пряма алфавітна специфікація

Таблиця 2 — Сполучні символи УДК

1.2 Огляд аналогів

Широкодоступних аналогів, які б повністю відтворювали функціонал — підбір кодів (або в нашому випадку класів) УДК та їх порівняння/перевірка, не знайдено. Під широким доступом маються на увазі комерційні або відкриті/безкоштовні рішення.

Втім, існують наукові роботи із схожою метою:

- ”Automatic classification of older electronic texts into the UDC” [4] — ця робота ставить своєю метою розробити модель для класифікації старих оцифрованих текстів.
- ”Service for assigning a UDC code to mathematical articles based on semantic technologies” [5] — дана робота намагається вирішити проблему класифікації математичних статей.
- ”A Hybrid Approach to Recommending Universal Decimal Classification Codes for Cataloguing in Slovenian Digital Libraries” [6] — ця робота має на меті додаток, який буде пропонувати бібліотекарям УДК шифри, тим самим значно спрощуючи їх роботу.
- ”Automated Subject Identification using the Universal Decimal Classification: The ANN Approach” [7] — мета цієї роботи така ж сама що й в попереднього прикладу.

- ”Porter advanced method for the Universal Decimal Classification” [8] — в процесі цієї роботи розроблюється додаток, який буде пропонувати бібліотекарям УДК шифри, тим самим значно спрощуючи їх роботу.

1.3 Аналіз проблеми та постановка задачі

Мета роботи — розробити ПЗ для підбору та перевірки коректності УДК шифрів НР. Під підбором розуміється наступний функціонал — на вході маємо текст роботи, на виході — список класів та підкласів УДК (далі — класів), до яких ця робота належить. Перевірка — на вході маємо текст роботи та шифр, на виході — згенерований шифр та оцінку схожості наданого та згенерованого шифрів, яким саме чином ця оцінка буде отримана, та як вона буде виглядати розглянемо в наступних розділах.

Програму можна узагальнити до такої специфікації — на вході маємо текст роботи та опціонально шифр, на виході маємо список класів та, якщо був наданий шифр, порівняльну оцінку зі згенерованими класами.

Через те що наукові роботи можуть бути доступні у різноманітних форматах, варто уточнити що ПЗ буде приймати їх у форматі plain-text на англійській мові.

Це можна розглядати як проблему класифікації [9, 10] — маємо множину об’єктів, які певним чином розподілені на класи (підмножини). Задача класифікації — різновид машинного навчання [11] (англ. machine learning, далі —– ML). Розрізняють два типи навчання:

- навчання з учителем (англ. supervised learning) — комп’ютеру надається набір даних (data set), в якому даним вже надано бажаний результат — саме за ним комп’ютер і буде навчатись.
- навчання без учителя (англ. unsupervised learning) — набір даних, наданий комп’ютеру, ніяк не помічається — комп’ютер сам має розбити дані на деякі групи (clustering).

Класи заздалегідь відомі, тож навчання без учителя, в нашому випадку, не підходить. Крім того можна використати існуючи класифіковані роботи, як набір

даних для навчання. Але отримання достатнього набору даних вимагає великих зусиль — потрібно отримати декілька робіт для кожного класу та підкласу та їх різних комбінацій. Тож пропонується простіший алгоритм:

1. Витягнемо з тексту роботи ключові слова. [12]
2. Порівняємо ключові слова із термінами та поняттями у каталозі УДК.

Останній крок є дещо проблематичним через те що доступ до офіційного каталогу здійснюється на платній основі — такий варіант не підходить для даного випадку через відсутність фінансування. Але існують безкоштовні та/або відкриті каталоги [13, 14], які можна використати замість платної версії. Авжеж вони мають свої недоліки, наприклад: менша частота оновлень, менший корпус інформації, тощо. Не дивлячись на ці недоліки вони є достатнім ресурсом для початкової версії ПЗ. Також за бажанням ПЗ може бути відредагованим у майбутньому щоб використовувати платну версію.

Висновки до розділу 1

Була розглянута система УДК - задача, яку ця робота намагається вирішити/спростити, є трудомісткою та займає час. Через це доцільне створення інструмента, який буде хоча б частково автоматизувати цю задачу.

Широкодоступних аналогів не знайдено, наукові роботи із схожою метою в більшості розглядають лише часткові рішення — вони орієнтуються на деяку підмножину НР.

2 ПРОЄКТУВАННЯ

2.1 Зовнішнє проєктування

Розробнику потрібна можливість тренувати модель. Користувачу — отримувати припущення від натренованої моделі припущень, та опціонально порівнювати це припущення із припущенням користувача.

1. Тренування моделі припущень:
    - вхід:
        - модель припущень,
        - текст;
    - вихід — нова модель припущень.
2. Отримання припущень від моделі:
    - вхід:
        - модель припущень,
        - текст;
    - вихід — список класів.
3. Отримання припущень від моделі та порівняння із наданим шифром УДК:
    - вхід:
        - модель припущень,
        - текст,
        - список класів УДК;
    - вихід:
        - список класів,
        - ступінь відповідності класів.

Розглянуті сценарії можна формалізувати у наступній use-case діаграмі (рис. 1).

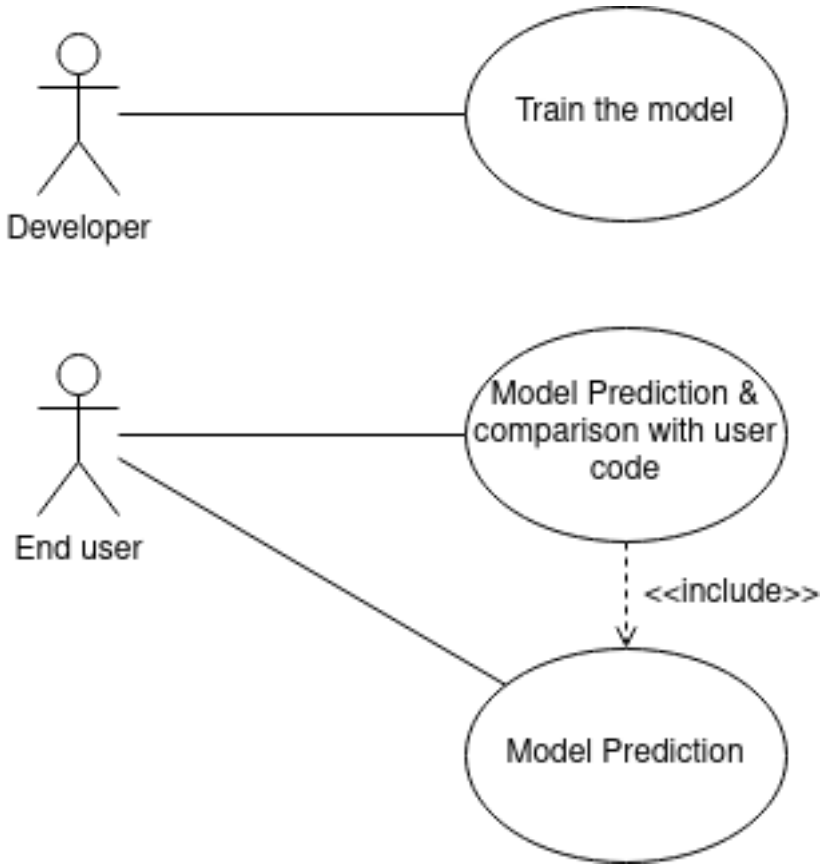

Рисунок 1 — Діаграма прецедентів

2.2 Внутрішнє проєктування

В додатку можна виділити наступні основні частини:

- інтерфейс користувача — цей модуль буде перетворювати вхідні дані користувача на внутрішнє представлення та перевіряти коректність наданих даних. Також цей модуль відповідає за надання результату користувачу
- бекенд
  - тренування моделі — цей модуль буде додавати нові ключові слова до класів, знайдених у наданому тексті
  - отримання припущення від моделі — цей модуль буде надавати можливі класи до наданого тексту

– порівняння списків класів УДК — цей модуль буде порівнювати два списки класів УДК, та надавати ступінь їх відповідності

Також можна виділити такі типи даних (замість передавання рядка або числа між внутрішніми процедурами, зручніше передавати спеціалізовану структуру):

- наукова робота/текст,
- клас УДК.

2.3 Проєктування архітектури системи

Програму розроблено ітераціями. На кожну з ітерацій виділено окрему секцію. Є два типи ітерацій:

1. Реалізація нового функціоналу,
2. Вдосконалення старого функціоналу.

Для простоти тестування було вирішено почати розробку з інтерфейсу користувача. З попередніх секцій знаємо що це буде інтерфейс командного рядка (CLI - Command Line Interface).

Найпростішим варіантом є перевірка аргументів командного рядка у функції main, і після цього виклик із цими аргументами внутрішньої реалізації.

UML-діаграма для такої моделі виглядає наступним чином (рис. 2). Кінцева реалізація буде мати складнішу ієрархію модулів, але ця секція фокусується на розробці користувацького інтерфейсу, тому на цьому етапі реалізація представлена як дуже простий клас.

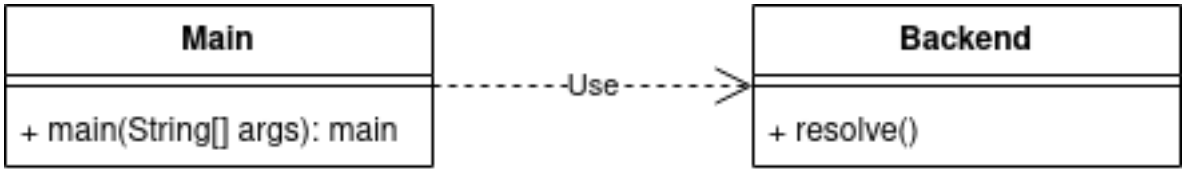

Рисунок 2 — Діаграма високорівневого розподілення обов’язків

Алє таке рішення не є оптимальним — заради простоти початкової реалізації втрачається легкість подальшої заміні інтерфейсу. Тому доцільно виділити модуль який буде відповідати за взаємодію з користувачем. В нашому випадку це буде зчитування аргументів командного рядку, але за потреби він може бути замінений на графічний або веб-інтерфейс, тощо. Тепер діаграма виглядає так (рис. 3).

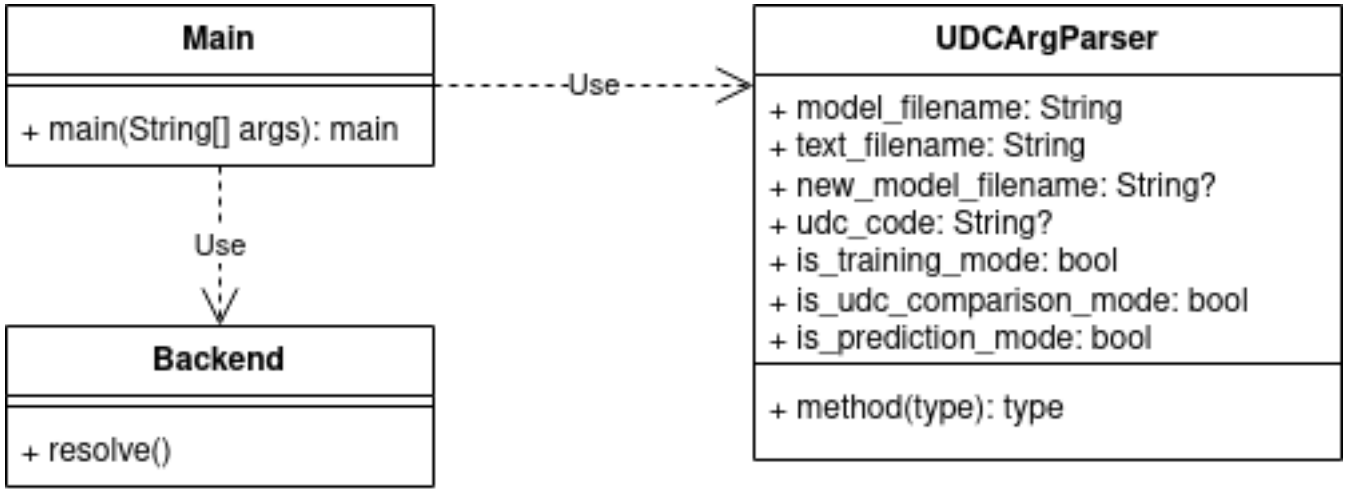

Рисунок 3 — Виділення модуля, який відповідає за розпізнавання аргументів командного рядка

Тепер перед тим як викликати внутрішню реалізацію, main викликає клас UDCArgParser, який у свою чергу опрацьовує аргументи командного рядку.

Хоча ця реалізація вирішує проблему відповідальності, вона додає декілька нових:

- по-перше, UDCArgParser зчитує аргументи комадного рядка у конструкторі із глобальних змінних, це порушує принцип Dependency Inversion [15, 16];
- по-друге, змінні is_training_mode, is_udc_comparison_mode та is_prediction_mode є взаємовиключними.

Для вирішення другої проблеми, потрібно скористатися поліморфізмом [17, 18, 19] і замінити ці три змінні на одну, яка буде приймати значення одного з трьох підкласів. Іншими словами, треба заміни три взаємовиключні булеві змінні на одну

змінну типу перечислення. З цими змінами UML-діаграма виглядає наступним чином (рис. 4).

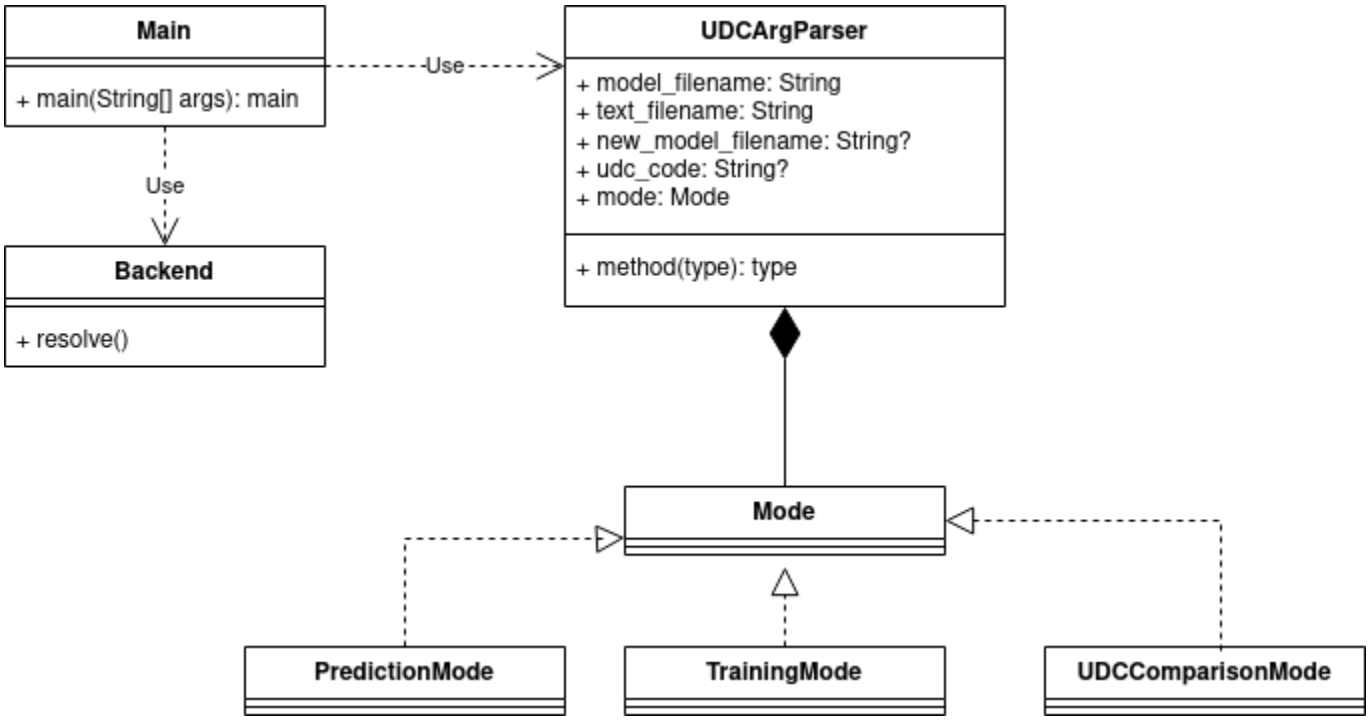

Рисунок 4 — Виділення типів, які відповідають за внутрішне представлення режиму роботи програми

Тепер клас Mode та його підкласи відповідають за представлення режиму, але клас UDCArgParser має змінні new_model_filename та udc_code, які в залежності від значення Mode завжди будуть NULL, тому доцільно перенести усі значення до підкласів Mode, перетворюючи тим самим це перечислення на алгебраічний тип данних [20].

Доцільно помітити що алгебраічні типи даних відсутні в ООП та мові Python, тому вони будуть ємульовані з допомогою наслідування [21, 22].

Таким чином UDCArgParser дійсно буде мати тільки одну відповідальність — перетворення аргументів командного рядка на об’єкт Mode. А Mode у свою чергу відповідає за представлення даних, які будуть надані внутрішній реалізації. Після цих змін UML-діаграма буде виглядати так (рис. 5).

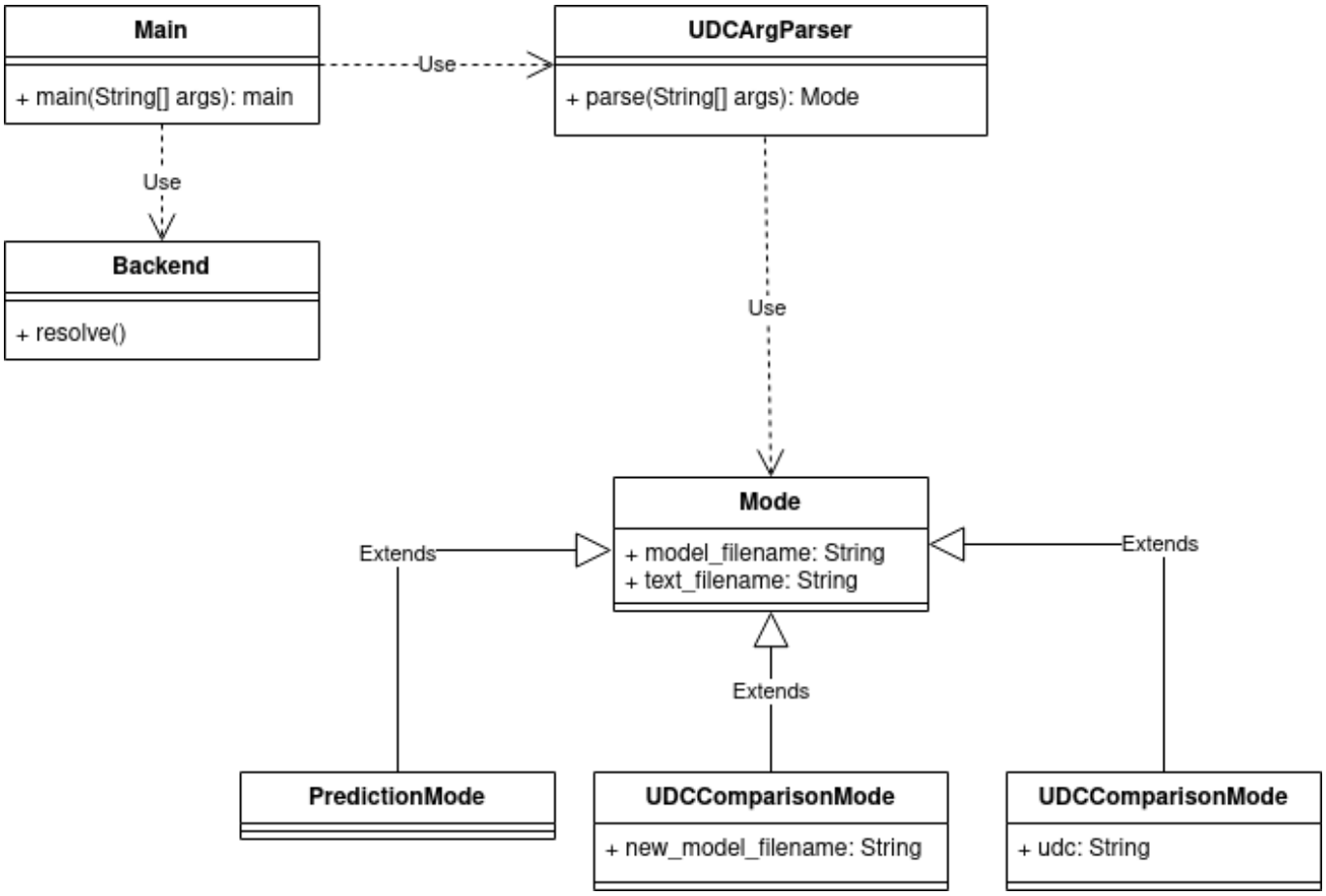

Рисунок 5 — Перенесення атрибутів режиму роботи до відповідних типів

Нова версія набагато краще попередніх, особливо першої. Втім, не усі недоліки були всунуті.

Тип Mode містить поля model_filename та text_filename — вони будуть передані в Backend (це не показано на діаграмах, тому теж є недоліком). Внутрішній реалізації потрібен вміст цих файлів, насправді внутрішня реалізація навіть не повинна знати що це вміст файлів, а не, наприклад, вміст текстового поля у графічному інтерфейсі, тощо. Тому доцільно:

1. Прибрати слово filename з обох полів,
2. Виділити типи для цих значень — таким чином ми зможемо робити майбутні зміни лише в одному місці [23]

Також присутня змінна new_model_filename в UDCComparisonMode — вона так само не є важливою дла внутрішньої реалізації, а тільки для інтерфейсу кори-

стувача. Через те що для внутрішньої реалізації це буде одним з результатів (також можуть бути припущення щодо класів УДК та точності припущення, зробленного користувачем), це потрібно формалізувати в інтерфейсі класу Backend.

Крім того, з попереднього пункту видно що потрібно формалізувати тип результату виконання внутрішньої реалізації. Втім, через те що мається чітка відовідність між підкласами Mode та типом (типами) результата внутрішньої реалізації доцільно розділити ці дії (режими роботи додатку) на окремі методи в класах Main та Backend.

У новій діаграмі (рис. 6) виділено типи UdcPredictorModel та UdcPredictorInput-Text, також явно показано залежність внутрішньої реалізації від типу Mode.

UDCArgParser тепер має відповідальність за зчитування файлів за наданими іменами, але саме зчитування оброблюється внутрішньою бібліотекою мови Python, тому це не є порушення принципу єдиної відповідальності [24]. Ця залежність не показана на діаграмі тому що вона подразумевається/implied (я не знаю как перевести єто слово, googletranslate предлагает вообще не то) як атомарний функціонал мови, як і багато інших дещо простіших речей.

Для вирішення інших проблем, описаних в попередній ітерації, краще розпочати з конкретизації внутрішньої реалізації. Через те що кожному з вхідних підтипів Mode буде відповідати новий формат результату, доцільно розділити метод resolve, на три різні. Крім того, ми можемо позбавитися залежності між внутрішньою реалізацією та типом Mode — методи можуть приймати декілька параметрів, щодо результатів, функції в мові Python (та інших сучасних) можна повертати декілька значень [25].

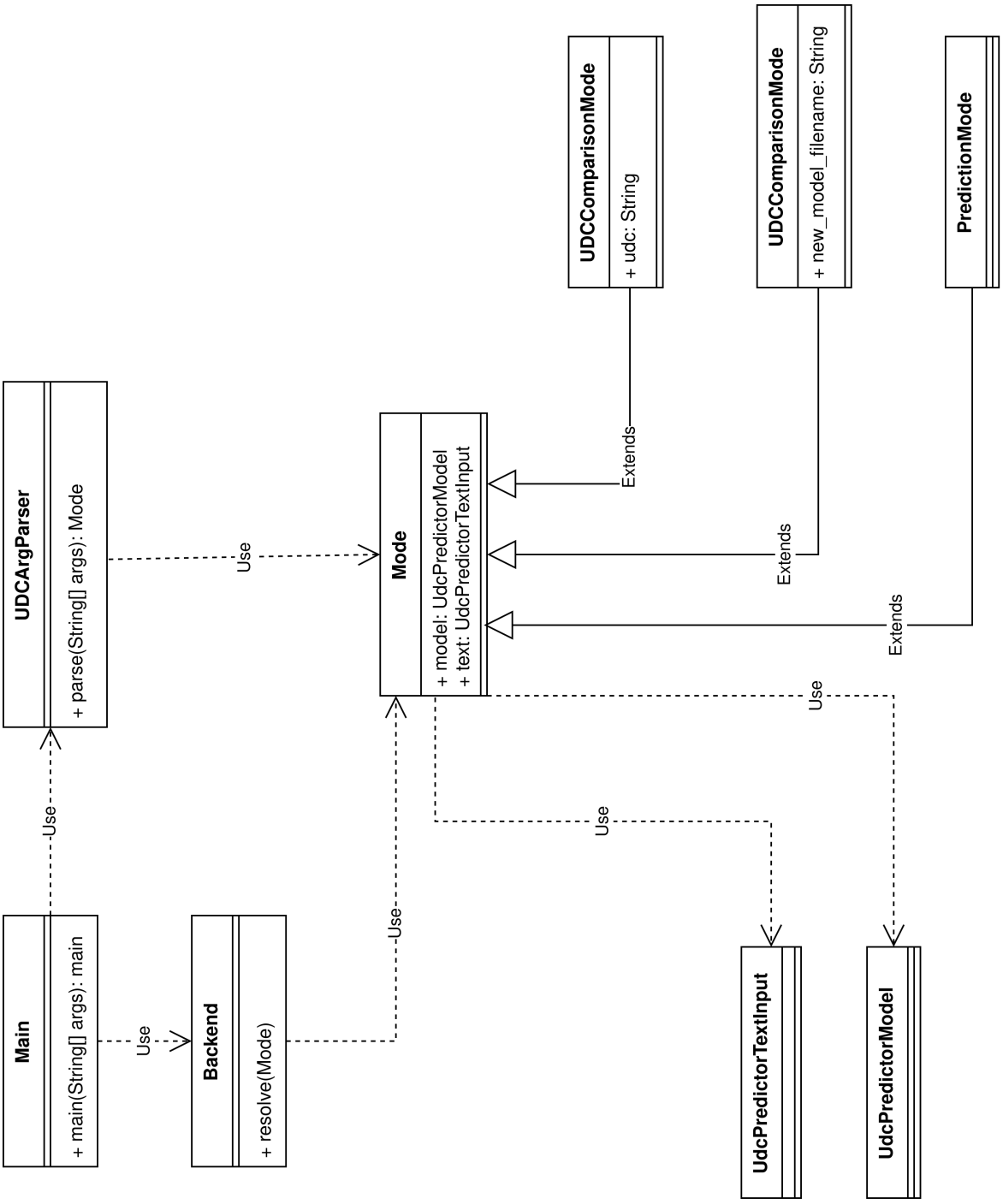

Рисунок 6 — Інкапсуляція типів вхідних даних

Через ці зміни, доцільно частково відмінити попередню зміну — тип Mode буде мати імена файлів, а клас Main буде зчитувати та записувати їх. Це рішення є покращенням через те що тип UDCComparisonMode все ще має ім’я файлу, насправді його можна було б замінити на дескріптор файлу, алє тоді отримання ресурсу (файлу), та його звільнення (закриття) відбувалося б у різних методах та класах — це порушує RAII [26]. Крім того, на даний момент клас Main не мав явної відповідальності (крім посередництва між іншими класами), тому ми можемо додати цю роботу до цього модуля. Також внутрішня реалізація буде напряму залежати від типів UdcPredictorTextInput та UdcPredictorModel.

Нова версія (рис. 7) вирішує вище описані проблеми наступним чином:

- внутрішня реалізація тепер має два методи — припущення та тренування, вони напряму залежать від типів UdcPredictorModel та UdcPredictorInputText.
- тип/клас/модуль UdcCode відподідає за представлення шифрів УДК та їх порівняння, модуль Main перетворює рядок отриманий від UdcArgParser на цей тип.
- усі взаємодії із файлами відбуваются у класі Main.

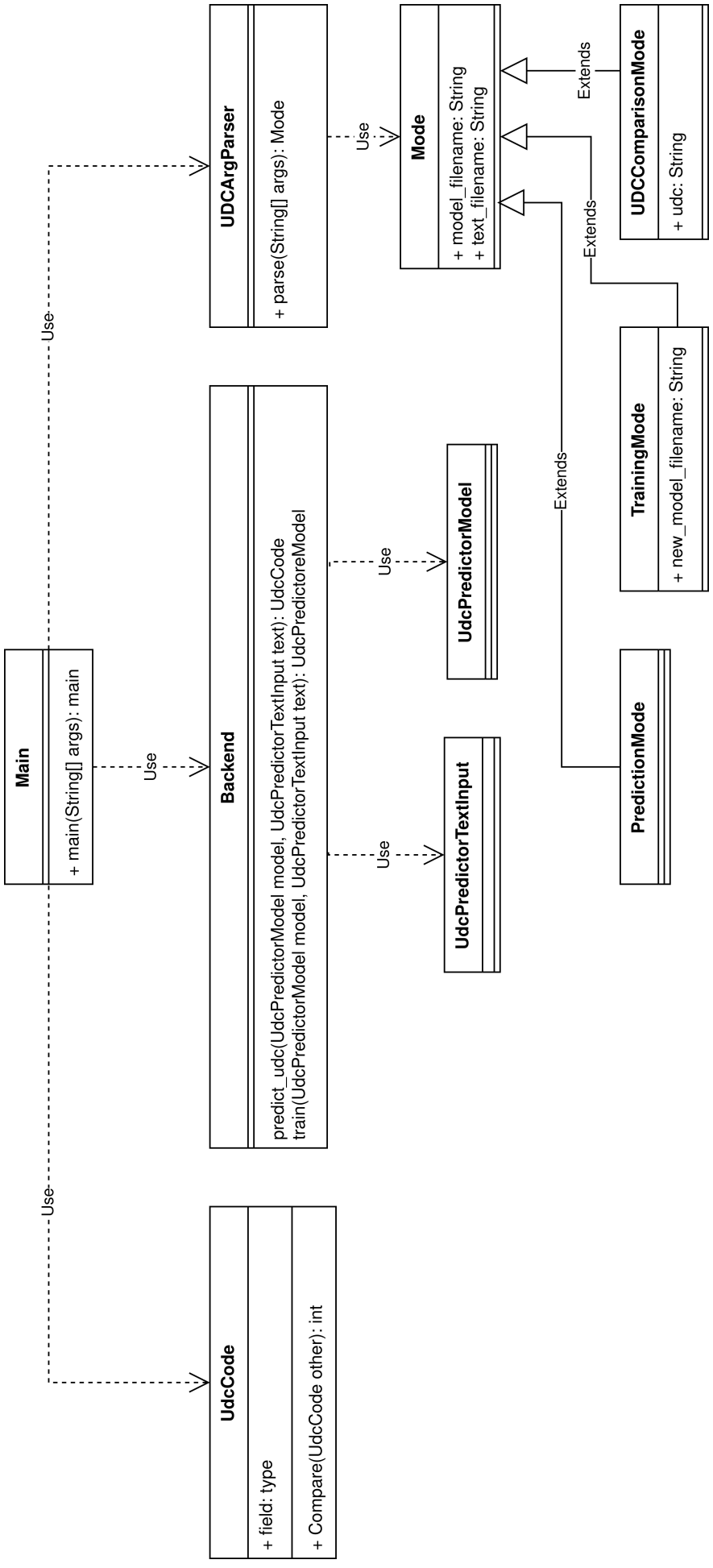

Рисунок 7 — Кінцева версія архітектури додатку

2.4 Вибір засобів програмування

Для реалізації даного ПЗ було обрано мову Python, як найпопулярнішу в даному напрямі. Крім того, через те що ця мова була й залишається однією з самих популярних багато років поспіль [27, 28, 29, 30] (рис. 8), вона має багатий набір різноманітних бібліотек та велику громаду, що дозволяє швидко знаходити рішення навіть для складних проблем.

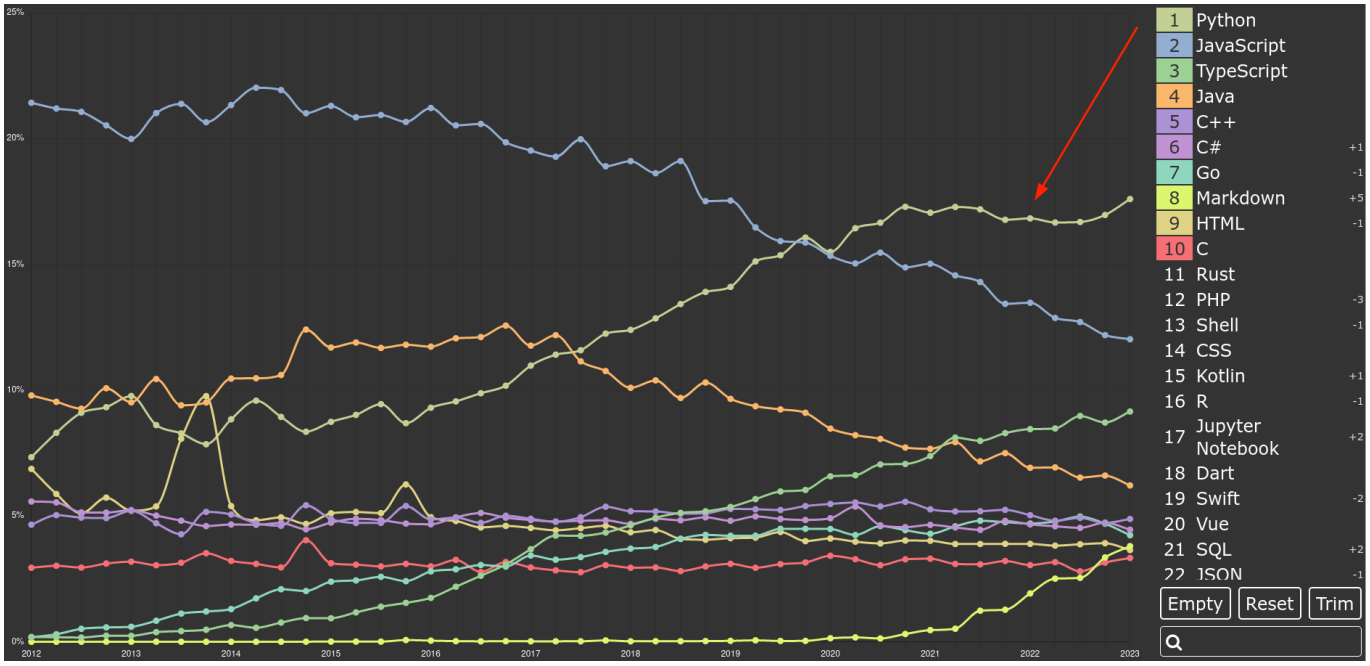

Рисунок 8 — Скриншот графіку популярності мов програмування з веб-ресурсу languish

Для вирішення задачі отримання ключових слів для заданого тексту потрібен такий функціонал:

- токенізація,
- видалення стоп слів [31],
- розмічування частин мови [32],
- розпізнавання іменованих сутностей [33].

Альтернативою може бути бібліотека, яка б надавала готовий алгоритм, наприклад:

- RAKE (Rapid Automatic Keyword Extraction) [34],
- YAKE (Yet Another Keyword Extractor) [35],
- spaCy [36],
- NLTK (Natural Language Toolkit) [37],
- TextBlob [38],
- Gensim [39].

YAKE потребує тренування на конкретному наборі даних — не підходить для цього випадку з тієї самої причини що й класифікація.

TextBlob є високорівневою обгорткою над NLTK, тому їх доцільно розглядати разом. Це бібліотеки загального призначення, але їх можна відкинути з наступних причин:

- вони програють spaCy за швидкістю,
- spaCy має кращі моделі,
- spaCy має більш просунуті функції.

Хоча spaCy не має готового алгоритму витягування ключових слів, його дуже легко реалізувати.

Алгоритм, який використовує RAKE працює наступним чином — текст розбивається на слова, потім перевіряється частота слів, та ступінь їх появи з іншими, слова сортуються за обома критеріями з попереднього кроку.

spaCy не надає готово алгоритму виділення ключових слів, але дає можливість розбити текст на слова, розмічати їх як частини мови, розпізнати іменовані сутності.

Алгоритм RAKE суто статистичний і має більшу похибку — він може вважати за ключові слова такі, які не є вагомими та навпаки виключати важливі.

Gensim не має деяких важливих алгоритмів для реалізації витягування ключових слів, а ті які є в цій бібліотеці були розроблені, в першу чергу, для інших цілей і не є оптимальними для нашої задачі. З цього виходить що spaCy є найкращим вибором:

- має весь потрібний функціонал;
- має моделі;
- швидше за інші бібліотеки.

Висновки до розділу 2

Були розглянуті різні типи користувачів, сценарії використання додатку цими користувачами. Завдяки цьому функціонал ПЗ був розподілений в залежності від сценаріїв використання. Таке розподілення є корисним для розробки архітектури програми та, в подальшому, написання реалізації.

Було розроблено високорівневу архітектуру програми — виділено модулі та типи, які будуть використані при розробці програми. Завдяки використаному ітераційному підходу розробки архітектури було помічено та виправлено можливі проблеми ще до написання коду.

Окрема секція приділена вибору засобів програмування. Було порівняно сучасні популярні бібліотеки для мови Python та обрано найкращий для реалізації задачи. Мову Python було обрано як варіант з найбільшою підтримкою бібліотек для машинного навчання.

3 РОЗРОБКА ПРОГРАМИ

3.1 Розробка інтерфейсу користувача

Для простоти виконання буде використано інтерфейс командного рядка (CLI). Для цієї задачі в мові python є спеціалізований модуль argparse.

З розробленої use-case діаграми можна сказати шо є два режими роботи (стільки ж скільки й акторів). Через те що режими тільки два, доцільно використати прапор. Основним сценарієм є припущення, тому прапор буде використовуватися для тренування ”training”.

Модель та текст присутні на вході кожного випадку, тому їх можна зробити позиційними аргументами.

Список класів є тільки в одному випадку, тому це буде опціональним аргументом, який буде прийматися тільки якщо відсутній прапор ”training”, цей аргумент буде помічатись як ”udc”. Для припущення результат буде надаватися в stdout у наступному форматі: список класів та ступінь їх відповідності, якщо використано ”udc”.

Маємо такі сценарії використання:

1. Тренування моделі

```
$ udc-classifier \
    model.model \
    scientific-work.txt \
    --training updated-model.model
```

Listing 1: Тренування моделі

- udc-classifier — назва виконуваного файлу
- model.model — назва файлу існуючої моделі
- scientific-work.txt — назва файлу з текстом наукової роботи
- updated-model.model — назва файлу, до якого буде записано нову модель

2. Отримання припущення

```
$ udc-classifier model.model scientific-work.txt
```

Listing 2: Отримання припущення

```
539.120 94 084.3
```

Listing 3: Припущення (класи, які можуть підійти до наданого тексту)

- udc-classifier — назва виконуваного файлу
- model.model — назва файлу існуючої моделі
- scientific-work.txt — назва файлу з текстом наукової роботи

3. Отримання припущення та порівняння класів

```
$ udc-classifier \
    model.model \
    scientific-work.txt \
    --udc ''913(574.22)"19"(084.3)
```

Listing 4: Отримання припущення та порівняння класів

```
539.120 94 084.3
0.3
```

Listing 5: Припущення

- udc-classifier — назва виконуваного файлу
- model.model — назва файлу існуючої моделі
- scientific-work.txt — назва файлу з текстом наукової роботи
- третій рядок прикладу — класи, які можуть підійти до наданого тексту
- другий рядок — вхідний шифр УДК
- четвертий рядок — ступінь відповідності класів (від 0 до 1, де 0 — відсутні співпадіння, 1 — повне співпадіння)

3.2 Перетворення ключових слів на класи УДК

В наведених у попередніх розділах відкритих каталогах УДК кількість ключових слів для кожного з підкласів дуже мала (близько 10 на кожен підклас), тому буде згенеровано свій каталог. Алгоритм створення наступний:

1. Беремо ключові слова з існуючої роботи (ті які надав автор),
2. Витягуємо ключові слова з тексту з допомогою обраного алгоритму,
3. Порівнюємо ключові слова з п.1 та п.2 та додаємо до каталогу ті, які входять до обох списків.

Точність такого підходу буде напряму залежати від кількості розглянутих робіт та кількості отриманих ключових слів. Через обмежений час такі каталоги будуть створені не для усіх підкласів, та деякі класи будуть мати меншу вибірку.

Для розуміння формату моделі, яка буде у подальшому використовуватися для припущень, доцільно деталізувати цей алгоритм. Так як модель буде змінюватися незалежно від коду — це дані, які будуть зберігатися окремо від коду — у файлі. Через це потрібно обрати формат серіалізації [40, 41].

3.2.1 Вибір формату серіалізації

Для порівняння популярних форматів можна звернутися до статті «A Comparison Of Serialization Formats» [42] на веб-ресурсі mbedded.ninja [43].

Розглянемо критерії за якими ця стаття порівнює формати, та оберемо ті, які є важливими:

- стисліcть — цю характеристику можна вважати однією з складових читабельності, тому для простоти порівняння вона не буде включена у порівняльну таблицю;
- читабельність — хоча дані, які будуть серіалізуватися здебільшого орієнтовані на читання не людиною, а комп’ютером, іноді можуть

виникати ситуації коли буде читабельність є важливою. Наприклад — налагодження;

- підтримка мов програмування — ця характеристика є важливою для сумісності [44] з можливими розширеннями на інших мовах програмування;
- структура даних — ця характеристика має відносно низьку важливість. Вона має значення тільки у тих випадках, коли оцінка дуже низька, що свідчить про погану гнучкість і необхідність незручного перетворення деяких структур даних;
- швидкість — ця характеристика дуже залежна від реалізації тому не буде порівнюватися. Крім того, в цьому додатку швидкість не є критичною характеристикою, відповідно це стосується і складових додатку;
- стандартизація — без цієї характеристики підтримка різних мов програмування має мало сенсу.

Представимо важливу для нас частину даних з наведеного раніше ресурсу у вигляді таблиці (3) для простішого вибору формату серіалізації.

Одразу ж доцільно відкинути CSV [45] за низьку оцінку в структурі даних — підтримуются тільки таблиці, та Protobuf [46] за дуже низьку оцінку в читабельності.

Наступним кроком відкидаємо JSON [47] та XML [48] через погану читабельність.

Залишається обрати з TOML [49] та YAML [50]. Підтримка мов програмування у обох варіантів має однакову оцінку тому будемо обирати за іншими характеристиками. Найпростіше буде скласти інші оцінки та обрати варінт із більшою сумою, за таким порівняння виграє TOML (9+9+9=27) проти YAML (7+10+8=25).

Формат	Підтримка мов програмування	Читабельність	Структура даних	Стандартизація
CSV	9/10	5/10	3/10	3/10
JSON	9/10	5/10	6/10	9/10
Protobuf	7/10	1/10	8/10	9/10
TOML	6/10	9/10	9/10	9/10
XML	9/10	5/10	9/10	10/10
YAML	6/10	7/10	10/10	8/10

Таблиця 3 — Порівняння форматів серіалізації

Можна також обрати той варіант який виграє по більшій кількості показників, тут також виграє TOML - два проти одного.

Можна було б використати складніші способи порівняння, наприклад для кожної з обраних характеристик можна додавати множник важливості, і порівнювати суму значень з множниками. Алє на першому етапі розробки для цього додатку це рішення не є практичним — набагато краще було б отримати відгуки реальних користувачів, і вже на його основі можливо змінювати формат серіалізації, або додавати підтримку нового.

Було обрано TOML (акронім ”Tom’s Obvious, Minimal Language”). Крім вже оглянутих характеристик, можна додати що цей формат має синтаксис дещо схожий на Python, це можна вважати плюсом тому що дозволяє розробнику, та можливу користувачи витрачати меньше єнергії на перехід від коду до серіалізованих даних [51]. Ця ситуація є схожою на JavaScript та JSON, тож якщо цей додаток був би написаний на JavaScript то JSON мав би додаткову перевагу.

Крім того слід додати що вибір популярного та добре стандартизованого формату дозволяє мати більшу сумісність з іншими форматами завдяки конверторам з одного формату в інший [52, 53, 54].

3.2.2 Формат моделі

Під час тренування моделі клас УДК буде вважатися таким, що відповідає ключовому слову (або навпаки), якщо ключове слово підібране програмою наявне у списку ключових слів, підібраних автором. Ця відповідність між ключовим словом та класом буде додана до моделі. Класи УДК в даному випадку будуть надані користувачем для тренування.

Крім того, в подальшому, для звуження множини рекомендованих класів, можна буде додати систему рангування відповідальності між ключовим словом та класом.

По перше, доцільно допускати до тренування лише підмножину ключових слів підібраних програмою [55]. На даний момент допускається 50 найпопулярніших

ключових слів — такий критерій було обрано здебільшого довільно, через обмежений час — за можливості доцільно було б порівняти різні способи вибірки [56] підмножин ключових слів.

По друге, можна рангувати [57] кожну пару з класу УДК та ключового слова в відповідальності до того як часто зустрічається таке ключове слово у текстах із відповідним класом УДК. Найпростішим рішенням буде використання кількості використань ключового слова у усіх текстах використаних для тренування (із таким класом УДК), тобто просто сумувати кількість використань у кожному тексті. Такий підхід скоріш за все покращить точність, але він не є ідеальним — різні тексти будуть мати різну довжину, тому кількість використань буде пропорційно змінюватись. Крім того, частота використання слів може мати різну пропорцію навіть у текстах схожих розмірів, тому доцільно було б використовувати відносну частоту — кількість використання слова у тексті, поділену на кількість використань усіх ключових слів. Можна використовувати й інші формули — частоту відносно усього текста тощо, усі вони будуть точніше ніж проста кількість, але щоб підібрати найкращу доцільно проводити порівняння на великих наборах даних.

В цілому усі розглянуті варіанти мають спільні характеристики — пара з ключового слова та класу УДК, деякі підходи також додають вагу до кожної пари для обмеження кількості рекомендованих класів — можна обирати N класів із найбільшою вагою, або усі класи, вага яких перевищує деякий поріг — який з цих підходів є кращім теж доцільно перевіряти на практиці.

Таку модель можна зберігати у вигляді таблиці із цих трьох (у деяких випдках двох значень). Завдяки обраному відносно універсальному формату (де)серіалізації, натренована модель може бути використана у інших програмах, написаних на різних мовах програмування.

Для додатку, розроблюваного у даній роботі, було вирішено додати для кожного запису вагу, а саме — кількість використань ключового слова в усіх текстах із наведеним класом, які були використані під час тренування.

3.2.3 Алгоритм перетворення ключових слів на класи УДК

Ця секція пропонує початковий алгоритм, який може буде покращений, або заміненний при наявності необхідного часу.

Першим кроком є витягування ключових слів з тексту — цьому кроку буде приділено окремий розділ цієї роботи.

Наступним кроком потрібно залишити лише ті ключові слова, які є у списку, наданому користувачем.

Далі для кожного з класів УДК, наданих користувачем, додамо до моделі запис для кожного з залишившихся ключових слів із цим класом, ключовим словом, та кількістю використань цього слова. Якщо запис із таким класом та ключовим словом вже присутній — просто збільшуємо лічильник використань такого ключового слова.

Цей алгоритм можно представити математичною формулою (1).

$$
\begin{aligned}
F :=& \{\ (k,\ n(k, K)) \mid k \in K,\ K = K_p \cap K_u\ \} \\
M' :=& \{\ (k,\ f) \quad \mid (k, f) \qquad\qquad \forall((k, f') \notin F \wedge (k, f) \in M) \\
& \qquad\qquad\qquad\qquad\quad \vee ((k, f) \in F \wedge (k, f') \notin M) \\
& \qquad , \ (k,\ f_f + f_m) \ \forall(k, f_f) \in F \\
& \qquad\qquad\qquad\qquad\quad \wedge (k, f_m) \in M \\
& \},
\end{aligned}
\tag{1}
$$

де F — ключові слова (та кількість їх використань), які були підібрані програмою та присутні у списку, який надав користувач;

K_p — ключові слова підібрані програмою;

K_u — ключові слова надані користувачем;

$n(k,\ K)$ — скільки разів k зустрічається в K;

M — старе значення моделі (такий самий формат я і в F);

M' — нове значення моделі;

f' — будь-яке значення f (wildcard).

3.2.4 Алгоритм витягування ключових слів

У додатку використано функціонал бібліотеки spaCy, тому ця секція розглядає алгоритм [58] на рівні абстракції [59, 60] з використанням цієї бібліотеки. Огляду реалізації використаного функціоналу буде приділено окрему секцію.

Для витягування ключових слів використано наступні кроки:

1. Виберемо з тексту іменникові групи — для цього використано функціонал бібліотеки SpaCy. Під іменниковою групою мається на-увазі словосполучення, яке описує один об'єкт, наприклад «злий чорний кіт».
2. Виключемо іменникові групи у яких є стоп-слова [31].
3. Видалимо розділови знаки із залишившихся груп.
4. Видалимо усі групи, де довжина тексту не перевищує одного символу.
5. Перетворимо усі літери до нижнього регістру.
6. Підрахуємо частоту використання кожної з груп.
7. Відсортуємо групи за частою їх використання в порядку спадання.
8. Виберемо N найпопулярніших.

Математично це може бути записано такою формулою (2).

$$
\begin{aligned}
keywords := \{\ & x \\
& \mid n \in N \ , \ x = lowercase(n \setminus P) \\
& \qquad , \ x \cap S = \varnothing \\
& \qquad \wedge \ |text(x)| > 1 \\
& \qquad \wedge \ rank(x) <= c \\
\},&
\end{aligned}
\tag{2}
$$

де n — іменникова група;

N — множина іменникових груп;

S — множина стоп слів;

w — слово в іменниковій групі;

P — множина знаків пунктуації;

rank — функція, яка повертає позицію елемента у множині за популярністю у порядку спадання;

c — кількість ключових слів, які потрібно обрати.

3.2.5 Огляд алгоритму spaCy noun_chunks

Бібліотека spaCy має натреновані моделі [61, 62], які зазвичай складаються з:

- токенізатору [63, 64] — цей процесс відповідає за виділення окремих слів з тексту,
- розмічувального алгоритму [65] — цей алгоритм розмічує слова за частинами мови [32],
- лемаматизатору [66] — цей компонент відповідає за лемматизацію [67] — процесс приведення слова до лемми [68] — нормальної форми слова,
- парсеру [69] — ця частина алгоритму розпізнає синтаксичні залежності між словами.
- розпізнавачу іменованих сутностей [33, 70]

Цю послідовність можно представити графічно (рис. 9). Токенайзер завжди виконується першим тому що усі послідуючі крокі працюють з окремими словами. Інші компоненти зазвичай ніяк не залежать один від одного, тому порядок їх виконання не є важливим. Втім, бібліотека підтримує і сторонні моделі, які можуть мати додаткові кроки, і вже вони можуть мати фіксований порядок виконання відносно один одного та стандартних кроків.

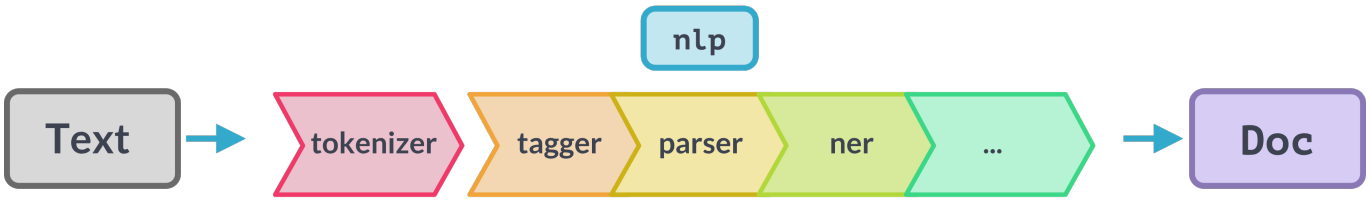

Рисунок 9 — Послідовність розбірки тексту моделями бібліотеки spaCy

Після обробки стандартним алгоритмом над документом можна проводити додаткові трансформації, однією з них є функція noun_chunks [71] — вона повертає усі іменникові групи [72] в тексті.

Функція noun_chunks працює наступним чином [73]:

1. Залишаємо лише ті слова, які помічені як іменники, займенники або власні ім'я — цю інформацію надає розмічувач частин мови.
2. Обходимо залишені слова:
   (a) Якщо слово вже було використано в групі – пропускаємо.
   (b) Якщо слово помічено як залежне з однією з наступних залежностей – додаємо це, та наступне слово як іменникову групу. Ці залежності проставляє парсер [74]:
      i. "oprd" (Object Predicate) [75] — присудок,
      ii. "nsubj" (Nominal Subject) [76] — називний підмет,
      iii. "dobj" (Direct Object) [77] — прямий додаток,
      iv. "nsubjpass" (Passive Nominal Subject) [78] — називний підмет в орудному відмінку,
      v. "pcomp" (Prepositional Complement) [79] — прийменникове доповнення,
      vi. "pobj" (Prepositional Object) — прийменниковий додаток,

vii. ”dative” (Dative) [80] — давальний відмінок,

viii. ”appos” (Appositional Modifier) [81] — прикладка,

ix. ”attr” (Attribute) [82] — означення,

x. ”ROOT” [83] — корінь речення, зазвичай головний присудок або дієслово речення.

(c) Якщо слово помічено залежністю «conj» (сполучення) [84, 85] (наприклад «та», «або»), додаємо це, та інші слова які помічені як залежні в цій групі.

3.3 Порівняння списків класів УДК

Для скорочення часу розробки було вирішено спростити функціонал додатку, та замість шифрів УДК приймати класи УДК. Крім скорочення часу розробки, це також дозволяє використовувати програму із будь-якими системами класифікації, що у свою чергу є розширенням функціоналу.

Через те що ключові слова є множиною, вирішено використати простий спосіб порівняння — коефіцієнт Жаккара [86]. Ця формула порівнює схожість двух скінченних множин та представляє собою частку від ділення розміру перетину [87] на розмір об’єднання [88] двох множин. Результатом є число від нуля до одиніці, нуль зустрічається у тих випадках коли множини не мають жодного спільного єлемента, а одиниця означає що множини однакові. Таким чином програма буде відповідати значеннями від нуля до одиниці, і чим вище значення тим вище співпадіння між класами УДК, які обрав користувач та тими, що пропонує програма.

Такий функціонал дозволить не порівнювати обидва списки вручну, а просто подивитися на результат порівняння, і якщо похибка не є вагомою, на думку користувача, він може використати будь-який з двох списків або обидва.

Формула виглядає наступним чином:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Висновки до розділу 3

Було розроблено користувацький інтерфейс, розроблено алгоритм перетворення ключових слів на класи УДК. Також було порівняно різні методи серіалізації даних, та було обраний найкращий для цільової проблеми. Розроблено алгоритм витягування ключових слів з тексту за домопомогою бібліотеки spaCy. Крім того, розглянуто алгоритм, за яким ця бібліотека разбирає текст що дозволяє нам витягувати з нього ключові слова та виконувати над ним інші трансформації.

4 ТЕСТУВАННЯ

4.1 Тестування користувацького інтерфейсу

Найпростішою частиною программи для тестування є користувацький інтерфейс. По-перше, доцільно описати можливі вхідні дані. При нормальному використанні програма очікує два обов'язкових аргументи — model_filename (путь до файлу моделі) та text_filename (путь до файлу із текстом для класифікації).

Крім того програма може приймати три опціональних параметри:

- training — цей параметр означає що програма має відпрацювати у режимі тренування, за цим параметром надається путь файлу для збереження нової моделі.
- udc — цей параметр приймає класи, які можуть бути використані для порівняння із класами підібраними програмою або для тренування моделі, в залежності від обраного режиму роботи.
- keywords — за цим параметром передається набір ключових слів для використання під час тренування моделі.

Слід додати що для кожного з режимів необхідний конкретний набір опціональних аргументів, а саме:

- для тренування — training, udc та keywords,
- для припущення та порівняння — udc,
- для припущення не потрібен жоден з опціональних аргументів.

Також є додаткові аргументи '-h' та '--help' — обидва виводять на екран інформацію про програму. Якщо програму запущено із некоректними аргументами – виводится рядок, в якому написані можливі аргументи, та рядок із помилкою.

Для тестування пропонується розробити таблицю, де рядками будуть тести, а стовпцями будуть аргументи, значення кліток — присутність або відсутність зазна-

ченого аргумента у зазначеному тесті. Також слід додати один стовпець для неочікуваного аргументу. Слід зазначити що усі випадки де аргументи присутній, будуть використовувати коректне значення для цього аргументу. Такий спосіб тестування можна вважати таким, що належить до методу сірої скриньки — усі ці тести тестують тільки один з модулів — ця інформація доступна лише розробнику, але ми не виключаємо комбінації які належать до одних й тих самих класів еквівалентності.

Насправді через кількість можливіх комбінацій ($2^6 = 64$), доцільно виключити деякі тести, а навіть групи.

По перше з документації бібліотеки argparse нам відомо що при вказаному аргументі -h або --help завжди буде виведено інформаційне повідомлення, незалежно від усіх інших параметрів, тому доцільно залишити один тест де усі парамети крім цього відсутні, а в усіх інших тестах не вказувати цей параметр, таким чином ми залишаємо лише ($2^5 + 1 = 33$) тестів, що майже вдвічи менше.

Також, нам відомо що при виявлені неочікуваного параметру буде показане повідомлення про помилку незалежно від інших аргументів. Тому доцільно залишити один тест із коректними неопціональними параметрами та вказаним неочікуваним аргументом, в усіх інших випадках неочікуваний параметр вказувати не будемо. Таким чином ми залишаємо тільки ($2^4 + 2 = 18$) тестів, це знову майже вдвічи меньше.

До того ж, ми знаємо що відсутність будь-якого з обов'язкових параметрів призведе до помилки, тому немає сенсу тестувати випадки де один з них відсутній, а інший — ні. Загалом, достатньо залишити лише один тест, де відсутні ці та будь-які інші параметри, а в усіх інших вказувати обов'язкові параметри. Завдяки цьому ми знову зменшуємо кількість тестів — до ($2^3 + 3 = 11$). Таким чином ми виділили велику групу тестів з однаковими результатами та замінили їх на лише 3. Крім того, пропущені тести насправді перевіряли б бібліотечний код, що не має сенсу тому що він протестований розробниками. Решта тестів перевіряє вже функціонал розроблений спеціально для додатку.

#	--training	--udc	--keywords	очікування
1	тільки -h			повне інформаційне повідомлення
2	text_filename, model_filename та неочікуваний параметр			короткe інформаційне повідомлення та повідомлення про помилку
3	text_filename та model_filename відсутні			
4	-	-	+	
5	-	+	+	
6	+	-	-	
7	+	-	+	
8	+	+	-	
9	-	-	-	припущення класів
10	-	+	-	припущення та порівняння
11	+	+	+	тренування

Таблиця 4 — Тести комбінацій параметрів (присутність та відсутність)

Через представлення груп тестів лише одним варіантом з кожної, можна сказати що ці тести були розроблені із використанням классів еквівалентності [89] — завдяки тому що нам відомо як реалізована внутрішня логіка, ми знаємо що деякі значення призведуть до однакового результата незалежно від інших параметрів. Таке використання знань про внутрішню реалізацію говорить про використання методу білої скриньки [90].

При виконанні розроблених тестів отримуємо очікувані результати (рис. 10 та 11).

Рисунок 10 — Скриншот виконання тестів із корректною роботою програми (1, 9, 10, 11 у таблиці)

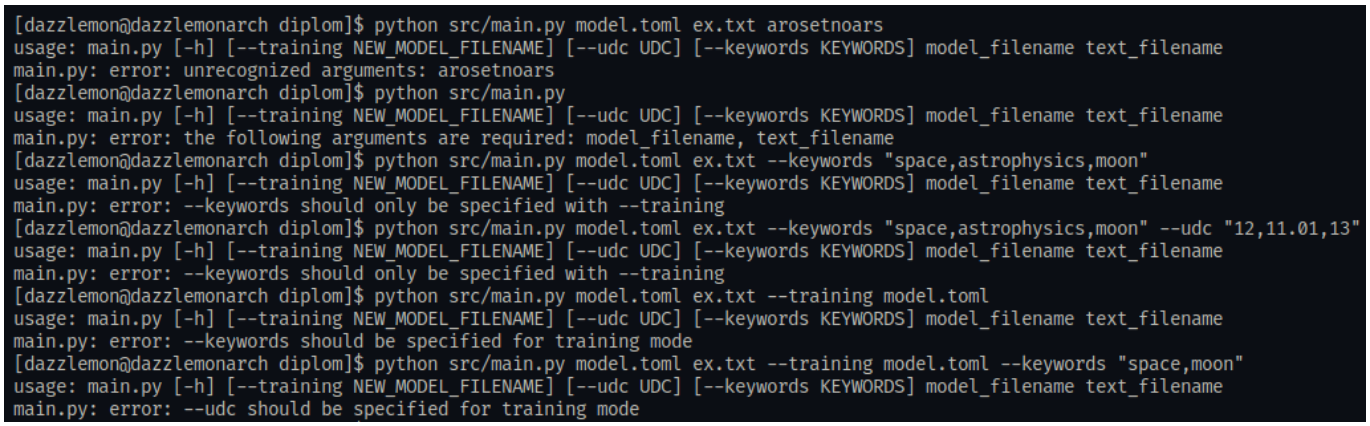

Рисунок 11 — Скриншот виконання тестів із некорректними вхідними даними (2, 3, 4, 5, 6, 7 та 8 у таблиці)

Також можна додати окремий тест щоб перевірити випадок, коли файл не є доступним, таке може статися на декількох єтапах, але результат завжди буде одна-

ковим, тому достатньо продемонструвати один раз вказавши некорректне ім’я для обох файлів для режиму припущення. При виконанні отримуємо помилку «Файл не знайдено» (рис. 12).

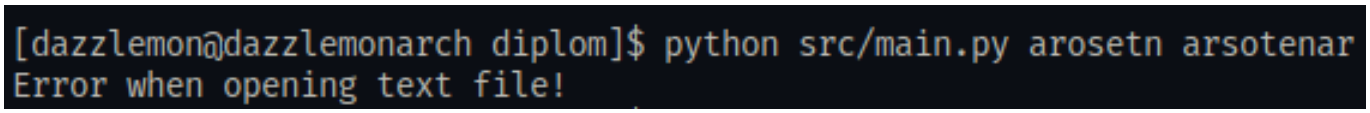

Рисунок 12 — Скриншот виконання тесту із некоректними назвами файлів

Також можна перевірити випадок, коли файл існує, алє його формат некоректний (рис. 13).

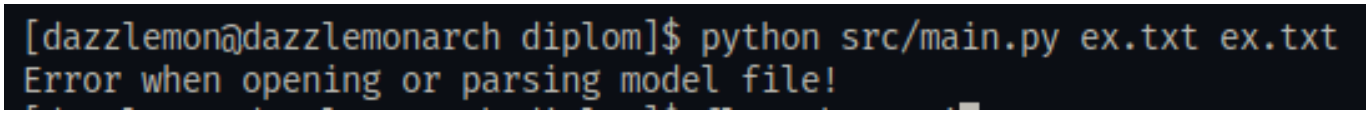

Рисунок 13 — Скриншот виконання тесту із некоректним форматом моделі

Цільовий функціонал розробленого додатку неможливо оцінити бінарно працюючий або непрацюючий, або навіть дискретно. Хоча й дійсно додаток може не працювати, алє процювати ідеально він не може тому що не існує стандарту за яким би можна було перевірити коректність роботи. Натомість можливо порівняти результати які надає розроблена програма із тими, які були підібрані авторами робіт, або бібліотекарями. Натомість, таке порівняння не є можливим у рамках цієї роботи тому що для того щоб зробити вагомі висновкі необхідний достатньо великий набір даних — при маленький вибірці програмі може просто «пощастити» і вона порекомендує дуже точно, також може бути і навпаки, алє з великим набором даних такі похибки не будуть заважати середньому результату.

4.2 Тестування використаних алгоритмів

Під час написання реалізації додатку було протестовано внутрішні алгоритми, а саме: підбору ключових слів (рис. 14). Для отримання цього скриншоту було модіфіковано вихідний кід програми щоб виводити список ключових слів та кількість їх використань одразу після його заповненння.

```
[dazzlemon@dazzlemonarch diplom]$ python src/main.py model.toml ex.txt
mode.model_filename='model.toml'
mode.text_filename='ex.txt'
predict

19 iras 16293b
11 order
9 iras
9 figure
8 lee et al
8 methanol
8 example
7 fact
7 section
6 molecules
6 isotopic ratios
6 methoxymethanol
5 isotopically substituted species
5 isotopologues
5 ethanol
5 table
5 m ol 2 vec
5 hydrogen peroxide
4 isotopic composition
4 methyl formate
4 h2 co
4 magnitude
4 grain surfaces
4 13 ch3 ocho
3 detection
3 space
3 chemistry
3 massachusetts institute
3 technology
3 cambridge
3 acetaldehyde
3 dimethyl ether
3 isotopically substituted molecules
3 molecular formation
3 isotopic information
3 gpr
3 sulfur
3 chloromethane
3 astrochemical study
3 species
3 high abundance
3 fig
3 co
3 methoxyethane
3 ethane
3 cs2
3 ch3 ch2 och2 d
2 rare isotopologues
2 interstellar sources
2 source b

class = 11
weight = 68

class = 12
weight = 68

class = 15
weight = 53

class = 12.01
weight = 15
```

Рисунок 14 — Скриншот виконання тесту із виведенням ключових слів

На вході маємо модель (для тесту неважлий зміст цієї моделі, тільки її наповнення), та текст; на виході очікуємо ключові слова без стоп-слів та розділових знаків, та кількість їх використань.

На наведеному скриншоті можна побачити що загалом підібрані ключові слова відповідають визначенню. Можна також помітити що деякі з підібраних слів не можна вважати ключовими (напр. fig та figure — позначення рисунку, table — позначення таблиці, example — приклад, fact — факт), алє це не є проблемою тому що цей список використовується для отримання перетину із ключовими словами, наданими користувачем при тренуванні, тож вони будуть записані як належні до якогось з класів тільки у тому випадку коли користувач вважає це доцільним.

Для перевірки заповнення моделі під час тренування можна обрати декілька ключових слів з попереднього скриншоту, надати їх від користувача, і в результаті ми очикуємо що попередньо пуста модель буде мати пари з цих ключових слів та наданих класів. Також в цьому ж тесті можна додати ключові слова, яких немає на скриншоті щоб показати що вони ні на що не впливають.

На вході маємо:

- пусту модель;
- текст з попепереднього тесту;
- класи: 11, 12, 11.01;
- ключові слова: space, chemistry, astrophysics, donut, pie, cake.

На виході очікуємо такі записи:

- space, 11, 3;
- space, 12, 3;
- space, 11.01, 3;
- chemistry, 11, 3;

- chemistry, 12, 3;
- chemistry, 11.01, 3.

В результаті (рис. 15) можна побачити що підбираються тільки ті з ключових слів, які присутні в обох списках — тому, що надав користувач і тому, який підібрала програма.

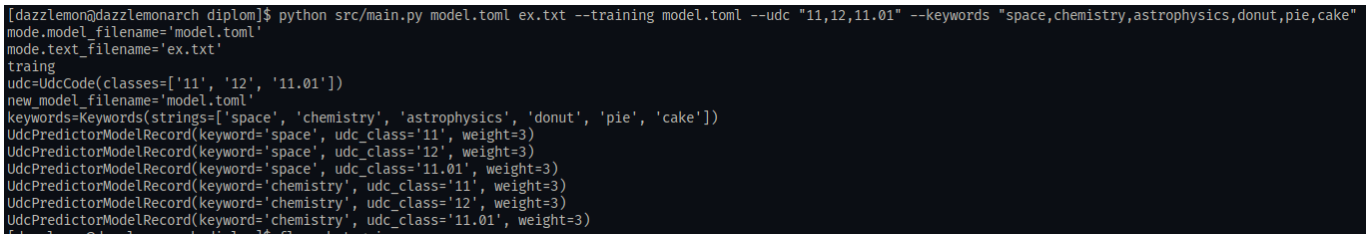

Рисунок 15 — Скриншот демонстрації тренування моделі

Якщо використати той самий список класів в режимі порівняння класів, то отримаємо повне співпадіння (рис. 16 — «similarity (0-1, more is better) = 1.0»).

На вході маємо модель із записами наведеними у результаті попереднього тесту, і класи, які були використані при тренувані, на виході очикуємо повне співпадіння, тобто «similarity = 1».

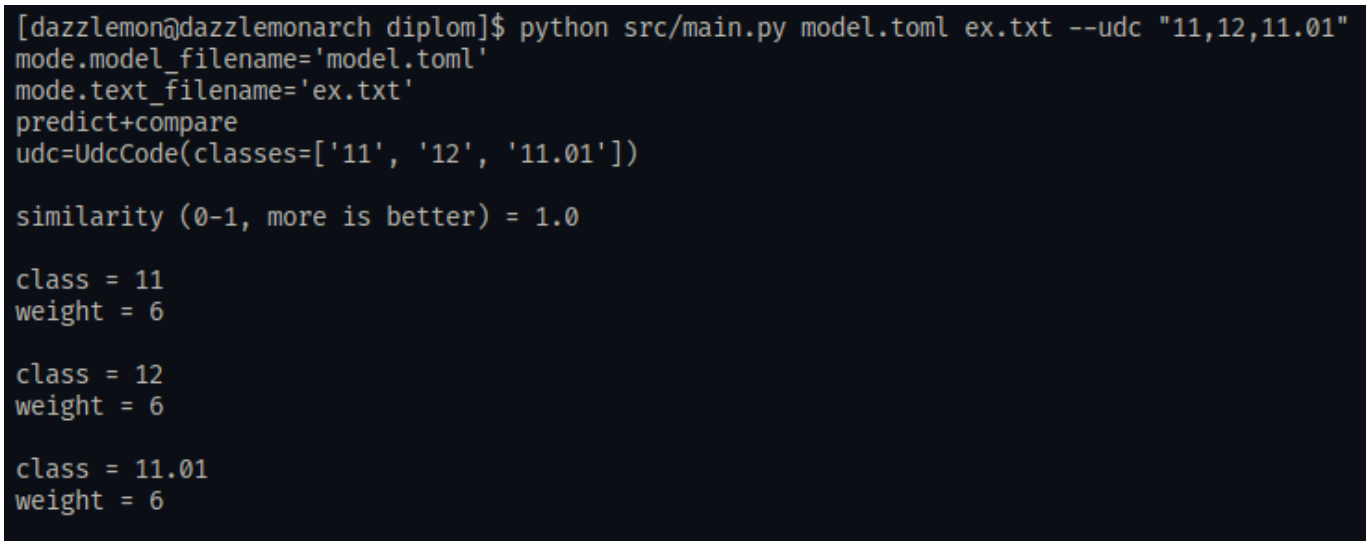

Рисунок 16 — Скриншот режиму порівняння класів

Висновки до розділу 4

В цьому розділі був протестований переважно користувацький інтерфейс. Через велику кількість комбінацій вхідних даних, кількість тестів була зменшена завдяки

розбиттю на еквівалентні класи. Також, був описаний можливий підхід до тестування функціоналу. Усі виконані тести дали очікуваний результат.

5 ПОШИРЕННЯ ДОДАТКУ

5.1 Вибір способу створення виконуваного файлу з вихідного коду на мові Python

Python це здебільшого інтерпретована мова програмування. Насправді некоретно казати що саме мова є інтерпретованою або компільованою, бо мова лише описує синтаксис та семантику, а код може бути і інтерпретованим і компільованим (напр. Dart [91]), алє саме в мові Python офіційний інструмент для запуску програм — інтерпретатор.

Розроблюваний додаток орієнтований не тільки на просунутих користувачів, тому доцільно загорнути додаток в автономний пакет, щоб звільнити користувачів від встановлення інтерпретатора та бібліотек.

Для вирішення цієї задачі можна розглянути декілька підходів:

- PyInstaller [92] — один з головних претендентів на вирішення задачі. На перший погляд, ця python бібліотека повністю підходить під вказані вимоги — вона збирає усі скрипти, усі модулі, усі бібліотеки та інтерпретатор в один комплект, який може бути запущеним кінцевим користувачем без додаткових дій. Втім, є деякі недоліки:
  - це рішення не є кросплатформенним, тобто «зібрати» виконуваний файл для цільової платформи можна тільки на цій платформі
  - деякі платформи зовсім не протестовані

  Хоча це рішення є дуже гарним, доцільно розглянути й інші, перед тим як робити кінцевий вибір.

- Nuitka [93] — python компілятор, так само не є кросплатформеним. Крім того, це рішення менш популярне ніж PyInstaller, та вимагає більшої кількості залежностей. Алє, це рішення може бути корисним, якщо додаток, зібраний з допомогою PyInstaller буде виконуватись дуже повільно.

- py2exe [94] — цей інструмент орієнтований лише на одну операційну систему — Windows, з цієї причини він є гіршим за інші запроновані рішення.

- Cython [95] — самий популярний python компілятор. Хоча він також не має прямої можливості компілювати на інші платформи, він дозволяє компілювати в код на мові C, який у свою чергу можна компілювати на різні цільові платформи з однієї. Незважаючи на ці переваги, такий підхід не виграє по зручності PyInstaller, але стоїть на приблизно тому самому рівні, завдяки перевагам над Nuitka.

- Frozen binaries [96] — функціонально те саме що й PyInstaller.

- Docker [97] — ця система контейнеризації хоча й сама зручна та стандартизована з представлених, вимогає встановлення додаткового ПЗ — через це, це решіння не має сенсу, бо ми намагаємося уникнути встановлення додаткового ПЗ та спростити встановлення розроблюваного додатку.

- Cloud-based deployment [98] — найкраща сумістність з різними платформами, але вимагає додаткових ресурсів, та постійного підключення до інтернету.

Для більш наглядного порівняння можна представити ці дані таблиці (5).

Назва	Кросплатформеність	Тип	Windows	Linux	macOS	FreeBSD/NetBSD
PyInstaller	ні	Пакетувальник	7/8/10/11	+	>10.15	+**
Nuitka	ні	Компілятор	+	+	+	+
py2exe	ні	Пакетувальник	+	-	-	-
Cython	так*	Компілятор	+	+	+	+
Frozen binaries	ні	Пакетувальник	+	+	+	+
Docker	так	Контейнер	+	+	+	+
Cloud-based deployment	так	веб	+	+	+	+

* - можна компілювати не у нативний код, а в код на мові C, а вже цей код можна крос-компілювати.
** - Неофіційна підтримка
Nuitka потребує окремого встановлення компілятора gcc

Таблиця 5 — Порівняння способів пакетування додатків написаних на Python.

З цієї таблиці можна побачити що усі варіанти мають підтримку усіх сучасних версій ОС Windows. Якщо відкинути py2exe, то будь-який з варіантів підтримує усі сучасні/популярні ОС. Через це залишается обирати з варіантів, які дозволяють збирати виконувані файли з під однієї платформи на усі інші. Docker вже було виключено через те що він замінює одну залежніст від додаткового ПЗ на інше. Розгортання веб ресурсу також виключено через додаткові витрати. Залишається лише Cython, який підтримує компіляцію на будь-які цілові платформи з під однієї робочої.

На практиці виявилося що крос-компіляція з Cython хоча й можлива, не є зручною. Головною причиною є погана сумісність компіляторів — python на windows компілюється із допомогою компілятора MSVC, а крос-компіляція з linux на windows проводится із допомогою mingw. Цю проблему можна вирішити перекомпілювавши python із іншим компілятором, алє набагато легше буде використовувати цільові ОС для створення виконуваних файлів. Для цього доцільно використати PyInstaller.

Для створення виконуваного файлу потрібно виконати такі кроки:

1. Перейти на цільову платформу.
2. Встановити python — для перевірки було використано версію 3.10, але скоріш за все підійдуть й інші.
3. Встановити необхідні залежності:

```
> pip install spacy
> python -m spacy download en_core_web_lg
> pip install pyinstaller
```

Listing 6: Встановлення залежностей

4. Зібрати виконуваний файл за допомогою PyInstaller:

```
> pyinstaller ^
    src\main.py ^
    -F ^
```

```
--dist build ^
--collect-all en_core_web_lg
```

Listing 7: Збірка виконуваного файлу

Цей приклад використовувася на ОС Windows, але він буде працювати і на інших платформах. Єдина суттева зміна — символ продовження строки (^), для віндовс це '^', для unix - '\'.

В цілому, цей підхід вирішує проблему додаткових залежностей — на виході маємо лише один виконуваний файл. Але можливі деякі покращення:

- Зараз мовна модель запаковується в виконуваний файл. Для більшої універсальності можна зробити так, щоб модель знаходилась в окремій папці поряд із виконуваним файлом — таким чином користувач за бажанням міг би замінити модель на іншу.
- Можна створити docker-контейнери з необхідними платформами [99, 100, 101] та використовувати їх для збірки виконуваних файлів на різні цільові платформи з під однієї платформи розробки.

Висновки до розділу 5

Через те що додаток орієнтований на простих користувачів, важливо зробити поширення, встановлення та використання програми якомога простішим для кінцевого користувача. Це є реальною проблемою для цього додатку через те що обрані засоби програмування, а саме мова програмування Python, роблять поширення складнішим ніж в можливих альтернативах. Стається це через те що офіційний спосіб використання цієї мови очікує що кінцевий користувач має на своєму пристрої інтерпретатор, а розробник у свою чергу поширює вихідний код програми. Така парадигма не є зручною бо користувачу потрібно окремо встановлювати конкретну версію інтерпретатора, а розробнику доцільно якомога швидше оновлювати програму для підтримки останніх версій інтерпретатора. Крім того, доцільно й підтримувати попередні версії щоб старі користувачі могли не оновлювати інтерпретатор кожного

разу. Тож вцілому це ускладнює і розробку і використання (а саме встановлення) додатку.

Розроблюваний додаток не є першим випадком такої проблеми, тому за історію мови Python було розроблено немало способів для вирішення цієї проблеми. Ці специфічні для мови Python способи, та деякі інші рішення було розглянуто та порівнянно щоб обрати найкращий. Перший обраний варіант виявився складнішим у використанні ніж очикувалося, тому кінцевий вибір було змінено.

В підсумку, підібране рішення дозволяє достатньо зручно поширювати додаток на різні платформи. Крім того, наведено можливі покращення для автоматизації частини роботи для поширення ПЗ.

ЗАГАЛЬНІ ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

В процесі написання роботи було розроблено додаток, який може рекомендувати класи УДК для наданих текстів після тренування. Такий додаток може бути корисним для часткової автоматизації класифікації великих бібліотек — користувачу достатньо класифікувати тільки частину наукових робіт, а для решти можна використовувати рекомендації програми.

Вцілому, розроблений додаток можна вважати гарною основою для подальшого розвитку — завдяки добре розподіленій архітектурі, будь-яка частина може бути дуже просто замінена, що у свою чергу дозволяє продовжувати ітераційні покращення.

Важливо також зазначити що під час розробки для економії часу було вирішено спростити функціонал порівняння шифрів УДК до порівняння списків класів УДК. Через це усі використання шифрів УДК було замінено звичайними списками класів УДК, а через те що класи УДК представлення простими рядками це списки рядків. Завдяки цьому рішенню програма стає більш загальною і дозволяє робити припущення не тільки у вигляді класів УДК, а й у вигляді класів будь-яких систем класифікації за умови використання моделі, натренованій на відповідних даних.

Велика частина очевидних покращень складається з порівняння альтернативних підходів до використаних рішень — через обмежений час, частина рішень була зроблена без достатнього обгрунтування та/або достатнього порівняння альтернатив. Крім того, для порівняння частини альтернатив необхіден великий набір данних, здобуття якого не є практичним або навіть можливим в межах цієї роботи.

Інший путь покращень — користувацький інтерфейс. Розроблений функціонал можна вважати перевіркою доцільності такого додатку. Алє для комфортного користування додатком можна зробити багато покращень. Можна зробити додаток більш модульним — на даний момент усе, крім моделі підбору класів УДК на основі ключових слів, запаковане в виконуваному файлі. Алє використані засоби програмування дозволяють додати більше можливостей для конфігурації, наприклад можна

дозволити використовувати різні мовні моделі — це дозволить по-перше за бажаням використовувати простіші язикові моделі, які займають менше місця. Крім того, це дозволить використовувати додаток із текстами на інших мовах, таким чином цей додаток перетворюється на фреймворк. Також, на основі розробленого інтерфейсу користувацького рядка можна розробити графічний інтерфейс — більшості користувачів буде набагато зручніше використовувати додаток саме таким чином.

З точки зору розробки головним покращенням була б автоматизація поширення додатку, а саме його пакування у виконуваний файл. Це можна вирішити із допомогою контейнеризації цільових платформ та використання їх із додатками безперервної інтеграції.

Також можна додати рекомендацію, яка включає полегшення і розробки, і використання — розгортання додатку у вигляді веб-ресурсу. Це не було зроблено у межах роботи через відсутність фінансування.

Крім того, було б доцільно додати більше стандартів у вигляді вимог до написання коду, бажано з автоматизованими перевірками, це б суттево покращило якість вихідного коду, що в свою чергу б полегшило подальшу розробку та підтримку додатку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] *Natural language processing*. https://en.wikipedia.org/wiki/Natural_language_processing. Звертання 15 червня 2023 р.

[2] *Universal Decimal Classification*. https://en.wikipedia.org/wiki/Universal_Decimal_Classification. Звертання 15 червня 2023 р.

[3] *UDC Structure & Tables*. https://udcc.org/index.php/site/page?view=about_structure. Звертання 15 червня 2023 р.

[4] Matjaž Kragelj та Mirjana Kljajić Borštnar. «Automatic classification of older electronic texts into the Universal Decimal Classification–UDC». В: *Journal of Documentation* 77.3 (2021), с. 755—776. ISSN: 0022-0418. DOI: 10.1108/JD-06-2020-0092. URL: https://www.emerald.com/insight/content/doi/10.1108/JD-06-2020-0092/full/html.

[5] Damir Albertovich Almukhametov та Olga Avenirovna Nevzorova. «Service for assigning a UDC code to mathematical articles based on semantic technologies». В: *Proceedings of the 24th Scientific Conference "Scientific Services & Internet – 2022"*. Січ. 2022. DOI: 10.20948/abrau-2022-28. URL: https://www.researchgate.net/publication/365667926_Service_for_assigning_a_UDC_code_to_mathematical_articles_based_on_semantic_technologies.

[6] Mladen Borovič, Milan Ojstersek та Damjan Strnad. «A Hybrid Approach to Recommending Universal Decimal Classification Codes for Cataloguing in Slovenian Digital Libraries». В: *IEEE Access* (серп. 2022). DOI: 10.1109/ACCESS.2022.3198706. URL: https://www.researchgate.net/publication/362704649_A_hybrid_approach_to_recommending_Universal_Decimal_Classification_codes_for_cataloguing_in_Slovenian_digital_libraries.

[7] Aditi Roy та Saptarshi Ghosh. «Automated Subject Identification using the Universal Decimal Classification: The ANN Approach». В: *SRELS Journal of Information Management* (трав. 2023). DOI: 10.17821/srels/2023/v60i2/170963.

URL: https://www.researchgate.net/publication/370786883_Automated_Subject_Identification _ using _ the _ Universal _ Decimal _ Classification _ The _ ANN _ Approach.

[8] Fiodor Tretyakov та Liya Serebryanaya. «Porter advanced method for the Universal Decimal Classification». В: *Central European Researchers Journal* 2.1 (). URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiy47m_9IP_AhVmlYsKHXukD40QFnoECA0QAQ&url=https%3A%2F%2Fceres-journal.eu%2Fdownload.php%3Ffile%3D2016_01_04.pdf&usg=AOvVaw0BnWLUqo-UGqhJ6_IC_xjl.

[9] *Multiclass Classification*. https://en.wikipedia.org/wiki/Multiclass_classification. Звертання 15 червня 2023 р.

[10] *Statistical Classification*. https://en.wikipedia.org/wiki/Statistical_classification. Звертання 15 червня 2023 р.

[11] *Machine Learning*. https://en.wikipedia.org/wiki/Machine_learning. Звертання 15 червня 2023 р.

[12] *Keyword extraction*. https://en.wikipedia.org/wiki/Keyword_extraction. Звертання 15 червня 2023 р.

[13] *Universal Decimal Classification summary*. https://udcsummary.info/php/index.php. Звертання 15 червня 2023 р.

[14] *Universal Decimal Classification Summary Linked Data*. https://udcdata.info/. Звертання 15 червня 2023 р.

[15] *Dependency inversion principle*. https://en.wikipedia.org/wiki/Dependency_inversion_principle. Звертання 15 червня 2023 р.

[16] *SOLID*. https://en.wikipedia.org/wiki/SOLID. Звертання 15 червня 2023 р.

[17] *Polymorphism (computer science)*. https://en.wikipedia.org/wiki/Polymorphism_(computer_science). Звертання 15 червня 2023 р.

[18] *Subtyping*. https://en.wikipedia.org/wiki/Subtyping. Звертання 15 червня 2023 р.

[19] *Dynamic dispatch*. https://en.wikipedia.org/wiki/Dynamic_dispatch. Звертання 15 червня 2023 р.

[20] *Algebraic data type*. https://en.wikipedia.org/wiki/Algebraic_data_type. Звертання 15 червня 2023 р.

[21] Steve McConnel. *Code Complete*. 2nd ed. 2004. Гл. 34.4 Program into Your Language, Not in It, с. 843. ISBN: 978-0-7356-1967-8.

[22] *Composition over inheritance: road to algebraic data types*. https://developers.mews.com/composition-over-inheritance/. Звертання 15 червня 2023 р.

[23] Steve McConnel. *Code Complete*. 2nd ed. 2004. Гл. 5.3 Design Building Blocks: Heuristics, с. 107. ISBN: 978-0-7356-1967-8.

[24] *Single-responsibility principle*. https://en.wikipedia.org/wiki/Single-responsibility_principle. Звертання 15 червня 2023 р.

[25] *Tuples and Sequences*. https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences. Звертання 15 червня 2023 р.

[26] *Resource acquisition is initialization*. https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization. Звертання 15 червня 2023 р.

[27] *TIOBE Index*. https://www.tiobe.com/tiobe-index/. Звертання 15 червня 2023 р.

[28] *PYPL PopularitY of Programming Language*. https://pypl.github.io/PYPL.html. Звертання 15 червня 2023 р.

[29] *The top programming languages*. https://octoverse.github.com/2022/top-programming-languages. Звертання 15 червня 2023 р.

[30] *Languish - Programming Language Trends*. https://tjpalmer.github.io/languish/. Звертання 15 червня 2023 р.

[31] *Stop word*. https://en.wikipedia.org/wiki/Stop_word. Звертання 15 червня 2023 р.

[32] *Part-of-speech tagging*. https://en.wikipedia.org/wiki/Part-of-speech_tagging. Звертання 15 червня 2023 р.

[33] *Named-entity recognition*. https://en.wikipedia.org/wiki/Named-entity_recognition. Звертання 15 червня 2023 р.

[34] *rake-nltk*. https://csurfer.github.io/rake-nltk/_build/html/index.html. Звертання 15 червня 2023 р.

[35] *yake*. https://pypi.org/project/yake/. Звертання 15 червня 2023 р.

[36] *spaCy*. https://spacy.io/. Звертання 15 червня 2023 р.

[37] *Natural Language Toolkit*. https://www.nltk.org/. Звертання 15 червня 2023 р.

[38] *TextBlob: Simplified Text Processing*. https://textblob.readthedocs.io/en/dev/. Звертання 15 червня 2023 р.

[39] *Gensim: Topic modelling for humans*. https://radimrehurek.com/gensim/. Звертання 15 червня 2023 р.

[40] *Serialization*. https://en.wikipedia.org/wiki/Serialization. Звертання 15 червня 2023 р.

[41] *Comparison of data-serialization formats*. https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats. Звертання 15 червня 2023 р.

[42] *A Comparison Of Serialization Formats*. https://blog.mbedded.ninja/programming/serialization-formats/a-comparison-of-serialization-formats/. Звертання 15 червня 2023 р.

[43] *mbedded.ninja*. https://blog.mbedded.ninja/. Звертання 15 червня 2023 р.

[44] *Interoperability*. https://en.wikipedia.org/wiki/Interoperability. Звертання 15 червня 2023 р.

[45] *Comma-separated values*. https://en.wikipedia.org/wiki/Comma-separated_values. Звертання 15 червня 2023 р.

[46] *Protocol Buffers*. https://en.wikipedia.org/wiki/Protocol_Buffers. Звертання 15 червня 2023 р.

[47] *JSON*. https://en.wikipedia.org/wiki/JSON. Звертання 15 червня 2023 р.

[48] *XML*. https://en.wikipedia.org/wiki/XML. Звертання 15 червня 2023 р.

[49] *TOML*. https://en.wikipedia.org/wiki/TOML. Звертання 15 червня 2023 р.

[50] *YAML*. https://en.wikipedia.org/wiki/YAML. Звертання 15 червня 2023 р.

[51] Steve McConnel. *Code Complete*. 2nd ed. 2004. Гл. 5.2 Key Design Concepts, с. 78. ISBN: 978-0-7356-1967-8.

[52] *json2toml*. https://github.com/KenanY/json2toml. Звертання 15 червня 2023 р.

[53] *xmltodict*. https://github.com/martinblech/xmltodict. Звертання 15 червня 2023 р.

[54] *x2js*. https://github.com/abdolence/x2js. Звертання 15 червня 2023 р.

[55] *Sampling (statistics)*. https://en.wikipedia.org/wiki/Sampling_(statistics). Звертання 15 червня 2023 р.

[56] *Sampling methods*. https://en.wikipedia.org/wiki/Sampling_(statistics)#Sampling_methods. Звертання 15 червня 2023 р.

[57] *Weight function*. https://en.wikipedia.org/wiki/Weight_function. Звертання 15 червня 2023 р.

[58] *Algorithm*. https://en.wikipedia.org/wiki/Algorithm. Звертання 15 червня 2023 р.

[59] *Abstraction (computer science)*. https://en.wikipedia.org/wiki/Abstraction_(computer_science). Звертання 15 червня 2023 р.

[60] *Abstraction layer*. https://en.wikipedia.org/wiki/Abstraction_layer. Звертання 15 червня 2023 р.

[61] *Trained Models & Pipelines*. https://spacy.io/models. Звертання 15 червня 2023 р.

[62] *Language Processing Pipelines*. https://spacy.io/usage/processing-pipelines/. Звертання 15 червня 2023 р.

[63] *Tokenization*. https://en.wikipedia.org/wiki/Lexical_analysis#Tokenization. Звертання 15 червня 2023 р.

[64] *Tokenizer*. https://spacy.io/api/tokenizer. Звертання 15 червня 2023 р.

[65] *Tagger*. https://spacy.io/api/tagger. Звертання 15 червня 2023 р.

[66] *Lemmatizer*. https://spacy.io/api/lemmatizer. Звертання 15 червня 2023 р.

[67] *Lemmatisation*. https://en.wikipedia.org/wiki/Lemmatisation. Звертання 15 червня 2023 р.

[68] *Lemma (morphology)*. https://en.wikipedia.org/wiki/Lemma_(morphology). Звертання 15 червня 2023 р.

[69] *DependencyParser*. https://spacy.io/api/dependencyparser. Звертання 15 червня 2023 р.

[70] *EntityRecognizer*. https://spacy.io/api/entityrecognizer. Звертання 15 червня 2023 р.

[71] *Doc.noun_chunks*. https://spacy.io/api/doc#noun_chunks. Звертання 15 червня 2023 р.

[72] *Noun phrase*. https://en.wikipedia.org/wiki/Noun_phrase. Звертання 15 червня 2023 р.

[73] *spaCy/spacy/lang/en/syntax_iterators.py*. https://github.com/explosion/spaCy/blob/master/spacy/lang/en/syntax_iterators.py. Звертання 15 червня 2023 р.

[74] *en_core_web_lg Label Scheme*. https://spacy.io/models/en#en_core_web_lg-labels. Звертання 15 червня 2023 р.

[75] *Predicate (grammar)*. https://en.wikipedia.org/wiki/Predicate_(grammar). Звертання 15 червня 2023 р.

[76] *nsubj*. https://universaldependencies.org/u/dep/nsubj.html. Звертання 15 червня 2023 р.

[77] *direct object*. https://en.wiktionary.org/wiki/direct_object. Звертання 15 червня 2023 р.

[78] *nsubj:pass*. https://universaldependencies.org/u/dep/nsubj-pass.html. Звертання 15 червня 2023 р.

[79] *Prepositional phrases*. https://en.wikipedia.org/wiki/Adpositional_phrase#Prepositional_phrases. Звертання 15 червня 2023 р.

[80] *Dative case*. https://en.wikipedia.org/wiki/Dative_case. Звертання 15 червня 2023 р.

[81] *appos*. https://universaldependencies.org/u/dep/appos.html. Звертання 15 червня 2023 р.

[82] *Grammatical modifier*. https://en.wikipedia.org/wiki/Grammatical_modifier. Звертання 15 червня 2023 р.

[83] *root*. https://universaldependencies.org/u/dep/root.html. Звертання 15 червня 2023 р.

[84] *Conjunction (grammar)*. https://en.wikipedia.org/wiki/Conjunction_(grammar). Звертання 15 червня 2023 р.

[85] *conj*. https://universaldependencies.org/u/dep/conj.html. Звертання 15 червня 2023 р.

[86] *Jaccard index*. https://en.wikipedia.org/wiki/Jaccard_index. Звертання 15 червня 2023 р.

[87] *Intersection (set theory)*. https://en.wikipedia.org/wiki/Intersection_(set_theory). Звертання 15 червня 2023 р.

[88] *Union (set theory)*. https://en.wikipedia.org/wiki/Union_(set_theory). Звертання 15 червня 2023 р.

[89] *Equivalence partitioning*. https : / / en . wikipedia . org / wiki / Equivalence _ partitioning. Звертання 15 червня 2023 р.

[90] *White-box testing*. https://en.wikipedia.org/wiki/White-box_testing. Звертання 15 червня 2023 р.

[91] *Dart programming language*. https://dart.dev/. Звертання 15 червня 2023 р.

[92] *PyInstaller*. https://pyinstaller.org. Звертання 15 червня 2023 р.

[93] *Nuitka*. https://nuitka.net/. Звертання 15 червня 2023 р.

[94] *py2exe*. http://www.py2exe.org/. Звертання 15 червня 2023 р.

[95] *Cython*. https://cython.org. Звертання 15 червня 2023 р.

[96] *cpython/Tools/freeze*. https://github.com/python/cpython/tree/main/Tools/freeze. Звертання 15 червня 2023 р.

[97] *Docker*. https://www.docker.com/. Звертання 15 червня 2023 р.

[98] *Cloud computing*. https://en.wikipedia.org/wiki/Cloud_computing. Звертання 15 червня 2023 р.

[99] *Docker Windows Image*. https://hub.docker.com/_/microsoft-windows. Звертання 15 червня 2023 р.

[100] *Docker Windows base OS images*. https://hub.docker.com/_/microsoft-windows-base-os-images. Звертання 15 червня 2023 р.

[101] *Docker-OSX*. https://github.com/sickcodes/Docker-OSX. Звертання 15 червня 2023 р.

ДОДАТКИ

”udc_predictor_text_input.py”

```
"""
UdcPredictorTextInput and everything related to it
"""
from pathlib import Path


UdcPredictorTextInput = str


def read_text(filename: str) -> UdcPredictorTextInput:
    """
    Given filename read that file
    and return its contents parsed as UdcPredictorTextInput.
    """
    # I don't know the encoding beforehand, and it should be ok anyways
    # pylint: disable-next=unspecified-encoding
    return Path(filename).read_text()
```

”udc_predictor_model.py”

```
"""
UdcPredictorModel and everything related to it
"""
from dataclasses import dataclass
from pathlib import Path
from typing import List, Dict
import tomlkit
from functional import seq
from udc_code import UdcClass
```

```
@dataclass
class UdcPredictorModelRecord:
    """
    Internal representation for model record.
    """
    keyword: str
    udc_class: UdcClass
    weight: int


    @classmethod
    def from_dict(cls, the_dict: Dict[str, str]) -> 'UdcPredictorModelRecord':
        """
        Parse udc predictor model record from dict.
        """
        return cls(
            the_dict['keyword'],
            the_dict['udc_class'],
            the_dict['weight']
        )


    def to_dict(self) -> Dict[str, str]:
        return {
            'keyword': self.keyword,
            'udc_class': self.udc_class,
            'weight': self.weight,
        }
```

```
@dataclass
class UdcPredictorModel:
    """
    Internal representation for the udc predictor model.
    """
    records: List[UdcPredictorModelRecord]


    @classmethod
    def from_dict(cls, records: List[Dict[str, str]]) -> 'UdcPredictorModel':
        """
        Parse udc predictor model from dict.
        """
        return cls(seq(records).map(UdcPredictorModelRecord.from_dict))


    def to_dict(self) -> List[Dict[str, str]]:
        return seq(self.records).map(lambda record: record.to_dict()).to_list()


def read_model(filename: str) -> UdcPredictorModel:
    """
    Given filename read that file
    and return its contents parsed as UdcPredictorModel.
    """
    return UdcPredictorModel.from_dict(
        # I don't know the encoding beforehand, and it should be ok anyways
```

```
        # pylint: disable-next=unspecified-encoding
        tomlkit.loads(Path(filename).read_text())['record']
    )


def write_model(filename: str, model: UdcPredictorModel):
    """
    Write `model` to `filename`
    """
    # pylint: disable-next=unspecified-encoding
    Path(filename).write_text(tomlkit.dumps({'record': model.to_dict()}))
```

”mode.py”

```
"""Data classes for app modes"""
from abc import ABC
from dataclasses import dataclass
from udc_code import UdcCode
from keywords import Keywords

@dataclass
class Mode(ABC):
    """Base class for app modes"""
    model_filename: str
    text_filename: str

@dataclass
class PredictionMode(Mode):
    """Prediction mode"""
```

```
@dataclass
class UDCComparisonMode(Mode):
    """Prediction and udc comparison mode"""
    udc: UdcCode


@dataclass
class TrainingMode(Mode):
    """Training mode"""
    new_model_filename: str
    udc: UdcCode
    keywords: Keywords
```

”keywords.py”

```
"""Internal representation for keywords"""
from dataclasses import dataclass
from typing import List

Keyword = str


@dataclass
class Keywords:
    """
This class is responsible for parsing keywords from string.
"""
    strings: List[str]


    @classmethod
    def from_string(cls, string: str) -> 'Keywords':
        """
```

```
        Parses an object of this class from string.
        """
        return cls(string.split(','))
```

”udc_predictor.py”

```
"""
Core logic of the application
"""

from typing import List, Tuple
import spacy
from functional import seq
from udc_predictor_text_input import UdcPredictorTextInput
from udc_predictor_model import UdcPredictorModel, UdcPredictorModelRecord
from udc_code import UdcCode
from keywords import Keywords

def extract_keywords(
    text: UdcPredictorTextInput, top_k=50
) -> List[Tuple[str, int]]:
    """
    Extract keywords from text (scientific work)
    """
    nlp = spacy.load("en_core_web_lg")
    doc = nlp(text)

    return seq(doc.noun_chunks)\
        .filter(lambda chunk: not contains_stop_words(chunk))\
        .map(remove_chunk_punctuation)\
```

```
        .map(strip_punctuation)\
        .filter(lambda chunk_text: len(chunk_text) > 1)\
        .count_by_value()\
        .sorted(lambda x: x[1], True)\
        .take(top_k)


def remove_chunk_punctuation(chunk):
    """Remove punctuation marks from the chunk"""
    return ' '\
        .join(seq(chunk)\
            .filter(lambda token: not token.is_punct)\
            .map(lambda token: token.text)\
        )\
        .strip()\
        .lower()


def strip_punctuation(noun_text):
    """Strips noun text from possible punctuation"""
    return noun_text.rstrip('.')


def contains_stop_words(chunk):
    """Whether chunk has any stopwords"""
    return seq(chunk).exists(lambda token: token.is_stop)


def predict(text: UdcPredictorTextInput, model: UdcPredictorModel) -> UdcCode:
```

```
    """
    predict UdcCode for `text` using `model`.
    """
    classes = []
    keywords = extract_keywords(text)

    for keyword, f in keywords:
        for record in model.records:
            if record.keyword == keyword:
                classes.append((record.udc_class, record.weight))

    return seq(classes)\
        .reduce_by_key(lambda a, b: a + b)\
        .sorted(lambda x: x[1], True)\
        .to_list()


def train(
    text: UdcPredictorTextInput,
    model: UdcPredictorModel,
    udc: UdcCode,
    keywords: Keywords,
) -> UdcPredictorModel:
    """
    Build upon `model` using `text`, `udc`, and `keywords`.
    """
    new_model_records = seq(extract_keywords(text))\
        .where(lambda pair: pair[0] in keywords.strings)\
        .flat_map(lambda pair: seq(udc.classes)\
```

```
            .map(lambda cl: UdcPredictorModelRecord(pair[0], cl, pair[1]))
        )\
        .to_list()

    for old_record in model.records:
        for i, new_record in enumerate(new_model_records):
            if new_record.keyword == old_record.keyword \
                    and new_record.udc_class == old_record.udc_class:
                new_model_records[i] = UdcPredictorModelRecord(
                    new_record.keyword,
                    new_record.udc_class,
                    new_record.weight + old_record.weight
                )
                break
        else:
            new_model_records.append(old_record)

    for i in new_model_records:
        print(i)

    return UdcPredictorModel(new_model_records)
```

”udc_code.py”

```
"""UDC code representation"""
from dataclasses import dataclass
from typing import List

UdcClass = str
```

```
@dataclass
class UdcCode:
    """UDC code representation"""
    classes: List[UdcClass]



    @classmethod
    def from_string(cls, string: str) -> 'UdcCode':
        """
        Parses a string and returns an object of this class.

        :param string: Is expected to be the input of `__str__`
        """
        return cls(string.split(','))



    # TODO:
# override is supported in python 3.12
    # @override
    def __str__(self) -> str:
        return ','.join(self.classes)
```

”main.py”

```
"""UDC classes predictor for scientific works in english"""
import sys
from parse_args import parse_args
from udc_predictor_text_input import read_text
from udc_predictor_model import read_model, write_model
from udc_predictor import predict, train
```

```
from mode import PredictionMode, UDCComparisonMode, TrainingMode

def main():
    """main duh"""
    mode = parse_args()

    try:
        text = read_text(mode.text_filename)
    except Exception:
        print('Error when opening text file!')
        sys.exit(-1)

    try:
        model = read_model(mode.model_filename)
    except Exception:
        print('Error when opening or parsing model file!')
        sys.exit(-1)

    print(f'{mode.model_filename=}')
    print(f'{mode.text_filename=}')

    match mode:
        case PredictionMode(_, _):
            print('predict')
            print('')

            print_keywords(predict(text, model))
        case UDCComparisonMode(_, _, udc):
            print('predict+compare')
```

```
            print(f'{udc=}')
            print('')

            prediction = predict(text, model)
            just_classes = [cl for cl, _ in prediction]
            print(f'similarity (0-1, more is better) = '
                f'{jaccard(set(udc.classes), set(just_classes))}'
            )
            print('')

            print_keywords(prediction)
        case TrainingMode(
            _,
            _,
            new_model_filename,
            udc,
            keywords
        ):
            print('traing')
            print(f'{udc=}')
            print(f'{new_model_filename=}')
            print(f'{keywords=}')
            new_model = train(text, model, udc, keywords)
            write_model(new_model_filename, new_model)


def print_keywords(keywords_and_freqs):
    """
    Prints keywords and frequencies in pairs formatted for human understanding.
```

```
    """
    for the_class, count in keywords_and_freqs:
        print(f'class = {the_class}')
        print(f'weight = {count}')
        print('')


def jaccard(set_a, set_b):
    """Jaccard index"""
    intersection = set_a & set_b
    union = set_a.union(set_b)
    return len(intersection) / len(union)


if __name__ == '__main__':
    main()
```

”parse_args.py”

```
"""CLI tools for UDC predictor"""
import argparse
from mode import Mode, TrainingMode, PredictionMode, UDCComparisonMode
from udc_code import UdcCode
from keywords import Keywords

def parse_args() -> Mode:
    """App mode from CLI args"""
    parser = argparse.ArgumentParser(description='UDC Classifier')
    parser.add_argument('model_filename', help='the name of the model file')
    parser.add_argument('text_filename', help='the name of the input text file')
```

```
parser.add_argument(
    '--training',
    dest='new_model_filename',
    help='If this parameter is specified, the model will be trained. '
        'You should also specify --keywords and --udc along with it.',
)
parser.add_argument(
    '--udc',
    help='the input UDC code to compare against the predicted classes, '
        'or to be used for trainging (see --training)',
)
parser.add_argument(
    '--keywords',
    help='the input keywords to be used for training '
        '(string of words delimeted by a comma). '
        'You should also specify --training and --udc along with it.',
)
args = parser.parse_args()

arguments = {
    "model_filename": args.model_filename,
    "text_filename": args.text_filename
}

if args.new_model_filename:
    if args.keywords is None:
        parser.error('--keywords should be specified for training mode')
    if args.udc is None:
        parser.error('--udc should be specified for training mode')
```

```
    arguments["new_model_filename"] = args.new_model_filename
    arguments["udc"] = UdcCode.from_string(args.udc)
    arguments["keywords"] = Keywords.from_string(args.keywords)
    mode = TrainingMode
elif args.udc:
    if args.keywords is not None:
        parser.error('--keywords should only be specified with --training')
    arguments["udc"] = UdcCode.from_string(args.udc)
    mode = UDCComparisonMode
else:
    if args.keywords is not None:
        parser.error('--keywords should only be specified with --training')
    mode = PredictionMode

return mode(**arguments)
```