

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**  
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**  
**імені ІГОРЯ СІКОРСЬКОГО»**  
**Теплоенергетичний факультет**

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

Наталія АУШЕВА

«\_\_\_» \_\_\_\_\_ 2022 р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

**спеціальності 122 «Комп'ютерні науки»**

**освітня програма «Комп'ютерний моніторинг та геометричне**

**моделювання процесів і систем»**

**на тему: «Класифікація наукових текстових масивів за  
параметрами»**

Виконав:

студент ІV курсу, групи ТР-82

Войтко Анатолій Сергійович \_\_\_\_\_

Керівник:

Доцент, к.т.н.

Кузьмініх Валерій Олександрович \_\_\_\_\_

Рецензент:

Директор, НВП «Символ», к.т.н.

Сенченко В'ячеслав Родіонович \_\_\_\_\_

Засвідчую, що у цій дипломній роботі  
немає запозичень з праць інших авторів  
без відповідних посилань.

Студент \_\_\_\_\_

Київ – 2022 року

**Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Наталія АУШЕВА

(підпис)

” \_\_\_ ” \_\_\_\_\_ 2022р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

Войтку Анатолію Сергійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Класифікація наукових текстових масивів за параметрами

керівник роботи доцент, к.т.н. Кузьмініх Валерій Олександрович

(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2022р.

№ \_\_\_\_\_

2. Строк подання студентом роботи ”10” травня 2022р

3. Вихідні дані до роботи персональний комп’ютер під управлінням операційної системи Windows 10/11, сервер під управлінням операційної системи на базі Linux, мова програмування Python з використанням фреймворку PyQt, середовище розробки PyCharm, інструментарій для управління ізольованими Linux-контейнерами Docker, контейнер системи управління базами даних MariaDB, контейнер системи обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP RabbitMQ

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити автоматизовану систему для класифікації наукової інформації у реальному часі для подальшої оцінки рівня наукової діяльності та каталогізації й архівації даних; дослідити та обрати необхідні для реалізації програмні компоненти; дослідити існуючі аналоги системи

5. Перелік ілюстративного матеріалу відображений ілюстрації архітектури системи, діаграма роботи компонентів системи, структурна схема відношень між таблицями бази даних, діаграма класів системи, ілюстрації роботи систем

7. Дата видачі завдання "10" вересня 2022р.

### КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.22р.	
2.	Вивчення та аналіз задачі	13.10.21р.- 01.12.21р.	
3.	Розробка архітектури та загальної	02.12.21р.- 25.01.22р.	
4.	Розробка структур окремих підсистем	26.01.22р.- 21.04.22р.	
5.	Програмна реалізація системи	22.04.22р.- 15.05.22р.	
6.	Оформлення пояснювальної записки	16.05.22р.- 26.05.22р.	
7.	Захист програмного продукту	27.05.22р.	
8.	Передзахист	07.06.22р.	
9.	Захист	20.06.22р.	

Студент \_\_\_\_\_ Войтко А.С.  
(підпис) (прізвище та ініціали.)

Керівник роботи \_\_\_\_\_ Кузьмініч В.О.  
(підпис) (прізвище та ініціали.)

# АНОТАЦІЯ

**Структура та обсяг роботи.** Робота містить 163 сторінки, 34 рисунки, 24 літературних джерел, 1 додаток.

Дана дипломна робота присвячена розробці комплексної програмної системи для збору та збереження необхідних наукових даних для навчання, їх обробці та конвертації у необхідний формат для навчання, завантаження та навчання нейронних мереж, а також їх подальше використання у результуючій системі по класифікації наукових текстів у реальному часі. Метою розробленої системи є подальше оцінювання рівня наукової діяльності та каталогізація й архівація даних. Задачею роботи є обробка та подальше використання для навчання даних з наукових бібліографічних джерел і подальше присвоєння категорій необхідним текстовим масивам з можливістю статистично оцінити якість результату.

Програмний продукт створений на високорівневій мові програмування Python у середовищі PyCharm. Для користувацького графічного інтерфейсу була застосована бібліотека PyQt5, для побудови серверної частини був використаний інструментарій для управління ізольованими Linux-контейнерами Docker, система керування базами даних MariaDB для збереження даних, система адміністрування бази даних Adminer, система обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP RabbitMQ. Для побудови, навчання та використання нейронних мереж була застосована бібліотека TensorFlow та Keras.

**Ключові слова:** парсинг, лематизація, класифікація, статистичний аналіз, бібліографічні джерела інформації, інтелектуальний аналіз інформації.

# ABSTRACT

**Structure and scope of work.** The work contains 163 pages, 34 drawings, 24 literature sources, 1 appendice.

This thesis is devoted to the development of a comprehensive software system for collecting and storing the necessary scientific data for learning, processing and converting them into the necessary format for learning, downloading and learning neural networks, and their further use in the resulting real-time classification system. The purpose of the developed system is further assessment of the level of scientific activity and cataloging and archiving of data. The task of the work is to process and further use for learning data from scientific bibliographic sources and further assign categories to the necessary text arrays with the ability to statistically assess the quality of the result.

The software product is created in the high-level Python programming language in the PyCharm environment. The PyQt5 library was used for the user graphical interface, the server part used tools for managing isolated Linux-containers Docker, MariaDB database management system for data storage, Adminer database administration system, messaging system between software components based on AMQP standard RabbitMQ. The TensorFlow and Keras libraries were used to build, teach and use neural networks.

**Key words:** parsing, lemmatization, classification, statistical analysis, bibliographic sources of information, intellectual analysis of information.

# ЗМІСТ

АНОТАЦІЯ .....	4
ABSTRACT .....	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ І СКОРОЧЕНЬ.....	8
ВСТУП.....	10
1. ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ СИСТЕМИ КЛАСИФІКАЦІЇ НАУКОВИХ ТЕКСТОВИХ МАСИВІВ ЗА ПАРАМЕТРАМИ .....	13
1.1 Основні задачі.....	13
1.2 Задачі для системи збору даних.....	15
1.3 Задачі для системи обробки текстів .....	15
1.4 Задачі для системи навчання.....	16
1.5 Задачі для системи нейронної обробки.....	16
1.6 Задачі для системи наглядача .....	18
2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ.....	19
2.1 Огляд бібліографічних джерел .....	19
2.2 Огляд систем управління базами даних.....	21
2.3 Огляд підходів до нейронних мереж та обробці текстів.....	23
2.3.1 Огляд бібліотеки FastText .....	23
2.3.2 Огляд реалізації на Dask та sklearn.....	25
2.3.3 Огляд реалізації Zero Shot Classification через HuggingFace .....	27
2.3.4 Огляд реалізації на Tensorflow та Keras.....	28
2.4 Огляд інструментів міжмодульної комунікації .....	29
2.5 Огляд інструментів візуалізації .....	30
3. ЗАСОБИ РОЗРОБКИ .....	31
3.1 Бібліотека arXiv для Python.....	31
3.2 Бібліотека Stanza для Python .....	32
3.3 Бібліотека PyQt5 для Python.....	32
3.4 ORM sqlalchemy.....	33
3.5 Середовище розробки PyCharm.....	33
3.6 Платформа контейнеризації Docker .....	34
3.7 ОС Windows 11 .....	34
4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ .....	36

4.1. Архітектура системи .....	36
4.2. Додаток доступу до бази даних .....	38
4.3. Модуль вивантаження даних з бібліографічних джерел .....	40
4.4. Модуль попередньої обробки текстів .....	43
4.5. Модуль конвертації текстів у формат для навчання .....	44
4.6. Модуль навчання нейронної мережі .....	45
4.7. Автономний модуль обробки текстів.....	49
4.8. Модуль керування та нагляду автономними модулями обробки текстів ....	51
4.9. Користувацькі інтерфейси.....	53
5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ.....	57
5.1. Керівництво по встановленню системи.....	57
5.2. Інтерфейс адміністратора.....	58
5.3. Інтерфейс аналітика .....	66
ВИСНОВКИ.....	72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	73
ДОДАТОК А .....	75

# **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ І СКОРОЧЕНЬ**

ОС – Операційна система

АІС – Автоматизована інформаційна система

БД – База Даних

СУБД – Система управління базами даних

Пайплайн – Послідовні стадії перетворення даних

GPU – Graphics Processing Unit

CPU – Central Processing Unit

TPU – Tensor Processing Unit

API – Application Programming Interface

GUI – Graphic User Interface

ID – Identifier

ORM – Object-Relational Mapping

SQL – Structured Query Language

UML – Unified Modeling Language

CBOW – Continuous Bag-of-Words

OvR – One-vs-the-rest

NLI – Natural Language Inference

BoW – Bag of Words

TF – Term Frequency

IDF – Inverse Document Frequency

AMQP – Advanced Message Queuing Protocol

IDE – Integrated Development Environment

ReLU – Rectified Linear Unit

## ВСТУП

Проблема каталогізації та впорядкування інформації не зменшила свою актуальність навіть у інформаційну еру. Потік матеріалів, який постійно збільшується, неможливо обробляти у ручному режимі, а поточні потреби з пошуку та аналізу інформації ставлять все більші вимоги до рівня категоризації. І хоча наведені вище твердження справедливі для багатьох галузей, надалі буде розглядатися питання саме наукових текстів. Їх упорядкування необхідно як і для звичайних користувачів, які потребують для ознайомлення тексти певних категорій, так і для глобальної аналітики наукової діяльності чи метааналізу.

У цілому, проблематика класифікації текстів за визначеними ознаками надає змогу зацікавленим особам здійснювати поглиблений аналіз потоків наукової інформації. Подібні системи призначені для аналізу масивів текстових даних і на основі результатів кластеризації оцінювати національну та міжнародну активність у різних сферах за даними з бібліографічних джерел у автоматичному режимі, а при необхідності це можна здійснювати у реальному часі. Така діяльність корисна як і для оцінювання дослідницьких і наукових організацій, так і для автоматичних систем каталогізації та архівування.

Результати класифікації дозволяють автоматично поділити на потоки нову бібліографічну інформацію та на існуючі архіви наукових текстів, що знайде своє застосування у різноманітних сферах: від розподілу бюджетного, грантового та іншого фінансування до побудови систем для пошуку наукових текстів за напрямками наукової діяльності. Як одне з рішень даної проблеми є можливість потоково аналізувати наявні наукові роботи з бібліографічних джерел на предмет визначених їх тематики за допомогою методів машинного навчання.

Метою роботи є створення системи для вирішення задач класифікації наукових текстових масивів за заданими параметрами. Задля цього необхідно

вирішити низку задач, від дослідження наявних бази бібліографічних джерел і сучасних методів машинного навчання і обробки природної мови, до підготовки відповідних наборів даних наукових статей та розробки структури системи для зберігання та категоризації текстів на основі наявного набору даних. Така система має мати такі можливості, як отримувати, зберігати, аналізувати та оцінювати тематику текстів у автоматичному режимі, зберігати результати аналізу; аналізу та оцінки даних, порівнюючи роботи авторів з ключовими словами їх діяльності; відображення даних у вигляді таблиць і графіків; збереження інформації у БД.

Створений програмний продукт дозволить зберегти час при пошуку статистичних матеріалів і надає змогу доволі швидко перевіряти їх валідність. Розроблена система представлена у вигляді додатків з графічним інтерфейсом для адміністрування й звичайного користування та серверної частини, що буде виконувати усі необхідні операції із завантаження, зберігання, обробки текстових даних, навчання і використання нейронних мереж, моніторингу, надання і керування потоками даних. За допомогою інтерфейсу користувач може завантажувати на аналіз одинокі тексти та текстові масиви, переглядати оброблені дані, що зберігаються системою, та отримувати статистичні результати роботи системи.

Додаток створено на високорівневній мові програмування Python у середовищі PyCharm для ОС Windows 10 чи 11.

Для створення користувацького інтерфейсу використано бібліотеку PyQt5 і та середовище QtDesigner, що надає інструменти для зручного прототипування і реалізації інтерфейсів PyQt5.

В якості бази даних використана СУБД MariaDB на базі MySQL і ORM бібліотека sqlalchemy для зв'язку бази даних з концепціями об'єктно-орієнтованих мов програмування, а саме Python.

Завантаження первинних даних здійснюється за використанням бібліотеки Python wrapper for the arXiv API і внутрішнього функціоналу мови Python.

Для функціоналу обробки текстових даних використовуються бібліотека Stanza, а саме англійська модель і пайплан з токенизації та лематизації.

Нейронні мережі реалізовані через бібліотеки TensorFlow та Keras.

# **1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ СИСТЕМИ КЛАСИФІКАЦІЇ НАУКОВИХ ТЕКСТОВИХ МАСИВІВ ЗА ПАРАМЕТРАМИ**

Для виконання поставленої задачі необхідно пройти такі етапи:

- 1) Аналіз існуючих бібліографічних джерел на предмет кращого джерела даних для навчання.
- 2) Налаштування завантаження даних з обраного джерела.
- 3) Розробка архітектури модулю зберігання інформації та обміну повідомленнями між модулями.
- 4) Розробка архітектури модулів серверної частини.
- 5) Розробка архітектури зв'язку між модулями системи.
- 6) Розробка архітектури нейронної мережі.
- 7) Розробка інтерфейсу програми для адміністратора та аналітика.
- 8) Всебічне тестування.

## **1.1 Основні задачі**

Дана система повинна надавати користувачам можливості категоризації одиноких текстів і текстових масивів у реальному часі, мати можливість зберігати результати категоризації та переглядати статистичну інформацію за ними.

Система має мати можливість працювати як частина вже існуючої системи, отримуючи по API тексти у реальному часі та надсилання результатів обробки у відповідь, так і з готовою та закінченою базою наукових текстів.

При обробці текстів наукових статей та тезисів доповідей на наукових конференціях з метою подальшої класифікації наукових текстових масивів за заданими параметрами кожен науковий текст проходить через етапи обробки [1], такі як вказані на рисунку 1.1:

- переведення всіх літер у тексті до нижнього регістру,
- видалення цифр,
- необхідної пунктуації та інших спеціальних символів,
- видалення стоп слів,
- стеммінг,
- лематизація,
- векторизація.

Після цього вектор тексту направляється до відповідних нейронних мереж [2], натренованих на знаходження тематики та теми наукових текстів. У результаті на кожний текст виводиться інформація по тематикам і темам з відповідними ймовірностями, після чого ця статистична інформація доступна для подальшої роботи.

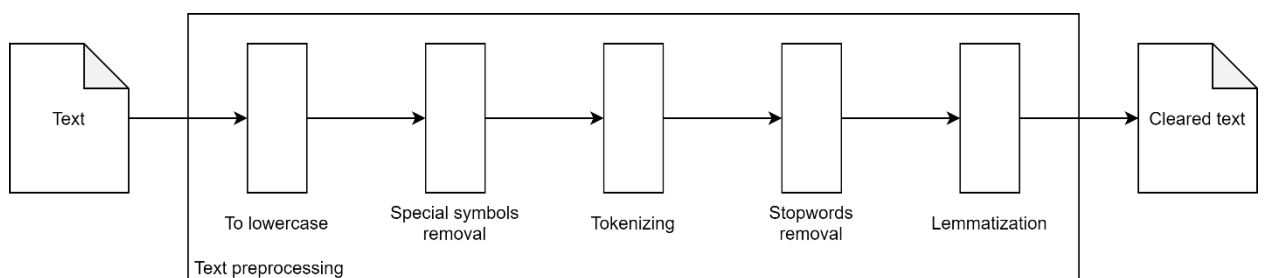


Рисунок 1.1 – Схема обробки тексту

Завершений програмний додаток з визначеним у завданні функціоналом може бути використаний для автоматичної категоризації наукових статей для подальшого використання отриманих даних як для автоматичних систем каталогізації та архівування, так і для створення профілю оцінювання наукових організацій й іншої статистичної обробки результатів.

## **1.2 Задачі для системи збору даних**

На базі модулю збору даних повинен бути реалізований парсер для отримання текстів з бібліографічної бази. Пошук даних повинен проходити за окремими запитами. Необхідно отримувати ID статті, назву статті, витяг, автора та категорії.

## **1.3 Задачі для системи обробки текстів**

Система обробки текстів має отримувати на вхід лише текст та налаштування для подальшої обробки, отримуючи на вихід текст у готовому до передачі на навчання форматі. Ця система має працювати як на GPU, так і на CPU. Швидкодія системи має бути достотною для обробки великої кількості текстів у реальному часі. Система має повністю використовувати можливості паралельного виконання коду для максимальної швидкодії.

## **1.4 Задачі для системи навчання**

Система навчання має мати можливість навчати нейронні мережі, візуалізувати, валідувати, зберігати та завантажувати результати навчання. Має працювати як на GPU, так і на CPU та можливе використання TPU. Необхідна можливість використання у середовищі Google Colab, з обмеженням сесії у 12 годин.

## **1.5 Задачі для системи нейронної обробки**

Модуль нейронної обробки, або юніт – ключова самостійна одиниця системи обробки текстів. Цей модуль має працювати самостійно, отримуючи певні параметри запуску, і реалізовувати можливість зміни текстових потоків на обробку. Він має складатися з модулю обробки текстів і модулю нейронної мережі, а також забезпечувати неперервний перегляд і приймання текстів через визначене API. На вихід буде отримуватись відповідь на запит через API. Для зберігання відповідності запитів до тексту прив'язується його ID, також у запиті може бути присутня додаткова мета-інформація, яка теж має бути збережена.

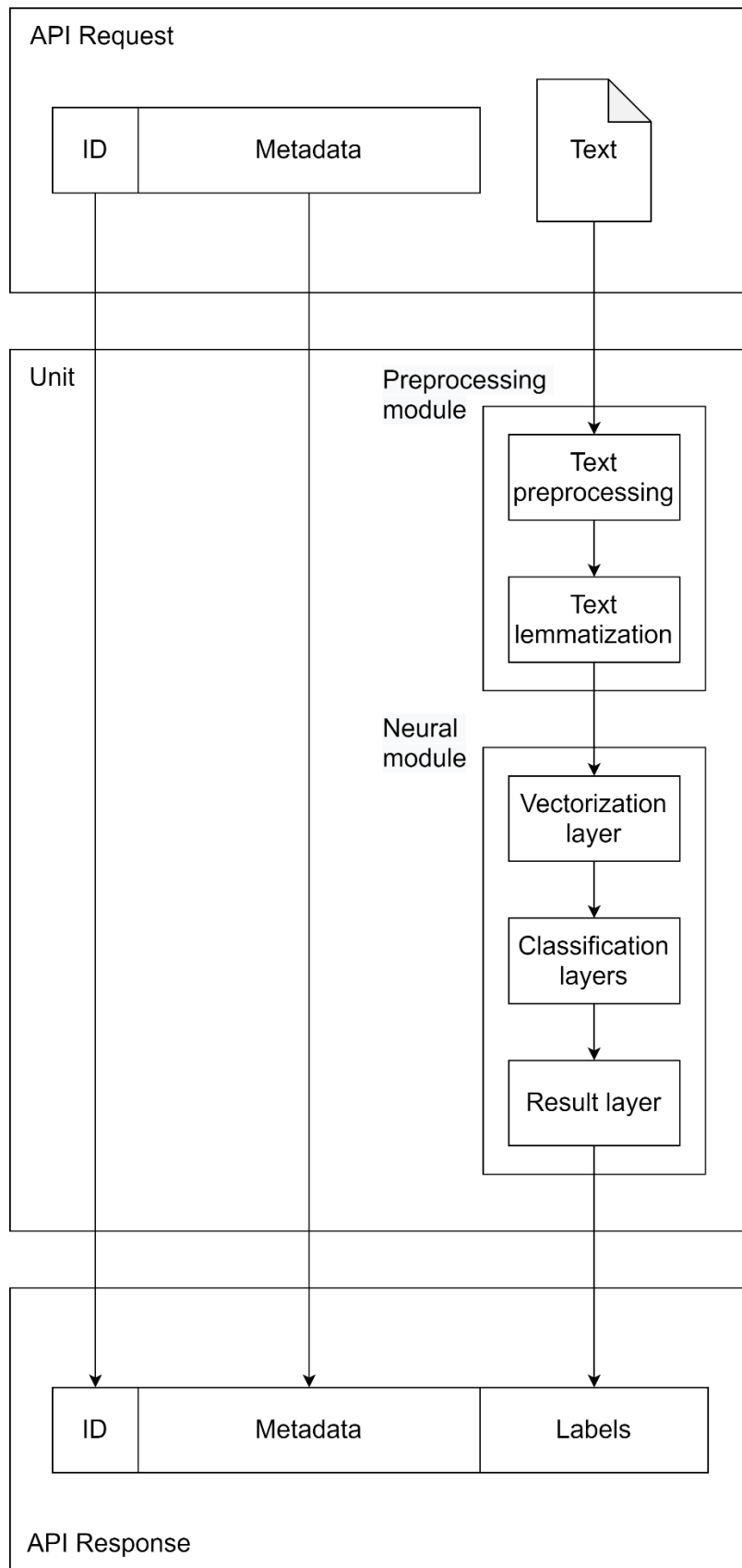


Рисунок 1.2 – Схема модулю нейронної обробки

## 1.6 Задачі для системи наглядча

Основою серверної частини комплексної системи, що розглянута у цій роботі, є модуль наглядча або менеджера. Він має відповідати за життєвий цикл під'єднаних до нього юнітів, моніторинг їх активності та прийом команд від адміністратора. Для цього модуль має бути також підключений до API.

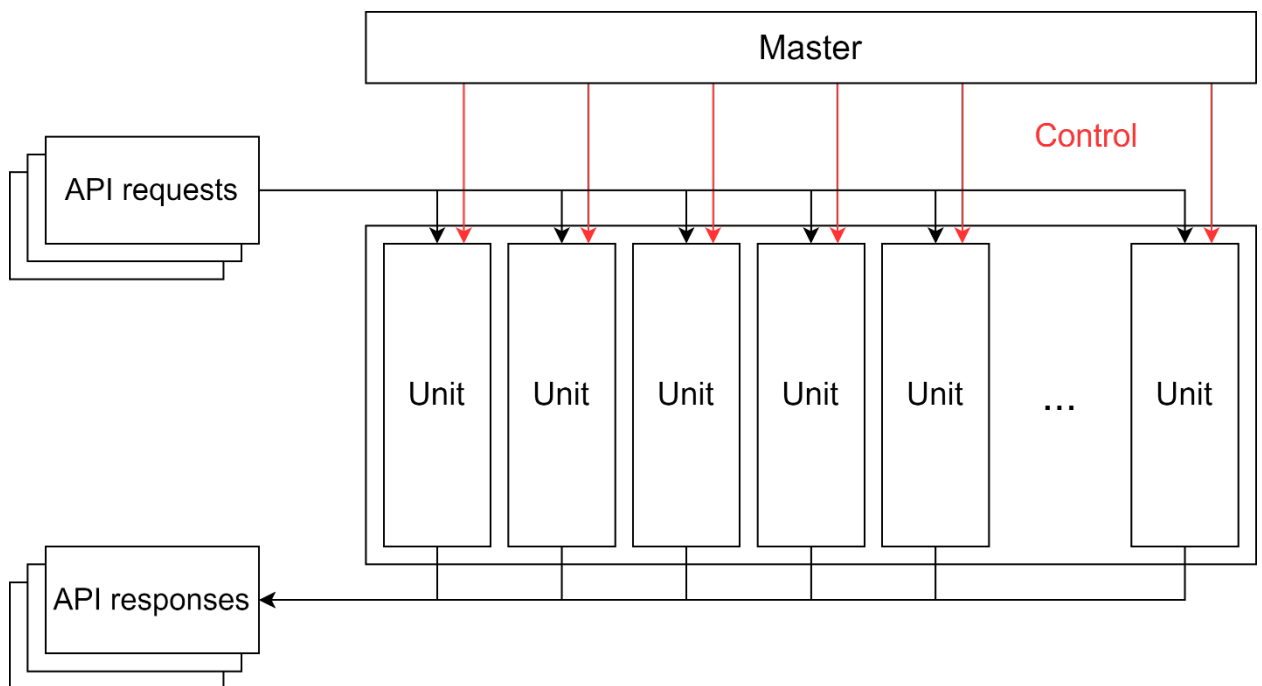


Рисунок 1.3 – Схема модулю наглядча

На рисунку 1.3 наведена схема роботи серверної частини комплексної системи з API запитом.

## 2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

На даний момент аналоги такої комплексної системи, яка виконує всі необхідні функції, існують лише у вигляді різних підходів до проектування нейронних мереж та інших аналогів окремих компонентів, які потребують з'єднання між собою. Через це далі будуть розглядатися аналоги окремих її компонентів.

### 2.1 Огляд бібліографічних джерел

Серед найпопулярніших баз наукової бібліографічної інформації можна розглядати такі джерела, як: Google Scholar, CORE, Scopus, arXiv. Саме вони і будуть розглянуті нижче.

Google Scholar – це вільна пошукова система, запущена у 2004 році, який індексує наукові роботи та різноманітну інформацію про них, пропонуючи безкоштовний доступ по ним [3]. Google Scholar включає в себе більшість рецензованих онлайн-журналів Європи та Америки найбільших наукових видавництв. За функціями ця пошукова система схожа на не розглянуті у цій роботі системи, такі як getCITED, Scirus та CiteSeerX. У супутніх даних присутні необхідні для навчання категорії, але основних категорії доволі мала кількість (8 штук), а детальних категорій доволі значна кількість, що також не підходить для подальшого навчання.

Також є ще один доволі популярний сервіс CORE, який об'єднує близько 125 мільйонів відкритих дослідницьких результатів [4]. До цих даних можна отримати доступ безкоштовно, хоча й існують обмеження на повторне використання. Але CORE доволі мало розповсюджений порівняно з Google Scholar і Scopus, а також

не зберігає інформацію про категорії у наукових текстах, що робить це джерело неактуальним у даному контексті.

Scopus, в якому зібрано близько 36 тисяч статей, є багатодисциплінарною базою даних рецензованої літератури з інструментами для відстеження, аналізу та візуалізації досліджень [5]. Необхідні для навчання з вчителем категорії є можливість доволі легко вилучати за допомогою Scopus Code, де кодування проходить чотирьома цифрами – дві перших формують загальну категорію, а дві останніх – підкатегорії. І хоча за допомогою такої гнучкої системи є можливість створювати різні конфігурації тематик і тем, закритість Scopus і невелика кількість текстів робить його доволі невдалим вибором для поставленої задачі.

Розглянутий надалі arXiv являє собою електронний архів наукових текстів, який збирає статті з фізики з астрономії, математики, інформатики, кількісної біології, фінансової математики та інших. Він був створений у 1991 році в Лос-Аламоській національній лабораторії, але на цей час він функціонує в Корнельському університеті. База даних arXiv містить тексти у галузях астрофізики, фізики, математики та статистики, нелінійних наук, інформатики, кількісної біології, кількісних фінансів, економіки та електротехніки. Зараз arXiv.org містить понад 2 мільйони статей, а також має деревовидну структуру категорій, з 8 основних і значна кількість додаткових, які можуть ділитися на більш детальніші категорії. Також всі статті у arXiv містять витяги, які являють собою основу для навчання.

Судячи з поданої вище інформації, arXiv найкраще підходить для використання в якості масиву даних для подальшого навчання з вчителем нейронних мереж. 2 мільйони статей буде більш чим достатньо, а деревовидна структура категорій дозволяє змінювати їх кількість, залишаючи усі теми повністю охопленими.

## 2.2 Огляд систем управління базами даних

У нас час існують значна кількість систем управління базами даних, кожна з яких має свої переваги та недоліки, і кожна з яких підходить під свій тип задач. Доволі важливо правильно обрати такий інструментарій, щоб не стикатися з фундаментальними архітектурними проблемами у подальшій реалізації системи.

Окрім усім відомої мови взаємодії з базою даних SQL, в останні роки в області баз даних з'явився ще один термін: бази даних NoSQL. SQL і NoSQL взаємодіють з різними типами баз даних. SQL – це підхід, який використовується для взаємодії з реляційними базами даних, тоді як NoSQL використовується для взаємодії з нереляційними базами даних. У реляційних базах даних дані зберігаються у різних таблицях, кожна з яких містить кілька записів. Ці таблиці з'єднані одна з одною за допомогою одного або кількох зв'язків. Також ключі визначають відношення між таблицями.

На відміну від реляційних баз даних, нереляційні бази даних або бази даних NoSQL — не зберігають дані в таблицях і записах. Натомість у цих типах баз даних структура зберігання даних розроблена й оптимізована для конкретних вимог. Замість SQL, який використовують реляційні бази даних, бази даних NoSQL використовують об'єктно-реляційне відображення (ORM) для полегшення зв'язку зі своїми даними. Чотири популярних типи баз даних NoSQL: орієнтовані на стовпці, орієнтовані на документи, пари ключ-значення і бази даних графів. Ці типи можливо використовувати окремо або комбінувати їх.

Для реалізації системи з класифікації наукових текстових масивів за параметрами нам необхідно зберігати, у першу чергу, інформацію для навчання, а саме тексти та зв'язані з ними категорії. Найбільш ефективно у контексті зайнятого місця буде зберігати тексти у форматі SQL, з'єднуючи їх з категоріями за допомогою ключів. Але для навчання нейронних мереж дані мають бути

представлені у вигляді масиву рядків, до яких зручно та швидко звертатися, і з яких легко формувати серії для навчання.

Тому у проекті для довготривалого зберігання отриманих з бібліографічного ресурсу даних було використано систему управління базами даних MariaDB, яка являє собою відгалуження від MySQL [6]. Ця система управління реляційною базою даних, створена групою переважно колишніх співробітників MySQL, які мали за основна мету працювати зі спільнотою вільного програмного забезпечення і випускати СУБД під ліцензією GPL, на відміну від нестабільного статусу ліцензії MySQL, яка тепер залежить від Oracle. Саме через цю відкритість цій системі була надана перевага у подальшій реалізації проекту.

У якості резервної бази даних була обрана SQLite, компактна СУБД яка зберігається лише в одному файлі та не потребує серверної частини [7]. Так, як усі дані зберігаються у одному файлі, програми, що використовують бібліотеку SQLite, можуть звертатися до бази даних за допомогою мови SQL без виділеного процесу СУБД. Це означає, що одночасні запити (або паралельні користувачі) повинні блокувати файл для безпечної зміни БД. Цей пункт досить важливий, оскільки безпосередньо зачіпає сферу застосування SQLite - якщо в основному використовується читання даних, тоді жодних проблем немає, але якщо необхідно робити значну кількість одночасних оновлень, то програма витратить більше часу на синхронізацію блокування файлів, ніж робити справжню роботу.

Як проміжний формат для зберігання оброблених для навчання даних був обраний DataFrame, який являє собою частину Python бібліотеки Pandas [8]. Це двовимірні, потенційно неоднорідні табличні дані, що змінюються. Структура їх даних також містить позначені осі (рядки та стовпці). Арифметичні операції вирівнюються на мітках рядків і стовпців. Є основною структурою даних Pandas. Для передачі її між частинами системи її екземпляр зберігається за допомогою модулю pickle стандартної бібліотеки Python.

## **2.3 Огляд підходів до нейронних мереж та обробці текстів**

Класифікація тексту є однією з популярних завдань у аналізі натурального тексту, яка дозволяє програмі класифікувати документи вільного тексту на основі попередньо визначених класів. Класи можуть бути засновані на темі, жанрі або настрої, у нашому випадку це теми наукових робіт.

Останнім часом прогрес досліджень щодо класифікації текстів досяг доволі високого рівня. Аналіз натурального тексту досяг значних результатів за допомогою методів глибокого навчання як передової технології для виконання таких завдань.

У цій сфері існує значна кількість різноманітних підходів, які відкриваються кожен день. Надалі будуть розглянуті деякі із них.

### **2.3.1 Огляд бібліотеки FastText**

У 2015 році лабораторія Facebook's AI Research (FAIR) відкрила світу свою бібліотеку fastText [9]. FastText – це бібліотека для вивчення представлення слів і класифікації тексту, модель дозволяє створити алгоритм навчання без учителя або з вчителем для отримання векторних представлених слів.

Для отримання векторного представлення слів одночасно використовуються моделі skipgram та CBOW. FastText надає швидшу роботу порівняно з іншими пакетами та моделями. Для моделі векторних уявлень слів використовується skipgram із негативним семплюванням, яке представляє собою спосіб створити для навчання векторної моделі негативні приклади, тобто показати їй кілька слів, які не є сусідами за контекстом. Для кожного позитивного прикладу (коли слова в

тексті стоять поруч, наприклад, “швидкий рух” ми підбираємо кілька негативних, наприклад “швидкий дім “швидка фізика”, “швидкий мікрофон”. Усього підбирається від декількох до десятків випадкових слів. Такий випадковий підбір кількох прикладів не потребує багато комп'ютерного часу та дозволяє прискорити роботу FastText. Також до основної моделі було додано subword-модель, для уявлення слова через ланцюжки символів ( $n$ -грами) з  $n$  від 3 до 6 символів від початку до кінця слова та слово цілком, що необхідно для мов з комплексними словами, наприклад, німецька. Наприклад, слово темно з  $n = 3$  буде представлено  $n$ -грамами “те, тем, емн, мно, но“ і послідовністю “темно“. Таким чином, для моделі є різниця між послідовністю <тем> у слові тем - і  $n$ -грамою тем із слова темно. Такий підхід дозволяє працювати з тими словами, які модель раніше не зустрічала.

В основі класифікатора лежить модель лінійної класифікації, що з архітектури схожа з моделлю SBOW. Чим більша кількість класів, тим більший час роботи лінійної моделі. Для оптимізації класифікатора використовується ієрархічний softmax, заснований на алгоритмі кодування Хаффмана.

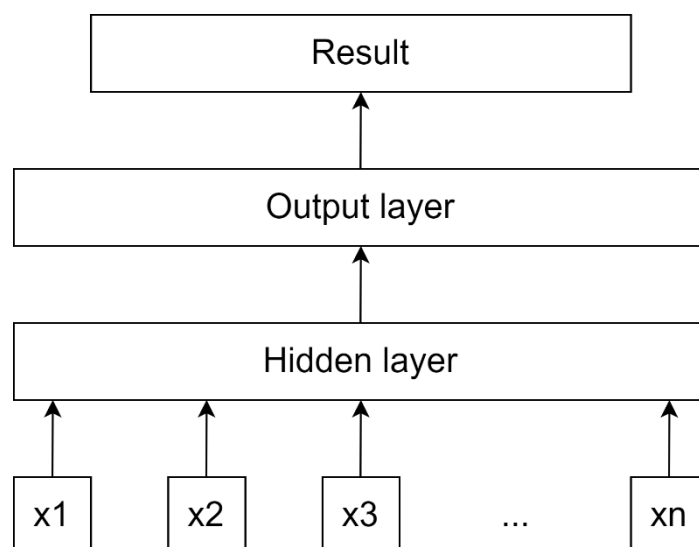


Рисунок 2.1 – Схема нейронної мережі fastText

Як нейрона мережа, модель являє собою просту мережу лише з одним шаром. Подання тексту в пакеті слів спочатку подається на шар відображення, де відображення вибираються для кожного окремого слова. Потім ці відображення слів усереднюються, щоб отримати єдине усереднене відображення для всього тексту. На прихованому шарі ми отримуємо кількість параметрів яка рівна добутку розміру відображення ( $dim$ ) елементів та розміру словника. Після усереднення ми маємо лише один вектор, який потім подається до лінійного класифікатора: ми застосовуємо Softmax до лінійного перетворення вихідних даних вхідного шару. Лінійне перетворення являє собою матрицю з  $dim \times output$ , де  $output$  – числом вихідних категорій.

Це доволі проста нейрона мережа, яка не підходить для такої комплексної задачі, як аналіз наукових текстів, але має свої ніші для використання, наприклад, дуже швидка категоризація малих за вмістом і словником текстів.

### 2.3.2 Огляд реалізації на Dask і sklearn

Для вирішення цієї проблеми можна також використовувати Dask [11]. Dask дозволяє нам використовувати можливості розподіленої обробки наших машин, а також лінійне оцінювання. Надалі буде розглянуто реалізацію [12] за допомогою класифікатора з бібліотеки Sklearn [13]. У цій реалізації був використаний CalibratedClassifierCV, який використовує калібрування ймовірності за допомогою ізотонічної або логістичної регресії, у нашому випадку він використовує Linear Support Vector Classification.

Цей класифікатор використовує перехресну перевірку як для оцінки параметрів класифікатора, так і для подальшого калібрування класифікатора. У нього є два варіанти роботи, за замовчуванням, для кожного поділу кроссвалідації

він поміщає копію базової оцінки до навчальної підмножини та калібрує її за допомогою тестової підмножини. Для передбачення ймовірності усереднюються для цих окремих каліброваних класифікаторів. У іншому випадку, перехресна перевірка використовується для отримання неупереджених передбачень, які потім використовуються для калібрування. Для прогнозування використовується базова оцінка, навчена з використанням усіх даних.

Для класифікації багатьох класів у даному прикладі була використана багатокласова стратегія “один проти решти”, або OvR. Ця стратегія полягає у підборі одного класифікатора для кожного класу. Для кожного класифікатора клас порівнюється з усіма іншими класами. На додаток до його обчислювальної ефективності (потрібні лише класифікатори кількістю у кількість класів), однією з переваг цього підходу є його інтерпретація. Оскільки кожен клас представлений лише одним і одним класифікатором, можна отримати знання про клас, перевірявши його відповідний класифікатор. Це найбільш часто використовувана стратегія для багатокласової класифікації та є справедливим вибором за замовчуванням.

Задля зменшення зашумленості у прикладі використовується фільтр стоп-слів від бібліотеки NLTK [14]. Стоп-слова, або порожні слова – це назви слів, що не мають значення, таких як артиклі, займенники, прийменники тощо, які фільтруються до або після обробки даних природною мовою. Створення англійських стоп-слів і використання цієї концепції у своєму дизайні приписують Гансу Петеру Луну, одному з піонерів інформаційного пошуку. Не існує остаточного списку стоп-слів, які включають усі інструменти обробки природної мови.

Але при цьому у розглянутій системі відсутня будь-яка лематизація, тобто усі подані на вхід тексти подаються у оригінальному вигляді, що може бути корисним для деяких типів нейронних мереж що залежать від контексту.

Підсумовуючи, такий підхід надає точність прогнозування категорій дорівнює 62.6%, що є достатнім значенням для багатокласової класифікації, але не є задовільним для даної системи.

### 2.3.3 Огляд реалізації Zero Shot Classification через HuggingFace

Hugging Face дозволяє користувачам створювати, навчати та розгортати нейронні моделі машинного навчання за допомогою відкритого коду.

Традиційно навчання з нульовим режимом (ZSL) найчастіше відноситься до досить специфічного типу завдання: вивчити класифікатор на одному наборі міток, а потім оцінити на іншому наборі міток, яких класифікатор ніколи раніше не бачив.

Останнім часом, особливо в обробці натуральної мови, це використовується набагато ширше, щоб означати змусити модель робити те, чому вона не була спеціально навчена робити. Добре відомим прикладом цього є робота GPT-2, де автори оцінюють мовну модель для таких завдань, таких як машинний переклад, без безпосереднього налаштування цих завдань.

Простіше кажучи, нульова класифікація відноситься до класу проблем машинного навчання, де ми хочемо, щоб наші моделі передбачали вихід для класів, з якими вона не стикалася під час навчання.

Основна модель тренується на завданні висновку з природної мови (NLI), яке приймає дві послідовності та визначає, чи суперечать вони один одному, не тягнуть за собою одне одного, чи ні.

Це можна адаптувати до задачі класифікації нульового результату, розглядаючи послідовність, яку ми хочемо класифікувати як одну послідовність NLI (називається передумовою), і перетворюючи мітку-кандидат на іншу (гіпотеза). Якщо модель передбачає, що побудована передумова тягне за собою

гіпотезу, то ми можемо прийняти це як передбачення, що мітка застосовується до тексту [15].

Цей метод належним чином показує себе відносно аналогічних на малій виборці текстів, але не підходить для точної категоризації, надаючи доволі багато зайвих категорій. Хоча його залежність від контексту та спеціальних символів може стати у нагоді при подальшому розвитку подібних нейронних мереж для категоризації наукових текстових масивів за заданими параметрами.

### **2.3.4 Огляд реалізації на TensorFlow та Keras**

TensorFlow – це безкоштовна програмна бібліотека з відкритим вихідним кодом для машинного навчання [16]. Ця бібліотека може використовуватися для різних цілей, але особлива увага приділяється навчанню та виведенню глибоких нейронних мереж. Він був розроблений командою Google Brain для внутрішнього використання Google, але програмне забезпечення було випущено під ліцензією Apache 2.0 у 2015 році.

Keras – бібліотека нейронних мереж з відкритим вихідним кодом, написана на Python [17]. Вона може працювати з Keras TensorFlow, Microsoft Cognitive Toolkit, R, Theano або PlaidML, і призначена для експериментів з глибокими нейронними мережами, орієнтований на зручність використання, модульність і розширюваність.

Але перед тим як посилати текст у нейронну мережу, його необхідно попередньо обробити. У даному випадку є можливість використати дві методики, мішок слів (BoW) перетворює текст у вектор ознак, підраховуючи кількість слів у документі. Це не враховує важливість слів. Також існує частота термінів – інверсна частота документів (TFIDF), яка заснована на моделі «мішок слів» (BoW), що

містить уявлення про менш релевантні та більш релевантні слова в документі. Важливість слова в тексті має суттєве значення для пошуку інформації. Періодичність терміну (TF) – це міра частоти слова у документі. TF визначається як відношення кількості слова в документі до загальної кількості слів у документі. Зворотна частота документів (IDF) – це міра важливості слова. Частота термінів (TF) не враховує важливість слів. Деякі слова, такі як «з», «і» тощо, можуть бути найчастішими, але вони мають незначне значення. IDF надає вагу кожному слову на основі його частоти в корпусі D. Частота термінів — інверсна частота документів це мікс TF та IDF. він надає більшу вагу слову, яке нечасто зустрічається у корпусі (всі документи) та надає більше значення тому слову, яке частіше зустрічається у документі.

Так, TFIDF векторизатор отримує на вхід текст, і видає на вихід матрицю характеристик.

Ці бібліотеки дозволяють самим будувати нейронні мережі будь-яких конфігурацій, встановлювати кількість шарів, кількість нейронів та їх тип. Ця гнучкість дозволяє успішно використати ці бібліотеки у нашому проекті.

## 2.4 Огляд інструментів міжмодульної комунікації

У якості основи для комунікації між компонентами було обрано Advanced Message Queuing Protocol (AMQP), тобто відкритий стандарт протоколу прикладного рівня для проміжного програмного забезпечення, орієнтованого на повідомлення. Характеристиками, які визначають AMQP, є орієнтація повідомлень, черга, маршрутизація, надійність і безпека.

AMQP визначає поведінку служби зв'язку та клієнта до такої міри, що робить реалізації різних постачальників взаємодіючими, так само, як SMTP, HTTP, FTP тощо створили взаємодіючі системи. Попередні стандартизації проміжного

програмного забезпечення діяли на рівні API (наприклад, Java Message Service) та були зосереджені на стандартизації взаємодії розробників з різними реалізаціями, а не на забезпеченні взаємодії між кількома реалізаціями. На відміну від JMS, який визначає API та набір поведінок, які повинна забезпечувати реалізація зв'язку, AMQP є протоколом "кабельного рівня". Протокол кабельного рівня — це термін для формату даних, які надсилаються по мережі у вигляді потоку байтів. Отже, будь-який інструмент, який може створювати та інтерпретувати повідомлення, що відповідають цьому формату даних, може працювати з будь-яким сумісним інструментом незалежно від мови реалізації.

У якості брокера було обрано RabbitMQ, на основі Open Telecom Platform, та написаний на мові Erlang, як двигун бази даних для зберігання повідомлень використовує Mnesia. Він доволі легкий для опанування, легковажний у контексті використання ресурсів і має багатофункціональний інструментарій.

## 2.5 Огляд інструментів візуалізації

Для взаємодії користувача з системою необхідно розробити користувацький інтерфейс. У випадку мови Python є вибір з двох найбільш популярних варіантів — Tkinter та PyQt.

PyQt, як і Tkinter – це набір інструментів для віджетів графічного інтерфейсу користувача (GUI) [18]. Вони обидва доволі схожі між собою, але при цьому PyQt дозволяє більш тісні інтеграції з різноманітними додатками, а також краще структуровану документацію, через що він і був обраний у якості основного інструментарію для реалізації користувацького інтерфейсу. Для реалізації було використано саме PyQt 5 через її стабільність та більшу підтримку спільнотою, на відміну від нової версії PyQt 6.

## 3 ЗАСОБИ РОЗРОБКИ

Для розробки даної програмної системи було використано доволі багато сторонніх інструментів і сервісів, а саме реляційну базу даних MariaDB на базі MySQL для зберігання проміжної інформації та SQLite як резервну базу даних. Для адміністрування бази даних був використаний модуль Adminer. У якості зв'язку між компонентами системи був використаний програмний брокер повідомлень на основі стандарту AMQP – RabbitMQ. Уся серверна частина використовує програмне забезпечення для автоматизації розгортання і керування програмами Docker.

Програмний код системи написаний на мові програмування Python у середовищі PyCharm. Нейрона частина використовує пару бібліотек TensorFlow і Keras, обробка тексту проходить через бібліотеку Stanza. Користувацький інтерфейс написано з використанням фреймворку PyQt. Для завантаження даних була обрана бібліотека arXiv для Python.

### 3.1 Бібліотека arXiv для Python

Бібліотека arXiv для високорівневої мови програмування Python являє собою базову бібліотеку-обгортку для запитів до arXiv API. Вона дозволяє користуватися як і базовим пошуком, так і створювати свої клієнти для запитів різної частоти та типів. Ця бібліотека підтримує усі встановлені ліміти на вивантаження даних і не дає системи arXiv заблокувати користувача за неправомірне користування її ресурсами.

## 3.2 Бібліотека Stanza для Python

Stanza являє собою бібліотеку для аналізу природної мови для мови програмування Python. Вона містить інструментарій, що використовується для перетворення людської натуральної мови у списки речень та слів, для лематизації слів, встановлення частин мови та морфологічних ознак. Також можливе використання задля розбору синтаксичної структури, мовних залежностей та розпізнавання різних типів іменованих сутностей. Бібліотека надає свій функціонал для використання на більше ніж 70 мов.

Stanza створена на базі високоточних компонентів нейронної мережі на основі бібліотеки PyTorch, які забезпечують ефективне навчання та оцінку за Вашими власними анотованими даними. Бібліотека отримує максимальну швидкодію, якщо запустити програмне забезпечення з нею на комп'ютері з підтримкою GPU.

Крім того, бібліотека Stanza включає інтерфейс мови Python до пакету Java CoreNLP та отримує з нього деякі додаткові функції, такі як відповідність мовним шаблонам.

## 3.3 Бібліотека PyQt5 для Python

Для розробки інтерфейсу була використана бібліотека PyQt саме 5 версії. PyQt – це бібліотека зв'язування на Python набору інструментів Qt із відкритим вихідним кодом, який також функціонує як кросплатформний фреймворк для розробки додатків. Qt – популярна платформа C++ для написання програм із

графічним інтерфейсом для всіх основних настільних, обільних і вбудованих платформ (підтримує Linux, Windows, MacOS, Android, iOS, Raspberry Pi тощо).

### **3.4 ORM sqlalchemy**

SQLAlchemy являє собою бібліотеку з відкритим вихідним кодом, написана мовою програмування Python і яка використовується для роботи з базами даних і об'єктно-реляційного їх відображення. Це спосіб відображення об'єктної архітектури інформаційної системи на базу даних (або інший елемент системи) реляційного характеру. Реалізація такого відображення використовується, серед іншого, в у випадку, коли створена система базується на об'єктно-орієнтованому підході, а система баз даних працює на зв'язках. Існує ряд проблем з продуктивністю, пов'язаними з ORM, але використання такого підходу дає захищеність від так званих SQL-ін'єкцій.

### **3.5 Середовище розробки PyCharm**

PyCharm – одна з найпопулярніших IDE для мови Python. Вона дозволяє аналізувати код і містить графічний налагоджувач. Вона також надає змогу керувати модульними тестами, інтегрувати програмне забезпечення для контролю версій та підтримує веб-розробку за допомогою Django та інших фреймворків. Розроблений чеською компанією JetBrains, це кросплатформне програмне забезпечення, яке працює на Windows, Mac OS X і GNU/Linux. Він доступний у

професійній версії, випущеній за власною ліцензією, і у спільному виданні, випущеному під ліцензією Apache. Для розробки була використана професіональна версія.

### **3.6 Платформа контейнеризації Docker**

Використаний у цьому проекті Docker – це програмна платформа з відкритим вихідним кодом, що реалізує віртуалізацію на рівні операційної системи. По суті, Docker пропонує автоматизовані процеси розробки додатків в ізольованих просторах користувача, званих програмними контейнерами. Програмне забезпечення використовує технології ядра Linux, такі як контрольні групи та простори імен ядра, щоб дозволити незалежним програмним контейнерам працювати в одній і тій же операційній системі. Це дозволяє уникнути використання додаткових обчислювальних ресурсів, які потрібні віртуальній машині. Для зручності конфігурація усього проекту знаходиться у файлі `docker compose`, який дозволяє піднімати систему однією командою.

### **3.7 ОС Windows 11**

Windows 11 це остатня на момент створення проекту версія операційної системи Microsoft Windows, представлена 24 червня і випущена 05 жовтня 2021 року і є наступником Windows 10. Саме вона була використана при розробці та

відладці системи. Кожен з модулів даної системи має змогу працювати на її основі та виконувати поставлені задачі у повному обсязі.

# 4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

## 4.1 Архітектура системи

Система складається з восьми основних компонентів: модуль вивантаження даних з бібліографічних джерел (Extractor), модуль попередньої обробки текстів (Processor), модуль конвертації текстів у формат для навчання (Converter), модуль навчання нейронної мережі (Learner), автономний модуль обробки текстів (Unit), модуль керування і нагляду автономними модулями обробки текстів (Master), інтерфейси користувача та адміністратора, а також усі серверні застосунки.

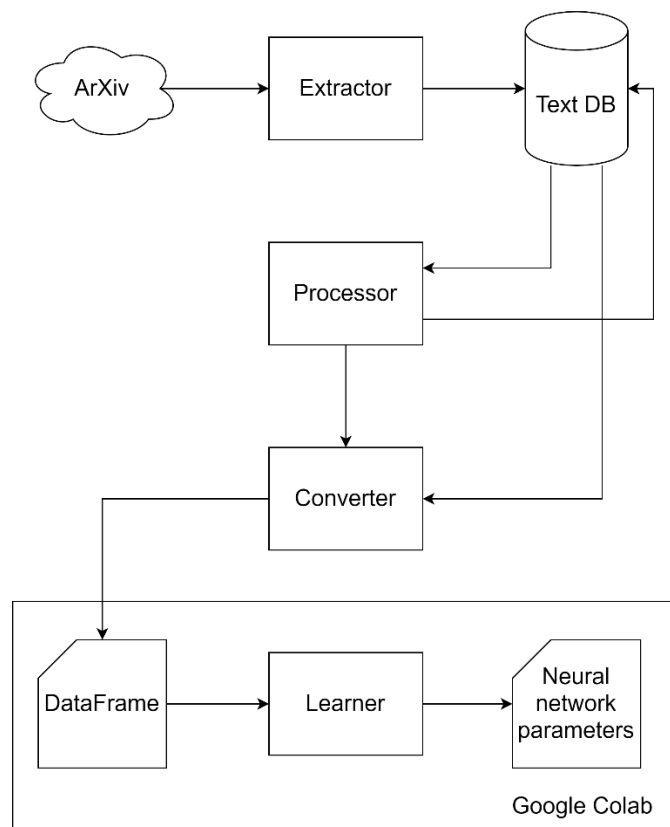


Рисунок 4.1 – Схема початкової обробки текстів

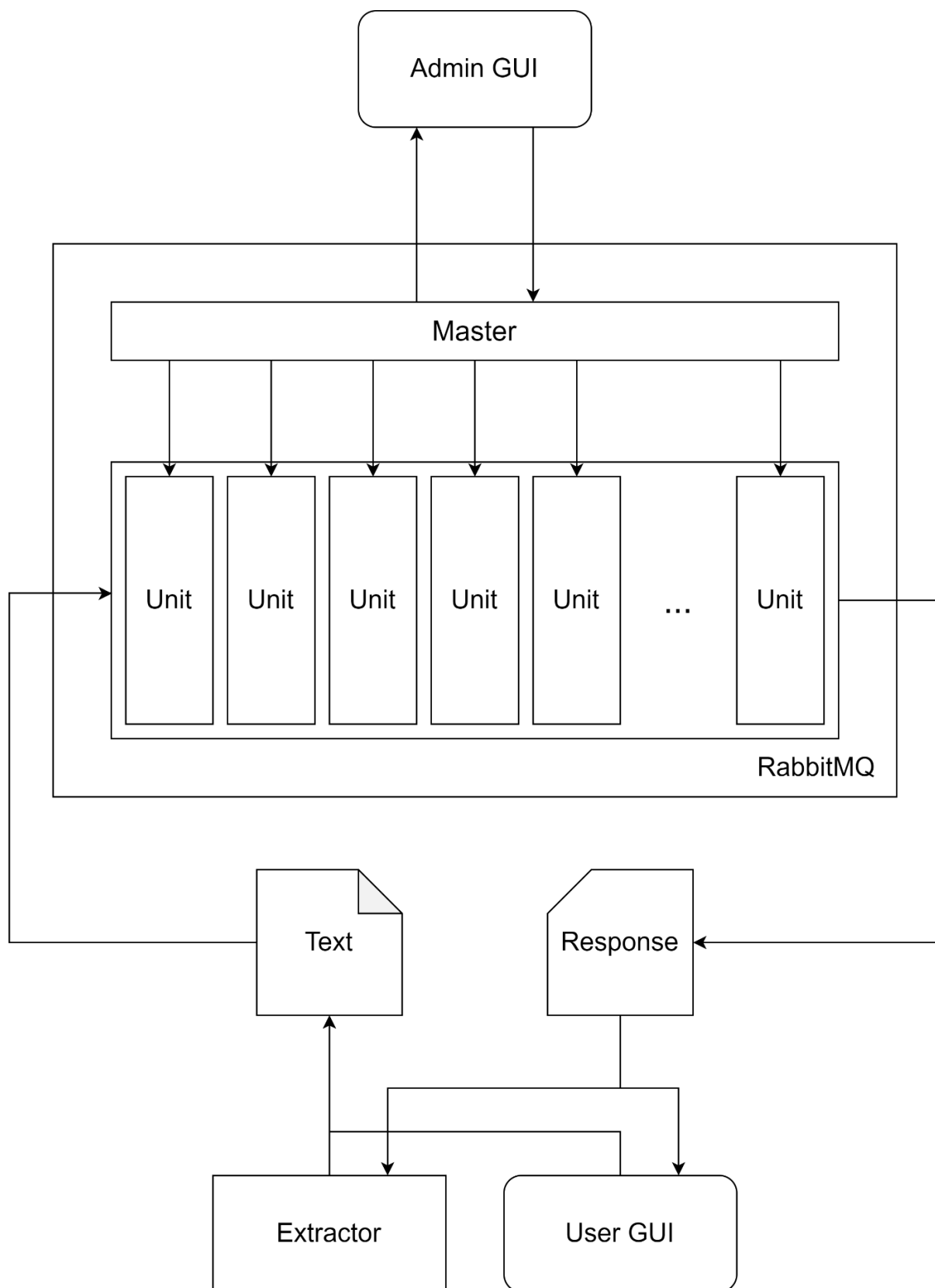


Рисунок 4.2 – Загальна схема роботи системи

На рисунку 4.1 та рисунку 4.2 наведені схеми первинної обробки масиву текстів та загальну схему роботи серверної частини, та її зв'язок з клієнтськими застосунками.

## 4.2 Додаток доступу до бази даних

Цей додаток відіграє ключову роль у комплексній системі класифікації наукових текстових масивів за параметрами. Саме через допомогою нього виконується під'єднання до обраного типу бази даних, виконуються універсальні та специфічні запити. Для уніфікації та полегшення впровадження інших типів баз до системи використовується батьківський клас `ConnectDB`, нащадки якого й містять специфічні інструкції щодо окремих випадків баз даних. За уніфікацію самого зв'язку між об'єктами баз відповідає бібліотека `sqlalchemy` [19].

Таблиці бази даних створюються або відновлюються через технологію ORM за допомогою представлення їх структури через класи. Кожна з таблиць має наслідувати її для включення у базу даних. Поля таблиць відображаються через поля класів типу `Column`, який сам по собі також може мати різні типи, такі як `Integer`, `Float`, `String` та `Boolean`. Зв'язки між таблицями також представляються у вигляді полів класів типу `relationship`, з вказанням класу до якого має вести зв'язок. Наприклад, поле `relationship('Authors')` задає відповідний зв'язок таблиці поточного класу до класу `Authors` (назва класу пишеться як рядок).

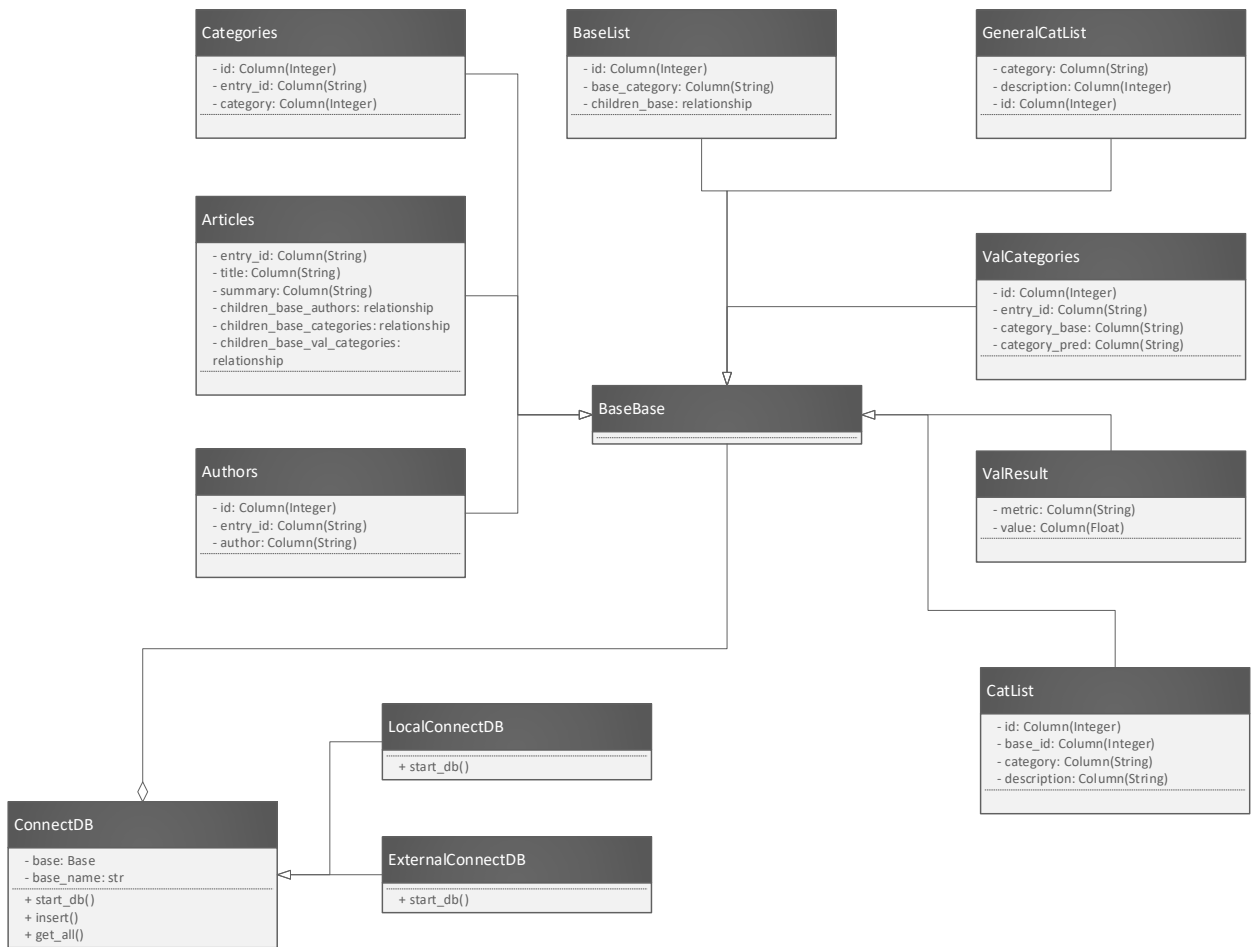


Рисунок 4.3 – Діаграма класів додатку доступу до бази даних

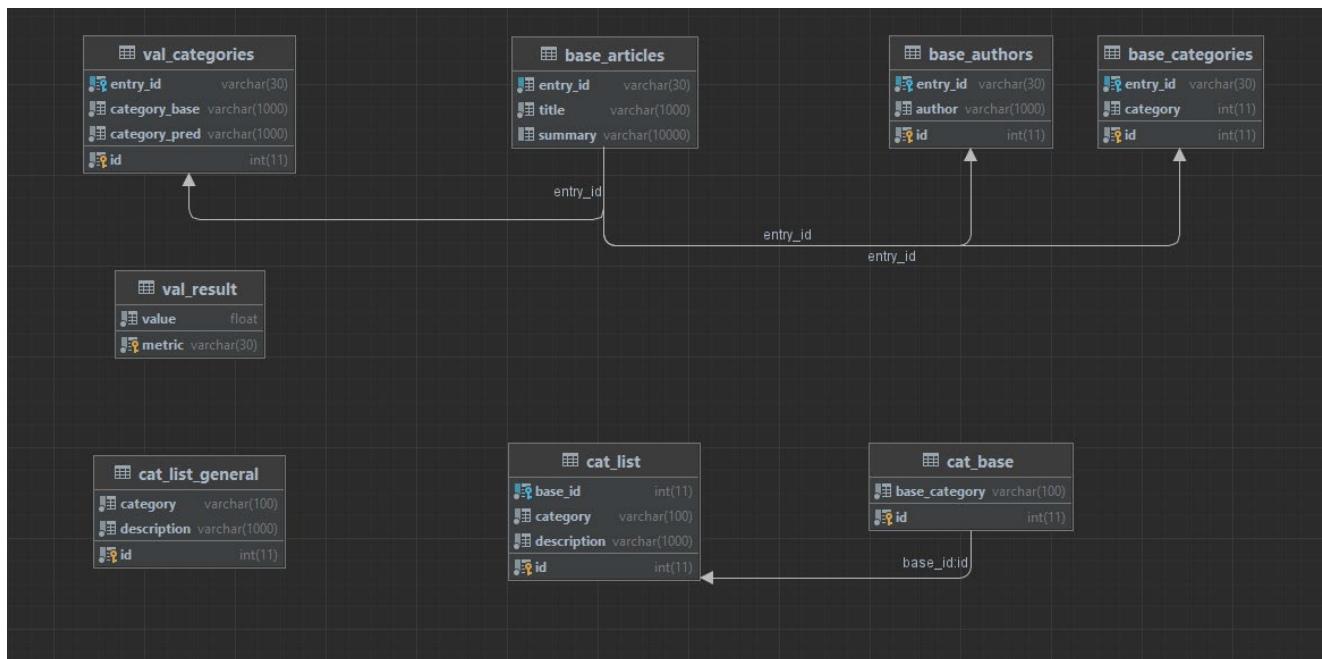


Рисунок 4.4 – Діаграма бази даних

Як і класи, що напряду відповідають за зв'язок з базами даних, так і класи, що відповідають за створення таблиць представлені на рисунку 4.3.

### 4.3 Модуль вивантаження даних з бібліографічних джерел

Модуль вивантаження даних з бібліографічних джерел повністю охоплює процес запиту, завантаження і передачі на зберігання текстових масивів з потенційно великої кількості джерел, які для цього мають використовувати інтерфейс Loader. На момент реалізації проекту був розроблений лише клас запиту даних з сервісу arXiv через його достатність та повну відповідність встановленим критеріям.

Завантаження може відбуватись у будь-яку базу даних, що наслідує клас ConnectDB, а саме у локальну (SQLite) чи зовнішню (MariaDB) базу даних.

Вивантажені тексти автоматично розподіляються по трьом таблицям з зв'язком через `entry_id`, а саме по таблиці текстів, таблиці авторів і таблиці категорій.

Запит даних через відповідну бібліотеку проходить через аналогічний формат запиту, як і через arXiv API. За допомогою префіксів можна отримати результати по:

- `ti` – Заголовок,
- `au` – Автор,
- `abs` – Анотація,
- `co` – Коментар,
- `jr` – Посилання на журнал,
- `cat` – Категорія,
- `rn` – Номер звіту,
- `id` – Id (замість цього використовується список `id_list`),
- `all` – Все вищесказане.

Наприклад `“all:electron”` чи `“cat:cs.DC”`. Таким чином було вивантажено усі тексти з бази на поточний момент часу.

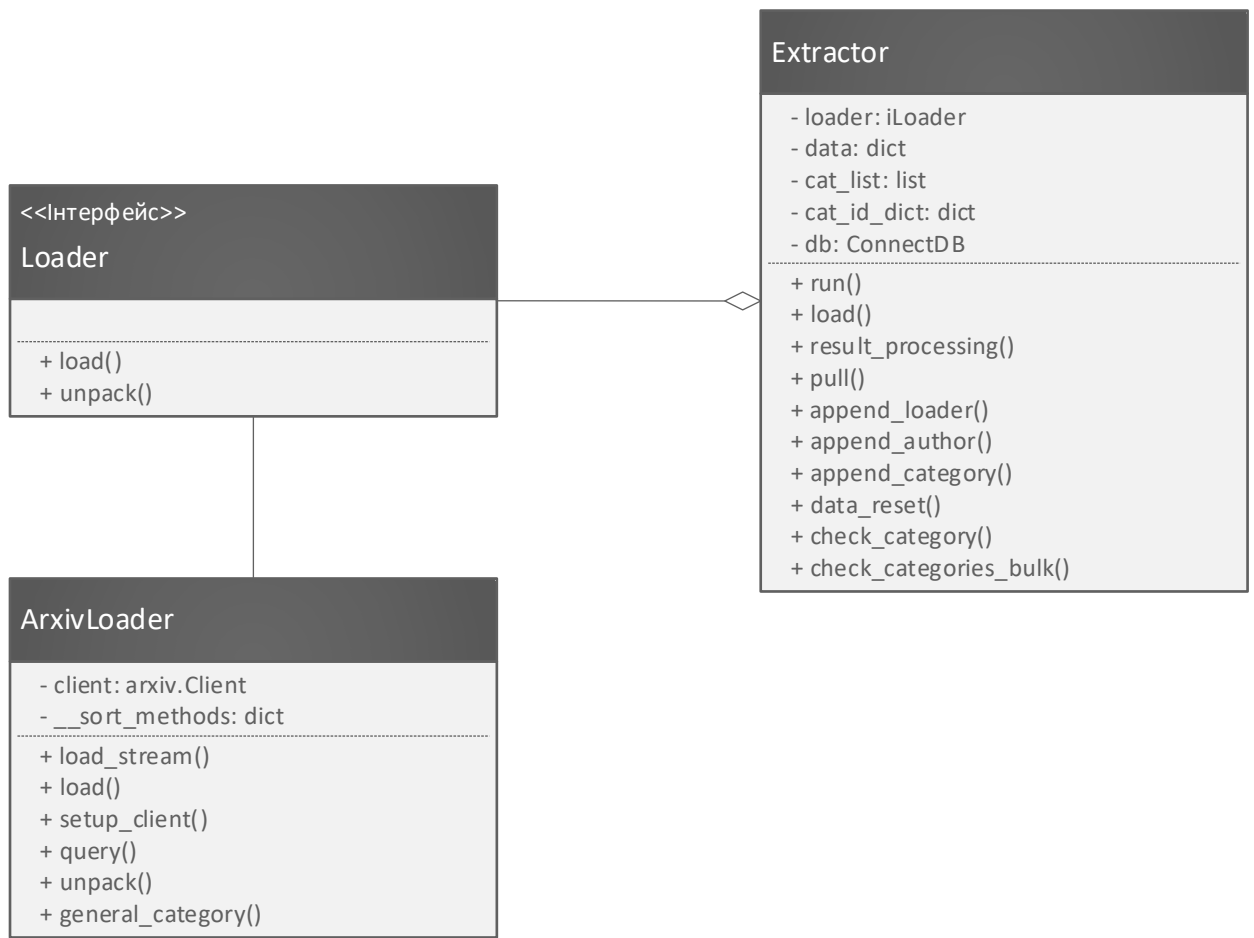


Рисунок 4.5 – Діаграма класів модулю вивантаження даних з бібліографічних джерел

На рисунку 4.5 наведена діаграма класів у обмеженому вигляді для розглянутого вище модуля системи.

За допомогою інтерфейсу реалізується універсальне підключення різних джерел наукових текстових даних для подальшої первинної обробки, зберігання та навчання нейронної мережі.

## 4.4 Модуль попередньої обробки текстів

Модуль попередньої обробки текстів це відносно малий модуль, який створює пайплайн обробки тексту, тобто послідовність за якою функцією виконують дії над текстовою інформацією одна за одним.

Відповідно модуль безпосередньо обробки тексту (Processor) складається з двох класів, TextCleaner та TextPreprocessor. Перший, TextCleaner, очищає текст від усієї зайвої символічної інформації, проводить базове спрощення слів по словнику, наприклад could've – could have, aren't – are not, ain't – is not, couldn't – could not, 'cause – because, can't – cannot, виправляє помилки у написанні, такі як colour – color, travelling – traveling, theatre – theater, centre – center, favourite – favorite, counselling – counseling і проводить іншу підготовку задля передачі наступному етапу обробки тексту максимально очищені та знешумлені вхідні дані.

Після цього етапу очищений текст передається у TextPreprocessor, де він проходить токенізацію лематизацію через бібліотеку Stanza [20], та видалення стоп-слів через їх збірник у бібліотеці NLTK [14]. Це доволі повільний процес, який може займати тижні робочого часу, тому для зменшення часових рамок рекомендується використовувати GPU, у випадку даного проекту – Tesla P100 16GB.

Так як нейронна модель, що була використана, не потребує збереження контекстної інформації про речення, оригінальні форми слів чи спеціальні символічні позначки – такий формат обробки є доцільним. Але для інших типів нейронних мереж він може не підходити, тому завжди є можливість змінювати параметри обробки кожного з двох модулів, видаляти чи додавати інші пункти обробки.

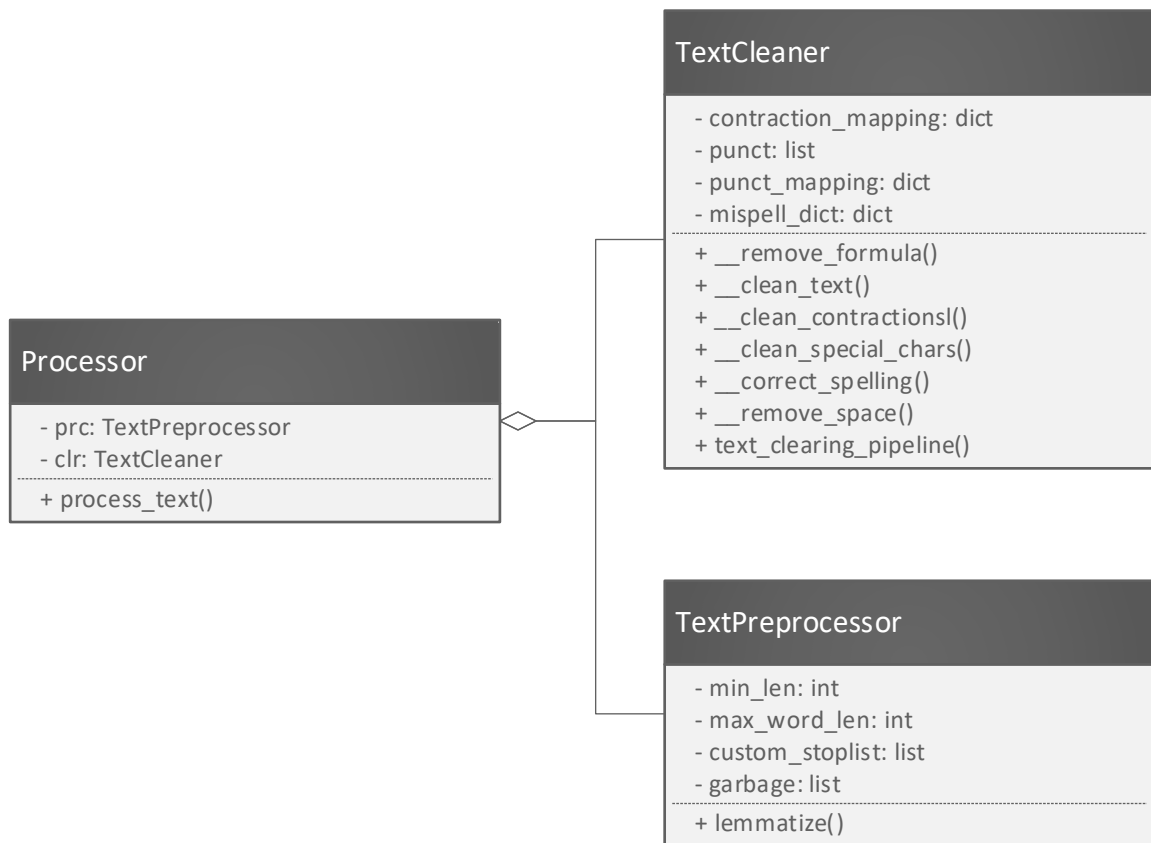


Рисунок 4.6 – Діаграма класів модулю попередньої обробки текстів

Цей модуль використовується як і для підготовки текстів до формату, необхідного для навчання, так і при аналізі текстів у реальному часі.

## 4.5 Модуль конвертації текстів у формат для навчання

Для конвертації текстів у формат для навчання використовується відповідний модуль. Він за допомогою модулю попередньої обробки текстів та бібліотеки

pandas конвертує тексти у відповідний формат DataFrame, який після відвантажується у файл.

У існуючих технічних обмеженнях конвертація текстів виконується у сервісі Google Colab, у форматі Jupyter Notebook.

## 4.6 Модуль навчання нейронної мережі

Даний модуль використовується для навчання нейронної мережі та існує лише в лінійному вигляді команд, які послідовно виконуються, Jupyter Notebook. За допомогою бібліотек TensorFlow і Keras відбувається саме навчання, бібліотеки sklearn, matplotlib, pandas та numpy виконують допоміжні функції. У цьому модулі закладається архітектура нейронної мережі, а саме:

- TextVectorization – шар для векторизації тексту, який на вхід отримує текстові дані (набір слів), і на вихід віддає вектор даного тексту. Навчається окремо від інших шарів нейронної мережі на усіх текстах. Має проходити перенавчання при додаванні нових можливих слів.
- Перший захований щільно-пов'язаний шар з 1024 нейронами, який застосовує функцію активації випрямленої лінійної одиниці (ReLU).
- Другий захований щільно-пов'язаний шар з 512 нейронами, який застосовує ReLU.
- Третій захований щільно-пов'язаний шар з 512 нейронами, який застосовує ReLU.
- Четвертий захований щільно-пов'язаний шар з 512 нейронами, який застосовує ReLU.

- Останній шар, кількість виходів, у якому рівна кількості необхідних нам категорій. Значення кожного виходу відповідає кожній вірогідності того, що текст відноситься до цієї категорії. Як функцію активації застосовує сигмовидну функцію.

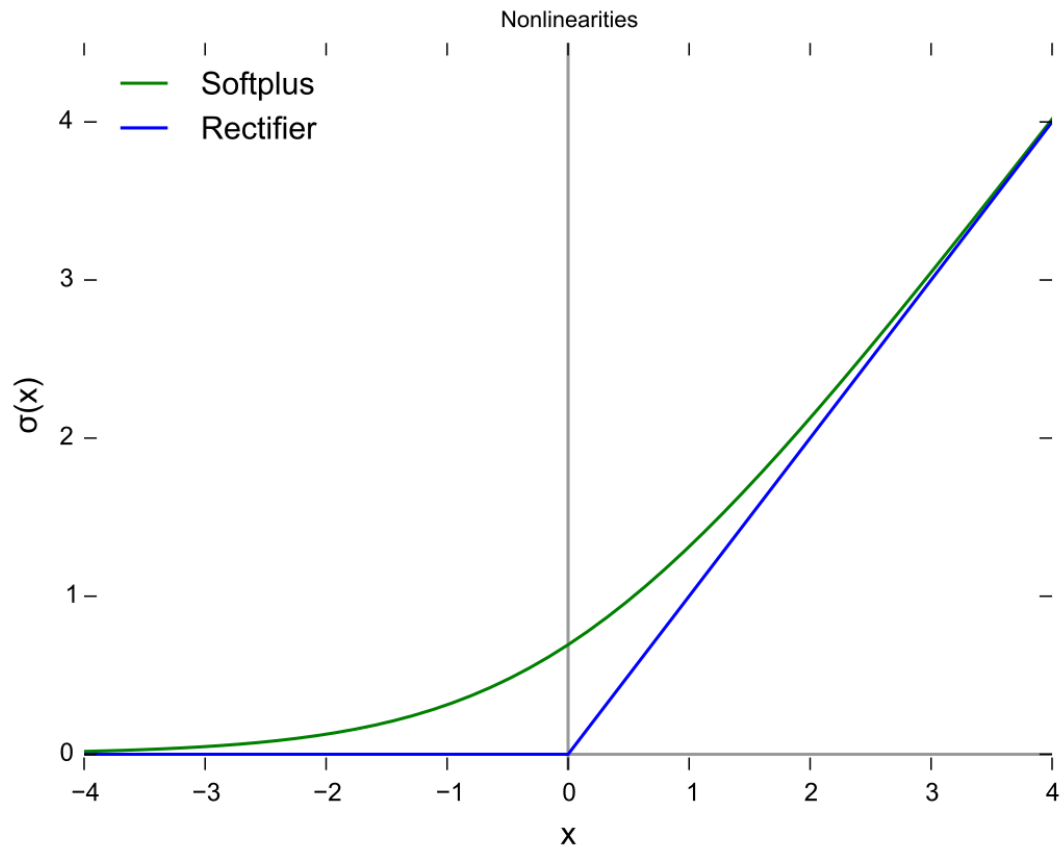


Рисунок 4.7 – Графік активації ReLU

Вигляд функції ReLU можна бачити на рисунку 4.6. Навчання глибоких нейронних мереж було неможливим з використанням сигмовидних функцій активації. Саме ReLU сприяла навчанню глибоких мереж, і з тих пір вона загально використовується як функцію активації для прихованих шарів. Її перевага над сигмовидними функціями полягає у зменшенні кількості часу для навчання,

складності у перенасичені, іншими словами, як тільки сигмовидна частина досягає лівого або правого плато, майже безглуздо здійснювати зворотний прохід через неї, оскільки значення похідної дуже близька до 0, ReLU насичується лише тоді, коли вхідний сигнал менше 0. І навіть це насичення можна усунути за допомогою нещільних ReLU. Для дуже глибоких мереж насичення перешкоджає навчанню, тому ReLU забезпечує гарну альтернативу, а також при стандартній активації градієнт сигмоїди становить деяку частку від 0 до 1. Якщо у Вас багато шарів, вони множаться і можуть дати загальний градієнт, який є експоненціально малим, тому кожен крок градієнтного спуску створюватиме лише невелика зміна ваг, що призводить до повільної конвергенції (проблема зникаючого градієнта).

З цього ми отримаємо, що модель, навчена за допомогою ReLU, швидко сходиться і, отже, займає набагато менше часу порівняно з моделями, натренованими на сигмовидних функціях і продуктивність моделі значно краща при навчанні за допомогою ReLU.

Для навчання текстовий масив поділяється на 80% текстів на навчання, 10% текстів для перевірки навчання і 10% текстів для валідації. Тексти для перевірки беруть участь у навчанні, на відміну від повністю незнайомих текстів для валідації, що робить процес валідації доволі точним. Але через наявність декількох категорій у одному тексті та великою різницею між їх представленістю ми не можемо допомогти розподілити ці категорії на рівних в усіх трьох частинах, тому можливі малоймовірні ситуації коли якість навчання напряду залежить від випадкового процесу розподілення, що було подолано декількома ітераціями розподілення і порівнянням результатів.

Навчання відбувалось у 10 епох, кожна з яких демонструвала кращі результати на текстах для перевірки, але гірші результати на текстах для валідації.

У якості функції втрат було використана бінарна кросентропія. Бінарна кросентропія – це функція втрат, яка використовується у завданнях двійкової класифікації. Це завдання, які відповідають на запитання лише з двома варіантами

(так чи ні, А чи Б, 0 чи 1, ліворуч чи праворуч). На кілька незалежних таких запитань можна відповісти одночасно, як у класифікації з кількома мітками або в бінарній сегментації зображень, що і було використано.

Формально ця втрата дорівнює середньому втрат категорійної кросентропії для багатьох завдань з двома категоріями.

У якості оптимізатора був обраний алгоритм Адама. Оптимізація Адама – це метод стохастичного градієнтного спуску, який базується на адаптивній оцінці моментів першого та другого порядку. За даними [22] метод “ефективний у обчислювальному відношенні, потребує малої пам’яті, інваріантний до діагонального масштабування градієнтів і добре підходить для задач, які є великими з точки зору даних/параметрів”, що повністю відповідає потребам поточного проекту по класифікації наукових текстових масивів.

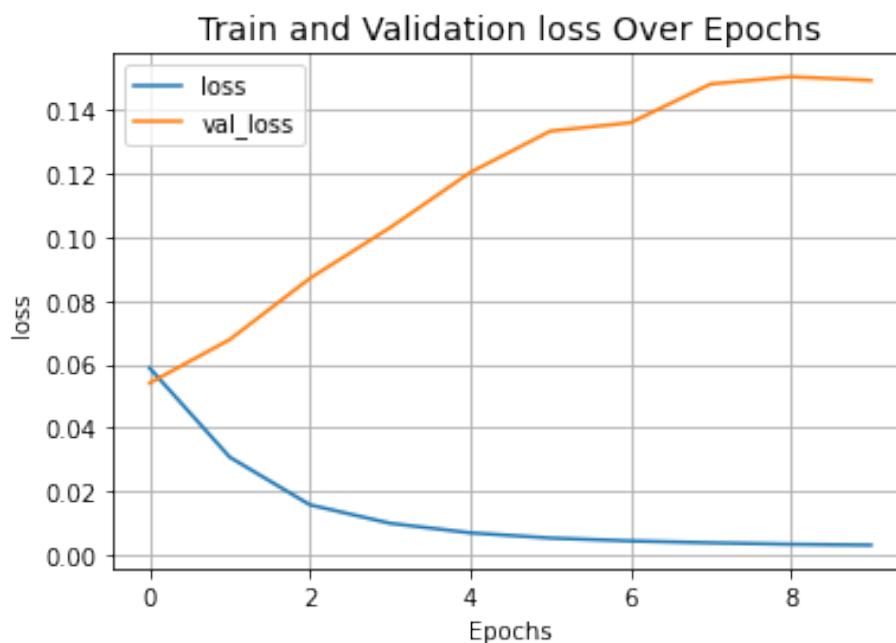


Рисунок 4.8 – Графік порівняння функції втрат для текстів перевірки та валідації

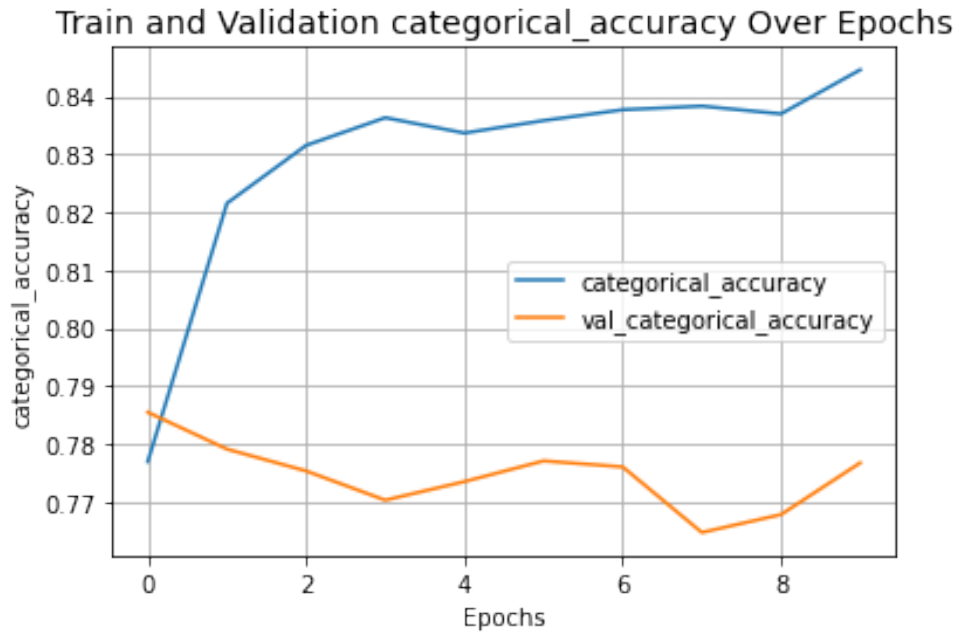


Рисунок 4.9 – Графік порівняння точності для текстів перевірки та валідації

На рисунку 4.8 та рисунку 4.9 наведені візуалізовані метрики навчання нейронної мережі. За допомогою них можна обрати оптимальний етап навчання, який краще за інших справляється з поставленою задачею.

## 4.7 Автономний модуль обробки текстів

Автономний модуль обробки текстів є базовою одиницею комплексної системи, яка сама по собі може обробляти данні. Він складається з модулю обробки текстів, моделі нейронної мережі та під'єднаний до системи комунікації з іншими компонентами RabbitMQ. При старті через параметри командної стрічки існує можливість вказувати назву черг, з якої він буде брати тексти на обробку, та у яку

він буде надавати результати. Від цього також залежить алгоритм роботи з метаданими, які надходять разом з текстом.

Робота модулю опирається на постійне опитування черги отримання і реакцію `on_message()` на кожне повідомлення. Постійне опитування встановлюється бібліотекою `pika`, та підтримується за допомогою сигналів життєдіяльності.

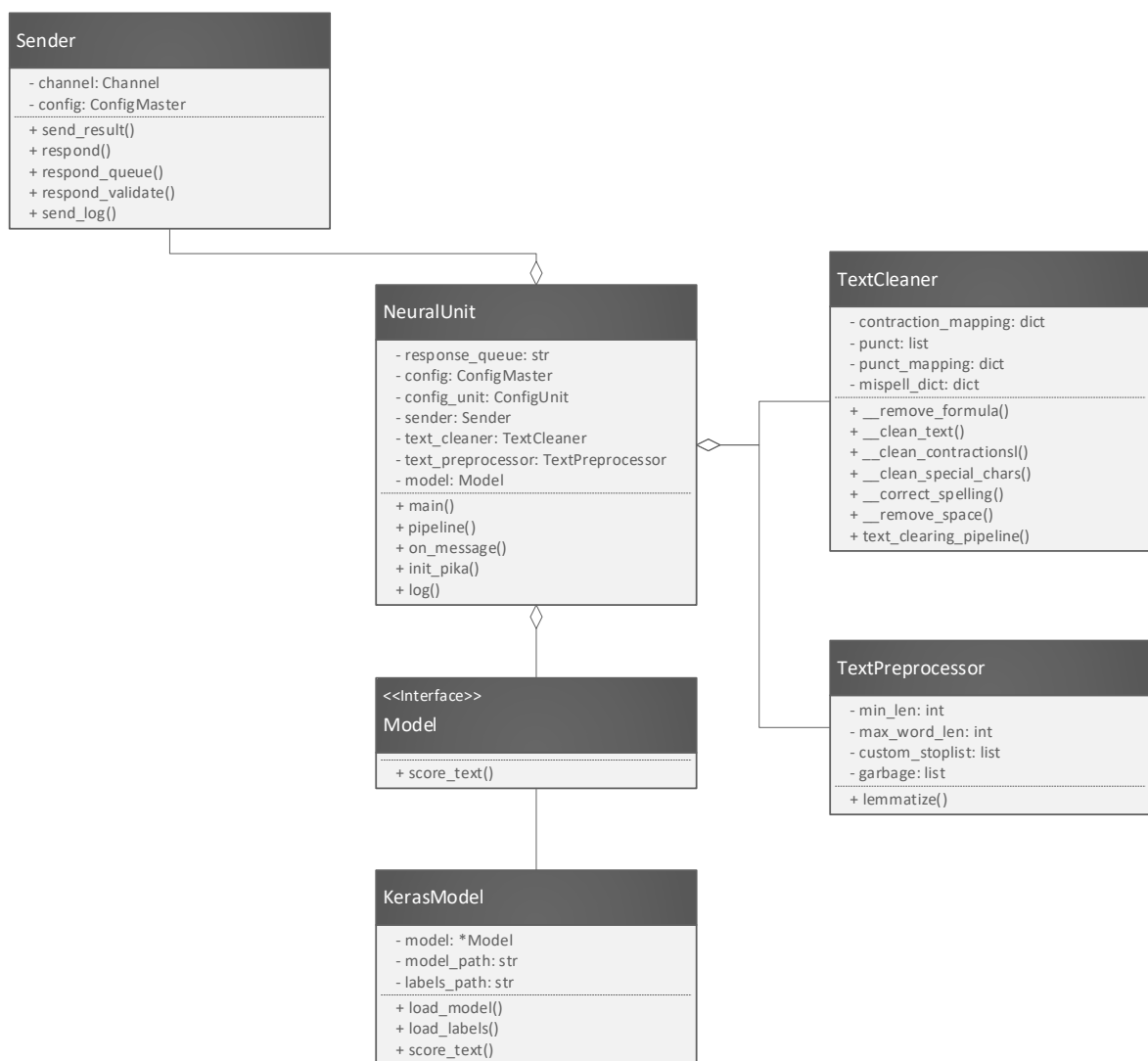


Рисунок 4.10 – Діаграма класів автономного модулю обробки текстів

Усі відповіді, які надсилаються, йдуть через клас Sender саме для цього модулю.

Підтримується робота з будь-якими нейронними моделями через інтерфейс Model.

## **4.8 Модуль керування та нагляду автономними модулями обробки текстів**

Цей модуль є основною одиницею адміністрування системи, яка з'єднана з інтерфейсом адміністратора, виконує його команди та звітує про поточний стан роботи кожної з частин системи за допомогою системи комунікації з іншими компонентами RabbitMQ.

Для своєї роботи він потребує допоміжний модуль Clock, який працює як тактовий генератор і встановлює частоту оновлення інформації. У наявних умовах оптимальна частота – оновлення раз у 2 секунди.

Він виконує такі функції, як:

- Моніторинг завантаження процесора кожним з юнітів.
- Моніторинг завантаження пам'яті кожним з юнітів.
- Моніторинг стану кожного з юнітів.
- Перезавантаження юнітів, автоматичне та за запитом.
- Пауза та відновлення роботи юнітів за запитом.
- Зміна черги оновлення через параметри командної стрічки за запитом.
- Зміна черги оновлення через параметри командної стрічки за запитом.
- Додавання і видалення юнітів за запитом.

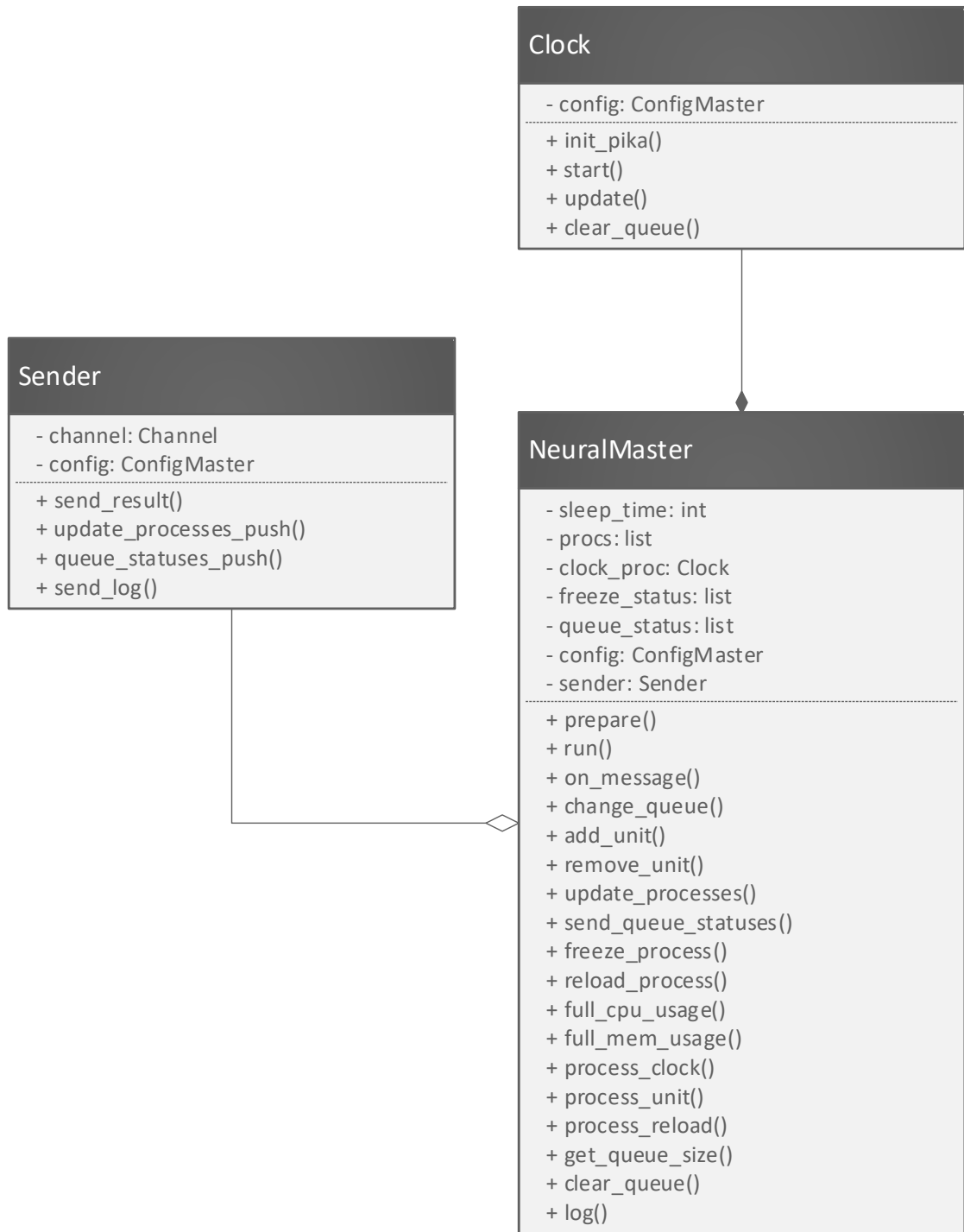


Рисунок 4.11 – Діаграма класів модуля керування і нагляду автономними модулями обробки текстів

На рисунку 4.11 наведена діаграма класів для розглянутого вище модуля системи.

## **4.9 Користувацькі інтерфейси**

Усього системою передбачено два типи користувацького інтерфейсу: інтерфейс адміністратора та аналітика. Обидва написані за допомогою фреймворка PyQt версії 5.

Інтерфейс адміністратора підтримують три потоки оновлення інтерфейсу, які забирають інформацію про стан юнітів, про стан черг і лог роботи та обробки.

Інтерфейс користувача підтримує один потік, який виконує обчислення та оновлює інформацію у базі даних.

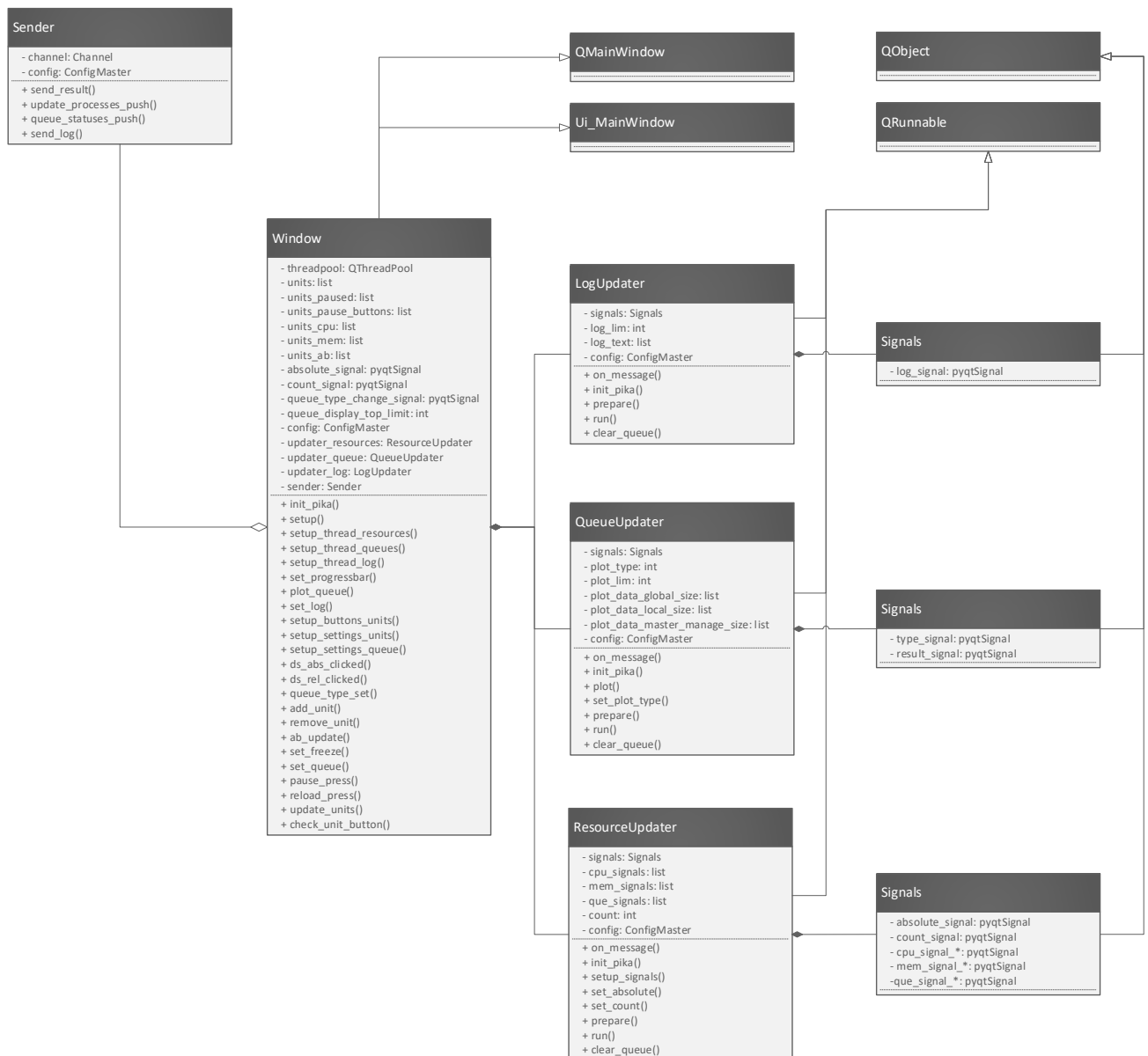


Рисунок 4.12 – Діаграма класів інтерфейсу адміністратора

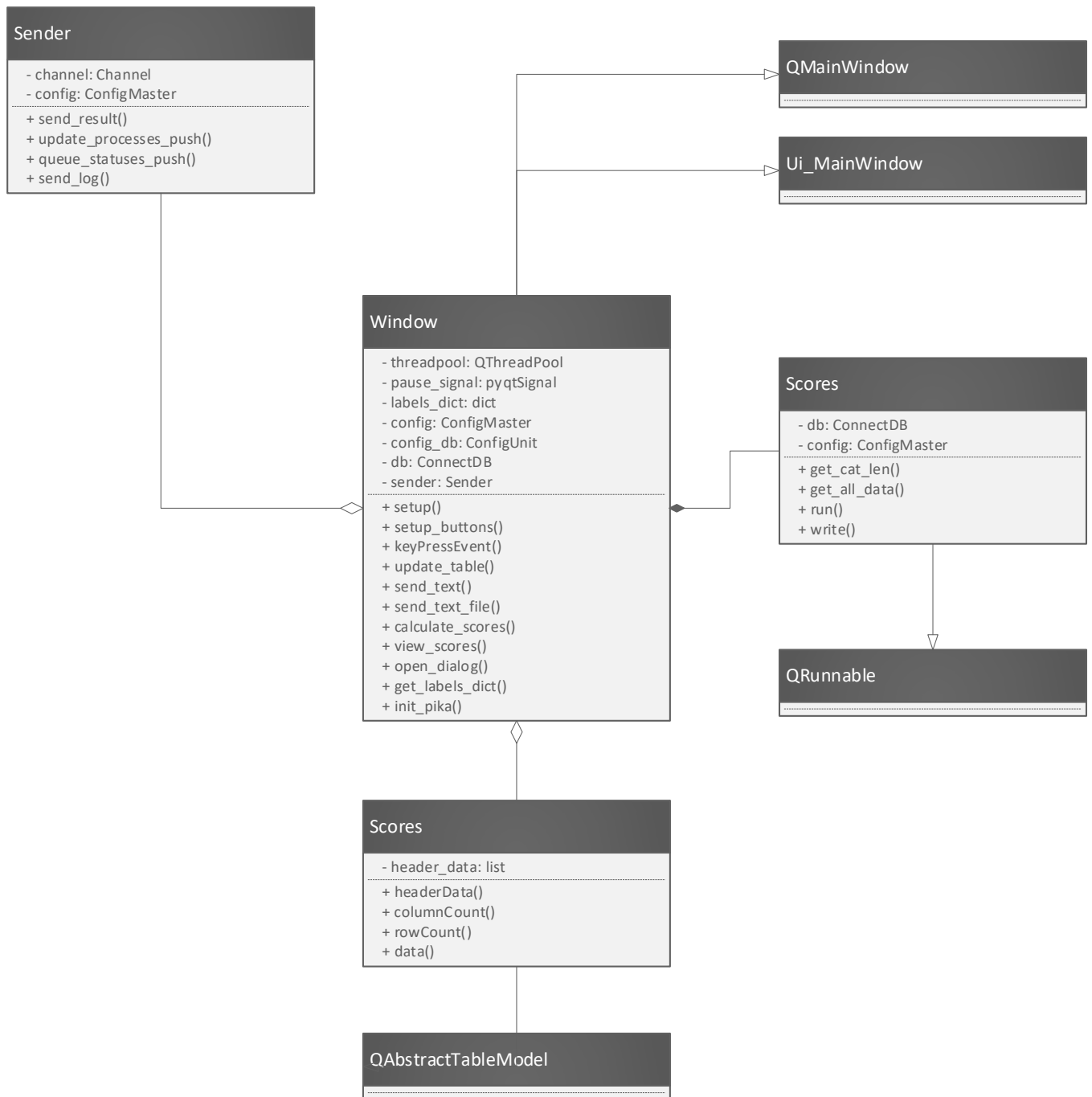


Рисунок 4.13 – Діаграма класів інтерфейсу користувача

Через особливості роботи PyQt необхідно було створити класи-нащадки QRunnable для зберігання сигналів, які можуть бути ініціалізовані лише при старті програми.

Для відображення таблиці у користувацькому інтерфейсі був створений нащадок базового класу моделі таблиці.

## 5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для того щоб програмний продукт працював належним чином необхідно дотримуватися основних вимог при інсталяції та приймати рекомендації щодо користування додатками.

### 5.1 Керівництво з встановлення системи

Для автоматизації розгортання серверної частини використовується інструментарій Docker, а саме Docker Compose. Compose це інструмент для запуску багатоконтейнерних систем Docker. За допомогою Compose можна використовувати файл YAML для налаштування служб, потім за допомогою однієї команди запускаються усі служби зі своєї конфігурації. Compose працює в усіх випадках: підготовка, розробка, тестування.

Для розгортання серверної частини необхідно здійснити таке:

- 1) Завантажити застосунок Docker на сервер.
- 2) Завантажити контейнери з репозиторію – `docker pull`.
- 3) Завантажити конфігураційний файл відповідно до серверу.
- 4) Запустити проект – `docker compose -f ~/project.yml up`.

Для розгортання клієнтської частини необхідно:

- 1) Завантажити інтерпретатор Python 3.9.
- 2) По необхідності створити віртуальне середовище.
- 3) Встановити усі необхідні пакети – `pip install -r requirements.txt`

4) Запустити файл main\_user.py чи main\_admin.py для запуску інтерфейсу аналітика чи адміністратора відповідно.

## 5.2 Інтерфейс адміністратора

Після запуску програмного забезпечення адміністратора зустрічає подібний екран, який вказано на рисунку 5.1.

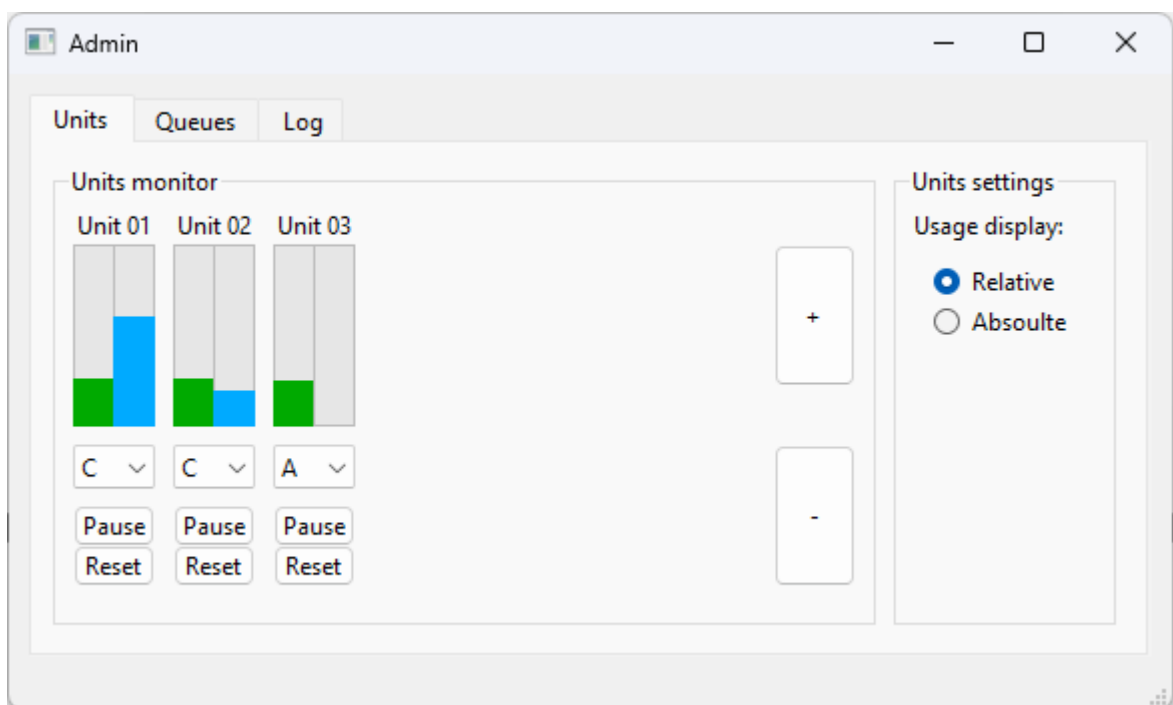


Рисунок 5.1 – Перший екран інтерфейсу адміністратора

На першій вкладці адміністратор бачить кількість запущених юнітів, використання ними ресурсів серверу (пам'ять відображена зеленим стовпчиком, використання процесору зеленим стовпчиком).

За допомогою кнопок “+” та “-” можна змінювати кількість запущених процесів у більшу та меншу сторону. Мінімальний ліміт – 0 процесів, максимальний – 7 процесів. Необхідно зауважити про те, що найбільша швидкодія на використанні ресурси досягається при кількості юнітів, яка дорівнює трьом.

Використовуючи кнопку “Pause” можна поставити певний юніт на паузу. При цьому цей процес не буде використовувати процесор, але буде займати системну пам'ять. Також кнопка “Pause” змінить свою назву на кнопку “Start”. Налаштування збережуться навіть при перезапуску додатка.

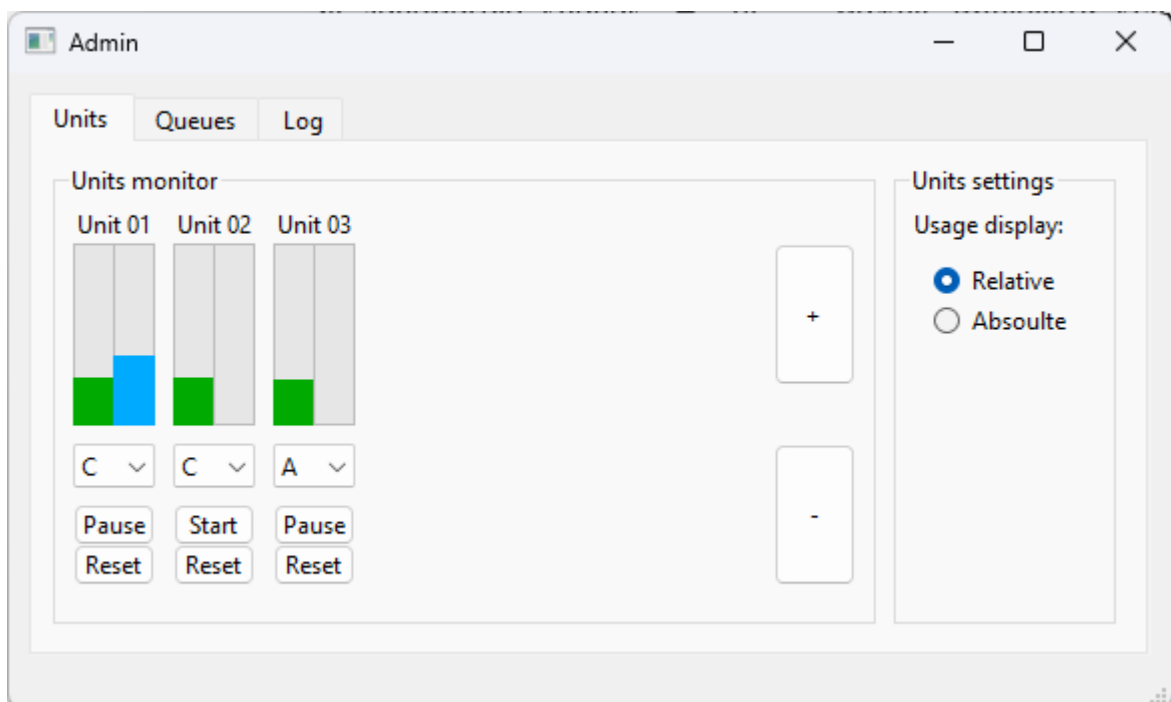


Рисунок 5.2 – Інтерфейс адміністратора, юніт 2 стоїть на паузі

Використовуючи кнопку “Reset” можна повністю перезапустити процес у примусовому порядку. Такий перезапуск ніяк не зачепить цілісність даних, текст на обробці перезапущеного юніту повернеться у чергу.

За допомогою меню вибору A/B/C можна обрати чергу текстів на обробку. Черга A відповідає за чергу користувацького інтерфейсу. Черга B відповідає за чергу завантаження текстів у базу даних. Черга C відповідає за чергу на валідацію навчання. При цьому юніт буде перезапущений з новими налаштуваннями. Налаштування також зберігаються на сервері та будуть відновлені після перезапуску додатку.

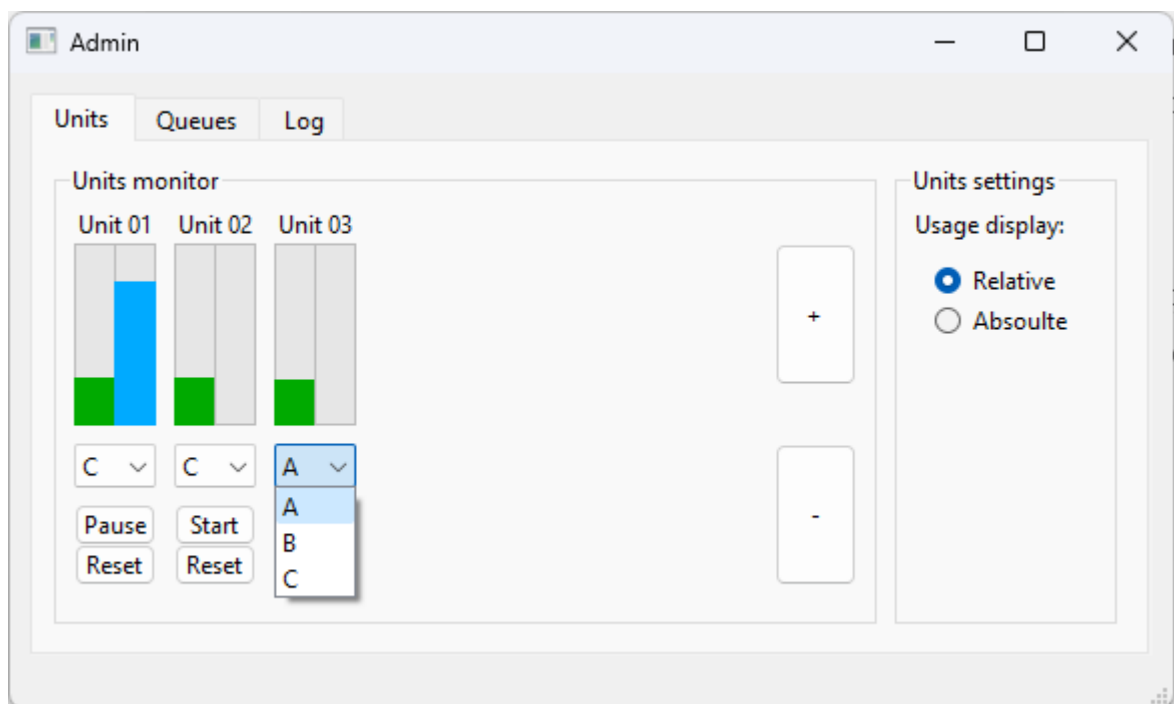


Рисунок 5.3 – Інтерфейс адміністратора та меню вибору черги

За допомогою радіокнопок у полі Usage display можна обрати тип відображення інформації про навантаження на сервер. Режим Relative обчислює

доступне вікно ресурсів для кожного з юнітів і відображає їх використання відносно цього числа, режим Absolute натомість дає абсолютні показники використання ресурсів. Як можна бачити з вікна на рисунку 5.5, при збільшенні кількості юнітів, зменшується вільний простір для кожного з юнітів. Юніти 3 – 6 не використовують процесор через відсутність інформації у відповідній черзі A.

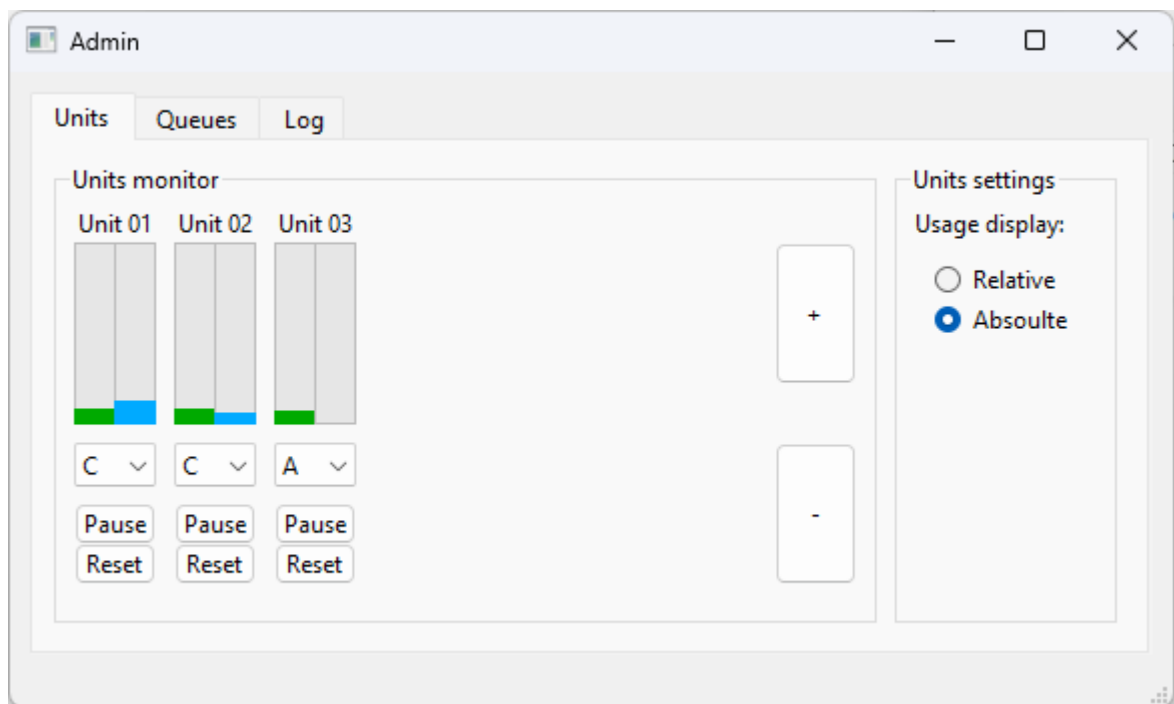


Рисунок 5.4 – Інтерфейс адміністратора, абсолютне відображення використанню ресурсів

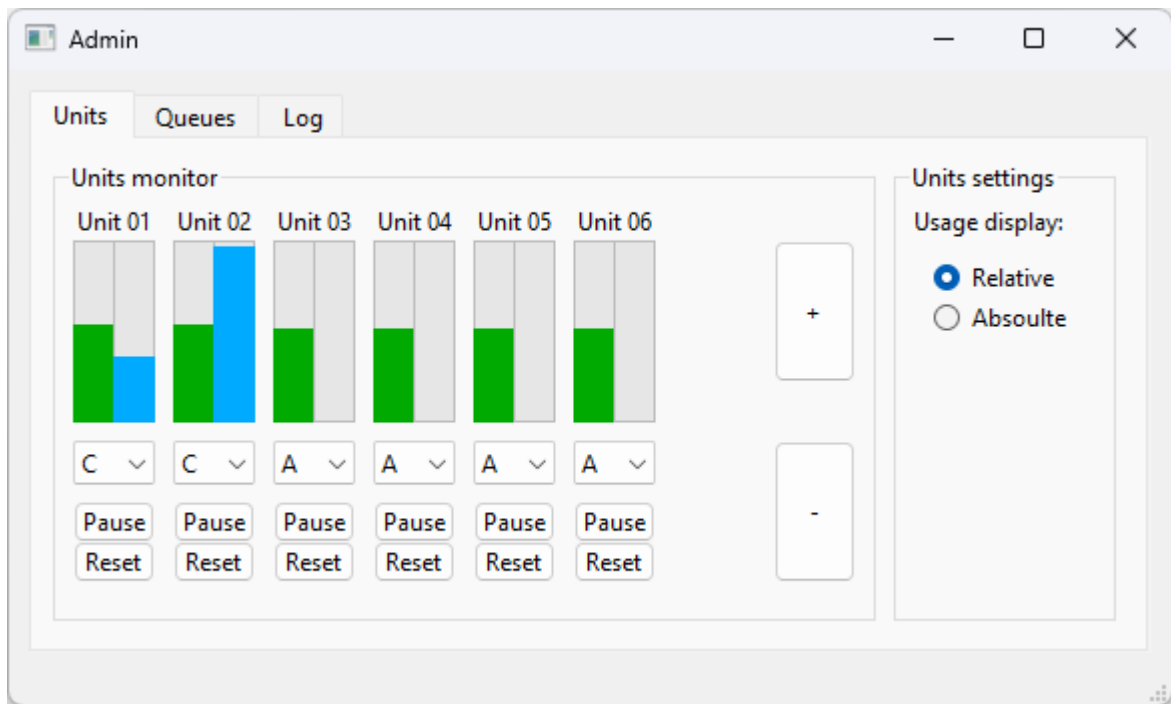


Рисунок 5.5 – Інтерфейс адміністратора, відносне відображення, 6 юнітів

На другій вкладці Queues відображена інформація про стан обраних черг та їх наповненість повідомленнями. Усього доступні такі три черги:

- Global – черга текстів на валідацію або на завантаження у базу даних.
- Personal – черга текстів аналітиків.
- Command – черга команд до модулю керування і нагляду автономними модулями обробки текстів.

На графіку відображена зміна черги відносно поточного моменту і до 50 часових одиниць у минуле. Кожна часова одиниця – це час опитування модулю керування і нагляду автономними модулями обробки текстів, у нашому випадку 2 секунди.

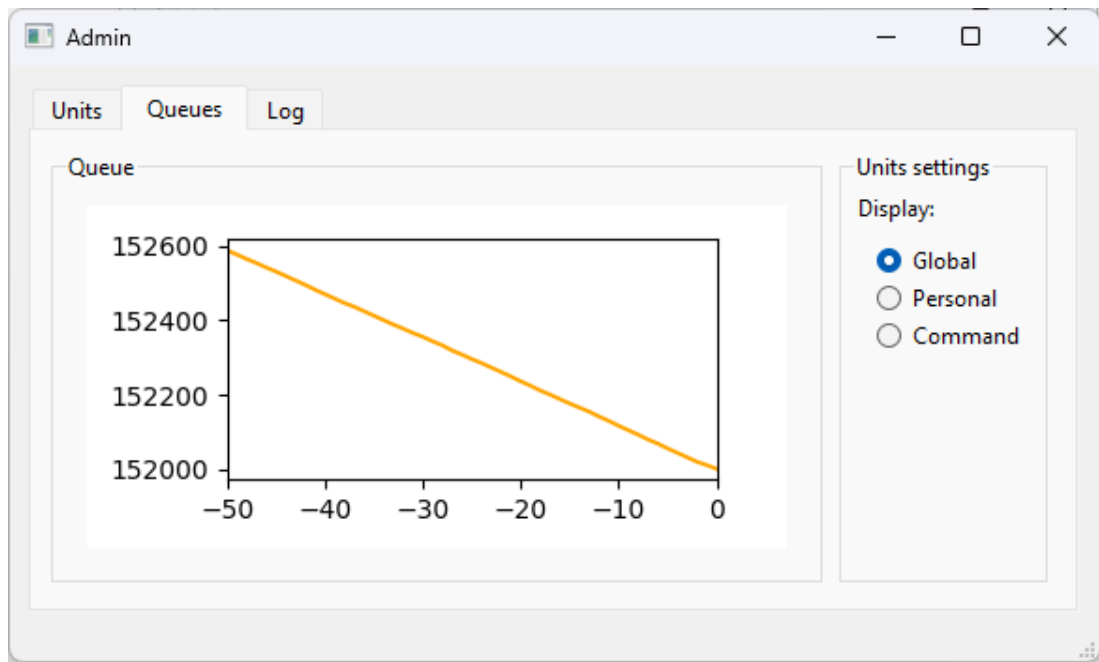


Рисунок 5.6 – Інтерфейс адміністратора, черга текстів на валідацію



Рисунок 5.7 – Інтерфейс адміністратора, черга текстів користувачів, додано 1000 текстів

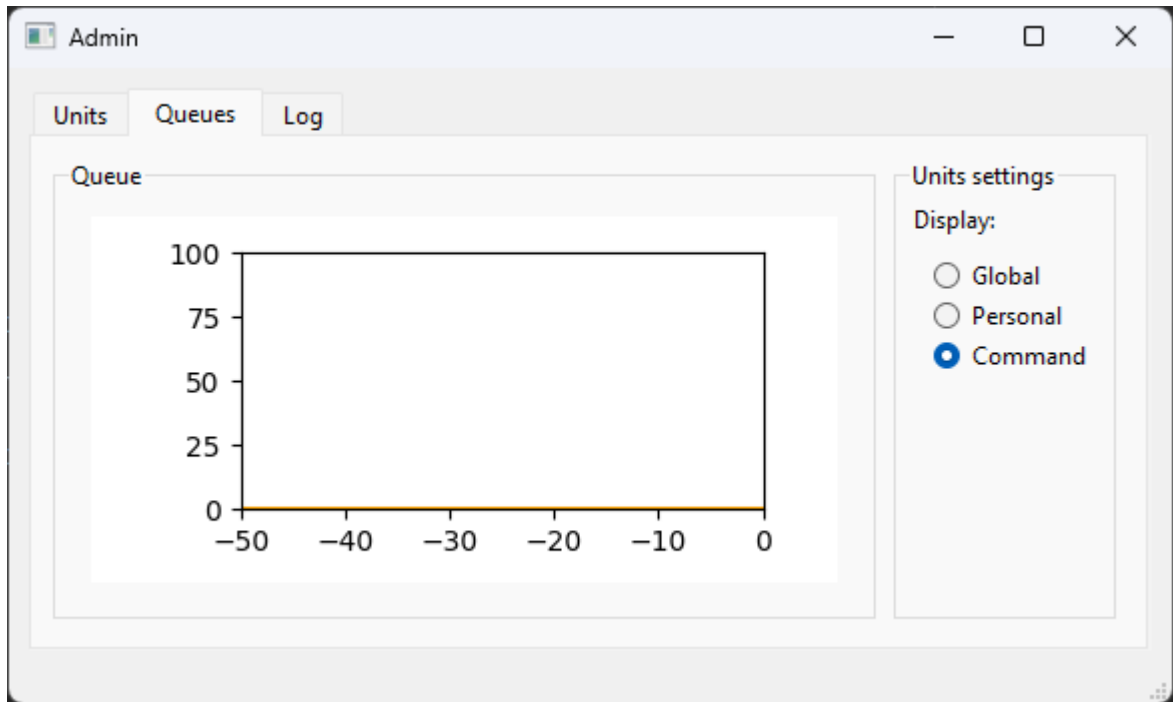


Рисунок 5.8 – Інтерфейс адміністратора, черга команд

Порожня черга команд показує те, що модуль керування не перевантажений та обробляє їх максимально швидко, що можна бачити на рисунку 5.8.

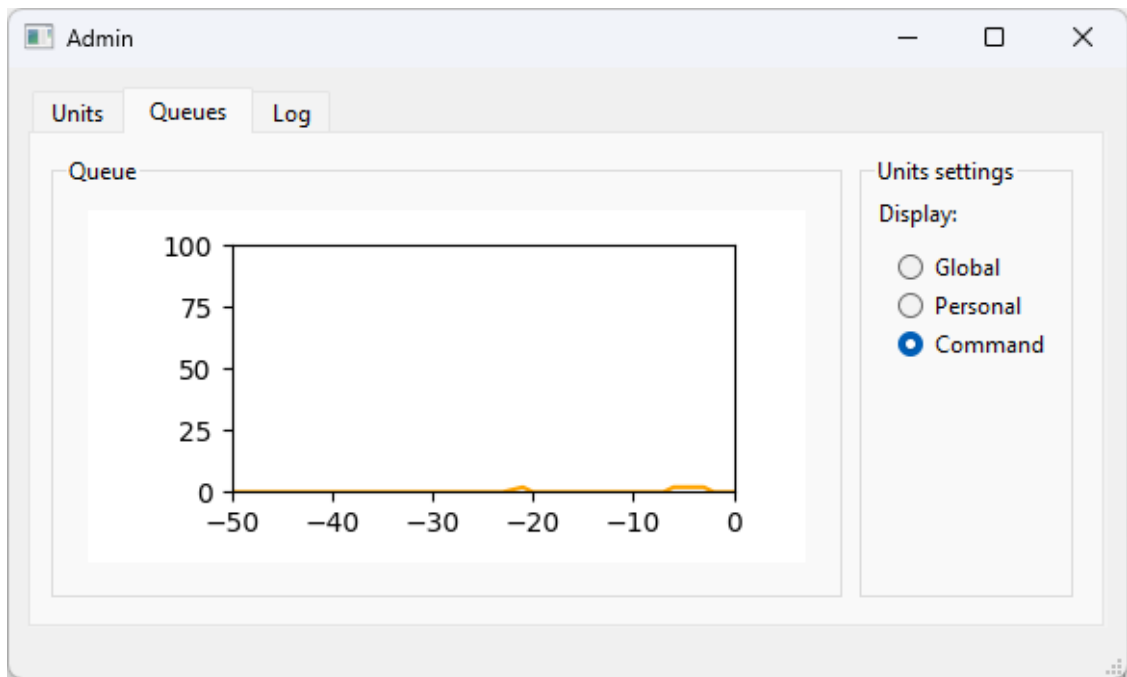


Рисунок 5.9 – Інтерфейс адміністратора, легке перевантаження черги команд

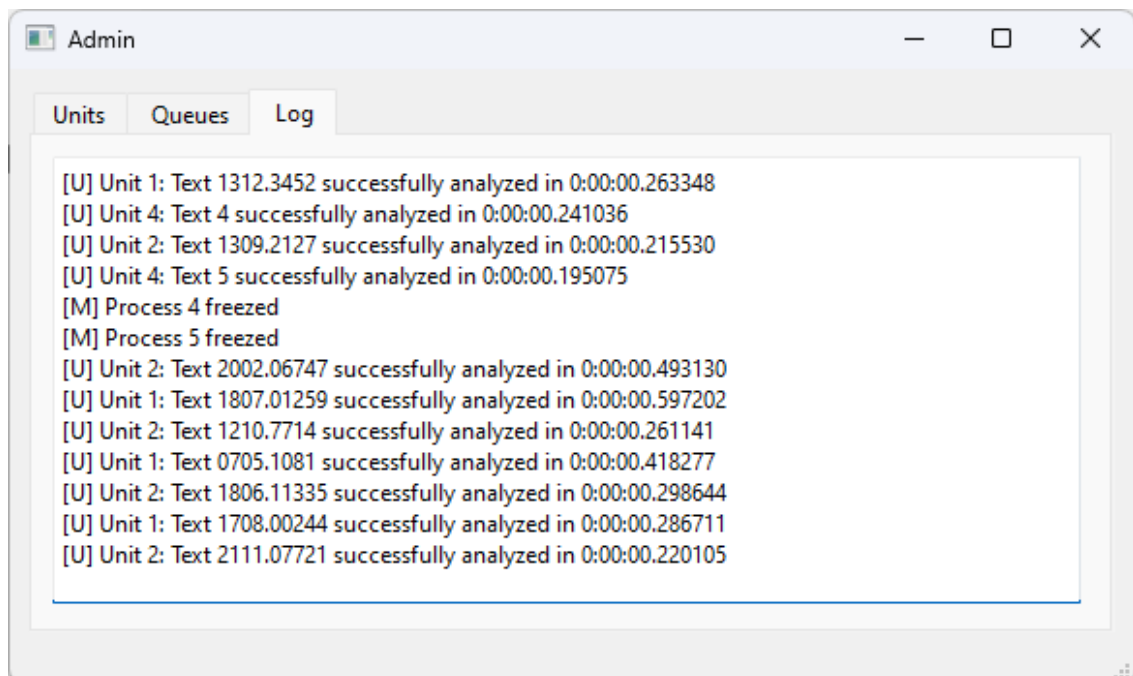


Рисунок 5.10 – Інтерфейс адміністратора, вкладка повідомлень

На третій вкладці Log знаходиться вивід повідомлень від усіх компонентів системи. На рисунку 5.10 можна побачити повідомлення про те, як і від юнітів, про успішний аналіз текстів із вказаним id і часом обробки певного тексту, так і повідомлення від наглядача, що заморозив 4 та 5 юніт. Як можна бачити, повідомлення від них більше не поступають.

### 5.3 Інтерфейс аналітика

Інтерфейс аналітика зустрічає користувача подібним екраном до того що вказаний на рисунку 5.11.

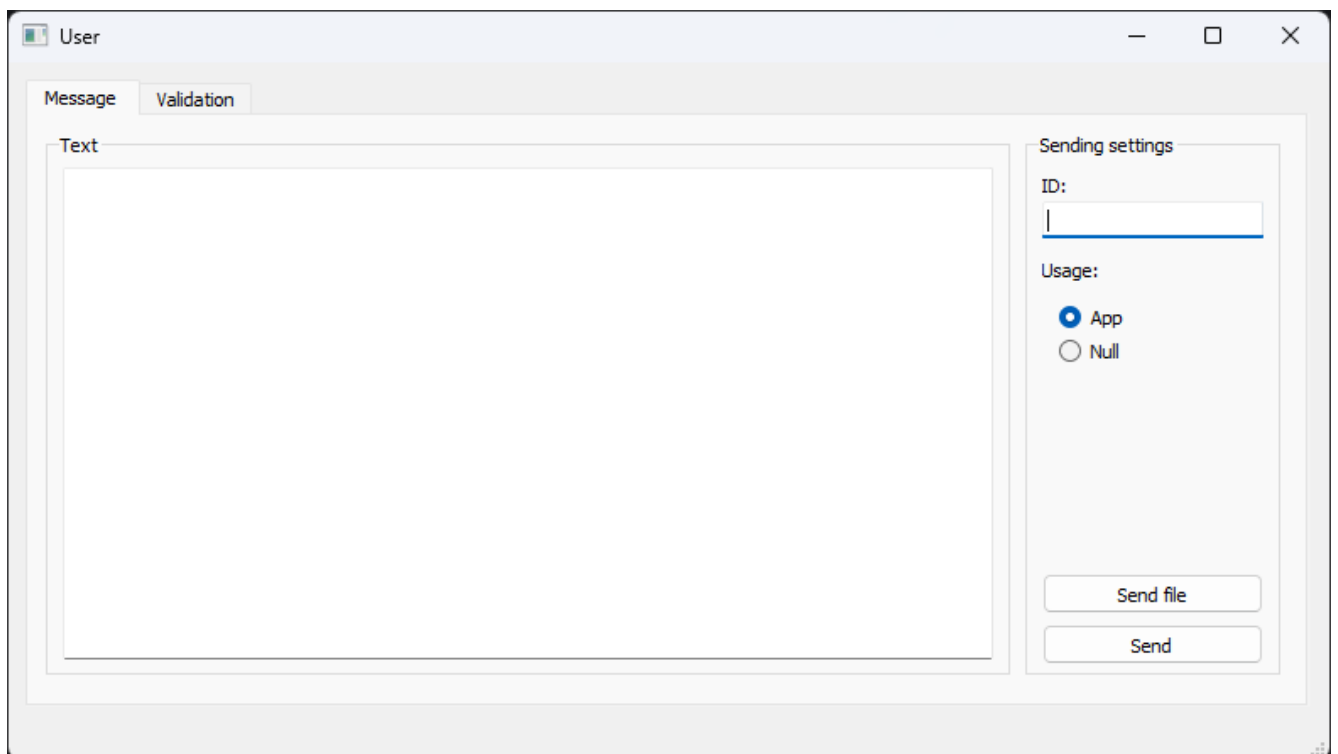


Рисунок 5.11 – Інтерфейс користувача

У поле текст вводиться текст, який буде в подальшому проаналізовано. У поле id вводиться id тексту, яке необхідно через багатопотоковість системи аналізу для синхронізації. Радіокнопки App та Null обирають адрес відправки результатів – вибравши App аналітик отримує результат саме у цю програму, вибравши Null результат не буде відіслано, це необхідно для відладки та демонстрацій.

Текст з полів відправляється кнопкою Send. Також присутня можливість завантажити текст з файлу .csv, де перша колонка – ідентифікатори, а друга – тексти. Радіокнопки App та Null також впливають на тексти з файлів.

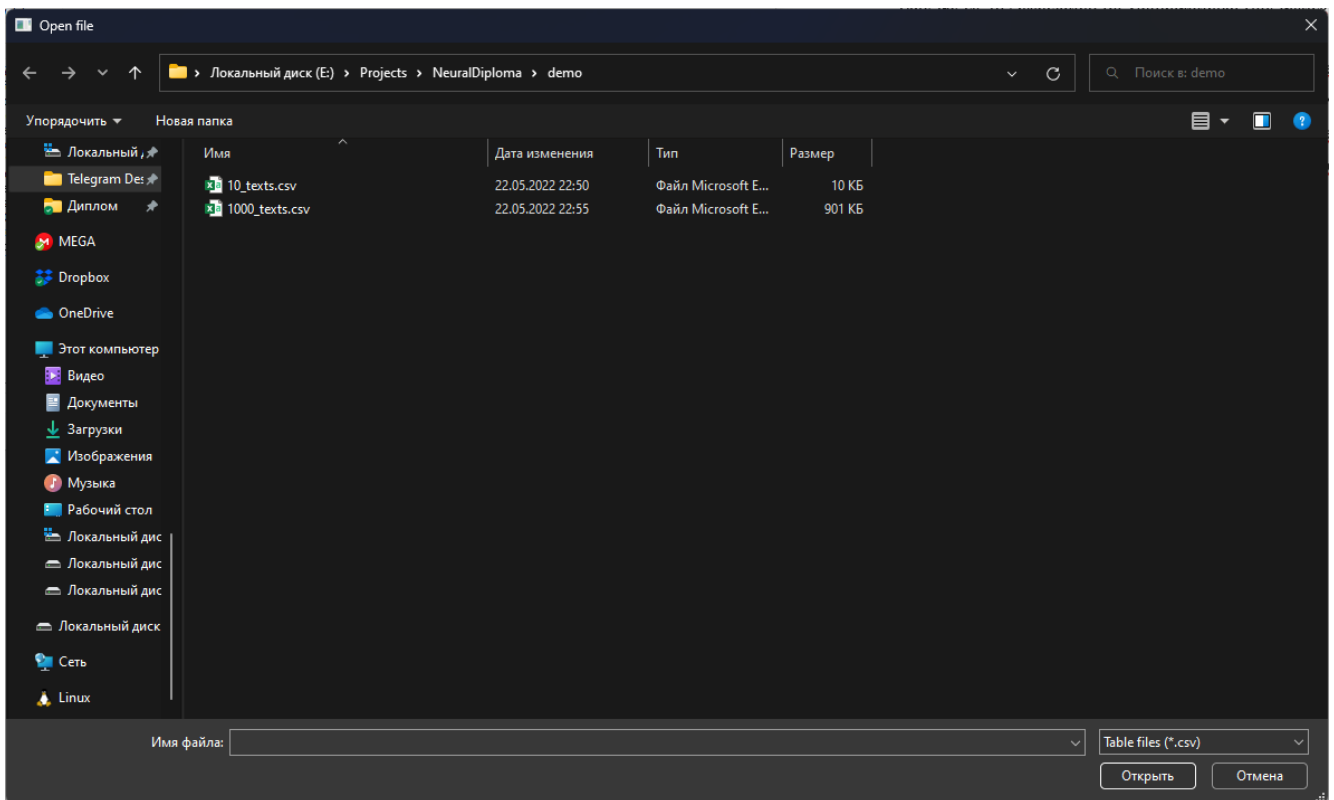


Рисунок 5.12 – Інтерфейс користувача, вікно вибору файлу

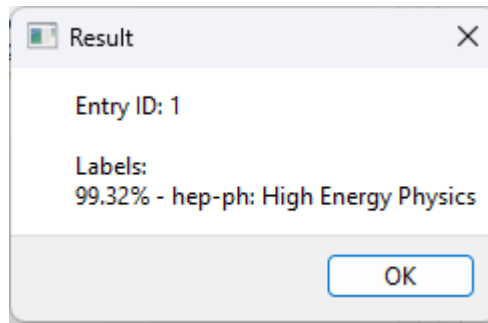


Рисунок 5.13 – Інтерфейс користувача, результат обробки

В наступній вкладці, *Validation*, відображена таблиця валідації, яка зібрана з таблиць текстів та таблиць категорій. У ній присутні стовпці *Entry ID*, які відповідають ID тексту, *Title* що відповідає заголовку, *Abstract* що відповідає анотації та два стовпці категорій – *Original* та *Predicted*, для справжніх та виданих системою категорій відповідно. Вони представлені у вигляді масивів цифр задля більш наглядного порівняння оригінальних та наданих категорій при перегляді людиною.

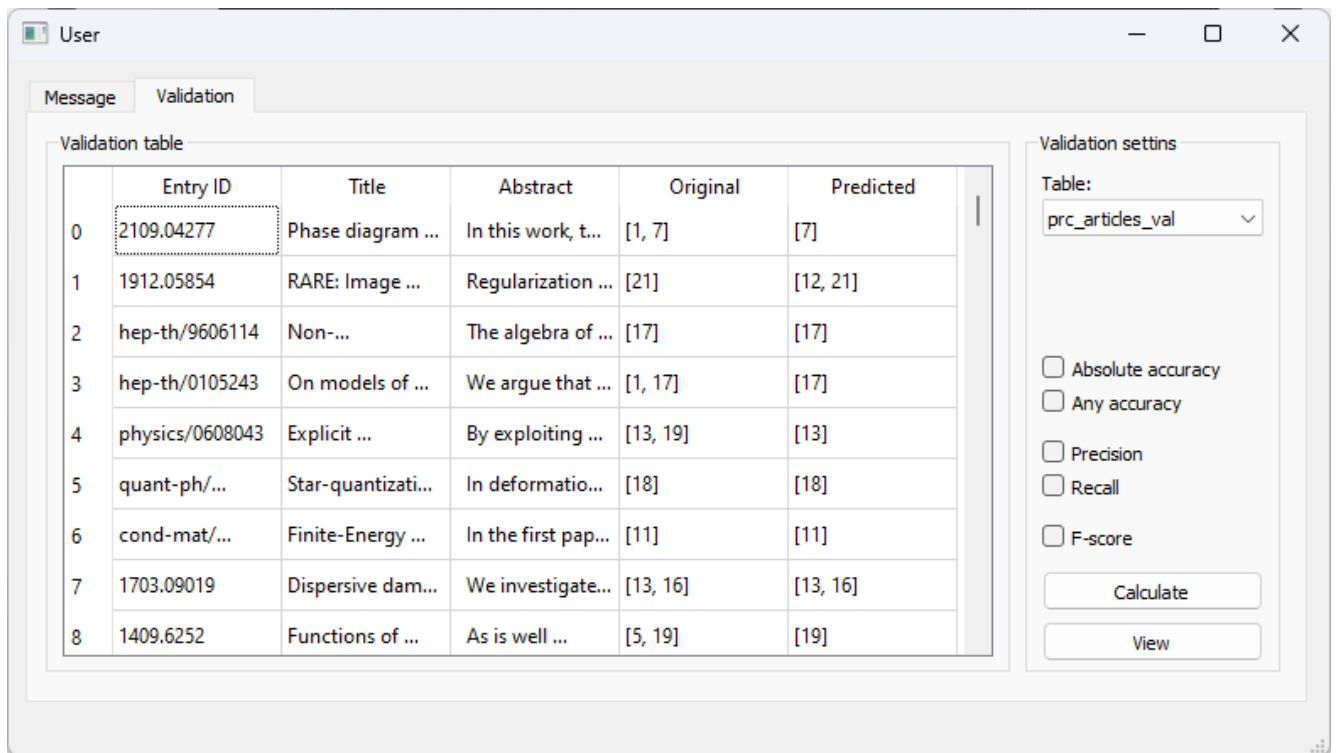


Рисунок 5.14 – Інтерфейс користувача, валідація

У меню Table можна обрати яка саме таблиця буде відображатись, для поточного проекту присутня лише одна таблиця для валідації. Під вибором таблиці також є чекбокси з параметрами, по яким буде відображена валідація результатів навчання, якщо натиснути на View. Кнопка Calculate робить переобчислення на основі обраної таблиці. Усі результати обчислень зберігаються на сервері чи у локальну базу даних.

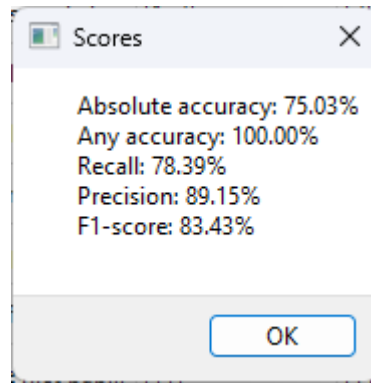


Рисунок 5.15 – Результати валідації

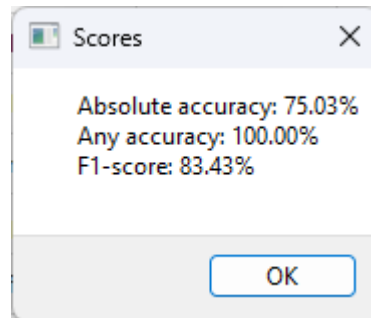


Рисунок 5.16 – Результати валідації, менша кількість параметрів

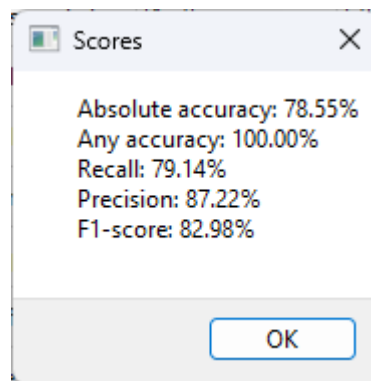


Рисунок 5.17 – Результати валідації, після оновлення

На рисунку 5.15 наведені неактуальні результати валідації, які після оновлення таблиці валідацій через кнопку Calculate актуалізуються і відображаються на рисунку 5.17.

## ВИСНОВКИ

У ході дипломної роботи було спроектовано та реалізовано систему, яка проводить аналізування і оцінювання діяльності організацій за інформацією з міжнародних бібліографічних джерел. Етапи, які були реалізовані:

- 1) Аналіз існуючих бібліографічних джерел на предмет кращого джерела даних для навчання.
- 2) Налаштування забору даних з обраного джерела.
- 3) Розробка архітектури модулю зберігання інформації та обміну повідомленнями між модулями.
- 4) Розробка архітектури модулів серверної частини.
- 5) Розробка архітектури зв'язку між модулями системи.
- 6) Розробка архітектури нейронної мережі.
- 7) Розробка інтерфейсу програми для адміністратора та аналітика.
- 8) Всебічне тестування.

Розроблена система виконує усі функції, які були зазначені у вимогах у повному обсязі та з необхідною швидкістю.

У ході виконання даної роботи був отриманий практичний досвід з проектування складних архітектур, розробки API на основі брокерів повідомлень, проектування і використання нейронних мереж, створення менеджерів і систем моніторингу та створення комплексних інтерфейсів.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Murugan Anandarajan, Chelsey Hill, Thomas Nolan. Text Preprocessing: Maximizing the Value of Text Data – Practical Text Analytics, 2019
2. Peter G. Zhang. Neural Networks for Classification: A Survey – Georgia State University, 2000
3. Google Scholar : веб-сайт. URL: <https://scholar.google.com/> (дата звернення: 11.04.2022).
4. CORE : веб-сайт. URL: <https://core.ac.uk/> (дата звернення: 12.04.2022).
5. Scopus : веб-сайт. URL: <https://www.scopus.com/> (дата звернення: 12.04.2022).
6. MariaDB : веб-сайт. URL: <https://mariadb.com/> (дата звернення: 12.04.2022).
7. SQLite : веб-сайт. URL: <https://www.sqlite.org/index.html> (дата звернення: 13.04.2022).
8. Pandas : веб-сайт. URL: <https://pandas.pydata.org/> (дата звернення: 14.04.2022).
9. FastText : веб-сайт. URL: <https://fasttext.cc/> (дата звернення: 15.04.2022).
10. Under the Hood : веб-сайт. URL: <https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3> (дата звернення: 15.04.2022).
11. Dask : веб-сайт. URL: <https://dask.org/> (дата звернення: 16.04.2022).
12. EDA and Multi Label Classification for arXiv : веб-сайт. URL: <https://www.kaggle.com/code/kobakhit/eda-and-multi-label-classification-for-arxiv/notebook> (дата звернення: 16.04.2022).
13. sklearn : веб-сайт. URL: <https://scikit-learn.org/stable/> (дата звернення: 16.04.2022).
14. NLTK : веб-сайт. URL: <https://www.nltk.org/> (дата звернення: 17.04.2022).
15. Zero Shot Classification with HuggingFace Pipeline : веб-сайт. URL: <https://www.kaggle.com/code/foolofatook/zero-shot-classification-with-huggingface-pipeline/notebook> (дата звернення: 17.04.2022).

16. TensorFlow : веб-сайт. URL: <https://www.tensorflow.org/> (дата звернення: 18.04.2022).
17. Keras : веб-сайт. URL: <https://keras.io/> (дата звернення: 18.04.2022).
18. QT : веб-сайт. URL: <https://www.qt.io/> (дата звернення: 19.04.2022).
19. SQLAlchemy : веб-сайт. URL: <https://www.sqlalchemy.org/> (дата звернення: 19.04.2022).
20. Stanza : веб-сайт. URL: <https://stanfordnlp.github.io/stanza/> (дата звернення: 20.04.2022).
21. Google Colab : веб-сайт. URL: <https://research.google.com/colaboratory/> (дата звернення: 20.04.2022).
22. Adam: A Method for Stochastic Optimization: веб-сайт. URL: <https://arxiv.org/abs/1412.6980> (дата звернення: 20.04.2022).
23. Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In Association for Computational Linguistics (ACL) System Demonstrations. 2020.
24. Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.

# ДОДАТОК А

Класифікація наукових текстових масивів за параметрами

Текст програмного модулю

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТР82360\_22А

Аркушів 89

Київ – 2022

## **`_db_connection.py`**

```
class ConnectDB:
    def __init__(self, base, base_name, config_path=None):
        self.base = base
        self.base_name = base_name

        Session = self.start_db(config_path)
        self.session = Session()

    def start_db(self, config_path):
        ...

    def insert(self, table, data: list[dict]):
        data_object = []

        for record in data:
            data_obj = table(**record)
            data_object.append(data_obj)

        self.session.add_all(data_object)
        self.session.commit()

    def get_all(self, table):
        result = self.session.query(table).all()

        data_object = [vars(r) for r in result]
```

```
return data_object
```

### **db\_external.py**

```
# Internal
```

```
from addons.config import ConfigDBExternal
```

```
from db.db_connection._db_connection import ConnectDB
```

```
# External
```

```
from sqlalchemy import create_engine
```

```
from sqlalchemy.orm import sessionmaker
```

```
class ExternalConnectDB(ConnectDB):
```

```
    def __init__(self, base, base_name, config_path=None):
```

```
        super().__init__(base, base_name, config_path)
```

```
    def start_db(self, config_path):
```

```
        db_config = ConfigDBExternal(config_path)
```

```
        user = db_config.user
```

```
        password = db_config.password
```

```
        hostname = db_config.hostname
```

```
        dbname = db_config.dbname
```

```
        charset = db_config.charset
```

```

        engine =
create_engine(f'mysql+pymysql://{user}:{password}@{hostname}/{dbname}?charset=
{charset}')

        self.base.metadata.create_all(engine, checkfirst=True)

        session = sessionmaker(bind=engine)

        return session

```

## **db\_local.py**

```

# Base
import os

# Internal
from addons.config import ConfigDBLocal, ConfigProject
from db.db_connection._db_connection import ConnectDB

# External
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

class LocalConnectDB(ConnectDB):
    def __init__(self, base, base_name, config_path=None):
        super().__init__(base, base_name, config_path)

```

```

def start_db(self, config_path):
    project_config = ConfigProject(config_path)
    db_config = ConfigDBLocal(config_path)
    path = os.path.join(project_config.project_path,
                        db_config.path,
                        self.base_name)

    engine = create_engine(f'sqlite:/// {path}')
    self.base.metadata.create_all(engine, checkfirst=True)
    session = sessionmaker(bind=engine)

    return session

```

## **extractor.py**

```

# Internal

from addons.general_category import general_category
from db.db_init.db_init_base import Articles, Authors, Categories, CatList, BaseList,
GeneralCatList

class Extractor:
    data = None

    cat_list = []
    cat_id_dict = {}

    def __init__(self, db_base, loader):

```

```

self.db_base = db_base
self.loader = loader

self.data_reset()
self.form_data()

def run(self, query, count: int = float('inf')):
    self.data_reset()
    self.loader.load_stream(query, count, self.result_processing)

def result_processing(self, result):
    if self.check_categories_bulk(result['categories']):
        self.append_loader(result)
        self.append_author(result)
        self.append_category(result)

def form_data(self):
    self.cat_list = []

    for row in self.db_base.get_all(GeneralCatList):
        cat_id = row['id']
        category = row['category']

        self.cat_id_dict[category] = cat_id
        self.cat_list.append(category)

def pull(self, table, insert_dict):

```

```

self.db_base.insert(table, insert_dict)

def append_loader(self, result):
    data = [{'entry_id': result['entry_id'],
            'title': result['title'],
            'summary': result['summary']}
           ]

    self.pull(Articles, data)

def append_author(self, result):
    for author in result['authors']:
        data = [{'entry_id': result['entry_id'],
                'author': author}
               ]

        self.pull(Authors, data)

def append_category(self, result):
    for category in result['general_categories']:
        data = [{'entry_id': result['entry_id'],
                'category': category}
               ]

        self.pull(Categories, data)

```

```
def data_reset(self):
    self.data = {
        'loader': [],
        'authors': [],
        'categories': [],
    }

def check_category(self, category):
    if category in self.cat_list:
        return True

    return False

def check_categories_bulk(self, categories):
    for category in categories:
        if category in self.cat_list:
            return True

    return False
```

## **loader\_axriv.py**

```
# Internal
```

```
from addons.perf_count import perf_count
from addons.config import ConfigArxiv
```

```

# External
import arxiv

class ArxivLoader:
    client = None

    __sort_methods = {
        'relevance': arxiv.SortCriterion.Relevance,
        'lastUpdatedDate': arxiv.SortCriterion.LastUpdatedDate,
        'submitted_date': arxiv.SortCriterion.SubmittedDate,
    }

    def __init__(self, config_path=None, default_sort=None):
        self.default_sort = default_sort if default_sort else 'submitted_date'
        self.arxiv_config = ConfigArxiv(config_path)

        self.setup_client()

    @perf_count
    def load_stream(self, query: str, size: int, func):
        for result in self.client.results(arxiv.Search(query=query, max_results=size)):
            func(self.unpack(result))

    @perf_count
    def load(self, query: str, size: int):

```

```

results = self.query(query, size, self.default_sort)
return [self.unpack(result) for result in results]

def setup_client(self):
    self.client = arxiv.Client(page_size=self.arxiv_config.bath_size,
                               delay_seconds=self.arxiv_config.delay_seconds,
                               num_retries=self.arxiv_config.num_retries
                               )

def query(self, query: str, max_results: int, sort: str):
    search = arxiv.Search(
        query=query,
        max_results=max_results,
        sort_by=self.__sort_methods[sort],
    )

    return search.results()

def unpack(self, result: arxiv.Result):
    result_dict = {
        'entry_id': result.get_short_id(),
        'doi': result.doi,

        'authors': [str(author) for author in result.authors],
        'title': result.title,
        'summary': result.summary,
    }

```

```

        'categories': result.categories,

        'general_categories': self.general_category(result.categories)
    }

    return result_dict

    @staticmethod
    def general_category(categories):
        return [a.split('.')[0] for a in categories]

if __name__ == '__main__':
    loader = ArxivLoader()
    results = loader.query('all', 10, 'submitted_date')

    for result in results:
        print(loader.unpack(result))

```

### **preprocess\_text.py**

```

# External
import nltk
from nltk.corpus import stopwords

import stanza

```

```

class TextPreprocessor:
    min_len = 2
    max_word_len = 40

    custom_stoplist = [
    ]

    garbage = [
    ]

    def __init__(self):
        nltk.download('stopwords')
        stanza.download('en')

        self.nlp = stanza.Pipeline(lang='en', processors='tokenize, lemma')
        self.english_stopwords = stopwords.words("english")

    def lemmatize(self, text, return_tokens=False):
        lemmatized_doc = self.nlp(text)
        lemmatized_tokens = []

        for sentences in lemmatized_doc.sentences:
            for word in sentences.words:
                if word.lemma is not None \

```

```

        and word.lemma not in self.english_stopwords \
        and word.lemma not in self.custom_stoplist \
        and self.min_len < len(word.lemma) < self.max_word_len:
            lemmatized_tokens.append(word.lemma)

    if return_tokens:
        return lemmatized_tokens
    else:
        return ''.join(lemmatized_tokens)

```

## **processor.py**

```

# Internal
from pipeline.clear_text import TextCleaner
from pipeline.preprocess_text import TextPreprocessor

class Processor:
    def __init__(self):
        self.prc = TextPreprocessor()
        self.clr = TextCleaner()

    def process_text(self, text: str) -> str:
        text = self.clr.text_clearing_pipeline(text)
        text = self.prc.lemmatize(text)

```

```
return text
```

## **convertor.py**

```
# Base
```

```
from multiprocessing import Pool
```

```
from collections import defaultdict
```

```
# Internal
```

```
from db.db_init.db_init_base import Articles, Authors, Categories, CatList,  
GeneralCatList
```

```
# External
```

```
from tqdm import tqdm
```

```
import pandas as pd
```

```
class Convertor:
```

```
    labels_id = {}
```

```
    def __init__(self, db_base, text_pipeline):
```

```
        self.db_base = db_base
```

```
        self.text_pipeline = text_pipeline
```

```
        self.get_labels_id()
```

```

def data_pipeline(self, save_path):
    """
    Extract data from database to dataframe

    :param queue: multiprocessing usage
    :param text_pipeline: function that applied to text
    :param save_path: path to dataframe pickle file
    :return:
    """
    data = self.get_data()
    data_dict = self.group_data(data)

    # text_data = self.preprocess_data(data_dict)
    # data_dict['text'] = text_data

    data_df = self.convert_to_df_encode(data_dict)
    self.save_to_file(data_df, save_path)

def get_data(self):
    data = self.db_base.session \
        .query(Articles.entry_id, Articles.title, Articles.summary, Categories.category) \
        .select_from(Articles) \
        .join(Categories, Categories.entry_id == Articles.entry_id) \
        .all()

    return data

```

```

def get_labels_id(self):
    data = self.db_base.get_all(GeneralCatList)

    self.labels_id = {}
    for row in data:
        self.labels_id[row['id']] = row['category']

def preprocess_data(self, data_dict):
    processed_texts = []
    for text in tqdm(data_dict['text']):
        processed_texts.append(self.text_pipeline(text))

    return processed_texts

def group_data(self, data):
    text_cat_dict = defaultdict(list)
    text_dict = {}

    for row in tqdm(data):
        entry_id = row[0]
        title = row[1]
        abstract = row[2]
        category = row[3]

        full_text = title + '\n' + abstract

        text_cat_dict[full_text].append(category)

```

```
text_dict[full_text] = entry_id
```

```
data_list = [  
    {  
        'entry_id': text_dict[full_text],  
        'text': full_text,  
        'categories': text_cat_dict[full_text]  
    }  
    for full_text in text_dict.keys()  
]
```

```
data_dict = {'entry_id': [],  
            'text': [],  
            'categories': []  
            }
```

```
for row in data_list:  
    data_dict['entry_id'].append(row['entry_id'])  
    data_dict['text'].append(row['text'])  
    data_dict['categories'].append(row['categories'])
```

```
return data_dict
```

```
@staticmethod
```

```
def convert_to_df(data_dict):  
    return pd.DataFrame.from_dict(data_dict)
```

```

def convert_to_df_encode(self, data_dict):
    id_count = len(self.labels_id.keys())
    column_names = ['entry_id', 'text']
    column_names.extend(self.labels_id.values())

    rows = []
    for t in tqdm(zip(data_dict['entry_id'], data_dict['text'], data_dict['categories'])):
        entry_id = t[0]
        text = t[1]
        categories = t[2]

        one_hot_categories = [0] * id_count
        for category in categories:
            one_hot_categories[category - 1] = 1

        row = [entry_id, text]
        row.extend(one_hot_categories)

        rows.append(row)

    return pd.DataFrame(rows, columns=column_names)

```

`@staticmethod`

```

def save_to_file(data_df, save_path, protocol=4):
    data_df.to_pickle(save_path, protocol=protocol)

```

## **unit.py**

```
# Base
```

```
import gc
```

```
import sys
```

```
import json
```

```
# Internal
```

```
from db.db_init.db_init_base import BaseBase, base_name
```

```
from db.db_connection.db_external import ExternalConnectDB
```

```
from addons.timer import Timer
```

```
from addons.config import ConfigMaster, ConfigUnit
```

```
from models.keras_model_base import KerasModel
```

```
from pipeline.clear_text import TextCleaner
```

```
from pipeline.preprocess_text import TextPreprocessor
```

```
from messengers.messenger_unit import Sender
```

```
# External
```

```
import pika
```

```
if sys.platform == "linux" or sys.platform == "linux2":
```

```
    config_path_project = 'conf/project.cfg'
```

```
    config_path_master = 'conf/server.cfg'
```

```
else:
```

```
config_path_project = '../conf/project.cfg'  
config_path_master = '../conf/server.cfg'
```

```
class NeuralUnit:
```

```
    logger_prefix = '[U]'
```

```
    response_queue = None
```

```
    def __init__(self):
```

```
        self.num = sys.argv[1]
```

```
        if '-queue' in sys.argv[2:]:
```

```
            self.type = 'queue'
```

```
        elif '-validate' in sys.argv[2:]:
```

```
            self.type = 'validate'
```

```
        else:
```

```
            self.type = 'personal'
```

```
        self.config = ConfigMaster(config_path_master)
```

```
        self.config_unit = ConfigUnit(config_path_master)
```

```
        self.connection, self.channel = self.init_pika()
```

```
        self.sender = Sender(self.channel, self.config)
```

```
        self.text_cleaner = TextCleaner()
```

```

self.text_preprocessor = TextPreprocessor()

self.models = {
    'base': KerasModel(self.config_unit.base_model_path),
}

self.main()

def main(self):
    if self.type == 'queue' or self.type == 'validate':
        self.channel.basic_consume(
            queue=self.config.n_queue_process_queue_bulk,
on_message_callback=self.on_message)

        self.response_queue = self.config.n_queue_response_queue_bulk
    elif self.type == 'personal':
        self.channel.basic_consume(
            queue=self.config.n_queue_process_queue,
on_message_callback=self.on_message)

        self.response_queue = self.config.n_queue_response_queue
    else:
        raise Exception('Wrong process type')

    try:
        self.channel.start_consuming()
    except KeyboardInterrupt:
        self.channel.stop_consuming()

```

```
self.connection.close()
```

```
def pipeline(self, body):
```

```
    json_msg = json.loads(body, strict=False)
```

```
    text_id, text = json_msg['entry_id'], json_msg['text']
```

```
    timer = Timer()
```

```
    timer.start()
```

```
    text = self.text_cleaner.text_clearing_pipeline(text, lowercase=True)
```

```
    text = self.text_preprocessor.lemmatize(text)
```

```
    text_enc = text.encode('utf-8')
```

```
    labels = self.models['base'].score_text(text_enc)
```

```
    new_labels = {}
```

```
    for k, v in labels.items():
```

```
        if v >= self.config_unit.threshold:
```

```
            new_labels[k] = v
```

```
    labels = new_labels
```

```
    self.log(f'Text {text_id} successfully analyzed in {timer.finish()}')
```

```
    if self.type == 'personal':
```

```

        self.sender.respond(entry_id=text_id, labels=labels, null=json_msg['null'],
queue=self.response_queue)

        elif self.type == 'queue':

            self.sender.respond_queue(entry_id=text_id, labels=labels,
queue=self.response_queue)

            elif self.type == 'validate':

                self.sender.respond_validate(entry_id=text_id, labels=list(labels.keys()),
original_labels=json_msg['original_labels'],

                    queue=self.response_queue)

def on_message(self, ch, method, properties, body):

    self.pipeline(body)

    self.channel.basic_ack(method.delivery_tag)

    gc.collect()

    return

def init_pika(self):

    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}@{self.co
nfig.server}'

                    f'/?heartbeat=5')

    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()

    channel.basic_qos(prefetch_count=1)

    queue_process_queue = channel.queue_declare(

```

```
        self.config.n_queue_process_queue,  
        durable=True  
    )  
  
    queue_process_queue_bulk = channel.queue_declare(  
        self.config.n_queue_process_queue_bulk,  
        durable=True  
    )  
  
    queue_response_queue = channel.queue_declare(  
        self.config.n_queue_response_queue,  
        durable=True  
    )  
  
    queue_response_queue_bulk = channel.queue_declare(  
        self.config.n_queue_response_queue_bulk,  
        durable=True  
    )  
  
    queue_logger_base = channel.queue_declare(  
        self.config.n_queue_logger_base,  
        durable=True  
    )  
  
    return connection, channel
```

```
def log(self, msg):
```

```
msg = self.logger_prefix + f' Unit {int(self.num) + 1}: ' + msg
print(msg)
self.sender.send_log(msg)

if __name__ == '__main__':
    print('[*] Base unit started. Waiting for messages. To exit press CTRL+C')
    unit = NeuralUnit()
```

### **master.py**

```
# Base
import json
import sys
import subprocess
import time

# Internal
from addons.config import ConfigMaster
from messengers.messenger_master import Sender

# External
import nltk
import psutil
import pika
```

```

class NeuralMaster:
    logger_prefix = '[M]'

    sleep_time = 20

    procs = []
    nltk.download("stopwords")

    clock_proc = None

    freeze_status = [False] * 7
    queue_status = ['A'] * 7

    def __init__(self, config_path_master):
        self.config = ConfigMaster(config_path_master)

        self.channel, \
        self.connection, \
        self.queue_master_manage, \
        self.queue_master_monitor, \
        self.queue_logger_base \
            = self.init_pika()

        self.sender = Sender(self.channel, self.config)

        self.clock_proc = self.process_clock()

```

```

self.prepare()
self.run()

def init_pika(self):
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

    queue_process_queue = channel.queue_declare(
        self.config.n_queue_process_queue,
        durable=True
    )

    queue_process_queue_bulk = channel.queue_declare(
        self.config.n_queue_process_queue_bulk,
        durable=True
    )

    queue_master_manage = channel.queue_declare(
        self.config.n_queue_master_manage,
        durable=True
    )

```

```

queue_master_monitor = channel.queue_declare(
    self.config.n_queue_master_monitor,
    durable=True
)

queue_master_monitor = channel.queue_declare(
    self.config.n_queue_master_monitor_queue,
    durable=True
)

queue_logger_base = channel.queue_declare(
    self.config.n_queue_logger_base,
    durable=True
)

return channel, connection, queue_master_manage, queue_master_monitor,
queue_logger_base

def prepare(self):
    self.clear_queue(self.config.n_queue_logger_base)
    self.clear_queue(self.config.n_queue_master_manage)
    self.process_unit()

def run(self):
    self.channel.basic_consume(
        queue=self.config.n_queue_master_manage,
        on_message_callback=self.on_message)

```

```

try:
    self.channel.start_consuming()
except KeyboardInterrupt:
    self.channel.stop_consuming()

self.connection.close()

def on_message(self, ch, method, properties, body):
    json_msg = json.loads(body, strict=False)
    type = json_msg['type']

    if type == 'update_request':
        self.update_processes(self.procs)
        self.send_queue_statuses()
    elif type == 'freeze_process_request':
        self.freeze_process(json_msg['id'])
    elif type == 'unfreeze_process_request':
        self.freeze_process(json_msg['id'], unfreeze=True)
    elif type == 'reload_process_request':
        self.reload_process(json_msg['id'])
    elif type == 'change_bulk_status_request':
        self.change_queue(json_msg['id'], json_msg['queue'])
    elif type == 'add_unit_request':
        self.add_unit()
    elif type == 'remove_unit_request':
        self.remove_unit()

```

```

self.channel.basic_ack(method.delivery_tag)
return

def change_queue(self, num, queue):
    proc = self.procs[num]
    proc.kill()

    self.queue_status[num] = queue

    self.reload_process(num)

def add_unit(self):
    if len(self.procs) < 7:
        proc = subprocess.Popen([sys.executable, self.config.unit, str(len(self.procs))])
        self.procs.append(proc)

def remove_unit(self):
    proc = self.procs[-1]
    self.procs.remove(proc)
    proc.kill()

def update_processes(self, procs):
    for p in enumerate(procs):
        process_id = p[0]
        process = p[1]

        mem_usage = self.full_mem_usage(process)

```

```

cpu_usage = self.full_cpu_usage(process)

self.sender.update_processes_push(process_id,
                                   mem_usage,
                                   cpu_usage,
                                   self.freeze_status[process_id],
                                   self.queue_status[process_id])

def send_queue_statuses(self):
    q_global_size = self.get_queue_size(self.config.n_queue_process_queue_bulk)
    q_local_size = self.get_queue_size(self.config.n_queue_process_queue)
    q_master_manage_size =
self.get_queue_size(self.config.n_queue_master_manage)

    self.sender.queue_statuses_push(q_global_size, q_local_size,
q_master_manage_size)

def freeze_process(self, num, unfreeze=False):
    process = psutil.Process(self.procs[num].pid)
    children = process.children(recursive=True)

    if not unfreeze:
        process.suspend()
        self.freeze_status[num] = True
        for child in children:
            child.suspend()
        self.log(f'Process {num + 1} frozen')
    else:

```

```
process.resume()
self.freeze_status[num] = False
for child in children:
    child.resume()
self.log(f'Process {num + 1} resumed')
```

```
def reload_process(self, num):
```

```
    process = self.procs[num]
    process.kill()
```

```
    if self.queue_status[num] == 'A':
```

```
        proc = subprocess.Popen([sys.executable, self.config.unit, str(num)])
```

```
    elif self.queue_status[num] == 'B':
```

```
        proc = subprocess.Popen([sys.executable, self.config.unit, str(num), '-queue'])
```

```
    elif self.queue_status[num] == 'C':
```

```
        proc = subprocess.Popen([sys.executable, self.config.unit, str(num), '-validate'])
```

```
    else:
```

```
        proc = subprocess.Popen([sys.executable, self.config.unit, str(num)])
```

```
    self.procs[num] = proc
```

```
    self.log(f'Process {num + 1} reloaded')
```

```
def full_cpu_usage(self, parent):
```

```
    parent = psutil.Process(parent.pid)
```

```
    parent_cpu = psutil.Process(parent.pid).cpu_percent(interval=0.1)
```

```

children_cpu = 0
children = parent.children(recursive=True)
for child in children:
    children_cpu += psutil.Process(child.pid).cpu_percent(interval=0.1)

return parent_cpu + children_cpu

def full_mem_usage(self, parent):
    parent = psutil.Process(parent.pid)
    parent_mem_prc = psutil.Process(parent.pid).memory_percent()

    children_mem_prc = 0
    children = parent.children(recursive=True)

    for child in children:
        children_mem_prc += psutil.Process(child.pid).memory_percent()

    return parent_mem_prc + children_mem_prc

def process_clock(self):
    clock_proc = subprocess.Popen([sys.executable, self.config.clock], shell=False)

    return clock_proc

def process_unit(self):
    for i in range(self.config.instance_count):
        proc = subprocess.Popen([sys.executable, self.config.unit, str(i)])

```

```
self.procs.append(proc)
time.sleep(self.sleep_time)
```

```
def process_reload(self):
```

```
    for p in self.procs:
        self.procs.remove(p)
        p.kill()
```

```
self.process_unit()
```

```
def get_queue_size(self, queue) -> int:
```

```
    queue_task = self.channel.queue_declare(
        queue,
        durable=True,
        passive=True
    )
```

```
    return queue_task.method.message_count
```

```
def clear_queue(self, queue):
```

```
    self.channel.queue_purge(queue)
```

```
def log(self, msg):
```

```
    msg = self.logger_prefix + ' ' + msg
    print(msg)
    self.sender.send_log(msg)
```

```
if __name__ == "__main__":
    if sys.platform == "linux" or sys.platform == "linux2":
        config_path_project = 'conf/project.cfg'
        config_path_master = 'conf/server.cfg'
    else:
        config_path_project = './conf/project.cfg'
        config_path_master = './conf/server.cfg'

    master = NeuralMaster(config_path_master)
```

## **clock.py**

```
# Base
import sys
import time
import json

# Internal
from addons.config import ConfigMaster

# External
import pika

class Clock:
    logger_prefix = '[C]'
```

```

def __init__(self, config_path_master):
    self.config = ConfigMaster(config_path_master)

    self.channel, \
    self.queue_clock \
        = self.init_pika()

def init_pika(self):
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

    queue_clock = channel.queue_declare(
        self.config.n_queue_master_manage,
        durable=True
    )

    return channel, queue_clock

def start(self):
    while True:
        self.update()

```

```

        print(f {self.logger_prefix} Update request send with timeout
{self.config.timeout}')

        time.sleep(self.config.timeout)

def update(self):
    message = {'type': 'update_request',
              }

    json_message = json.dumps(message)

    self.channel.basic_publish(exchange="",
body=json_message.encode(encoding='UTF-8'),
                               routing_key=self.config.n_queue_master_manage)

def clear_queue(self, queue):
    self.channel.queue_purge(queue)

if __name__ == "__main__":
    if sys.platform == "linux" or sys.platform == "linux2":
        config_path_project = 'conf/project.cfg'
        config_path_master = 'conf/server.cfg'
    else:
        config_path_project = './conf/project.cfg'
        config_path_master = './conf/server.cfg'

    clock = Clock(config_path_master)
    clock.start()

```

## messenger\_master.py

```
# Base
```

```
import json
```

```
class Sender:
```

```
    def __init__(self, channel, config):
```

```
        self.channel = channel
```

```
        self.config = config
```

```
    def send_result(self, queue, message):
```

```
        self.channel.basic_publish(exchange="", body=message.encode(encoding='UTF-8'),  
                                   routing_key=queue)
```

```
    def update_processes_push(self, process_id, process_mem, process_cpu, frozen,  
queue):
```

```
        message = {'type': 'progress_bar_update',
```

```
                   'id': process_id,
```

```
                   'process_mem': process_mem,
```

```
                   'process_cpu': process_cpu,
```

```
                   'frozen': frozen,
```

```
                   'queue': queue
```

```
        }
```

```
        json_message = json.dumps(message)
```

```
        self.send_result(self.config.n_queue_master_monitor, json_message)
```

```

def queue_statuses_push(self, q_global_size, q_local_size, q_master_manage_size):
    message = {'type': 'queue_statuses',
               'q_global_size': q_global_size,
               'q_local_size': q_local_size,
               'q_master_manage_size': q_master_manage_size,
               }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_master_monitor_queue, json_message)

def send_log(self, msg):
    self.send_result(self.config.n_queue_logger_base, msg)

```

## **messenger\_unit.py**

# Base

import json

class Sender:

```

def __init__(self, channel, config):

```

```

    self.channel = channel

```

```

    self.config = config

```

```

def send_result(self, queue, message):

```

```

    self.channel.basic_publish(exchange="", body=message.encode(encoding='UTF-8'),

```

```
routing_key=queue)
```

```
def respond(self, entry_id, labels, null, queue):
```

```
    message = {'entry_id': entry_id,  
              'labels': str(labels),  
              'null': null  
              }
```

```
    json_message = json.dumps(message)  
    self.send_result(queue, json_message)
```

```
def respond_queue(self, entry_id, labels, queue):
```

```
    message = {'entry_id': entry_id,  
              'labels': str(labels),  
              }
```

```
    json_message = json.dumps(message)  
    self.send_result(queue, json_message)
```

```
def respond_validate(self, entry_id, labels, original_labels, queue):
```

```
    message = {'entry_id': entry_id,  
              'labels': str(labels),  
              'original_labels': str(original_labels)  
              }
```

```
    json_message = json.dumps(message)  
    self.send_result(queue, json_message)
```

```
def send_log(self, msg):
    self.send_result(self.config.n_queue_logger_base, msg)
```

### **keras\_model\_base.py**

```
# External
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import load_model
```

```
class KerasModel:
```

```
    tokenizer = None
```

```
    model = None
```

```
    def __init__(self, model_path):
```

```
        self.model_path = model_path
```

```
        self.load_model()
```

```
    def load_model(self):
```

```
        self.model = load_model(self.model_path)
```

```
    def score_text(self, text):
```

```
        pred_tags_scores = self.model.predict([text])[0]
```

```
        pred_tags = {int(pair[0] + 1): pair[1] for pair in list(enumerate(pred_tags_scores))}
```

```
return pred_tags
```

### **gui\_admin/main\_admin.py**

```
# Base
```

```
import sys
```

```
import traceback
```

```
# Internal
```

```
from main_ui import Ui_MainWindow
```

```
from messengers.messenger import Sender
```

```
from updaters.update_resources import ResourceUpdater
```

```
from updaters.update_queue import QueueUpdater
```

```
from updaters.update_log import LogUpdater
```

```
from addons.config import ConfigMaster
```

```
# External
```

```
import pika
```

```
from PyQt6.QtWidgets import QMainWindow, QApplication
```

```
from PyQt6.QtCore import QThreadPool
```

```
# pyuic6 -o main_ui.py -x main.ui
```

```
class Window(QMainWindow, Ui_MainWindow):
```

```
    threadpool = QThreadPool()
```

```
    units = None
```

```
    units_paused = None
```

```
    units_pause_buttons = None
```

```
    units_cpu = None
```

```
    units_mem = None
```

```
    units_ab = None
```

```
    absolute_signal = None
```

```
    count_signal = None
```

```
    queue_type_change_signal = None
```

```
    queue_display_top_limit = 100
```

```
    def __init__(self, config_path_master, parent=None):
```

```
        super().__init__(parent)
```

```
        self.setupUi(self)
```

```
        self.config = ConfigMaster(config_path_master)
```

```
        self.units_count = self.config.instance_count
```

```
        self.channel, self.connection, self.queue_master_manage = self.init_pika()
```

```

self.sender = Sender(self.channel, self.config)

self.updater_resources = ResourceUpdater(self.config)
self.updater_queue = QueueUpdater(self.config)
self.updater_log = LogUpdater(self.config)

self.setup()
self.update_units()

def init_pika(self):
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

    queue_master_manage = channel.queue_declare(
        self.config.n_queue_master_manage,
        durable=True
    )

    return channel, connection, queue_master_manage

def setup(self):
    self.setup_buttons_units()

```

```

self.setup_settings_units()
self.setup_settings_queue()

self.units = [self.u1, self.u2, self.u3, self.u4, self.u5, self.u6, self.u7]
self.units_paused = [False, False, False, False, False, False, False]
self.units_pause_buttons = [self.u1_pause, self.u2_pause, self.u3_pause,
self.u4_pause,
                                self.u5_pause, self.u6_pause, self.u7_pause]
self.units_ab = [self.u1_ab, self.u2_ab, self.u3_ab, self.u4_ab, self.u5_ab,
self.u6_ab, self.u7_ab]

self.units_cpu = [self.u1_p_cpu, self.u2_p_cpu, self.u3_p_cpu, self.u4_p_cpu,
self.u5_p_cpu, self.u6_p_cpu, self.u7_p_cpu]
self.units_mem = [self.u1_p_mem, self.u2_p_mem, self.u3_p_mem,
self.u4_p_mem,
self.u5_p_mem, self.u6_p_mem, self.u7_p_mem]

self.setup_thread_resources()
self.setup_thread_queues()
self.setup_thread_log()

def setup_thread_resources(self):
    self.updater_resources.signals.cpu_signal_1.connect(lambda value:
self.set_progressbar(value, 1, 'cpu'))
    self.updater_resources.signals.cpu_signal_2.connect(lambda value:
self.set_progressbar(value, 2, 'cpu'))
    self.updater_resources.signals.cpu_signal_3.connect(lambda value:
self.set_progressbar(value, 3, 'cpu'))
    self.updater_resources.signals.cpu_signal_4.connect(lambda value:
self.set_progressbar(value, 4, 'cpu'))

```

```
self.updater_resources.signals.cpu_signal_5.connect(lambda value:
self.set_progressbar(value, 5, 'cpu'))
```

```
self.updater_resources.signals.cpu_signal_6.connect(lambda value:
self.set_progressbar(value, 6, 'cpu'))
```

```
self.updater_resources.signals.cpu_signal_7.connect(lambda value:
self.set_progressbar(value, 7, 'cpu'))
```

```
self.updater_resources.signals.mem_signal_1.connect(lambda value:
self.set_progressbar(value, 1, 'mem'))
```

```
self.updater_resources.signals.mem_signal_2.connect(lambda value:
self.set_progressbar(value, 2, 'mem'))
```

```
self.updater_resources.signals.mem_signal_3.connect(lambda value:
self.set_progressbar(value, 3, 'mem'))
```

```
self.updater_resources.signals.mem_signal_4.connect(lambda value:
self.set_progressbar(value, 4, 'mem'))
```

```
self.updater_resources.signals.mem_signal_5.connect(lambda value:
self.set_progressbar(value, 5, 'mem'))
```

```
self.updater_resources.signals.mem_signal_6.connect(lambda value:
self.set_progressbar(value, 6, 'mem'))
```

```
self.updater_resources.signals.mem_signal_7.connect(lambda value:
self.set_progressbar(value, 7, 'mem'))
```

```
self.updater_resources.signals.frz_signal_1.connect(lambda value:
self.set_freeze(value, 1))
```

```
self.updater_resources.signals.frz_signal_2.connect(lambda value:
self.set_freeze(value, 2))
```

```
self.updater_resources.signals.frz_signal_3.connect(lambda value:
self.set_freeze(value, 3))
```

```
self.updater_resources.signals.frz_signal_4.connect(lambda value:
self.set_freeze(value, 4))
```

```
self.updater_resources.signals.frz_signal_5.connect(lambda value:
self.set_freeze(value, 5))
```

```

        self.updater_resources.signals.frz_signal_6.connect(lambda value:
self.set_freeze(value, 6))

        self.updater_resources.signals.frz_signal_7.connect(lambda value:
self.set_freeze(value, 7))

        self.updater_resources.signals.que_signal_1.connect(lambda value:
self.set_queue(value, 1))

        self.updater_resources.signals.que_signal_2.connect(lambda value:
self.set_queue(value, 2))

        self.updater_resources.signals.que_signal_3.connect(lambda value:
self.set_queue(value, 3))

        self.updater_resources.signals.que_signal_4.connect(lambda value:
self.set_queue(value, 4))

        self.updater_resources.signals.que_signal_5.connect(lambda value:
self.set_queue(value, 5))

        self.updater_resources.signals.que_signal_6.connect(lambda value:
self.set_queue(value, 6))

        self.updater_resources.signals.que_signal_7.connect(lambda value:
self.set_queue(value, 7))

self.absolute_signal = self.updater_resources.signals.absolute_signal
self.absolute_signal.connect(self.updater_resources.set_absolute)

self.count_signal = self.updater_resources.signals.count_signal
self.count_signal.connect(self.updater_resources.set_count)

self.threadpool.start(self.updater_resources)

def setup_thread_queues(self):
    self.queue_type_change_signal = self.updater_queue.signals.type_signal

```

```

self.queue_type_change_signal.connect(self.updater_queue.set_plot_type)

self.updater_queue.signals.result_signal.connect(self.plot_queue)

self.threadpool.start(self.updater_queue)

def setup_thread_log(self):
    self.updater_log.signals.log_signal.connect(self.set_log)

    self.threadpool.start(self.updater_log)

def set_progressbar(self, value, n, t):
    if t == 'cpu':
        self.units_cpu[n - 1].setValue(value)
    elif t == 'mem':
        self.units_mem[n - 1].setValue(value)

def plot_queue(self, xy):
    x = xy[0]
    y = xy[1]

    self.plot_widget.canvas.axes.clear()
    if max(y) < self.queue_display_top_limit:
        self.plot_widget.canvas.axes.set_ylim([0, self.queue_display_top_limit])
    self.plot_widget.canvas.axes.set_xlim([-self.updater_queue.plot_lim, 0])
    self.plot_widget.canvas.axes.plot(x, y, color='orange')
    self.plot_widget.canvas.draw()

```

```

def set_log(self, log):
    self.plain_log.setPlainText(log)

def setup_buttons_units(self):
    self.u0_plus.clicked.connect(self.add_unit)
    self.u0_minus.clicked.connect(self.remove_unit)

    self.u1_pause.clicked.connect(lambda: self.pause_press(1))
    self.u1_reload.clicked.connect(lambda: self.reload_press(1))
    self.u1_ab.currentTextChanged.connect(lambda value: self.ab_update(1))

    self.u2_pause.clicked.connect(lambda: self.pause_press(2))
    self.u2_reload.clicked.connect(lambda: self.reload_press(2))
    self.u2_ab.currentTextChanged.connect(lambda value: self.ab_update(2))

    self.u3_pause.clicked.connect(lambda: self.pause_press(3))
    self.u3_reload.clicked.connect(lambda: self.reload_press(3))
    self.u3_ab.currentTextChanged.connect(lambda value: self.ab_update(3))

    self.u4_pause.clicked.connect(lambda: self.pause_press(4))
    self.u4_reload.clicked.connect(lambda: self.reload_press(4))
    self.u4_ab.currentTextChanged.connect(lambda value: self.ab_update(4))

    self.u5_pause.clicked.connect(lambda: self.pause_press(5))
    self.u5_reload.clicked.connect(lambda: self.reload_press(5))
    self.u5_ab.currentTextChanged.connect(lambda value: self.ab_update(5))

```

```

self.u6_pause.clicked.connect(lambda: self.pause_press(6))
self.u6_reload.clicked.connect(lambda: self.reload_press(6))
self.u6_ab.currentTextChanged.connect(lambda value: self.ab_update(6))

self.u7_pause.clicked.connect(lambda: self.pause_press(7))
self.u7_reload.clicked.connect(lambda: self.reload_press(7))
self.u7_ab.currentTextChanged.connect(lambda value: self.ab_update(7))

def setup_settings_units(self):
    self.ds_abs.toggled.connect(self.ds_abs_clicked)
    self.ds_rel.toggled.connect(self.ds_rel_clicked)

def setup_settings_queue(self):
    self.rb_global.toggled.connect(self.queue_type_set)
    self.rb_personal.toggled.connect(self.queue_type_set)
    self.rb_command.toggled.connect(self.queue_type_set)

def ds_abs_clicked(self):
    self.absolute_signal.emit(True)

def ds_rel_clicked(self):
    self.absolute_signal.emit(False)

def queue_type_set(self):
    if self.rb_global.isChecked():
        self.queue_type_change_signal.emit(1)

```

```

elif self.rb_personal.isChecked():
    self.queue_type_change_signal.emit(2)
elif self.rb_command.isChecked():
    self.queue_type_change_signal.emit(3)

def add_unit(self):
    self.units_count += 1
    self.update_units()

    self.sender.add_unit()

def remove_unit(self):
    self.units_count -= 1
    self.update_units()

    self.sender.remove_unit()

def ab_update(self, n):
    ab = self.units_ab[n - 1]
    self.sender.change_queue_status_unit(n - 1, ab.currentText())

def set_freeze(self, value, n):
    if not value:
        self.units_pause_buttons[n - 1].setText('Pause')
        self.units_paused[n - 1] = False
    else:
        self.units_pause_buttons[n - 1].setText('Start')

```

```
self.units_paused[n - 1] = True
```

```
def set_queue(self, value, n):
```

```
    if value == 'A':
```

```
        self.units_ab[n - 1].setCurrentIndex(0)
```

```
    elif value == 'B':
```

```
        self.units_ab[n - 1].setCurrentIndex(1)
```

```
    elif value == 'C':
```

```
        self.units_ab[n - 1].setCurrentIndex(2)
```

```
def pause_press(self, i):
```

```
    if not self.units_paused[i - 1]:
```

```
        self.units_pause_buttons[i - 1].setText('Pause')
```

```
        self.units_paused[i - 1] = False
```

```
        self.sender.freeze_process(i - 1)
```

```
    else:
```

```
        self.units_pause_buttons[i - 1].setText('Start')
```

```
        self.units_paused[i - 1] = True
```

```
        self.sender.unfreeze_process(i - 1)
```

```
def reload_press(self, i):
```

```
    self.sender.reload_process(i - 1)
```

```
def update_units(self):
```

```
    for i in range(len(self.units)):
```

```
        if i >= self.units_count:
```

```
            self.units[i].hide()
```

```

        else:
            self.units[i].show()

self.check_unit_button()
self.count_signal.emit(self.units_count)

def check_unit_button(self):
    if self.units_count == 7:
        self.u0_plus.setEnabled(False)
        self.u0_minus.setEnabled(True)
    elif self.units_count == 0:
        self.u0_plus.setEnabled(True)
        self.u0_minus.setEnabled(False)
    elif 0 < self.units_count < 7:
        self.u0_plus.setEnabled(True)
        self.u0_minus.setEnabled(True)
    else:
        raise Exception(f'Value of unit_count out of boundaries: {self.units_count}')

def except_hook(exc_type, exc_value, exc_tb):
    tb = ".join(traceback.format_exception(exc_type, exc_value, exc_tb))
    print('Error message:\n', tb)
    QApplication.quit()

if __name__ == "__main__":

```

```
config_path_master = '../conf/server.cfg'

sys.excepthook = except_hook
app = QApplication(sys.argv)
win = Window(config_path_master)
win.show()
sys.exit(app.exec())
```

### **gui\_admin/plot\_widget.py**

```
# External
from matplotlib.backends.backend_qtagg import FigureCanvas
from matplotlib.figure import Figure

from PyQt6.QtWidgets import QWidget, QVBoxLayout

class PlotWidget(QWidget):

    def __init__(self, parent=None):
        QWidget.__init__(self, parent)

        self.canvas = FigureCanvas(Figure())

        vertical_layout = QVBoxLayout()
```

```
vertical_layout.addWidget(self.canvas)

self.canvas.axes = self.canvas.figure.add_subplot(111)
self.canvas.axes.set_position([0.2, 0.2, 0.7, 0.7])
self.setLayout(vertical_layout)
```

### **gui\_admin/messenger.py**

```
# Base
```

```
import json
```

```
class Sender:
```

```
    def __init__(self, channel, config):
```

```
        self.channel = channel
```

```
        self.config = config
```

```
    def send_result(self, queue, message):
```

```
        self.channel.basic_publish(exchange="", body=message.encode(encoding='UTF-8'),
                                   routing_key=queue)
```

```
    def freeze_process(self, process_id):
```

```
        message = {'type': 'freeze_process_request',
                   'id': process_id,
                   }
```

```

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_master_manage, json_message)

def unfreeze_process(self, process_id):
    message = {'type': 'unfreeze_process_request',
              'id': process_id,
              }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_master_manage, json_message)

def reload_process(self, process_id):
    message = {'type': 'reload_process_request',
              'id': process_id,
              }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_master_manage, json_message)

def add_unit(self):
    message = {'type': 'add_unit_request',
              }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_master_manage, json_message)

def remove_unit(self):

```

```
message = {'type': 'remove_unit_request',  
          }
```

```
json_message = json.dumps(message)  
self.send_result(self.config.n_queue_master_manage, json_message)
```

```
def change_queue_status_unit(self, process_id, queue):
```

```
    message = {'type': 'change_bulk_status_request',  
              'id': process_id,  
              'queue': queue  
              }
```

```
    json_message = json.dumps(message)  
    self.send_result(self.config.n_queue_master_manage, json_message)
```

### **gui\_admin/update\_log.py**

```
# Base
```

```
import json
```

```
import sys
```

```
import subprocess
```

```
import time
```

```
# Internal
```

```
import multiprocessing
```

```
from addons.config import ConfigMaster
```

```

# External

from PyQt6.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
QGraphicsPixmapItem, QFileDialog

from PyQt6.QtCore import Qt, QLineF, pyqtSignal, QThread, pyqtSignal, QObject,
QRunnable

from PyQt6.QtGui import QPixmap, QPen

import pika

class Signals(QObject):
    log_signal = pyqtSignal(str)

class LogUpdater(QRunnable):
    signals = Signals()

    log_lim = 13
    log_text = []

    def __init__(self, config):
        super(LogUpdater, self).__init__()

        self.config = config

        self.channel, \
        self.connection, \

```

```

self.queue_logger_base \
    = self.init_pika()

def on_message(self, ch, method, properties, body):
    self.log_text.append(str(body)[2:-1])

    if len(self.log_text) > self.log_lim:
        self.log_text = self.log_text[-self.log_lim:]

    self.signals.log_signal.emit('\n'.join(self.log_text))

    self.channel.basic_ack(method.delivery_tag)
    return

def init_pika(self):
    parameters =
    pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                      f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

    queue_logger_base = channel.queue_declare(
        self.config.n_queue_logger_base,
        durable=True
    )

```

```

return channel, connection, queue_logger_base

def prepare(self):
    self.clear_queue(self.config.n_queue_logger_base)

def run(self):
    self.prepare()

    self.channel.basic_consume(
        queue=self.config.n_queue_logger_base,
on_message_callback=self.on_message)

    self.clear_queue(self.config.n_queue_logger_base)

    try:
        self.channel.start_consuming()
    except Exception as e:
        print(e)

    self.connection.close()

def clear_queue(self, queue):
    self.channel.queue_purge(queue)

```

## **gui\_admin/update\_queue.py**

```
# Base

import json

import sys

import subprocess

import time

# Internal

import multiprocessing

from addons.config import ConfigMaster

# External

from PyQt6.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
QGraphicsPixmapItem, QFileDialog

from PyQt6.QtCore import Qt, QLineF, pyqtSignal, QThread, pyqtSignal, QObject,
QRunnable

from PyQt6.QtGui import QPixmap, QPen

import pika

class Signals(QObject):

    type_signal = pyqtSignal(int)

    result_signal = pyqtSignal(tuple)

class QueueUpdater(QRunnable):
```

```

signals = Signals()

plot_type = 1

plot_lim = 50
plot_data_global_size = [0] * (plot_lim + 1)
plot_data_local_size = [0] * (plot_lim + 1)
plot_data_master_manage_size = [0] * (plot_lim + 1)

def __init__(self, config):
    super(QueueUpdater, self).__init__()

    self.config = config

    self.channel, \
    self.connection, \
    self.queue_master_monitor_queue, \
    = self.init_pika()

def on_message(self, ch, method, properties, body):
    json_msg = json.loads(body, strict=False)
    type = json_msg['type']

    if type == 'queue_statuses':
        q_global_size = json_msg['q_global_size']
        q_local_size = json_msg['q_local_size']
        q_master_manage_size = json_msg['q_master_manage_size']

```

```

self.plot_data_global_size = self.append_and_check(self.plot_data_global_size,
                                                    q_global_size)

self.plot_data_local_size = self.append_and_check(self.plot_data_local_size,
                                                    q_local_size)

self.plot_data_master_manage_size =
self.append_and_check(self.plot_data_master_manage_size,
                      q_master_manage_size)

self.plot()

self.channel.basic_ack(method.delivery_tag)

return

def append_and_check(self, queue, size):
    queue.append(size)

    if len(queue) > self.plot_lim + 1:
        queue = queue[-(self.plot_lim + 1):]

    return queue

def plot(self):
    x = list(range(-self.plot_lim, 1))

    if self.plot_type == 1:
        y = self.plot_data_global_size
    elif self.plot_type == 2:

```

```

        y = self.plot_data_local_size
    elif self.plot_type == 3:
        y = self.plot_data_master_manage_size
    else:
        raise Exception(f'Invalid plot type: {self.plot_type}')

    self.signals.result_signal.emit((x, y))

def set_plot_type(self, plot_type):
    self.plot_type = plot_type

def init_pika(self):
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

    queue_master_monitor_queue = channel.queue_declare(
        self.config.n_queue_master_monitor_queue,
        durable=True
    )

    return channel, connection, queue_master_monitor_queue

def prepare(self):

```

```

self.clear_queue(self.config.n_queue_master_monitor_queue)

def run(self):
    self.prepare()

    self.channel.basic_consume(
        queue=self.config.n_queue_master_monitor_queue,
        on_message_callback=self.on_message)

    self.clear_queue(self.config.n_queue_master_monitor_queue)

    try:
        self.channel.start_consuming()
    except Exception as e:
        print(e)

    self.connection.close()

def clear_queue(self, queue):
    self.channel.queue_purge(queue)

```

### **gui\_admin/update\_resources.py**

```

import json
import sys
import subprocess
import time

```

```

# Internal

import multiprocessing

from addons.config import ConfigMaster

# External

from PyQt6.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
QGraphicsPixmapItem, QFileDialog

from PyQt6.QtCore import Qt, QLineF, pyqtSignal, QThread, pyqtSignal, QObject,
QRunnable

from PyQt6.QtGui import QPixmap, QPen

import pika

class ResourceUpdater(QRunnable):
    absolute = False
    count = 5

    cpu_signals = []
    mem_signals = []
    que_signals = []

    signals = Signals()

    def __init__(self, config):
        super(ResourceUpdater, self).__init__()

```

```

self.config = config

self.channel, \
self.connection, \
self.queue_master_monitor, \
    = self.init_pika()

self.setup_signals()

def on_message(self, ch, method, properties, body):
    json_msg = json.loads(body, strict=False)
    type = json_msg['type']

    if type == 'progress_bar_update':
        p_id = json_msg['id']
        if 0 <= p_id <= 6:
            process_cpu = json_msg['process_cpu']
            process_mem = json_msg['process_mem']

            freeze_status = json_msg['frozen']
            queue_status = json_msg['queue']

            if self.absolute:
                cpu_value = int(process_cpu / multiprocessing.cpu_count())
                mem_value = int(process_mem)
            else:
                cpu_value = int(process_cpu / multiprocessing.cpu_count() * self.count)

```

```

        mem_value = int(process_mem * self.count)

        self.cpu_signals[p_id].emit(cpu_value)
        self.mem_signals[p_id].emit(mem_value)
        self.frz_signals[p_id].emit(freeze_status)
        self.que_signals[p_id].emit(queue_status)
    else:
        raise Exception(f'Value of unit_count out of boundaries: {json_msg["p_id"]}')
```

```

self.channel.basic_ack(method.delivery_tag)

return

def setup_signals(self):
    self.cpu_signals = [self.signals.cpu_signal_1, self.signals.cpu_signal_2,
self.signals.cpu_signal_3,
                        self.signals.cpu_signal_4, self.signals.cpu_signal_5,
self.signals.cpu_signal_6,
                        self.signals.cpu_signal_7]
    self.mem_signals = [self.signals.mem_signal_1, self.signals.mem_signal_2,
self.signals.mem_signal_3,
                        self.signals.mem_signal_4, self.signals.mem_signal_5,
self.signals.mem_signal_6,
                        self.signals.mem_signal_7]
    self.frz_signals = [self.signals.frz_signal_1, self.signals.frz_signal_2,
self.signals.frz_signal_3,
                        self.signals.frz_signal_4, self.signals.frz_signal_5,
self.signals.frz_signal_6,
                        self.signals.frz_signal_7]
    self.que_signals = [self.signals.que_signal_1, self.signals.que_signal_2,
self.signals.que_signal_3,
```

```
        self.signals.que_signal_4, self.signals.que_signal_5,
self.signals.que_signal_6,
        self.signals.que_signal_7]
```

```
def set_absolute(self, absolute):
```

```
    self.absolute = absolute
```

```
def set_count(self, count):
```

```
    self.count = count
```

```
def init_pika(self):
```

```
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)
```

```
    channel = connection.channel()
```

```
    channel.basic_qos(prefetch_count=1)
```

```
    queue_master_monitor = channel.queue_declare(
```

```
        self.config.n_queue_master_monitor,
```

```
        durable=True
```

```
    )
```

```
    return channel, connection, queue_master_monitor
```

```
def prepare(self):
```

```
    self.clear_queue(self.config.n_queue_master_monitor)
```

```

def run(self):
    self.prepare()

    self.channel.basic_consume(
        queue=self.config.n_queue_master_monitor,
        on_message_callback=self.on_message)

    self.clear_queue(self.config.n_queue_master_monitor)

    try:
        self.channel.start_consuming()
    except Exception as e:
        print(e)

    self.connection.close()

def clear_queue(self, queue):
    self.channel.queue_purge(queue)

```

### **gui\_user/main\_user.py**

```

# Base
import sys
import csv
import traceback

```

```

# Internal

from main_ui import Ui_MainWindow

from messengers.messenger import Sender

from updaters.update_queue import QueueUpdater

from db.db_init.db_init_base import BaseBase, base_name, GeneralCatList, ValResult,
ValCategories, Articles

from db.db_connection.db_external import ExternalConnectDB

from calculations.scores import Scores

from addons.config import ConfigMaster, ConfigDBExternal

# External

import pika

from PyQt5.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
QGraphicsPixmapItem, QFileDialog, QMessageBox, QTableView, QApplication,
QTableWidgetItem, QAction

from PyQt5.QtCore import Qt, QLineF, pyqtSignal, QThreadPool, QThread,
QAbstractListModel, QAbstractTableModel, QModelIndex

from PyQt5.QtGui import QPixmap, QPen

# pyuic6 -o main_ui.py -x main.ui

class ValTableModel(QAbstractTableModel):
    def __init__(self, data=[[[]]], parent=None):
        super().__init__(parent)

```

```
self.header_data = ['Entry ID', 'Title', 'Abstract', 'Original', 'Predicted']
self.data = data
```

```
def headerData(self, section: int, orientation: Qt.Orientation, role: int):
    if role == Qt.DisplayRole:
        if orientation == Qt.Horizontal:
            return self.header_data[section]
        else:
            return str(section)
```

```
def columnCount(self, parent=None):
    return len(self.data[0])
```

```
def rowCount(self, parent=None):
    return len(self.data)
```

```
def data(self, index: QModelIndex, role: int):
    if role == Qt.DisplayRole:
        row = index.row()
        col = index.column()
        return str(self.data[row][col])
```

```
class Window(QMainWindow, Ui_MainWindow):
    threadpool = QThreadPool()

    pause_signal = None
```

```

def __init__(self, config_path_project, config_path_master, parent=None):
    super().__init__(parent)
    self.setupUi(self)

    self.config = ConfigMaster(config_path_master)
    self.config_db = ConfigDBExternal(config_path_project)

    self.db = ExternalConnectDB(BaseBase, base_name, config_path_project)
    self.labels_dict = self.get_labels_dict()

    self.scores = Scores(self.db, config_path_master)

    self.channel, self.connection, self.queue_master_manage = self.init_pika()

    self.sender = Sender(self.channel, self.config)

    self.updater_queue = QueueUpdater(self.config, self.labels_dict)

    self.setup()

def setup(self):
    self.setup_buttons()
    self.setup_thread_queues()
    self.update_table()

def setup_thread_queues(self):

```

```

self.pause_signal = self.updater_queue.signals.pause_signal
self.pause_signal.connect(self.updater_queue.set_pause)

self.updater_queue.signals.dialog_signal.connect(self.open_dialog)

self.threadpool.start(self.updater_queue)

def setup_buttons(self):
    self.send.clicked.connect(self.send_text)
    self.send_file.clicked.connect(self.send_text_file)

    self.calculate.clicked.connect(self.calculate_scores)
    self.view.clicked.connect(self.view_scores)

def keyPressEvent(self, event):
    if event.key() == Qt.Key_F5:
        self.update_table()

def update_table(self):
    results = self.db.session.query(Articles.entry_id, Articles.title, Articles.summary,
                                   ValCategories.category_base, ValCategories.category_pred) \
        .select_from(Articles, ValCategories) \
        .join(ValCategories, ValCategories.entry_id == Articles.entry_id) \
        .limit(1000).all()

    model = ValTableModel(results)
    self.tableView.setModel(model)

```

```

self.tableView.show()

def send_text(self):
    entry_id = self.id_line.text()
    text = self.text.toPlainText()

    if entry_id != "" and text != "":
        if self.ds_app.isChecked():
            self.sender.send_text_app(entry_id, text)
        elif self.ds_null.isChecked():
            self.sender.send_text_null(entry_id, text)

def send_text_file(self):
    file_name = QFileDialog.getOpenFileName(self,
                                             'Open file',
                                             r'E:\Projects\NeuralDiploma\demo',
                                             'Table files (*.csv)')

    try:
        with open(file_name[0], newline="") as csvfile:
            file_reader = csv.reader(csvfile, delimiter=';', quotechar="")

            for row in file_reader:
                entry_id = row[0]
                text = row[1]

                if self.ds_app.isChecked():
                    self.sender.send_text_app(entry_id, text)

```

```

        elif self.ds_null.isChecked():
            self.sender.send_text_null(entry_id, text)
except FileNotFoundError:
    pass

def calculate_scores(self):
    self.threadpool.start(self.scores)

def view_scores(self):
    data = self.db.get_all(ValResult)
    data_sorted = {row['metric']: row['value'] for row in data}

    dlg = QMessageBox(self)
    dlg.setWindowTitle('Scores')

    text = ""

    if self.cb_abs_acc.isChecked():
        text += f'Absolute accuracy: {(data_sorted["Absolute accuracy"] * 100):.2f}%\n'
    if self.cb_any_acc.isChecked():
        text += f'Any accuracy: {(data_sorted["Any accuracy"] * 100):.2f}%\n'
    if self.cb_prc.isChecked():
        text += f'Recall: {(data_sorted["Recall"] * 100):.2f}%\n'
    if self.cb_rec.isChecked():
        text += f'Precision: {(data_sorted["Precision"] * 100):.2f}%\n'
    if self.cb_f_score.isChecked():
        text += f'F1-score: {(data_sorted["F1-score"] * 100):.2f}%\n'

```

```

if text != "":
    dlg.setText(text)
    dlg.exec()

def open_dialog(self, text):
    dlg = QMessageBox(self)
    dlg.setWindowTitle('Result')
    dlg.setText(text)
    dlg.exec()

self.pause_signal.emit(False)

def get_labels_dict(self):
    general_list_raw = self.db.get_all(GeneralCatList)
    general_list = {row['id']: row['category'] + ': ' + row['description'] for row in
general_list_raw}

    return general_list

def init_pika(self):
    parameters =
pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                    f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

```

```

queue_master_manage = channel.queue_declare(
    self.config.n_queue_master_manage,
    durable=True
)

return channel, connection, queue_master_manage

def except_hook(exc_type, exc_value, exc_tb):
    tb = ".join(traceback.format_exception(exc_type, exc_value, exc_tb))
    print('Error message:\n', tb)
    QApplication.quit()

if __name__ == "__main__":
    config_path_master = r'../conf/server.cfg'
    config_path_project = r'../conf/project.cfg'

    sys.excepthook = except_hook
    app = QApplication(sys.argv)
    win = Window(config_path_project, config_path_master)
    win.show()
    sys.exit(app.exec())

```

## **gui\_user/update\_queue.py**

```
# Base
import ast
import json
import sys
import subprocess
import time

# Internal
import multiprocessing
from addons.config import ConfigMaster

# External
from PyQt5.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
    QGraphicsPixmapItem, QFileDialog, QDialog, \
    QLabel, QPushButton, QVBoxLayout
from PyQt5.QtCore import Qt, QLineF, pyqtSignal, QThread, pyqtSignal, QObject,
    QRunnable, QEventLoop
from PyQt5.QtGui import QPixmap, QPen

import pika

class Signals(QObject):
    dialog_signal = pyqtSignal(str)
    pause_signal = pyqtSignal(bool)
```

```

class QueueUpdater(QRunnable):
    signals = Signals()

    pause = True

    def __init__(self, config, labels_dict):
        super(QueueUpdater, self).__init__()

        self.config = config

        self.labels_dict = labels_dict

        self.channel, \
        self.connection, \
        self.queue_response_queue, \
        = self.init_pika()

    def on_message(self, ch, method, properties, body):
        json_msg = json.loads(body, strict=False)

        entry_id = json_msg['entry_id']
        labels = ast.literal_eval(json_msg['labels'])

        if not json_msg['null']:
            labels_text = []
            for label in labels.keys():

```

```

    prob = labels[label]

    labels_text.append(f' {(prob * 100):.2f}% - {self.labels_dict[label] }')

    if labels_text:
        text = f'Entry ID: {entry_id}\n\n' \
              f'Labels:\n' + '\n'.join(labels_text)
    else:
        text = f'Entry ID: {entry_id}\n\n' \
              f'Labels: None'

    self.signals.dialog_signal.emit(text)

    while self.pause:
        time.sleep(1)
    self.pause = True

    self.channel.basic_ack(method.delivery_tag)
    return

def init_pika(self):
    parameters =
    pika.URLParameters(f'amqp://{self.config.username}:{self.config.password}'
                      f'@{self.config.server}/?heartbeat={self.config.heartbeat}')
    connection = pika.BlockingConnection(parameters)

    channel = connection.channel()
    channel.basic_qos(prefetch_count=1)

```

```

queue_response_queue = channel.queue_declare(
    self.config.n_queue_response_queue,
    durable=True
)

return channel, connection, queue_response_queue

def prepare(self):
    self.clear_queue(self.config.n_queue_master_monitor_queue)

def run(self):
    self.prepare()

    self.channel.basic_consume(
        queue=self.config.n_queue_response_queue,
        on_message_callback=self.on_message)

    self.clear_queue(self.config.n_queue_response_queue)

    try:
        self.channel.start_consuming()
    except Exception as e:
        print(e)

    self.connection.close()

def set_pause(self, pause):

```

```
self.pause = pause
```

```
def clear_queue(self, queue):  
    self.channel.queue_purge(queue)
```

### **gui\_user/messenger.py**

```
# Base
```

```
import json
```

```
class Sender:
```

```
    def __init__(self, channel, config):
```

```
        self.channel = channel
```

```
        self.config = config
```

```
    def send_result(self, queue, message):
```

```
        self.channel.basic_publish(exchange="", body=message.encode(encoding='UTF-8'),  
                                   routing_key=queue)
```

```
    def send_text_app(self, entry_id, text):
```

```
        message = {'entry_id': entry_id,
```

```
                   'text': text,
```

```
                   'null': False
```

```
        }
```

```
        json_message = json.dumps(message)
```

```

self.send_result(self.config.n_queue_process_queue, json_message)

def send_text_db(self, entry_id, text):
    message = {'entry_id': entry_id,
              'text': text,
              }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_process_queue, json_message)

def send_text_null(self, entry_id, text):
    message = {'entry_id': entry_id,
              'text': text,
              'null': True
              }

    json_message = json.dumps(message)
    self.send_result(self.config.n_queue_process_queue, json_message)

```

### **gui\_user/scores.py**

```

# Base
import ast
import json
import sys
import subprocess

```

```

import time

# Internal
from addons.config import ConfigMaster

from db.db_init.db_init_base import BaseBase, base_name, GeneralCatList,
ValCategories, ValResult

from db.db_connection.db_external import ExternalConnectDB

# External
from PyQt5.QtWidgets import QMainWindow, QApplication, QGraphicsScene,
QGraphicsPixmapItem, QFileDialog, QDialog, \
    QLabel, QPushButton, QVBoxLayout
from PyQt5.QtCore import Qt, QLineF, pyqtSignal, QThread, QObject, QRunnable,
QEventLoop
from PyQt5.QtGui import QPixmap, QPen

import sqlalchemy.exc
from sqlalchemy.dialects.mysql import insert

class Scores(QRunnable):
    def __init__(self, db, config_path):
        super(QRunnable, self).__init__()

        self.db = db
        self.config = ConfigMaster(config_path)

```

```
def run(self):

    abs_count = 0
    any_count = 0

    tp = 0
    tn = 0
    fp = 0
    fn = 0

    count = 0

    data_format = self.get_all_data()
    cats_count = self.get_cat_len()

    for row in data_format:
        cats_base = row[0]
        cats_pred = row[1]

        o_cats_vector = [0] * cats_count
        p_cats_vector = [0] * cats_count

        for i in range(cats_count):
            if i in cats_base:
                o_cats_vector[i] = 1
            if i in cats_pred:
                p_cats_vector[i] = 1
```

```

if sorted(o_cats_vector) == sorted(p_cats_vector):
    abs_count += 1

if any(cat in o_cats_vector for cat in p_cats_vector):
    any_count += 1

cat_pair = zip(o_cats_vector, p_cats_vector)

for pair in cat_pair:
    a = pair[0]
    b = pair[1]

    if a == b == 1:
        tp += 1
    elif a == b == 0:
        tn += 1
    elif a != b and b == 1:
        fp += 1
    elif a != b and a == 1:
        fn += 1

count += 1

pr = tp / (tp + fp)
rc = tp / (tp + fn)

```

```
f1 = tp / (tp + 0.5 * (fp + fn))
```

```
absl = abs_count / count
```

```
anya = any_count / count
```

```
result = {'Precision': pr,  
         'Recall': rc,  
         'F1-score': f1,  
         'Absolute accuracy': absl,  
         'Any accuracy': anya}
```

```
self.write(result)
```

```
def write(self, result):
```

```
    for cat in result.keys():
```

```
        insert_stmt = insert(ValResult).values(  
            metric=cat,  
            value=result[cat],  
        )
```

```
        on_duplicate_key_stmt = insert_stmt.on_duplicate_key_update(  
            value=result[cat],  
        )
```

```
self.db.session.execute(on_duplicate_key_stmt)
```

```
self.db.session.commit()
```

```

def get_cat_len(self):
    data = self.db.get_all(GeneralCatList)
    return len(data)

def get_all_data(self):
    data = self.db.get_all(ValCategories)

    data_format = []
    for row in data:
        data_format.append((ast.literal_eval(row['category_base']),
ast.literal_eval(row['category_pred'])))

    return data_format

if __name__ == '__main__':
    config_path = r'E:\Projects\NeuralDiploma\conf\project.cfg'
    config_path_server = r'E:\Projects\NeuralDiploma\conf\server.cfg'

    db_ext = ExternalConnectDB(BaseBase, base_name, config_path)

    scr = Scores(db_ext, config_path_server)
    scr.run()

```