Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка рішення для аналізу тендерної документації на платформі Smarttender

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Вихрущ Д.С. (прізвище та ініціали)

Керівник (підпис) Цебрій О.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) Готович В. А. (прізвище та ініціали)

Тернопіль
2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконана Вихрущ Дмитром Сергійовичем, студентом групи СП-41 Тернопільського національного технічного університету, присвячена розробці програмного рішення для автоматизованого аналізу тендерної документації з використанням технологій веб-скрапінгу. Обсяг роботи становить 69 сторінки, містить 8 рисунків, 4 додатки та список використаних джерел з 19 позицій.

Метою дослідження є створення інструменту для збору та аналізу даних з публічних тендерних платформ за допомогою технологій веб-скрапінгу. У роботі реалізовано систему автоматизованого парсингу тендерної документації, яка дозволяє витягувати ключові параметри: предмет закупівлі, бюджет, строки, кваліфікаційні вимоги. Для обробки отриманих даних застосовано методи машинного навчання та NLP (обробки природної мови), що дає змогу класифікувати тендери та виявляти критичні умови участі.

Розроблений інструмент демонструє високу ефективність у порівнянні з ручним аналізом, скорочуючи час обробки на 70–80%. Особливу увагу приділено обходженню анти-скрапінгових захистів та легальності отримання даних. Практична цінність роботи полягає у можливості масштабування рішення для різних тендерних майданчиків та інтеграції з CRM-системами. Результати дослідження можуть бути корисні для учасників закупівель, аналітиків ринку та розробників бізнес-рішень.

Отримані результати засвідчують доцільність подальшого розвитку створеної системи як окремого програмного продукту.

Ключові слова: Python, API-інтеграція, тендерний аналіз, веб-скрапінг, парсинг даних, машинне навчання, автоматизація закупівель, NLP-обробка.

ABSTRACT

The Bachelor's qualification thesis completed by Dmytro Vykhrushch, a student of group SP-41 at Ternopil Ivan Puluj National Technical University, is dedicated to the development of a software solution for automated analysis of tender documentation using web scraping technologies. The thesis consists of 69 pages, includes 8 figures, 4 appendices, and a list of references comprising 19 sources.

The aim of the research is to create a tool for collecting and analyzing data from public tender platforms through web scraping techniques. The work presents an automated system for parsing tender documents, which extracts key parameters such as procurement subject, budget, deadlines, and qualification requirements. For processing the extracted data, methods of machine learning and natural language processing (NLP) are applied, enabling the classification of tenders and the identification of critical participation conditions.

The developed tool demonstrates high efficiency compared to manual analysis, reducing processing time by 70–80%. Special attention is given to bypassing anti-scraping mechanisms and ensuring the legal compliance of data acquisition. The practical value of the project lies in the potential scalability of the solution to various tender platforms and its integration with CRM systems. The research results may be useful for procurement participants, market analysts, and business solution developers.

The results confirm the feasibility of further developing the system as a standalone software product.

Keywords: Python, API integration, tender analysis, web scraping, data parsing, machine learning, procurement automation, NLP processing.

ЗМІСТ

ЗМІСТ 6
ПЕРЕЛІК СКОРОЧЕНЬ 9
ВСТУП 10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 12
1.1 Електронні тендерні платформи та особливості документації 12
1.1.1 Огляд систем Smarttender, Prozorro, їх структура і призначення 14
1.1.2 Формат та типові елементи тендерної документації 15
1.1.3 Основні проблеми ручного аналізу: витрати часу, помилки, ризики ... 17
1.2 Технології веб-скрапінгу та автоматизованого збору даних 18
1.2.1 Інструменти для парсингу HTML-сторінок і API (Selenium, BeautifulSoup, Playwright) 19
1.2.2 Взаємодія з обмеженими платформами: CAPTCHA, throttling, legal aspects 20
1.3 NLP та машинне навчання в аналізі тендерів 21
1.3.1 Великі мовні моделі (LLM) у задачах витягу інформації 21
1.3.2 Виявлення вимог до документів у тендерному тексті 24
1.3.3 Побудова інструкцій (prompts) до LLM і приклади взаємодії 25
1.3.4 Обмеження та виклики у застосуванні LLM 26
1.4 Вимоги до системи аналізу тендерної документації 27
2 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ 29
2.1 Вибір технологій і загальна архітектура системи 29
2.2 Модуль збору даних (web-scraping) 30

2.2.1 Загальна логіка роботи модуля 31
2.2.2 Вхідні параметри 31
2.2.3 Внутрішня структура класу 31
2.2.4 Формат отриманих даних 32
2.2.5 Обробка вкладених документів 33
2.2.6 Приклад згенерованого результату 34
2.3 Модуль попередньої обробки тексту 34
2.3.1 Конвертація та уніфікація форматів документів 35
2.3.2 Витяг і попереднє очищення тексту 36
2.3.3 Розбиття тексту на фрагменти для аналізу 37
2.3.4 Вибір релевантних фрагментів за допомогою embedding-порівняння . 38
2.4 Аналіз тексту за допомогою великої мовної моделі 39
2.4.1 Побудова запиту до мовної моделі 39
2.4.2 Вибір моделі та формат відповіді 40
2.4.3 Вибір моделі 40
2.5 Створення веб-інтерфейсу та тестування 42
2.5.1 Створення REST-ендпойнту 43
2.5.2 Приклад використання 44
2.5.3 Розгортання сервісу у Docker 44
3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 46
3.1 Вплив діяльності людини на довкілля. 46
3.2 Заходи захисту обладнання від статичної електрики 47
ВИСНОВКИ 50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 51

ДОДАТКИ..... 53
Додаток А..... 54
Додаток Б 55
Додаток В..... 57
Додаток Г 61
Додаток Д..... 70

ПЕРЕЛІК СКОРОЧЕНЬ

UI – User Interface

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

URL – Uniform Resource Locator

JS – JavaScript

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

JSON – JavaScript Object Notation

REST – Representational State Transfer

LLM – Large Language Model

GPT – Generative Pre-trained Transformer

PDF – Portable Document Format

DOCX – Microsoft Word Open XML Document

Docker – Software platform for developing, shipping, and running applications in containers

ВСТУП

Із розвитком цифрових технологій процеси публічних закупівель в Україні та у світі зазнають суттєвих трансформацій. Відкриті електронні тендерні платформи, такі як Smarttender, Prozorro, Zakupki та інші, надають доступ до великої кількості структурованих і неструктурованих даних, які містять важливу інформацію про умови закупівель, вимоги до постачальників, бюджети та строки виконання контрактів. Однак, попри формальну відкритість і доступність, обробка тендерної документації залишається складним завданням, що потребує значних часових та інтелектуальних ресурсів.

Значна частина інформації у тендерних оголошеннях подається у вигляді текстових описів, що ускладнює автоматичну обробку та класифікацію. Учасники торгів, зокрема представники малого і середнього бізнесу, часто змушені витрачати багато часу на ручне опрацювання кожної пропозиції з метою оцінити релевантність, відповідність власним можливостям і наявність потенційних ризиків. У таких умовах актуальною стає потреба у створенні інструментів, які б автоматизували процес аналізу тендерної документації та надавали користувачам релевантну інформацію в зручному та зрозумілому форматі.

Автоматизація цього процесу можлива завдяки поєднанню кількох сучасних технологій, зокрема веб-скрапінгу, машинного навчання (ML), обробки природної мови (NLP) та великих мовних моделей (LLM – Large Language Models). Саме LLM сьогодні демонструють значний прогрес у розумінні текстових даних, що дозволяє не лише витягувати ключові параметри з документації, а й формувати семантичне уявлення про її зміст, визначати ризики, узагальнювати вимоги та робити прогнози на основі текстового контексту. Завдяки цьому великі мовні моделі відкривають нові можливості для побудови інтелектуальних систем, які здатні аналізувати документи майже на рівні людського експерта, але з набагато вищою швидкістю.

Однією з проблем, яка постає перед розробником такого інструменту, є вибір найбільш ефективної архітектури рішення. Різні способи побудови парсера, підходи до збирання даних, вибір NLP-бібліотек або моделей (наприклад, spaCy, BERT, GPT або LLaMA) дають різний рівень точності, продуктивності та адаптивності. Тому доцільно провести порівняльний аналіз декількох варіантів реалізації, щоб визначити оптимальний підхід з точки зору швидкодії, точності та стійкості до змін у структурі вебсторінок.

Ще одним важливим аспектом є юридичний і етичний компонент: з одного боку, веб-скрапінг може порушувати правила користування певних платформ, а з іншого — забезпечує прозорість та доступ до публічних даних, гарантованих законодавством. У межах даної роботи розглядаються тільки ті підходи, які відповідають принципам етичного збору даних і не порушують нормативно-правові акти.

Таким чином, дана кваліфікаційна робота спрямована на вирішення комплексного завдання: автоматизувати аналіз тендерної документації з одночасним підвищенням точності, швидкості та зручності отримання інформації. Метою є не лише створення технічно працездатного інструменту, а й забезпечення реальної практичної цінності для користувачів — бізнес-аналітиків, закупівельників, представників компаній, які беруть участь у торгах. Розроблена система має потенціал стати частиною ширшого екосередовища автоматизації закупівель і може бути інтегрована в CRM, ERP або спеціалізовані аналітичні модулі.

У процесі реалізації поставленої мети буде проведено аналіз існуючих рішень, здійснено порівняння сучасних моделей обробки текстових даних, визначено критерії ефективності системи та побудовано повноцінне прототипне програмне забезпечення з можливістю подальшого масштабування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Публічні закупівлі давно стали не просто адміністративною процедурою, а джерелом великих обсягів структурованої та неструктурованої інформації. Щодня на тендерних платформах з'являються сотні документів, серед яких – технічні вимоги, договірні умови, критерії відбору. Із цього масиву потрібно швидко та точно витягнути ключове: що купують, хто може подати заявку, які умови можуть викликати ризики, і найголовніше — список документів, потрібних для участі в тендері.

Все це — є задачею для сучасних інструментів аналізу тексту. Сьогодні ми маємо у розпорядженні не лише регулярні вирази та ключові слова, а й повноцінні мовні моделі, здатні «читати між рядків». Цей розділ — спроба окреслити, з чим ми маємо справу, що вже існує, і які технології можна використати, щоби зробити аналіз тендерної документації автоматизованим, швидким і розумним.

1.1 Електронні тендерні платформи та особливості документації

Сфера публічних закупівель в Україні за останнє десятиліття зазнала суттєвих змін у напрямку цифровізації та відкритості даних. Ключовим кроком стало впровадження електронних платформ для проведення тендерів, серед яких найбільш відомими є Prozorro — державна платформа, та Smarttender — одна з найбільших комерційних електронних торговельних систем.

Prozorro — це централізована система електронних закупівель, яка була запущена у 2016 році відповідно до Закону України «Про публічні закупівлі». Її головною метою є забезпечення прозорості торгів, запобігання корупції та спрощення процесу взаємодії між замовниками й постачальниками. У Prozorro

публікується вся інформація про закупівлі — від оголошення тендера до укладення договору. Всі документи та заявки є публічними, що створює сприятливе середовище для аналітики. Архітектура Prozorro побудована за принципом централізованої бази даних із можливістю доступу через авторизовані майданчики, серед яких Smarttender виступає одним із найбільших.

Smarttender.biz є комерційною платформою, яка підтримує як державні, так і комерційні закупівлі. Вона інтегрована з Prozorro, що дає змогу здійснювати повний цикл участі у публічних торгах через її інтерфейс. Окрім того, Smarttender розвиває власний функціонал, орієнтований на B2B-сегмент: внутрішні корпоративні закупівлі, тендери закритого типу, зберігання шаблонів документів, автоматичну перевірку учасників тощо [4].

З технічного боку, обидві системи мають схожі компоненти:

- публічний веб-інтерфейс для перегляду закупівель;
- особисті кабінети користувачів для подання пропозицій і документів;
- API-доступ для автоматизації взаємодії;
- сховище документів, у якому зберігаються тендерні файли, технічні завдання, вимоги тощо.

Особливістю обох платформ є наявність розгалуженої системи фільтрації та пошуку, яка, хоч і зручна для ручної роботи, але не забезпечує гнучкої інтеграції для повноцінного машинного аналізу. Саме тому виникає потреба у створенні окремих інструментів збору, парсингу та обробки тендерної інформації, зокрема — витягнення змісту тендерної документації, яка часто доступна лише у вигляді вкладених PDF- або DOC-файлів.

1.1.1 Огляд систем Smarttender, Prozorro, їх структура і призначення

Публічні закупівлі в Україні давно вийшли за межі паперових процедур і сьогодні повністю переведені в цифровий формат. Основним елементом цієї системи є електронні платформи, через які проходить весь цикл закупівель — від оголошення до укладання договору. Найбільш відомими серед них є державна система Prozorro та один з найбільших комерційних майданчиків — Smarttender.biz.

Prozorro — це офіційна електронна система публічних закупівель, яка працює під контролем держави та забезпечує юридично обов’язкову взаємодію між замовниками та постачальниками. На практиці Prozorro не є класичним веб-сайтом, а скоріше централізованою базою даних, до якої підключаються окремі електронні торгові майданчики (ЕТМ), такі як Smarttender, через стандартизований API. Уся інформація про тендери — від оголошень до підписаних договорів — зберігається в Prozorro, і будь-який користувач може отримати до неї доступ через інтерфейс одного з майданчиків.

Smarttender — один із таких майданчиків. Він виконує дві ролі: по-перше, як авторизований інтерфейс для участі в державних тендерах, а по-друге — як комерційна платформа для приватних закупівель. Ця двофункціональність робить Smarttender привабливим для бізнесу, який працює як із держсектором, так і з приватними партнерами. Окрім базового функціоналу (створення тендерів, подача пропозицій, завантаження документації), Smarttender також пропонує додаткові сервіси: шаблони документів, перевірку учасників, внутрішні тендери та аналітичні інструменти [4].

З точки зору структури, обидві системи мають схожі компоненти:

- веб-інтерфейс для пошуку та перегляду тендерів;
- особистий кабінет користувача;

- можливість завантаження та перегляду документації у вигляді прикріплених файлів (зазвичай PDF, DOCX, XLS);
- доступ до API, що дозволяє інтегруватися з зовнішніми програмами або автоматизувати подання заявок.

Хоча доступ до тендерних даних є відкритим, структура самих документів часто не є формалізованою. Багато важливої інформації (вимоги, критерії, обмеження) знаходяться всередині вкладених текстових файлів, написаних довільною мовою. Це ускладнює аналіз і порівняння тендерів, особливо коли таких документів десятки або сотні.

У цьому контексті Smarttender і Prozorro є не лише джерелами інформації, а й точками входу для автоматизації. Вивчення їхньої структури є ключовим етапом для розробки програмного рішення, яке зможе ефективно збирати, обробляти та аналізувати тендерні дані без втрати змісту.

1.1.2 Формат та типові елементи тендерної документації

У системах публічних закупівель, таких як Prozorro чи Smarttender, основна частина інформації для потенційного учасника торгів зосереджена не стільки в оголошенні, як у прикріпленій тендерній документації. Ці документи є ключовим джерелом вимог, умов та інструкцій, необхідних для подання повноцінної пропозиції. Саме тому їх формат і зміст мають вирішальне значення при спробі автоматизувати процес аналізу закупівель.

Найчастіше замовники публікують тендерну документацію у вигляді окремих файлів, які завантажуються на платформу як додатки до тендеру. Це можуть бути текстові документи, електронні таблиці або скановані PDF-файли, у яких міститься вся необхідна інформація щодо предмета закупівлі, кваліфікаційних

вимог до учасника, строків поставки, форм подачі документів, умов укладання договору тощо. Особливістю такої документації є відсутність єдиного шаблону або чітко стандартизованої структури. Різні замовники подають інформацію по-різному: хтось користується логічною розбивкою тексту на тематичні розділи, а хтось об'єднує всі умови в один суцільний документ без чітких заголовків чи форматування.

Попри різницю в оформленні, більшість тендерних документів мають спільні елементи, які повторюються з торгів у торги. Як правило, це опис предмета закупівлі з технічними характеристиками, очікувана вартість контракту, строк постачання або виконання робіт, вимоги до досвіду та документального підтвердження кваліфікації учасників, умови укладання та виконання договору, а також форми, які потрібно заповнити і подати разом із заявкою. Часто до документації додається ще й проект договору, який передбачено укласти із переможцем.

Оскільки ці дані подаються у вигляді тексту, часто неструктурованого і написаного довільною мовою, їх автоматичне опрацювання пов'язане з низкою викликів. Наприклад, навіть така, здавалося б, проста інформація як очікувана вартість закупівлі може бути подана в різних форматах, із різними одиницями виміру, а іноді — і в зовсім іншому документі, ніж основна специфікація. Те саме стосується кваліфікаційних вимог: вони можуть бути описані детально в окремому розділі або згадані побіжно в кількох абзацах, розкиданих по всьому документу.

Тендерна документація, хоч і є відкритою, не адаптована до машинної обробки. Вона створена"для людей" — і саме тому її потрібно перетворювати, у форму, зрозумілу для автоматизованих систем. Це, своєю чергою, формує технологічний запит на гнучкі інструменти обробки тексту, здатні працювати не з шаблоном, а з мовною логікою, змістом і контекстом документів.

1.1.3 Основні проблеми ручного аналізу: витрати часу, помилки, ризики

Попри те, що більшість даних у сфері публічних закупівель є відкритими, їх опрацювання вимагає значних зусиль. Користувач, який хоче подати пропозицію, змушений самостійно знаходити, завантажувати, читати й аналізувати тендерну документацію, обсяг якої іноді сягає кількох десятків сторінок. Звісно, для досвідчених учасників цей процес уже частково налагоджений, однак навіть вони витрачають чимало часу на щоденне опрацювання нових тендерів.

Головна проблема полягає в тому, що інформація в тендерних документах не завжди подається логічно або послідовно. Вона може бути розкидана по різних файлах, повторюватися, суперечити сама собі або бути сформульованою не зовсім однозначно. Наприклад, кваліфікаційні вимоги можуть згадуватися в одному абзаці загального опису, а потім уточнюватися в додатку, або ж відрізнятися між основним документом і формою заявки. В такому випадку навіть уважний читач не завжди може одразу зрозуміти, що саме вимагає замовник.

Ще одна суттєва складність — мовна варіативність. У різних галузях використовуються свої терміни, шаблони та формулювання. Замовники можуть писати «постачальник повинен мати досвід», або «не допускаються компанії, що не виконували аналогічні проєкти», або ще інакше — і всі ці варіанти по суті означають одне й те саме, але вимагають окремого прочитання, аналізу й інтерпретації. Через це значно ускладнюється побудова чіткої внутрішньої логіки документа, особливо якщо потрібно переглянути не один, а десятки таких файлів.

В умовах конкуренції навіть кілька зайвих годин, витрачених на аналіз, можуть вплинути на рішення брати участь у тендері чи ні. У малому й середньому бізнесі часто немає окремої людини, яка займається лише тендерами, тому аналіз документації виконується між іншими справами, швидко і з ризиком упустити важливу деталь. Помилки ж у таких випадках коштують дорого: невчасно поданий

документ, невідповідність формату, відсутність довідки — і заявка автоматично відхиляється.

Найтиповіші наслідки таких помилок — це технічне відхилення заявки через порушення формальних вимог, втрачена можливість участі через поспіх, неповне розуміння змісту умов або пропуск дискримінаційних положень, які могли би бути оскаржені ще до подання пропозиції.

1.2 Технології веб-скрапінгу та автоматизованого збору даних

У контексті роботи з тендерною інформацією першим кроком завжди є її отримання. Якщо йдеться про платформи на кшталт Prozorro чи Smarttender, то значна частина важливих даних знаходиться не в структурованих API-відповідях, а безпосередньо на вебсторінках або в прикріплених документах. Саме тому веб-скрапінг — тобто автоматизоване зчитування інформації з HTML-сторінок — залишається одним з основних інструментів доступу до таких даних.

На практиці веб-скрапінг передбачає імітацію дій користувача: відкриття сторінки, завантаження вмісту, знаходження потрібних елементів за тегами чи CSS-класами, і витяг тексту або посилань. Для цього найчастіше використовують такі бібліотеки, як BeautifulSoup, Selenium, Playwright, або повноцінні фреймворки типу Scrapy. Вибір інструменту залежить від складності сайту: якщо достатньо просто «прочитати» HTML, підійде легке рішення, але коли сайт динамічний, із завантаженням контенту через JavaScript, доводиться використовувати браузерні емулятори [2].

Окремим аспектом є обходження технічних обмежень: деякі сайти захищаються від автоматичного зчитування за допомогою CAPTCHA, обмеження частоти запитів, приховання структури сторінки. Тут допомагають ротація проксі, налаштування User-Agent і затримки між запитами.

У випадку Smarttender частина інформації доступна через API, але не всі поля зручні для аналізу — тому комбінування API і скрапінгу дозволяє охопити весь обсяг потрібних даних. В ідеальному сценарії структура сайту стабільна, і скрипти можуть регулярно оновлювати базу для подальшої обробки [4].

1.2.1 Інструменти для парсингу HTML-сторінок і API (Selenium, BeautifulSoup, Playwright)

Автоматизація збору даних із тендерних платформ зазвичай передбачає два основні підходи: парсинг HTML-сторінок або роботу через відкриті програмні інтерфейси (API). Вибір між ними визначається рівнем доступності даних, складністю структури вебсторінки та стабільністю адрес.

У випадку з платформою Smarttender доступ до детальної інформації про закупівлі можна отримати як через HTML-інтерфейс, так і через внутрішні запити, які виконуються динамічно в браузері. При аналізі конкретного тендера за ID 38246622 було виявлено, що дані завантажуються з адреси:
(https://smarttender.biz/PurchaseDetail/GetTenderModel/?tenderId=38246622)

Цей URL повертає структуровану JSON-відповідь з усіма основними параметрами тендеру, включно з його предметом, строками, бюджетом, організацією-замовником, типом процедури та переліком прикріплених документів. Оскільки доступ до цього ресурсу не потребував авторизації, він був використаний у роботі як джерело сирих даних для подальшої обробки [11].

Для отримання вмісту сторінок або виклику API застосовуються такі популярні інструменти, як Selenium, BeautifulSoup, Playwright, або звичайні HTTP-клієнти на кшталт requests у Python. Selenium дозволяє емулювати повноцінну роботу браузера — переходити по сторінках, чекати на завантаження елементів, взаємодіяти з кнопками чи випадаючими списками [2].

У моєму випадку, з огляду на відкритість ендпоінта, не було необхідності в браузерній автоматизації — достатньо було надіслати запит до API та отримати у відповідь JSON-об’єкт. Надалі ці об’єкти розбиралися, і з них витягувались ключові поля, які подавались на вхід наступним модулям — зокрема NLP-компоненту, що аналізував текст тендерної документації.

Таким чином, об’єднання традиційного парсингу та роботи з API дозволяє отримати максимально повну картину про тендер без залучення надлишкових технічних ресурсів.

1.2.2 Взаємодія з обмеженими платформами: CAPTCHA, throttling, legal aspects

У рамках цієї роботи не виникало потреби в обході захисних механізмів, таких як CAPTCHA, throttling чи авторизаційні бар’єри, оскільки необхідні дані були доступні без обмежень. Проте ці аспекти є невід’ємною частиною теми веб-скрапінгу, й тому їх варто окремо розглянути з позиції загального розуміння технологій автоматизованого збору інформації.

Багато сучасних вебплатформ застосовують активні захисти проти ботів. Щоб обійти ці обмеження, зазвичай використовують техніки, пов’язані з проксі-серверами, затримками між запитами, зміною user-agent або навіть повноцінною емуляцією браузера.

Окремим питанням є правова сторона доступу до інформації. Не всі ресурси дозволяють використання скрапінгу, навіть якщо дані на перший погляд є публічними. Умови використання сайту можуть містити обмеження на автоматизоване зчитування або подальше поширення зібраної інформації. У цьому контексті важливо, що платформи державних закупівель, зокрема Smarttender, базуються на законі «Про публічні закупівлі» та оперують відкритими даними. Це

знімає юридичні ризики при обробці таких даних, за умови, що вони використовуються в інформаційно-аналітичних цілях без спотворення чи комерційного переосмислення.

Хоча в межах цієї розробки не виникало потреби у використанні методів обходу захистів, їх розуміння залишається важливим — як для подальшого масштабування рішення, так і для потенційної адаптації під інші платформи з більш жорсткою політикою доступу до даних.

1.3 NLP та машинне навчання в аналізі тендерів

Аналіз тендерної документації неможливий без розуміння її текстового змісту. Оскільки більшість вимог подається у вільній формі, традиційні методи пошуку неефективні. Тут на перший план виходять технології обробки природної мови (NLP), які дозволяють автоматично витягувати ключові умови, знаходити критичні фрази та класифікувати закупівлі за змістом.

Поєднання NLP із машинним навчанням відкриває можливості для побудови адаптивних моделей, здатних навчатися на прикладах і підвищувати точність аналізу з часом. Зокрема, такі підходи дозволяють визначати тип закупівлі, та витягувати документи, потрібні для участі в тендері.

1.3.1 Великі мовні моделі (LLM) у задачах витягу інформації

Великі мовні моделі (Large Language Models, LLM), такі як GPT, BERT, Claude або LLaMA, стали ключовим інструментом в обробці текстових даних,

зокрема — для завдань витягу інформації. Їхня основна перевага полягає в здатності не лише аналізувати текст на рівні слів, але й «розуміти» його контекст, структуру й логіку викладу, що особливо важливо у сфері тендерної документації [5].

У класичних підходах до NLP витяг інформації здійснювався через заздалегідь визначені шаблони, правила або моделі машинного навчання з вузькою спеціалізацією. Натомість LLM працюють за принципом генеративного передбачення — вони «навчені» на величезних обсягах тексту й здатні відповісти на запит, витягти потрібні фрагменти, узагальнити чи переформулювати інформацію без жорстко заданих правил [1].

З технічної точки зору, більшість сучасних LLM побудовані на архітектурі трансформерів — нейронних мереж, які використовують механізм self-attention для того, щоб враховувати зв’язки між словами в контексті всього тексту, а не лише в межах фіксованого вікна. Це дозволяє моделі ефективно працювати з довгими документами, виявляти логічні залежності, дублювання, суперечності або пропущені елементи [3].

Архітектура Transformer складається з двох основних частин — енкодера та декодера, кожен з яких містить кілька однакових блоків. Енкодер відповідає за обробку вхідного тексту та побудову контекстуальних представлень (embedding), а декодер — за генерацію вихідного тексту або відповідей. Ключовими компонентами трансформера є багатоголовий self-attention, який дозволяє моделі паралельно враховувати зв’язки між різними словами, та позиційне кодування, що додає інформацію про порядок слів. Така архітектура є масштабованою, ефективною для навчання на великих корпусах і універсальною для різних завдань обробки природної мови.

Нижче наведено рисунок архітектури Transformer

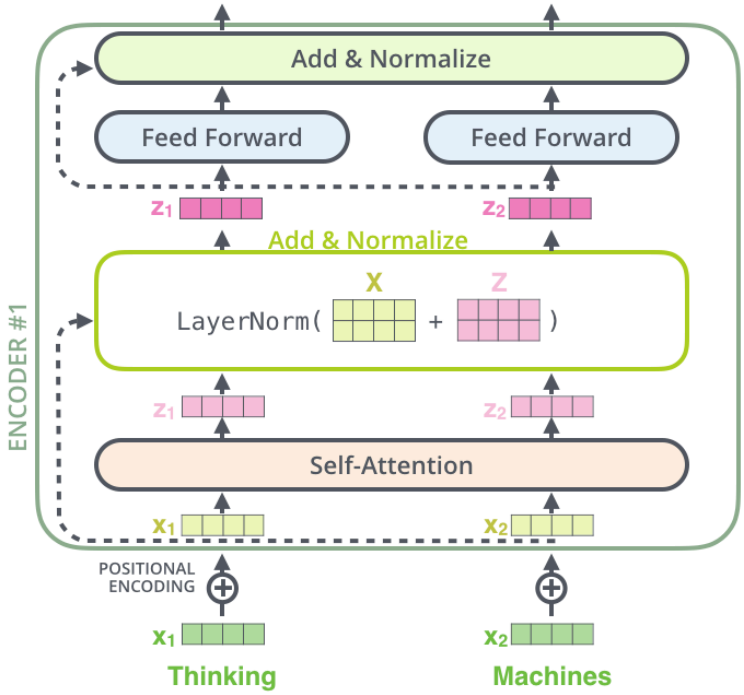


Рис 1.1 — Архітектура Transformer з механізмом self-attention

Для задачі витягу обов’язкових документів із тендерних текстів такі моделі підходять ідеально. Вони здатні знаходити речення на зразок «учасник повинен надати», «до заявки додається», «необхідно подати такі документи», навіть якщо ці фрази сформульовані інакше або розкидані по різних частинах тексту. Модель враховує як явні інструкції, так і контекст, у якому згадуються вимоги — що дає змогу значно підвищити точність та повноту витягу порівняно з простим пошуком ключових слів.

Таким чином, використання LLM у задачах інформаційного витягу дозволяє перейти від жорсткої формальної логіки до гнучкого семантичного аналізу, що особливо актуально у роботі з реальними тендерними документами, де немає стандартизованої структури.

1.3.2 Виявлення вимог до документів у тендерному тексті

Одним із практичних завдань при роботі з тендерною документацією є витяг списку документів, які потрібно подати для участі в процедурі закупівлі. У більшості випадків такі вимоги не подаються у стандартизованому вигляді, а розміщені у довільній формі в різних частинах тексту. Це ускладнює аналіз і часто потребує уважного прочитання всієї документації.

Зазвичай такі вимоги формулюються як: «учасник повинен надати», «до пропозиції додається», «необхідно подати такі документи» або подібні. Однак ці формулювання можуть бути подані з нюансами, умовами або в юридично складній формі, наприклад: «у разі наявності…», «якщо це передбачено законодавством…», «на підтвердження відповідності…». Через це автоматичне виявлення переліку потрібних документів вимагає не просто пошуку ключових слів, а розуміння контексту, в якому ці слова з’являються.

У межах цієї роботи для цього застосовуються великі мовні моделі, які дають змогу працювати з такими текстами не як зі звичайним набором слів, а як із логічно пов’язаними смисловими блоками. Модель аналізує, в яких розділах документації згадуються документи, як саме вони описані, і що саме з цього списку дійсно є обов’язковим, а не умовним. Це дозволяє автоматично знаходити й виокремлювати вимоги до пакету документів навіть у випадках, коли вони не подані у вигляді чітких списків.

Таке рішення економить значну кількість часу і може бути застосоване в системах підтримки прийняття рішень для учасників закупівель або аналітиків, які працюють з великими масивами тендерних даних.

1.3.3 Побудова інструкцій (prompts) до LLM і приклади взаємодії

Для того, щоб отримати від великої мовної моделі коректну відповідь, важливо правильно сформулювати запит (тобто prompt). У випадку з тендерною документацією, модель не має структурованої інформації, як у базі даних, тому результат напряму залежить від того, як саме поставити питання.

Наприклад, якщо задати занадто загальний запит типу «Що потрібно подати для участі в тендері?», модель може дати узагальнену відповідь, яка не стосується конкретного тексту. Натомість, якщо дати їй інструкцію на кшталт:
«Проаналізуй наведений нижче текст та виділи список документів, які є обов’язковими для подачі пропозиції», то результат буде набагато точніший. Бажано також вказати, у якому форматі бажано отримати відповідь — наприклад, нумерований список, або таблиця з колонкою “Назва документа” і “Примітки”.

В процесі експериментів я пробував різні формулювання інструкцій. В деяких випадках модель «перестаралася» і почала вигадувати документи, яких в тексті не було. Щоб уникнути цього, я додавав уточнення типу: «Використовуй тільки ті формулювання, які присутні в тексті. Не додавай нічого від себе.» Це допомогло зменшити кількість помилкових включень.

Ще один нюанс — іноді модель не розпізнає, що фрази «у випадку, якщо...» не означають обов’язковість. Через це до списку можуть потрапити документи, які є необов’язковими. Це питання можна частково вирішити через доопрацювання prompt’у, або ж валідацією результату вручну (наприклад, порівнюючи його з текстом оригіналу).

Загалом, побудова промптів — це не просто технічне завдання, а скоріше робота з мовною логікою. Один і той самий текст можна проаналізувати дуже по-різному залежно від формулювання інструкції. Тому тестування кількох варіантів запиту є обов'язковою частиною підходу до такої задачі.

1.3.4 Обмеження та виклики у застосуванні LLM

Хоча великі мовні моделі демонструють високі результати у роботі з текстами, при реальному аналізі тендерної документації виникають низка практичних обмежень, з якими доводиться рахуватись.

Найперше — це обсяг вхідного тексту. Тендерні файли часто містять по 20–30 сторінок, а іноді й більше. Спроба передати всю інформацію в одному запиті не завжди доречна: навіть якщо текст поміщається в межах технічного ліміту моделі, вона починає «розсіювати увагу» — дає загальні, неструктуровані відповіді, або ігнорує частину даних. Особливо це помітно, якщо документ написаний юридичною мовою з багатьма умовностями й повтореннями.

У межах цієї роботи я пробував кілька стратегій. Один із підходів — подавати модель текст по сторінках, витягуючи можливі згадки документів з кожної окремо. Метод працює, але не завжди — буває, що вимога починається на одній сторінці, а продовжується на наступній, і тоді модель не сприймає її як цілісну.

Також я тестував розбиття тексту по абзацах. Це дозволяє працювати більш точно, особливо коли інформація стисла й логічно завершена. Проте такий підхід потребує додаткової обробки — об’єднання результатів, відкидання дублікатів і ручної перевірки.

Третій і найбільш перспективний метод — це embedding-пошук. Суть полягає в тому, що документ ділиться на фрагменти (чанки), кожен з яких перетворюється у векторну форму. Далі обчислюється схожість кожного чанка до запиту, наприклад: «Які документи потрібно подати для участі в тендері?». Найбільш релевантні частини тексту передаються моделі, яка вже генерує відповідь на їх основі. Такий варіант дає змогу уникнути перевантаження моделі та зосередитися лише на дійсно важливих фрагментах.

Втім, і цей підхід має свої обмеження. Якщо у фрагменті тексту, який формально не схожий на запит, але містить важливу вимогу, embedding-порівняння

може просто не виявити його. У результаті окремі документи не потраплять у вибірку й, відповідно, не будуть згадані у відповіді моделі. Це особливо критично для «розпорошених» вимог, які не мають очевидної форми.

Тому, навіть при використанні сучасних методів, необхідно зберігати обережність і враховувати можливість пропуску. Для підвищення надійності система має або допускати післяаналіз, або поєднувати кілька стратегій одночасно.

1.4 Вимоги до системи аналізу тендерної документації

Система для автоматизованого аналізу тендерної документації повинна відповідати низці вимог, які забезпечують її точність, зручність та практичну користь. Основне завдання — допомогти користувачеві швидко орієнтуватися у великих обсягах тексту та отримати чітку відповідь щодо того, які саме документи необхідні для участі в тендері.

Найважливішою вимогою є надійність витягу обов’язкових документів. Пропуск навіть одного документа може мати серйозні наслідки для учасника тендеру, аж до дискваліфікації. Тому система повинна мати високу чутливість до формулювань, які вказують на вимоги до документації, навіть якщо вони сформульовані непрямо або подані у складній юридичній мові.

У той же час допускається помірна надмірність у результатах — тобто включення до списку документів таких, що не є обов’язковими. Такий компроміс є прийнятним, оскільки ризик від зайвого включення менший, ніж від пропуску. Користувач може самостійно переглянути перелік і відкинути зайве.

Крім точності аналізу, система повинна відповідати ще кільком ключовим вимогам:

- Масштабованість — здатність обробляти великі документи (30 сторінок і більше), а також пакетне завантаження кількох тендерів.
- Ясність результату — відповідь має бути зручною для користувача: зрозуміло структурованою, читабельною, бажано з можливістю подальшого збереження або експорту.
- Можливість інтеграції — у разі потреби, результати роботи мають легко інтегруватися у сторонні системи (наприклад, CRM або аналітичні панелі).
- Мінімальна залежність від вручну прописаних шаблонів — бажано, щоб система могла працювати з новими документами без потреби постійного оновлення правил.

Ці вимоги формують основу для розробки рішення, яке буде не просто демонстраційним, а дійсно корисним у практичному застосуванні

2 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

У цьому розділі розглянуто технічну реалізацію системи, яка виконує автоматизований аналіз тендерної документації на основі її унікального ідентифікатора. Рішення поєднує інструменти для збору даних із відкритих джерел (веб-скрапінг) та великі мовні моделі (LLM) для аналізу тексту і витягу обов'язкових документів. Результатом роботи системи є стисла структурована відповідь, яка включає загальну інформацію про тендер і список вимог до учасника.

2.1 Вибір технологій і загальна архітектура системи

Розроблена система виконує автоматизований аналіз тендерної документації за унікальним ідентифікатором тендеру. Вона поєднує компоненти для збору даних з відкритих веб-ресурсів і обробки тексту за допомогою великих мовних моделей. Кінцева мета — сформувати стисле, зрозуміле представлення основної інформації про тендер, а також список документів, необхідних для участі.

Архітектура системи складається з трьох основних компонентів:

- Збір даних — надсилання HTTP-запиту до відкритого API платформи Smarttender для отримання JSON-структури, що містить опис тендеру.
- Аналіз тексту — обробка тендерної частини за допомогою LLM (GPT або DeepSeek), формування короткого опису закупівлі та виявлення вимог до документації [10].
- Інтерфейс користувача — повернення результату у форматі текстової відповіді через FastAPI, з можливістю перегляду в браузері або обробки через API-запит.

У якості мови програмування обрано Python, як найбільш гнучкий інструмент для роботи з веб-запитами, JSON-даними, а також інтеграції зі сторонніми AI-сервісами. Для розгортання REST-сервісу використано FastAPI, що дозволяє легко створювати HTTP-інтерфейс і працювати як із веб-клієнтом, так і з API-запитами [7].

Система працює за простим принципом: користувач вводить тендерний ID, після чого вся подальша обробка виконується автоматично. У відповідь повертається структурована текстова інформація, яка може бути використана як людиною, так і інтегрована у більші системи (наприклад, CRM чи внутрішні аналітичні інструменти).

Використання великих мовних моделей дозволяє досягнути гнучкого й надійного аналізу тексту без потреби у жорстко прописаних шаблонах. Такий підхід також легко адаптується до змін у форматах вхідних даних.

2.2 Модуль збору даних (web-scraping)

Для отримання фактичної тендерної інформації, яка в подальшому аналізується системою, необхідно виконати автоматизований збір даних з публічного вебджерела. У цьому проєкті для цієї мети використовується відкритий інтерфейс платформи Smarttender, який дозволяє отримати JSON-структуру по кожному конкретному тендеру за його ID.

Щоб спростити доступ до таких даних і забезпечити зручність у подальшій обробці, було реалізовано окремий модуль у вигляді класу TenderScraper.

2.2.1 Загальна логіка роботи модуля

Клас TenderScraper відповідає за повний цикл роботи з тендером:

- Отримання JSON-даних з API Smarttender за вказаним ID тендеру.
- Обробка вкладених структур JSON (учасники, документи, етапи).
- Збір та форматування ключових полів у зручну текстову або словникову форму.
- Завантаження файлів тендерної документації у локальну директорію.

Цей підхід дозволяє чітко розмежувати логіку збору даних і логіку подальшої обробки LLM-моделлю.

2.2.2 Вхідні параметри

Модуль очікує на вхід лише один обов’язковий параметр — ідентифікатор тендеру (tenderId), який користувач може скопіювати з вебінтерфейсу платформи. Приклад такої адреси:
https://smarttender.biz/PurchaseDetail/GetTenderModel/?tenderId=38246622

2.2.3 Внутрішня структура класу

Клас TenderScraper реалізує кілька основних методів:

- get_data(url) — виконує HTTP-запит і повертає JSON-дані.

- download_document(url, file_name, tender_id) — зберігає локально файл тендерної документації.
- process_tender(data, tender_id) — перевіряє наявність документації та викликає метод завантаження.
- get_scraped_data(...) — основна логіка збору текстових полів (бюджет, гарантія, опис, етапи, учасники, контакти).
- scraper(tender_id) — основна точка входу; об'єднує всі попередні кроки в єдину функцію.

Створення об’єкта класу та використання методу для витягування даних представлено нижче на рисунку.

```
from scraper import TenderScraper

# Ідентифікатор тендера (приклад взято з відкритого доступу)
tender_id = "38246622"

# Створення об'єкта для збору даних
scraper = TenderScraper()

# Отримання результату обробки
result = scraper.scraper(tender_id)
```

Рис 2.1 — Приклад використання класу TenderScraper

2.2.4 Формат отриманих даних

JSON-структура, що повертається платформою Smarttender, містить складні вкладені поля. Наприклад:

- Budget — містить бюджетну суму, валюту, ПДВ.
- Documents — вкладені документи з посиланнями на завантаження.
- PaymentTerms, QualificationCriteriaGroups — опис умов участі, оплати.

Важливо, що не всі поля присутні одночасно — тому модуль містить базову обробку винятків (try/except) і перетворення None у порожні рядки.

Нижче представлений приклад сирих даних витягнених, за допомогою API Smarttender


```
"Title": "Поточний ремонт місць загального користування в Миколаївському ліцеї
"Description": "",
"StatusTitle": "Прийом пропозицій",
"StatusColor": "#33CC99",
"Budget": {
    "Amount": 126564.51,
    "AmountTitle": "126 564.51",
    "WithVat": true,
    "VatTitle": "з ПДВ",
    "CurrencyTitle": "грн.",
    "CurrencyHtmlTitle": "грн.",
    "CurrencyId": 0
},
"Contracts": [],
"LotFeatures": [],
"MinimalStepPercent": "0.5",
"MinimalStepAmount": "632,82",
"ParticipationCost": "408,00 грн.",
"ParticipationCostAmount": 408.0,
"Cancelled": false,
"ShowCancelReason": false,
```


Рис 2.2 — Приклад структури одного JSON-виводу з Smarttender

2.2.5 Обробка вкладених документів

Окремо реалізовано механізм пошуку та збереження документів із вкладених полів. Наприклад, при знаходженні запису з назвою "Тендерна документація", програма завантажує відповідний PDF-файл у локальну директорію, яка має назву, що відповідає ID тендеру. Це дозволяє зберігати сирий текст документів для подальшої обробки великими мовними моделями.

2.2.6 Приклад згенерованого результату

Після завершення збору інформації метод get_scraped_data() повертає зручну структуру, яку далі аналізує модель. Структура містить текстову інформацію про організатора, бюджет, вимоги, учасників, умови оплати та інші корисні поля для учасників тендеру.

```
Назва тендера: Поточний ремонт місць загального користування в Миколаївському ліцеї №22 м. Миколаєва  (код ДК 021:2015 45450000-6 – Інші завершальні будівельн
Дати етапів подачі на тендер: Назва стейджу: tendering Дата початку: 10.06.25 16:57 Дата закінчення: 18.06.25 00:00
Назва стейджу: auction Дата початку: 18.06.25 12:02 Дата закінчення:
Назва стейджу: qualification Дата початку:  Дата закінчення:

Організатор: Управління освіти Миколаївської міської ради
Категорія: Інші завершальні будівельні роботи
Номер в Prozorro: UA-2025-06-10-012059-a
Учасники тендера:
Умови оплати:
Бюджет: 126 564.51грн. з ПДВ
Опис Тендеру:
Гарантія:
Номенклатура: Поточний ремонт місць загального користування в Миколаївському ліцеї №22 м. Миколаєва 1 послуга по 27.10.2025 Згідно з документацією, Україна
Контакти: Ім'я: Анастасія Мозолевська Номер телефону: +0512374169
Підстави для відмови в участі: ["Наявність не виконаних зобов'язань за раніше укладеним договором", 'Наявність впливу на прийняття рішення замовника', 'Вчинен
Інформація про інші технічні, якісні та кількісні характеристики та вимоги до учасника: ['Строк дії тендерної пропозиції', 'Мова (мови) тендерної пропозиції',
Кваліфікаційні критерії: ['Наявність досвіду виконання аналогічного договору']
Документи переможця: {}
Мінімальний крок аукціона: 632,82
```

Рис 2.3 — Зразок згенерованого результату

2.3 Модуль попередньої обробки тексту

Після збору даних з API Smarttender система переходить до обробки текстових документів, які містяться у вкладеннях тендеру. Як правило, це файли у форматах .doc, .docx або .pdf, що потребують конвертації та очищення перед подальшим аналізом.

На цьому етапі реалізовано автоматичну обробку документів — конвертацію .doc у .docx, а потім у .pdf, з подальшим зчитуванням тексту за допомогою бібліотеки fitz. Текст очищується від зайвих символів, об'єднується та готується до розбиття на менші частини.

Оскільки повний текст тендерної документації може бути надто великим для обробки за один раз, передбачено поділ на логічні блоки (чанки). Такий підхід дозволяє передавати у LLM тільки частини тексту — наприклад, сторінки або абзаци.

Модуль забезпечує універсальну підготовку тексту до наступного етапу — аналізу великою мовною моделлю. Він гнучко працює з різними форматами, адаптується до змісту документа і допомагає уникати втрати інформації.

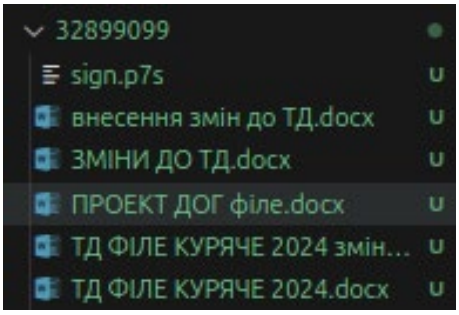
2.3.1 Конвертація та уніфікація форматів документів

Для подальшої обробки тендерної документації необхідно забезпечити єдиний формат усіх файлів. У Smarttender документи можуть мати різні розширення: .doc, .docx, .pdf. Щоб уникнути проблем із сумісністю та подальшим зчитуванням тексту, всі файли було приведено до одного формату — .pdf.

Цей процес реалізовано у вигляді функції convert_documents, яка проходиться по папці з тендерними файлами, знаходить документи у форматі .doc, конвертує їх у .docx за допомогою бібліотеки Spire.Doc, а потім — у .pdf через LibreOffice, запущений у headless-режимі (тобто без графічного інтерфейсу). Після успішної конвертації оригінальні файли видаляються, щоб уникнути дублювання [9].

Такий підхід дозволив автоматизувати підготовку документів до обробки, прибрати зайві ручні кроки та зменшити ризик виникнення помилок, пов’язаних із нестандартними форматами. Крім того, працювати з єдиним форматом набагато простіше на наступних етапах — наприклад, при витягу тексту або розмітці даних.

Нижче представлено скріншот результату виконання функції — видно, що всі документи в каталозі тендеру конвертовано до формату PDF:



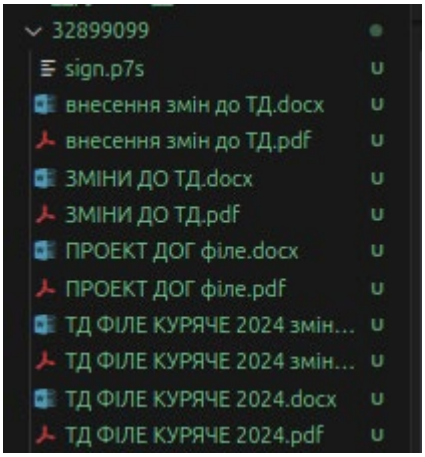


Рис 2.4 - 2.5 — папка з файлами тендеру до та після конвертації

Цей крок виконується автоматично при запуску аналізу для кожного нового тендера, ID якого вводить користувач.

2.3.2 Витяг і попереднє очищення тексту

Після конвертації документів у формат .pdf, наступним кроком є зчитування текстового вмісту. Для цього використано бібліотеку fitz (PyMuPDF), яка дозволяє працювати зі сторінками PDF-файлу окремо і витягувати з них текст [8].

В рамках реалізації, кожен документ відкривається покроково, сторінка за сторінкою. Текст, отриманий із кожної сторінки, очищується від службових символів, зайвих перенесень рядків, пробілів та непотрібних розривів. Також додатково відфільтровуються порожні рядки, залишки номерів сторінок та інші неінформативні фрагменти.

Особливу увагу приділено стабільності — якщо сторінка не містить тексту або документ зчитується некоректно, це фіксується у повідомленні про помилки. Такий підхід дозволяє не зупиняти всю обробку через один пошкоджений файл.

Після цього очищений текст об’єднується в один великий блок, який передається на поділ у наступному підпункті.

2.3.3 Розбиття тексту на фрагменти для аналізу

Після зчитування та очищення тексту з тендерних документів виникає потреба в його поділі на менші частини. Це пов’язано з обмеженнями на обсяг тексту, який може приймати LLM-модель за один раз, а також для кращої структурованості обробки.

У реалізованій системі для розбиття використовується компонент RecursiveCharacterTextSplitter з бібліотеки langchain. Він дозволяє розділити текст за реченнями або абзацами, з урахуванням пунктуації, і при цьому контролювати розмір кожного фрагмента (чанку). У моєму випадку використовувався розмір ~250 символів без перекриття (chunk_overlap=0) [6].

Також на практиці я пробував кілька підходів до розбиття:

- поділ тексту по сторінках PDF-документа;
- поділ по абзацах;
- поділ із застосуванням регулярних виразів;
- відбір найбільш релевантних фрагментів (за embedding-запитом).

Вибір підходу залежить від мети. Якщо потрібно просто передати модельному аналізу основний зміст — підійде абзацне або реченне розбиття. Проте в моєму випадку, переді мною стоїть завдання витягти відповідь на конкретне питання, можна шукати фрагменти, близькі до змісту запиту.

Нижче вставлено приклад сформованих чанків для конкретного тендера:

```python
from data_preprocess import read_documents
# ID тендера
tender_id = "32899099"

# Запускаємо функцію зчитування та обробки документів
chunks, error_msg = read_documents(tender_id)

# Вивід результатів
print("Отримано фрагментів тексту:", len(chunks))
for i, chunk in enumerate(chunks[:5]):  # виведемо перші 5
    print(f"\n--- Фрагмент {i+1} ---")
    print(chunk)

if error_msg.strip() != "Системі не вдалось прочитати наступні документи: ":
    print("\n⚠ Повідомлення про помилки:")
    print(error_msg)
```

✓ 3.4s

```
Отримано фрагментів тексту: 1143

--- Фрагмент 1 ---
Внесення змін до тендерної документації по предмету закупівлі: філе куряче

--- Фрагмент 2 ---
  Постачальник зобов'язується надати Замовнику товар: філе куряче заморожене, г
```

Рис 2.6 — приклад фрагментів розбитого тексту

2.3.4 Вибір релевантних фрагментів за допомогою embedding-порівняння

Після того, як усі фрагменти документа були згенеровані, постає завдання — визначити, які саме з них найбільше відповідають конкретному запиту. У моїй роботі цим запитом є, наприклад: «Які документи необхідні для участі в тендері?»

Щоб знайти відповідні частини тексту, використовується векторне представлення (embedding) як запиту, так і кожного фрагмента документа. Для цього застосовано бібліотеку OpenAIEmbeddings, яка дозволяє представити текст у вигляді вектора в багатовимірному просторі [5].

Вектори запиту та чанків порівнюються за допомогою косинусної подібності. Найбільш схожі фрагменти відбираються за найвищим значенням цієї метрики — зазвичай до 25 найрелевантніших.

Фрагменти, які потрапили до вибірки, потім передаються у вхід LLM-моделі. Це дозволяє зосередити увагу моделі лише на важливому контексті, зменшуючи навантаження і підвищуючи точність відповіді.

2.4 Аналіз тексту за допомогою великої мовної моделі

Фінальний етап обробки документації полягає у використанні великої мовної моделі (LLM) для аналізу відібраних текстових фрагментів. Модель повертає відповідь, сформовану на основі релевантного контексту.

У системі тестувались різні моделі, зокрема GPT-4 та DeepSeek R1. Результати показали, що навіть при неповному охопленні тексту модель здатна формувати осмислені та точні відповіді, що дозволяє автоматизувати рутинну аналітику тендерних документів [10].

2.4.1 Побудова запиту до мовної моделі

Для отримання коректної та структурованої відповіді від LLM-моделі важливо грамотно сформулювати запит (prompt). У рамках даної системи було розроблено спеціальний інструктивний промпт, який пояснює моделі її завдання — витягти перелік документів для участі у тендері на основі лише поданого фрагмента тексту.

Запит містить чіткі інструкції щодо мови, форми відповіді, заборони на узагальнення та необхідності обґрунтування кожного пункту. Також надається приклад оформлення списку, щоб уніфікувати результати.

Промпт є системним, тобто використовується в кожному виклику моделі, де {text} динамічно замінюється на конкретний фрагмент тендерної документації, отриманий після препроцесингу.

Промпт було сформульовано українською мовою, щоб уникнути втрати змісту при перекладі та забезпечити точність витягу формулювань із джерела.

Повний текст промпту, що використовується для взаємодії з LLM-моделлю, наведено у Додатку А.

2.4.2 Вибір моделі та формат відповіді

У реалізованій системі для аналізу тендерної документації використовуються великі мовні моделі. Основна модель — GPT-4o від OpenAI, яка має хорошу точність і стабільність при роботі з українською мовою. Також для порівняння результатів тестувалися різні моделі GPT та модель DeepSeek R1.

Модель отримує текстовий запит разом із фрагментами документа і повертає відповідь у вигляді структурованого списку. У випадку витягу документів це — нумерований перелік із короткими поясненнями, які саме документи потрібні.

Результат зберігається як звичайний текст і може бути показаний у веб-інтерфейсі або використаний далі в автоматизації.

2.4.3 Вибір моделі

Для аналізу відібраних фрагментів тендерної документації в системі використовується велика мовна модель GPT-4o від OpenAI. Вибір саме цієї моделі

був обґрунтований рядом факторів: високою якістю відповідей, підтримкою української мови, стабільною роботою з довгими запитами та адекватною вартістю обробки.

Під час розробки проводилось тестування кількох варіантів моделей. Однією з альтернатив, яку я перевірив, була DeepSeek R1 — нова open-source LLM з гарними результатами на загальних benchmark'ах. Проте в контексті української тендерної документації вона показала дещо гіршу точність. У відповідях траплялись неточності, пропуски частини вимог або небажані узагальнення. Це, ймовірно, пов’язано з тим, що ця модель менш натренована на україномовних юридичних чи бюрократичних текстах.

Також перевірялись моделі GPT-3.5-turbo та GPT-4 (classic). GPT-3.5 виявилась недостатньо чутливою до контексту і часто формулювала відповіді нечітко або занадто узагальнено. GPT-4 classic показала гарні результати, але при роботі з великими обсягами тексту (більше 10–15 тисяч токенів на запит) її використання ставало економічно недоцільним для реального застосування.

OpenAI Assistant API (через LangChain chain'и) також був протестований. Він дозволяє зручно обробляти довгі тексти через багатостадійну генерацію, але у поточному варіанті взаємодії асистенту передавався лише один цілий файл або один блок, що призводило до втрати повного контексту. Через це якість відповіді була нестабільною і неповною — модель не «бачила» всі вимоги одночасно. Зважаючи на це, я відмовився від такого підходу на користь прямої інтеграції з LLM [5].

У цьому контексті GPT-4o забезпечує оптимальний баланс між вартістю й якістю:

- Точно працює з українською мовою та юридичною термінологією;
- Дає структуровані, конкретні відповіді з мінімальними пропусками;
- Дозволяє працювати з великим обсягом контексту (до 128k токенів), що важливо при аналізі великих документів;

- Має порівняно низьку ціну за тисячу токенів, що робить її практично придатною для масштабованої системи.

Формат відповіді, який генерує модель, — це звичайний нумерований список документів з короткими поясненнями, сформований на основі конкретного фрагменту тендерної документації. Жодна додаткова постобробка над текстом не виконується — відповідь зберігається або одразу показується користувачеві у тому вигляді, в якому її надала модель.

Приклад повної відповіді GPT-4o наведено у Додатку Б.

2.5 Створення веб-інтерфейсу та тестування

Щоб готовий алгоритм можна було використовувати у реальних робочих процесах, його обгорнуто у невеликий веб-сервіс FastAPI. Така обгортка дає змогу звертатися до всієї системи одним HTTP-запитом: користувач передає лише tenderId, а у відповідь отримує сформований перелік документів та коротку довідку про тендер. Крім зручності інтеграції, FastAPI автоматично генерує Swagger-документацію, що полегшує ручне тестування й демонстрацію роботи сервісу [7].

Разом із самим ендпойнтом додано прості тести на pytest, які перевіряють, що сервіс коректно відповідає та повертає потрібні поля. Таким чином, інтерфейс забезпечує зв’язку між алгоритмом аналізу й практичним використанням у будь-якій зовнішній системі.

2.5.1 Створення REST-ендпойнту

Уся система взаємодіє з користувачем або іншими сервісами через один запит у форматі REST. Достатньо звернутися до адреси /analyze, підставивши значення tenderId ідентифікатор тендера зі Smarttender. У відповідь сервер повертає структурований результат: загальну інформацію про тендер і нумерований список документів, які необхідні для участі.

Цей підхід спрощує використання та дозволяє легко інтегрувати рішення в інші системи. У випадку помилок (наприклад, якщо ID не існує або сторінка не відкривається), сервер повертає відповідне повідомлення з кодом 500.

Також сервіс містить endpoint для перевірки статусу активності веб-сервісу.

Нижче представлено, як виглядає веб-інтерфейс системи

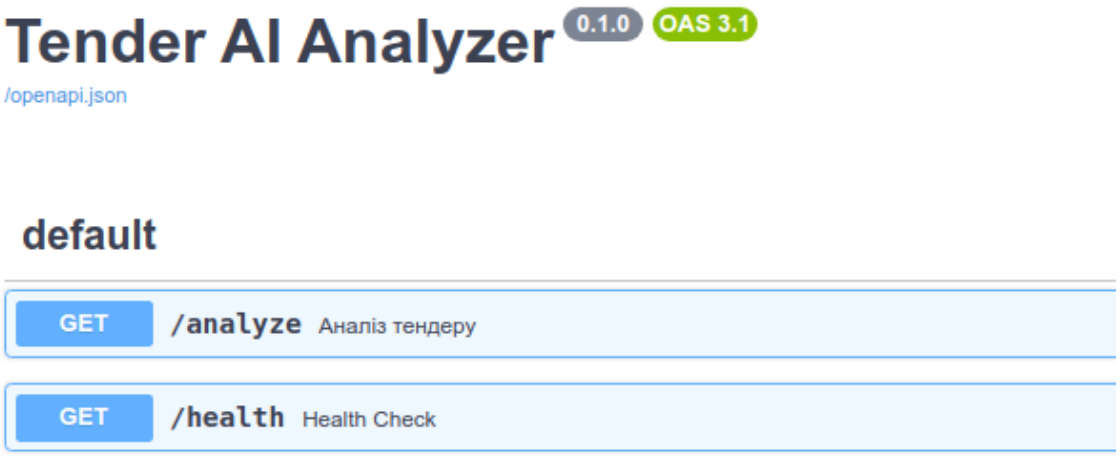


Рис 2.7 — скріншот інтерфейсу Swagger

2.5.2 Приклад використання

Щоб перевірити роботу сервісу, достатньо звернутися до нього за допомогою браузера або будь-якого інструменту для HTTP-запитів (наприклад, curl, Postman, або через Swagger-інтерфейс).

Приклад простого запиту з терміналу:

```
curl http://localhost:8000/analyze/32899099
```

Лістинг 2.1 – Фрагмент запиту в терміналі Linux

У відповідь повертається JSON, що містить:

- загальні дані про тендер (бюджет, строки, організатор тощо);
- нумерований список документів, які модель вважає обов’язковими для участі.

Приклад повної відповіді системи знаходиться у Додатку В

2.5.3 Розгортання сервісу у Docker

Щоб запустити систему в будь-якому середовищі без вручну встановлених залежностей, її зібрано у контейнер Docker. Це спрощує розгортання: не потрібно окремо встановлювати Python, бібліотеки чи сторонні утиліти — усе вже входить до складу контейнера.

Docker створює ізольоване середовище, в якому програма працює стабільно незалежно від операційної системи чи конфігурації комп’ютера. Завдяки цьому систему легко запускати як на локальній машині, так і на сервері.

У фрагменті нижче наведено приклад збірки Docker-контейнера для запуску програми:

```
docker build -t tender-ai .

docker run -d -p 8000:8000 \
  -e OPENAI_API_KEY=<ключ_OpenAI> \
  --name tender-ai-service tender-ai
```

Лістинг 2.2 — збірка образу та запуск контейнера

Ключ OPENAI_API_KEY передається як змінна середовища.

Після запуску сервіс доступний за адресою http://localhost:8000, а перевірити працездатність можна запитом до /health або /docs.

Такий підхід гарантує, що програма працюватиме однаково на локальному комп’ютері, тестовому сервері чи в хмарі, без складного налаштування оточення, та дозволяє максимально зручне розгортання на віддаленому сервері.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Вплив діяльності людини на довкілля.

Від самого початку своєї історії людина впливала на природу. Спочатку це був незначний вплив — наприклад, використання вогню чи вирубування лісу для землеробства. Але з розвитком техніки та промисловості цей вплив значно посилився. Сьогодні ми бачимо, як виробнича, транспортна, побутова й інша діяльність людей поступово змінює стан навколишнього середовища — і, на жаль, не в кращий бік.

У навчальній літературі з безпеки життєдіяльності підкреслюється, що надмірне втручання людини в природу призводить до таких негативних наслідків, як забруднення повітря і води, зникнення рослин і тварин, виснаження ґрунтів, накопичення відходів. Промислові підприємства викидають у повітря шкідливі речовини — оксиди вуглецю, сірки, азоту, важкі метали. Стоки з заводів та аграрних господарств потрапляють у річки та озера, отруюючи все живе.

Крім того, нераціональне використання природних ресурсів спричиняє руйнування екосистем. Люди вирубують ліси, висушують болота, забудовують зелені зони. Усе це знижує здатність природи до самовідновлення. Відбувається порушення балансу в біосфері, а це впливає і на нас самих — через зміни клімату, частіші стихійні лиха, проблеми зі здоров’ям.

Закон України «Про охорону навколишнього природного середовища» визначає захист довкілля як одну з головних функцій держави. Згідно з ним, кожен громадянин має право на безпечне для життя і здоров’я довкілля. Водночас, кожен зобов’язаний дбати про його збереження. Це підтверджується й іншими нормативними актами, зокрема положеннями про екологічну експертизу, державний моніторинг, екологічну освіту та право на доступ до екологічної інформації [16].

У стандартах управління навколишнім середовищем, зокрема ДСТУ ISO 14001, йдеться про те, що підприємства повинні брати відповідальність за наслідки своєї діяльності, зменшувати обсяги викидів, економно використовувати енергію та ресурси, впроваджувати безпечні технології [17].

Сучасна екологічна політика не повинна зводитися лише до боротьби з наслідками. Потрібно думати на перспективу — планувати так, щоб не шкодити природі сьогодні й не створювати загрозу для майбутніх поколінь. Навчальні курси з безпеки життєдіяльності підкреслюють важливість формування в суспільстві екологічної культури. Це означає не тільки знання, а й відповідальне ставлення до природи у щоденних діях — на роботі, вдома, у побуті.

Лише за умов усвідомлення людиною власної ролі в збереженні довкілля можливий гармонійний розвиток — коли економіка, суспільство і природа існують у рівновазі. Необхідно не просто зменшувати шкоду, а переосмислити сам підхід до споживання і виробництва. І почати варто з малого — із себе.

3.2 Заходи захисту обладнання від статичної електрики

Захист від статичної електрики — важливий компонент системи електробезпеки, особливо у виробничих умовах, де накопичення електростатичних зарядів може призвести до ураження персоналу електричним струмом, займання горючих матеріалів або виходу з ладу чутливої електроніки.

Одним із ключових методів запобігання таким загрозам є захисне заземлення — цілеспрямоване з’єднання неструмоведучих металевих частин електроустановок із землею. У разі пробою ізоляції струм миттєво відводиться в землю, що не дозволяє з’явитися небезпечній напрузі на корпусі обладнання. Для цього використовують як природні, так і штучні заземлювачі — труби, металеві кутики, смуги з відповідним поперечним перерізом.

Крім того, ефективним заходом є використання подвійної або посиленої ізоляції струмоведучих частин. Це дозволяє уникнути ураження струмом навіть при пошкодженні основної ізоляції.

Іонізація повітря застосовується у приміщеннях з чутливою технікою — іонізатори нейтралізують накопичений електростатичний заряд на поверхнях, запобігаючи розрядам.

Використання малої напруги — ще один метод, що знижує ризик ураження. У виробничих приміщеннях з підвищеною небезпекою рекомендується використовувати напругу 12 або 36 В для живлення ручного електроінструменту чи освітлення, що значно зменшує можливість утворення небезпечного струму через тіло людини.

Також застосовуються організаційні заходи, такі як встановлення сітчастих або суцільних огороджень струмоведучих частин, використання блокувань, захисного відключення (ПЗВ), навчання персоналу, маркування захисного заземлення тощо.

Комплекс цих технічних і організаційних заходів забезпечує ефективний захист працівників та обладнання від шкідливого впливу статичної електрики у промислових умовах.

Згідно з ДСН 3.3.6.042–99 «Державні санітарні норми і правила. Захист від електростатичних зарядів», основна мета таких заходів — попередити накопичення електричних зарядів, забезпечити їх безпечне відведення, а також зменшити ймовірність неконтрольованих розрядів [15].

Основні заходи безпеки:

- Використання антистатичних матеріалів — це покриття підлоги, одяг, взуття та інструменти, що не накопичують заряд або мають антистатичні властивості. Особливо це актуально для персоналу, що працює у вибухонебезпечних зонах [14].

- Іонізація повітря — встановлення іонізаторів, які створюють заряджені частинки, що нейтралізують електростатичні поля. Це важливо в умовах виробництва мікроелектроніки або полімерних матеріалів.
- Підтримання оптимальної вологості повітря — при відносній вологості нижче 40% значно зростає здатність поверхонь накопичувати статичний заряд. Оптимальний діапазон — 45–60% [12].
- Використання антистатичних пристроїв та браслетів — персонал, який працює з електронікою, повинен бути підключений до захисних антистатичних систем через спеціальні браслети або килимки.
- Регулярна перевірка стану заземлення та електростатичних властивостей матеріалів — за допомогою омметрів, тестерів зарядів або вимірювачів опору поверхні [17].
- Навчання персоналу — працівники повинні бути поінформовані про джерела утворення електростатичних зарядів і правильне поводження з антистатичними засобами.
- Організаційні заходи — у вибухонебезпечних зонах суворо регламентуються дії працівників, вимоги до одягу, обладнання та матеріалів [15].

Забезпечення цих заходів є обов’язковим не лише з погляду безпеки працівників, а й для збереження обладнання та запобігання виробничим втратам. У багатьох випадках навіть прості технічні рішення дозволяють уникнути серйозних наслідків.

ВИСНОВКИ

Сьогодні обробка інформації стає все більш актуальною в умовах зростаючих обсягів даних. Особливо це стосується сфер, де великі об’єми текстових документів потрібно аналізувати швидко і точно — наприклад, у публічних закупівлях. Саме тому виникла ідея створити систему, яка допомагає автоматично аналізувати тендерну документацію та визначати перелік документів, які потрібні для участі.

У процесі виконання дипломної роботи було розроблено програмне рішення, яке об’єднує технології веб-скрапінгу, обробки PDF-документів, попередньої підготовки тексту та аналізу за допомогою великої мовної моделі GPT-4o. Важливо було не лише добитись правильної роботи системи, але й протестувати різні підходи та моделі, щоб знайти найкращий варіант — зокрема, було порівняно GPT-3.5, GPT-4, DeepSeek R1.

Система вийшла зручною: достатньо ввести ID тендеру — і вже за декілька секунд користувач отримує структуровану відповідь з основними параметрами закупівлі та списком документів. Це рішення можна розгорнути через Docker, що значно спрощує розгортання у різних середовищах.

Під час роботи над проектом вдалося поглибити знання з Python, REST-сервісів, роботи з API, PDF-документами та LLM-моделями. Також з’явилось краще розуміння того, як подібні системи можуть впливати на реальні процеси в бізнесі.

Робота стала результатом навчання у Тернопільському національному технічному університеті імені Івана Пулюя, де я отримав не лише теоретичну базу, але й практичні навички, необхідні для створення повноцінного програмного продукту. Надалі систему можна вдосконалювати, інтегрувати з CRM або іншими бізнес-рішеннями, або ж адаптувати під інші сфери, де аналіз текстових документів займає багато часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chollet F. Deep Learning with Python. – 2nd ed. – Manning Publications, 2021. – 504 p.
2. Udofia E. Webscrapping: BeautifulSoup or Selenium? // Medium. – 2023. – URL: https://medium.com/@udofiaetietop/webscrapping-beautifulsoup-or-selenium-3467edb3c0d9
3. Jalammar J. The Illustrated Transformer – URL: https://jalammar.github.io/illustrated-transformer/
4. Smarttender.biz – URL: https://smarttender.biz/
5. OpenAI API Documentation – URL: https://platform.openai.com/docs/
6. LangChain Documentation – URL: https://docs.langchain.com/
7. FastAPI Documentation – URL: https://fastapi.tiangolo.com/
8. PyMuPDF (fitz) Documentation – URL: https://pymupdf.readthedocs.io/
9. Spire.Doc for Python – URL: https://github.com/eiceblue/Spire.Doc-for-Python
10. Eden AI. GPT-4o vs DeepSeek R1: Which is the best open-weight model? // Eden AI Blog. – 2024. – URL: https://www.edenai.co/post/gpt-4o-vs-deepseek-r1
11. Requests: HTTP for Humans – URL: https://docs.python-requests.org/
12. Березюк О. В., Лемешев М. С. Безпека життєдіяльності: навчальний посібник. Вінниця: ВНТУ, 2011. 204 с.
13. Скобло Ю. С., Соколовська Т. Б., Мазоренко Д. І., Тіщенко Л. М., Троянов М. М. Безпека життєдіяльності: навчальний посібник. Київ: Кондор, 2003. 424 с. ISBN 966-7982-10-6
14. Грибан В. Г., Негодченко О. В. Охорона праці: навчальний посібник. Київ: Центр учбової літератури, 2009. 280 с. ISBN 978-966-364-832-3

15. ДСН 3.3.6.042-99. Державні санітарні норми і правила «Захист від електростатичних зарядів». Затверджено МОЗ України 01.12.1999. URL: https://zakon.rada.gov.ua/rada/show/v0042282-99

16. Закон України «Про охорону навколишнього природного середовища» від 25.06.1991 № 1264-XII. URL: https://zakon.rada.gov.ua/laws/show/1264-12

17. ДСТУ ISO 14001:2015. Системи екологічного управління. Вимоги та настанови щодо застосування. URL: https://online.budstandart.com/ua/catalog/doc-page?id_doc=65284

18. ТЕМА 17. Електробезпека. Методи захисту в електроустановках // Безпека життєдіяльності та основи охорони праці. ТНТУ ім. І. Пулюя. URL: https://dl.tntu.edu.ua/content.php?cid=289205

19. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

Додаток А

Промпт для великої мовної моделі (LLM)

Ваше завдання — ретельно проаналізувати фрагмент тендерної документації, поданий нижче, і на основі лише цього тексту скласти вичерпний перелік документів, які учасник повинен підготувати для участі у тендері.

Важливі вимоги до виконання завдання:
1. Використовуйте **виключно наданий фрагмент тексту**. Не залучайте зовнішні джерела або попередні знання.
2. Текст для аналізу написаний **українською мовою**, відповідно й відповідь також повинна бути складена **українською**.
3. Інформація про документи може бути:
   - представлена **у вигляді текстового опису**;
   - сформульована як **вимога до заповнення певного поля постачальником**.
4. Уникайте узагальнень. **НЕ групуйте документи під загальними назвами** (наприклад, “повний пакет документів” або “всі підтверджуючі документи”).
   Замість цього, **розпишіть кожен документ окремим пунктом**, конкретизуючи його зміст або вимоги щодо нього.
5. За можливості, використовуйте **цитати або фрагменти** з наданого тексту як підставу для кожного пункту.
6. Подавайте перелік документів **у вигляді нумерованого списку**, з коротким описом кожного документа (формулювання може бути адаптовано, але зміст має зберігати суть вимоги).

Приклад формату відповіді:
1. Копія документа, що підтверджує повноваження особи на підписання тендерної пропозиції.
2. Довідка в довільній формі про відсутність підстав для відмови в участі у процедурі закупівлі відповідно до статті 17 Закону.
3. У разі надання товарів, що підлягають обов'язковій сертифікації, — копія сертифіката відповідності або декларації про відповідність.
4. Лист-гарантія, що підтверджує готовність виконати умови договору.

Ось текст для аналізу:
{text}

Де {text} — всі найбільш релевантні чанки з документу

Додаток Б

Приклад відповіді GPT-4o після аналізу тендерної документації

1. **Документально підтверджений досвід виконання аналогічних договорів**:
   - Скан-копія з оригіналу або з належним чином засвідченої копії аналогічного договору.
   - Скан-копія з оригіналу або з належним чином засвідченої копії акту прийому-передачі робіт (послуг) або акту наданих послуг, який засвідчує виконання робіт/надання послуг в повному обсязі відповідно до умов договору.
2. **Документи, що підтверджують повноваження представника учасника**:
   - Для фізичних осіб-підприємців: нотаріально посвідчена довіреність, паспорт громадянина України довіреної особи, витяг або виписка з Єдиного державного реєстру юридичних осіб, фізичних осіб-підприємців та громадських формувань, паспорт громадянина України фізичної особи-підприємця, який видав довіреність.
   - Для фізичних осіб: нотаріально посвідчена довіреність, паспорт громадянина України довіреної особи, паспорт громадянина України фізичної особи, яка видала довіреність.
   - Для юридичних осіб: документи, що підтверджують повноваження посадової особи або представника на підписання документів тендерної пропозиції.

3. **Кошторисна документація**, підписана уповноваженою особою учасника з проставлянням печатки.

4. **Копія свідоцтва про реєстрацію платника податку** або копія витягу з реєстру платників податку у відповідності до порядку оподаткування (платника ПДВ, єдиного податку тощо).

5. **Інформація, що підтверджує відсутність підстав для відмови в участі у процедурі закупівлі** згідно з вимогами п. 47 Особливостей.

6. **Проєкт договору** із зазначенням порядку змін його умов.

7. **Сканований оригінал Дозволу/Декларації про відповідність матеріально-технічної бази** вимогам законодавства з питань охорони праці в частині спроможності виконання робіт на висоті понад 1.3 метра.

8. **Інформаційна довідка з Єдиного державного реєстру осіб**, які вчинили корупційні або пов’язані з корупцією правопорушення, про відсутність інформації про корупційні або пов'язані з корупцією правопорушення, що видана на керівника учасника процедури закупівлі або фізичну особу, яка є учасником.

9. **Документи, складені іноземною мовою**, повинні мати автентичний переклад українською мовою, засвідчений нотаріально.

10. **Документи, що стосуються юридичної особи – нерезидента**, мають супроводжуватися перекладом на українську мову, засвідченим нотаріально, та легалізовані в установленому порядку, якщо інше не передбачено міжнародними договорами.

Додаток В

Приклад повної відповіді програми

Назва тендера: Поточний ремонт місць загального користування в Миколаївському ліцеї №22 м. Миколаєва (код ДК 021:2015 45450000-6 – Інші завершальні будівельні роботи)

Дати етапів подачі на тендер: Назва стейджу: tendering Дата початку: 10.06.25 16:57 Дата закінчення: 18.06.25 00:00

Назва стейджу: auction Дата початку: Дата закінчення:

Назва стейджу: qualification Дата початку: Дата закінчення:

Організатор: Управління освіти Миколаївської міської ради

Категорія: Інші завершальні будівельні роботи

Номер в Prozorro: UA-2025-06-10-012059-a

Учасники тендера:

Умови оплати:

Бюджет: 126 564.51грн. з ПДВ

Опис Тендеру:

Гарантія:

Номенклатура: Поточний ремонт місць загального користування в Миколаївському ліцеї №22 м. Миколаєва 1 послуга по 27.10.2025 Згідно з документацією, Україна

Контакти: Ім'я: Анастасія Мозолевська Номер телефону: +0512374169

Підстави для відмови в участі: Наявність не виконаних зобов'язань за раніше укладеним договором, Наявність впливу на прийняття рішення замовника, Вчинення корупційних правопорушень, Вчинення економічних правопорушень, Вчинення кримінальних правопорушень, Наявність зв'язку учасника з іншими учасниками процедури або замовником, Порушення справи про банкрутство, Наявність антикорупційної програми, Вчинення правопорушень, пов’язаних із застосуванням санкцій, Вчинення правопорушень, пов'язаних з використанням

дитячої праці чи будь-якими формами торгівлі людьми, Наявність інших підстав для відмови в участі у процедурі закупівлі,
Інформація про інші технічні, якісні та кількісні характеристики та вимоги до учасника: Строк дії тендерної пропозиції, Мова (мови) тендерної пропозиції, Інформування про субпідрядників/співвиконавців,
Кваліфікаційні критерії: Наявність досвіду виконання аналогічного договору,
Мінімальний крок аукціона: 632,82

Системі не вдалось прочитати наступні документи:

Необхідні документи для участі в тендері:
На основі наданого фрагменту тендерної документації, учасник повинен підготувати наступні документи для участі у тендері:

1. **Документально підтверджений досвід виконання аналогічних договорів**:
   - Скан-копія з оригіналу або з належним чином засвідченої копії аналогічного договору.
   - Скан-копія з оригіналу або з належним чином засвідченої копії акту прийому-передачі робіт (послуг) або акту наданих послуг, який засвідчує виконання робіт/надання послуг в повному обсязі відповідно до умов договору.

2. **Документи, що підтверджують повноваження представника учасника**:
- Для фізичних осіб-підприємців: нотаріально посвідчена довіреність, паспорт громадянина України довіреної особи, витяг або виписка з Єдиного державного реєстру юридичних осіб, фізичних осіб-підприємців та громадських формувань, паспорт громадянина України фізичної особи-підприємця, який видав довіреність.

- Для фізичних осіб: нотаріально посвідчена довіреність, паспорт громадянина України довіреної особи, паспорт громадянина України фізичної особи, яка видала довіреність.
- Для юридичних осіб: документи, що підтверджують повноваження посадової особи або представника на підписання документів тендерної пропозиції.

3. **Кошторисна документація**, підписана уповноваженою особою учасника з проставлянням печатки.

4. **Копія свідоцтва про реєстрацію платника податку** або копія витягу з реєстру платників податку у відповідності до порядку оподаткування (платника ПДВ, єдиного податку тощо).

5. **Інформація, що підтверджує відсутність підстав для відмови в участі у процедурі закупівлі** згідно з вимогами п. 47 Особливостей.

6. **Проєкт договору** із зазначенням порядку змін його умов.

7. **Сканований оригінал Дозволу/Декларації про відповідність матеріально-технічної бази** вимогам законодавства з питань охорони праці в частині спроможності виконання робіт на висоті понад 1.3 метра.

8. **Інформаційна довідка з Єдиного державного реєстру осіб**, які вчинили корупційні або пов’язані з корупцією правопорушення, про відсутність інформації про корупційні або пов'язані з корупцією правопорушення, що видана на керівника учасника процедури закупівлі або фізичну особу, яка є учасником.

9. **Документи, складені іноземною мовою**, повинні мати автентичний переклад українською мовою, засвідчений нотаріально.

10. **Документи, що стосуються юридичної особи – нерезидента**, мають супроводжуватися перекладом на українську мову, засвідченим нотаріально, та легалізовані в установленому порядку, якщо інше не передбачено міжнародними договорами.

Додаток Г

Основний код програми

DocumentAnalyzer.py

```
import os
import time
import fitz
import openai
import numpy as np

import data_preprocess as data_preprocess
from constants import OPENAI_KEY, initial_prompt, embed_queries
from scraper import TenderScraper

from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate

class TenderAnalyzer:
    def __init__(self, url):
        self.url = url
        openai.api_key = OPENAI_KEY
        os.environ["OPENAI_API_KEY"] = OPENAI_KEY

    def output_formatting(scraped_data:dict, result:str,
error_doc:str):
        output = ''
        for key, value in scraped_data.items():
            if key != 'Документи переможця':
                if isinstance(value, list):
                    output += f'\n{key}: {"".join(f"{str(val)}, "
for val in value)}'
                else:
                    output += f'\n{key}: {value}'
        output += '\n\n' + error_doc + '\n\nНеобхідні документи для
участі в тендері:\n' + result
        return output

    def analyze_tender(self):
        start = time.time()
        print("Starting tender analysis...")
        combined_text, scraped_data, error_doc =
TenderDataScraper.scrape(self.url)
        print("1")
        embedded_chunks, embedded_queries, chunks =
TenderDataScraper.embed_chunks(combined_text)
```

```
        print("111")
        relevant_chunks =
TenderDataScraper.get_relevant_chunks(embedded_chunks,
embedded_queries, chunks, query_key="Документація", top_k=128)
        print("1111")
        answer = DocumentAnalyzer.refine_method(relevant_chunks,
input_token_limit=12000)
        print("Total time to process data and get response: ",
time.time() - start)
        return TenderAnalyzer.output_formatting(scraped_data,
answer, error_doc)

class TenderDataScraper:
    def scrape(url:str):
        start = time.time()
        scraper_instance = TenderScraper()
        tender_id = url.split('/')[-1] if url.split('/')[-1].strip()
else url.split('/')[-2]
        scraped_data = scraper_instance.scraper(tender_id)
        try:
            combined_text, error_doc =
data_preprocess.extract_text_from_documents(tender_id)
            print("Total time to scrape a web page, download and
read document: ", time.time() - start)
            return combined_text, scraped_data, error_doc
        except FileNotFoundError:
            print("Total time to scrape a web page, download and
read document: ", time.time() - start)
            return 1, scraped_data, "Tender analyzation failed."

    def split_embed_langchain(combined_text:str):
        text_splitter = RecursiveCharacterTextSplitter(
            separators=[r'(?<=[.!?()])\s+(?=[A-ZA-Я])'],
            chunk_size=250,
            chunk_overlap=0,
            is_separator_regex=True
        )
        texts = text_splitter.create_documents([combined_text])
        docs = text_splitter.split_documents(texts)
        return docs

    def embed_chunks(chunks:list):
        query_documentation, query_requirements,
query_disqualification = [value for value in embed_queries.values()]
        start = time.time()
        chunks_query = chunks.copy()

        print("Number of chunks after removing duplicates: ",
len(chunks))
        chunks_query += [query_documentation] + [query_requirements]
+ [query_disqualification]
```

```
        openaiEmbeddings = OpenAIEmbeddings()
        embedded_chunks_query =
np.array(openaiEmbeddings.embed_documents(chunks_query))
        embedded_chunks = embedded_chunks_query[:-3, :]
        embedded_query_documentation = embedded_chunks_query[-3]
        embedded_query_requirements = embedded_chunks_query[-2]
        embedded_query_disqualification = embedded_chunks_query[-1]
        embedded_queries = {
            "Документація": embedded_query_documentation,
            "Вимоги для участі": embedded_query_requirements,
            "Причини дискваліфікації":
embedded_query_disqualification
        }
        print("Time to split and embed chunks: ", time.time() -
start)
        return embedded_chunks, embedded_queries, chunks

    def get_relevant_chunks(embedded_chunks, embedded_queries,
chunks, query_key="Документація", top_k=25):
        query_embedding = embedded_queries[query_key]
        query_norm = query_embedding /
np.linalg.norm(query_embedding)
        chunk_norms = embedded_chunks /
np.linalg.norm(embedded_chunks, axis=1)[:, np.newaxis]
        cosine_similarities = np.dot(chunk_norms,
query_norm.T).flatten()
        sorted_indices = np.argsort(cosine_similarities)[-
top_k:][::-1]
        selected_chunks = []
        for idx in sorted_indices:
            selected_chunks.append(chunks[idx])
        return selected_chunks

class DocumentAnalyzer:
    @staticmethod
    def refine_method(selected_chunks: list, input_token_limit: int
= 12000) -> str:
        start = time.time()
        PROMPT = PromptTemplate(
            template=initial_prompt,
            input_variables=["text"]
        )
        joined_text, total_len = "", 0
        for chunk in selected_chunks:
            if total_len + len(chunk) > input_token_limit:
                break
            joined_text += chunk + "\n"
            total_len += len(chunk)
        final_prompt = PROMPT.format(text=joined_text)
        llm = ChatOpenAI(model_name="gpt-4o", temperature=0.2)
        result_text = llm.invoke(final_prompt).content
```

```
        print("GPT-4o latency (s):", time.time() - start)
        return result_text
```

Scraper.py

```
import os
import requests

def none_to_str(obj):
    return '' if obj is None else obj

class TenderScraper:
    def __init__(self, title_keyword="Тендерна документація"):
        self.title_keyword = title_keyword

    def get_data(self, url):
        try:
            response = requests.get(url)
            response.raise_for_status()
            return response.json()
        except (requests.RequestException, ValueError):
            return {}

    def download_document(self, url, file_name, tender_id):
        file_path = f"{tender_id}/"
        try:
            response = requests.get(url)
            response.raise_for_status()
        except requests.RequestException:
            return
        if not os.path.exists(file_path):
            os.makedirs(file_path)
        full_path = os.path.join(file_path, file_name or
'unnamed_file')
        with open(full_path, 'wb') as file:
            file.write(response.content)

    def process_tender(self, data, tender_id):
        documents = data.get("Documents", [])
        for doc in documents:
            if doc.get("Title") == self.title_keyword and
"Documents" in doc:
                sub_documents = doc["Documents"]
                for sub_doc in sub_documents:
                    document_url = sub_doc.get("DownloadUrl")
                    file_name = sub_doc.get("FileName", "document")
                    if document_url:
                        self.download_document(document_url,
file_name, tender_id)
```

```
    def get_scraped_data(self, data, collect_winner_documents:
bool):
        budget = ''
        if data.get("Budget"):
            budget = (
                none_to_str(data["Budget"].get("AmountTitle"))
                + none_to_str(data["Budget"].get("CurrencyTitle"))
                + " " + none_to_str(data["Budget"].get("VatTitle"))
            )

        description = data.get("LotTitle", '')

        guarantee = ''
        if data.get("Guarantee"):
            guarantee = (
                none_to_str(data["Guarantee"].get("AmountTitle"))
                +
none_to_str(data["Guarantee"].get("CurrencyTitle"))
                + " " +
none_to_str(data["Guarantee"].get("VatTitle"))
            )

        nomenclatures_list = data.get("Nomenclatures", [])
        if nomenclatures_list:
            nomenclatures = '\n'.join(
                f"{none_to_str(item.get('Title'))}
{item.get('Count', '')} по "
                f"{none_to_str(item.get('DeliveryPeriodTo'))} "
                f"{none_to_str(item.get('DeliveryAddress'))}"
                for item in nomenclatures_list
            )
        else:
            nomenclatures = ''

        tender_dates = ''
        for stage in data.get('Stages', []):
            name_stage = stage.get('Name', '')
            start_date = stage.get('DateFrom', '')
            end_date = stage.get('DateTo', '')
            tender_dates += f"Назва стейджу: {name_stage} Дата
початку: {start_date} Дата закінчення: {end_date}\n"

        participants_info = ''
        winner_documents = {}
        for participant in data.get('Awards', []):
            participant_name = participant.get('ParticipantTitle',
'')
            participant_address =
participant.get('ParticipantAddress', '')
            participant_status = participant.get('Status', '')
```

```
            contact = participant.get('ContactPerson', {})
            participant_phone_number = contact.get('Phone', '')

            participants_info += (
                f"Ім'я: {participant_name} Адреса учасника:
{participant_address} "
                f"Контактний телефон учасника:
{participant_phone_number} "
                f"Статус учасника: {participant_status}\n"
            )
            if collect_winner_documents and participant_status ==
'Переможець':
                for doc_section in participant.get("Documents", []):
                    winner_documents[doc_section.get("Title", "Без
назви")] = [
                        el.get('FileName', 'unnamed') for el in
doc_section.get("Documents", [])
                    ]

        payment_details = ''
        for terms in data.get('PaymentTerms', []):
            payment_type = terms.get('PaymentTermTypeTitle', '')
            payment_event = terms.get('PaymentTermEventTitle', '')
            payment_description = terms.get('Description', '')
            payment_time = str(terms.get('Days', ''))

            payment_details += (
                f"Тип оплати: {payment_type} Вимоги для оплати:
{payment_event} "
                f"Період оплати: {payment_time} днів Опис:
{payment_description}\n"
            )

        criterion_dict = {}

        scraped_data = {
            'Назва тендера': data.get('Title', ''),
            'Дати етапів подачі на тендер': tender_dates,
            'Організатор': data.get('Organizer', {}).get('Name',
''),
            'Категорія': data.get('Category', {}).get('title', ''),
            'Номер в Prozorro': data.get('ProzorroNumber', ''),
            'Учасники тендера': participants_info,
            'Умови оплати': payment_details,
            'Бюджет': budget,
            'Опис Тендеру': description,
            'Гарантія': guarantee,
            'Номенклатура': nomenclatures,
            'Контакти': (
                f"Ім'я: {data.get('Organizer',
{}).get('ContactPerson', {}).get('Name', '')} "
```

```
                f"Номер телефону: {data.get('Organizer',
{}).get('ContactPerson', {}).get('Phone', '')}"
            ),
        }

        for element in data.get('QualificationCriteriaGroups', []):
            title = element.get("Title", "Без категорії")
            if title not in criterion_dict:
                scraped_data[title] = []
            criteria = [el.get("Title", "Без назви") for el in
element.get("Criteria", [])]
            scraped_data[title] += criteria

        if collect_winner_documents:
            scraped_data['Документи переможця'] = winner_documents

        if data.get('MinimalStepAmount'):
            scraped_data['Мінімальний крок аукціона'] =
data['MinimalStepAmount'].replace('\xa0', '.')

        return scraped_data

    def scraper(self, tender_id, collect_winner_documents=True):
        tender_url =
f"https://smarttender.biz/PurchaseDetail/GetTenderModel/?tenderId={t
ender_id}"
        data = self.get_data(tender_url)
        if not data:
            return {"error": "Не вдалося отримати дані по тендеру"}
        self.process_tender(data, tender_id)
        return self.get_scraped_data(data,
collect_winner_documents=collect_winner_documents)
```

data_preprocess.py

```
import os
from pdf2docx import Converter
from spire.doc import *
from spire.doc.common import *
from langchain.text_splitter import RecursiveCharacterTextSplitter
import subprocess
import fitz
import re


def convert_doc_to_docx(doc_file_path):
    with open(doc_file_path, "rb") as file:
        word_doc = Document()
```

```
        word_doc.LoadFromFile(file.name)
        word_doc.SaveToFile(file.name.replace('.doc', '.docx'))
        word_doc.Close()

    os.remove(doc_file_path)
    return doc_file_path.replace('.doc', '.docx')


def convert_pdf_to_docx(pdf_file_path):
    with open(pdf_file_path, "rb") as file:
        pdf_converter = Converter(file.name)
        pdf_converter.convert(file.name.replace(".pdf", ".docx"))
        pdf_converter.close()
    os.remove(pdf_file_path)
    return pdf_file_path.replace('.doc', '.docx')


def convert_all_documents(folder_path):
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.doc'):
            full_doc_path = os.path.join(folder_path, file_name)
            convert_doc_to_docx(full_doc_path)
    for file_name in os.listdir(folder_path):
        if file_name.endswith(".docx"):
            subprocess.call([
                '/usr/bin/libreoffice',
                '--headless',
                '--convert-to', 'pdf',
                '--outdir',
                folder_path,
                os.path.join(folder_path, file_name)
            ])


def split_text_with_langchain(full_text):
    splitter = RecursiveCharacterTextSplitter(
        separators=[r'(?<=[.!?()])\s+(?=[A-ZA-Я])'],
        chunk_size=250,
        chunk_overlap=0,
        is_separator_regex=True
    )

    text_docs = splitter.create_documents([full_text])
    split_docs = splitter.split_documents(text_docs)
    return split_docs


def extract_text_from_documents(folder_path):
    convert_all_documents(folder_path)
    extracted_paragraphs = []
```

```
    failed_docs_message = "Не вдалось прочитати наступні документи:
"

    for file_name in os.listdir(folder_path):
        if file_name.endswith(".pdf"):
            pdf_path = os.path.join(folder_path, file_name)
            try:
                pdf_doc = fitz.open(pdf_path)
                full_text = ""
                if not pdf_doc[0].get_text():
                    failed_docs_message += f"\nДокумент: {file_name}
не був зчитаний правильно"
                for page_index in range(len(pdf_doc)):
                    page = pdf_doc[page_index]
                    cleaned_text = (
                        page.get_text()
                        .replace('\n (', ' (')
                        .replace('\n(', '(')
                        .replace('-\n', '-')
                        .replace('\n', ' ')
                        .replace('\xa0', '')
                        .replace('\t', ' ')
                    )
                    if re.search(r"[0-9][0-9]", cleaned_text[-3:-
1]):
                        cleaned_text = cleaned_text[:-3]
                    if re.search(r"[1-9]", cleaned_text[-2:-1]):
                        cleaned_text = cleaned_text[:-2]
                    full_text += cleaned_text
                split_chunks = split_text_with_langchain(full_text)
                unique_chunks = [chunk.page_content for chunk in
split_chunks]
                for text_chunk in set(unique_chunks):
                    extracted_paragraphs.append(text_chunk)
            except:
                print("Система не змогла відкрити документ: ",
file_name)

    return extracted_paragraphs, failed_docs_message
```

Додаток Д

Диск із квалівікаційною роботою бакалавра