

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» _____ 2025 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Система автоматичної перевірки оформлення бакалаврських
робіт»

Виконав:
студент IV курсу, групи КА-13
Підберезка Артем Віталійович _____

Керівник:
асистент кафедри ММСА, Канцедал Георгій Олегович _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Наталія Василівна _____

Консультант з нормоконтролю:
доцент, к. ф.-м. н. Статкевич Віталій Михайлович _____

Рецензент:
професор кафедри ШІ, д.т.н.,
проф. Данилов Валерій Якович _____

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

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Підберезці Артему Віталійовичу

1. Тема роботи «Система автоматичної перевірки оформлення бакалаврських робіт», керівник роботи Канцедал Георгій Олегович асистент кафедри ММСА, затверджені наказом по університету від «26» травня 2025 р. № 1759-с
2. Термін подання студентом роботи: 11.06.2025.
3. Вихідні дані до роботи: навчально-методична література.
4. Зміст роботи: дослідження предметної області, теоретичний апарат, програмна реалізація, функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., к.е.н., доцент	19.05.2025	01.06.2025

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики дослідження	25.03.2025	Виконано
2	Дослідження літератури за темою бакалаврської дипломної роботи (БДР)	30.03.2025	Виконано
3	Реалізація програмного продукту	07.05.2025	Виконано
4	Опис теоретичної частини	13.05.2025	Виконано
5	Опис практичної частини	18.05.2025	Виконано
6	Оформлення звіту БДР	03.06.2025	Виконано
7	Розрахунок економічної частини	01.06.2025	Виконано
8	Проходження нормативного контролю	30.05.2025	Виконано
9	Перевірка на антиплагіат	31.05.2025	Виконано
10	Підготовка презентації	04.06.2025	Виконано
11	Попередній захист БДР	06.05.2025	Виконано
12	Захист БДР		

Студент

Артем ПІДБЕРЕЗКА

Керівник

Георгій КАНЦЕДАЛ

РЕФЕРАТ

Дипломна робота: 138 с., 24 рис., 9 табл., 3 дод., 29 джерел.

РЕГУЛЯРНІ ВИРАЗИ, PYTHON, PDF, ОФОРМЛЕННЯ, FITZ,
АВТОМАТИЗОВАНА ПЕРЕВІРКА, ВЕБ-ЗАСТОСУНОК

Тема: Система автоматичної перевірки оформлення бакалаврських робіт

Об'єкт дослідження: процес автоматичної перевірки оформлення кваліфікаційних робіт.

Предмет дослідження: методи автоматичного виявлення помилок оформлення бакалаврських робіт у PDF-документах.

Мета роботи: створити систему для автоматичної перевірки відповідності оформлення дипломних робіт згідно державного стандарту України.

У роботі досліджено проблему автоматизації перевірки технічних вимог до оформлення кваліфікаційних робіт, представлених у форматі PDF. Розроблено програмний продукт на мові програмування Python, що дозволяє здійснювати автоматичну перевірку ключових структурних елементів документа: наявності обов'язкових заголовків, відступів, нумерації сторінок, а також коректного оформлення рисунків, таблиць і формул. Результатом роботи є система, що реалізує алгоритми виявлення помилок при оформленні дипломних робіт і доступна як веб-сервіс.

ABSTRACT

Thesis: 138 pages, 24 figures, 9 tables, 3 appendices, 29 references.

REGULAR EXPRESSIONS, PYTHON, PDF, FORMATTING, FITZ,
AUTOMATED CHECKING, WEB APPLICATION

Title: System for automatic verification of bachelor's thesis format

Object of study: automatic process of checking the formatting of qualification papers.

Subject of study: methods for automatically detecting formatting errors in bachelor's theses in PDF format.

Objective: to develop a system for automatic verification of thesis formatting compliance according to the state standard of Ukraine.

This thesis addresses the issue of automating the verification of technical requirements for formatting qualification papers presented in PDF format. A software solution was developed in the Python programming language that enables automatic checking of key structural elements in the document: presence of required headings, indentation, page numbering, and correct formatting of figures, tables, and formulas.

The result is a system that implements algorithms for detecting formatting errors in theses and is available as a web service.

ЗМІСТ

ВСТУП.....	9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	11
1.1 Актуальність теми.....	11
1.2 Аналіз можливих рішень.....	12
1.2.1 Використання шаблону LateX для оформлення.....	12
1.2.2 Використання LLM (Великі мовні моделі).....	14
1.2.3 Алгоритмічний підхід.....	18
1.2.4 Підхід з використанням зображень.....	20
1.2.5 Порівняння підходів.....	21
1.3 Висновки до розділу 1.....	22
РОЗДІЛ 2 ТЕОРЕТИЧНИЙ АПАРАТ.....	24
2.1 Теоретичні основи регулярних виразів.....	25
2.1.1 Формальні мови, граматики та Ієрархія Хомського.....	25
2.1.2 Скінченні автомати, регулярні мови та вирази.....	29
2.2 Регулярні вирази в Python.....	32
2.2.1 Огляд модуля re.....	33
2.2.2 Основні функції модуля re.....	34
2.2.3 Спеціальні символи та метасимволи модуля re.....	35
2.3 Обробка PDF-документів у Python за допомогою бібліотеки fitz.....	38
2.3.1 Загальний опис бібліотеки.....	39
2.3.2 Структури, текст та об'єкти в бібліотеці fitz.....	40
2.4 Висновки до розділу 2.....	43
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ.....	45

3.1	Визначення основних вимог.....	46
3.2	Обґрунтування вибору мови програмування та програмного забезпечення.....	48
3.3	Основні виклики програмної реалізації.....	49
3.4	Налаштування середовища розробки.....	51
3.5	Реалізація веб-застосунку.....	52
3.6	Обмеження реалізованого продукту.....	54
3.7	Реалізація алгоритмів перевірки.....	55
3.7.1	Парсинг змісту.....	57
3.7.2	Класифікація заголовків на рівні.....	59
3.7.3	Перевірка нумерації сторінок.....	60
3.7.4	Перевірка змісту на обов'язкові заголовки.....	61
3.7.5	Перевірка відступів.....	61
3.7.6	Перевірка стилю тексту.....	63
3.7.7	Перевірка початку рядка на недопустимі символи.....	64
3.7.8	Перевірка наявності прізвища після календарного плану.....	65
3.7.9	Перевірка рисунків у документі.....	66
3.7.10	Перевірка таблиць у документі.....	68
3.7.11	Перевірка формул.....	70
3.7.12	Перевірка розміщення заголовків першого рівня.....	72
3.7	Висновки до розділу 3.....	73
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ.....		75
4.1	Постановка задачі проектування.....	75
4.2	Обґрунтування функцій програмного продукту.....	76
4.3	Обґрунтування системи параметрів програмного продукту.....	80

4.4 Аналіз експертного оцінювання параметрів.....	83
4.5 Аналіз рівня якості реалізації основних функціональних компонентів системи.....	87
4.6 Економічний аналіз варіантів розробки програмного продукту.....	89
4.7 Вибір кращого варіанту програмного продукту за техніко-економічним рівнем.....	93
4.8 Висновки до розділу 4.....	94
ВИСНОВКИ.....	96
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	99
ДОДАТОК А. ПРОГРАМНИЙ КОД.....	102
ДОДАТОК Б. ГРАФІЧНІ МАТЕРІАЛИ.....	128
ДОДАТОК В. ДОВІДКА ПРО ВПРОВАДЖЕННЯ.....	138

ВСТУП

У сучасній системі вищої освіти зростає потреба не лише в забезпеченні академічної доброчесності, але й у дотриманні встановлених стандартів оформлення освітніх і наукових документів. Одним із таких стандартів є ДСТУ — державний стандарт, що регламентує вимоги до структури, оформлення та подачі текстових матеріалів, зокрема курсових, дипломних і бакалаврських робіт. Невиконання цих вимог часто стає причиною повернення робіт на доопрацювання, що затримує захист і створює додаткове навантаження як на викладачів так і на студентів.

З розвитком дистанційного навчання та збільшенням кількості студентських робіт, які необхідно перевіряти, зростає актуальність автоматизації процесів контролю відповідності документації до вимог ДСТУ. Ручна перевірка форматування, наявності обов'язкових розділів, правильності оформлення заголовків, списків, таблиць та інших елементів документа є тривалою і ресурсоємкою задачею, яка не завжди гарантує повну об'єктивність.

Метою цієї роботи є розробка автоматизованої системи перевірки відповідності оформлення студентських робіт вимогам чинного державного стандарту. Така система повинна забезпечити швидкий і точний аналіз документів, ідентифікацію порушень та формування звітів, що дозволяють студентам і викладачам оперативно реагувати на зауваження.

Реалізація автоматичної перевірки передбачає використання сучасних інструментів для обробки PDF-файлів, аналізу структурних елементів документів, а також застосування алгоритмів для виявлення форматних відхилень. Це забезпечує уніфікованість оцінювання, зменшує вплив людського фактора та підвищує ефективність освітнього процесу.

Загалом, впровадження таких автоматизованих рішень є важливим

кроком на шляху до підвищення якості освіти, підтримки стандартів і оптимізації адміністративного навантаження в закладах вищої освіти.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

У сучасних умовах цифровізації освіти дедалі більшої ваги набуває автоматизація процесів, пов'язаних з оцінюванням та оформленням навчальних і кваліфікаційних робіт. Одним з обов'язкових елементів підготовки дипломної роботи є її відповідність чинним державним стандартам оформлення документів — зокрема, вимогам ДСТУ. Проте перевірка відповідності дипломної роботи цим стандартам вручну потребує значного часу та зусиль як з боку студентів, так і з боку викладачів, що може призводити до помилок та суб'єктивності в оцінюванні.

Автоматична перевірка дипломних робіт на відповідність ДСТУ дозволить значно підвищити ефективність та об'єктивність цього процесу. Такий підхід сприяє стандартизації вимог, мінімізує людський фактор, забезпечує швидке виявлення порушень форматування та спрощує процес підготовки роботи до захисту. Крім того, автоматизована система може служити інструментом навчання, допомагаючи студентам своєчасно виправляти помилки та краще розуміти вимоги до оформлення наукових робіт.

Однією з ключових переваг впровадження таких систем є забезпечення уніфікованого підходу до контролю якості документів, що особливо важливо у вищих навчальних закладах з великою кількістю студентів. Застосування сучасних технологій обробки текстів, зокрема методів обчислювальної лінгвістики, аналізу структурованих документів, а також елементів машинного навчання, відкриває широкі можливості для створення інтелектуальних систем, здатних самостійно ідентифікувати відхилення від

стандарту, надавати рекомендації щодо їх виправлення та формувати звітність.

Крім того, автоматизовані засоби перевірки оформлення можуть бути інтегровані в загальні системи управління освітнім процесом, що дозволяє підвищити зручність користування та централізувати контроль над виконанням вимог. Такий підхід узгоджується із загальною стратегією цифрової трансформації освіти в Україні та відповідає положенням Державної освітньої цифрової стратегії [1].

1.2 Аналіз можливих рішень

У контексті автоматизації перевірки відповідності дипломних робіт вимогам державних стандартів (зокрема ДСТУ) виникає необхідність ретельного аналізу наявних технічних та програмних рішень, які можуть бути використані або адаптовані для реалізації такої системи. Цей розділ має на меті розглянути існуючі підходи, технології та інструменти, що забезпечують автоматизовану перевірку оформлення документів.

1.2.1 Використання шаблону LateX для оформлення

На сьогодні у багатьох університетах світу активно впроваджуються засоби автоматизації оформлення та перевірки дипломних робіт. Одним з найпоширеніших підходів є використання шаблонів у форматі LaTeX (мова розмітки даних для високоякісного оформлення документів), які дозволяють забезпечити відповідність роботи структурним та технічним вимогам закладу

вищої освіти. Такі шаблони не лише полегшують процес підготовки документа, а й частково автоматизують перевірку на відповідність формальним критеріям: наявність усіх необхідних розділів, правильність форматування, структура бібліографії тощо.

Наприклад, у Zhejiang University (Китай) застосовується шаблон **ZJUTHesis** [2, 3], який підтримує автоматичну перевірку структури документа та бібліографічних посилань. У Tampere University (Фінляндія) студенти користуються шаблоном **tau-latex-thesis-template** [4], що також містить інструменти для автоматичного генерування глосаріїв, бібліографії та контролю форматування. В Amirkabir University of Technology (Іран) та Tel-Aviv University (Ізраїль) використовуються адаптовані шаблони, які відповідають вимогам конкретних факультетів і підтримують багатомовність.

Університет Штутгарта (Німеччина) впровадив **Scientific Thesis Template** [5], який широко використовується в технічних університетах Німеччини та забезпечує уніфікованість оформлення дипломних робіт.

Використання таких шаблонів дозволяє уникати типових помилок в оформленні, забезпечити відповідність робіт установленим стандартам, а також зменшити навантаження на наукових керівників у частині перевірки структури та формальних вимог.

Узагальнюючи, можна сказати, що використання шаблонів на базі LaTeX є ефективним інструментом для часткової автоматизації процесу оформлення дипломних робіт, особливо у технічних та природничих університетах. Вони забезпечують відповідність базовим структурним вимогам, дозволяють уникати типових помилок і значно полегшують як підготовку документа, так і його первинну перевірку. Проте такі шаблони не охоплюють повний спектр вимог, передбачених державними стандартами (зокрема ДСТУ), що обмежує їх застосування в умовах української освітньої системи без додаткової адаптації.

Тому для забезпечення повної автоматизованої перевірки відповідності

дипломних робіт ДСТУ доцільно дослідити і розробити більш гнучкі та інтелектуальні програмні рішення. Такі системи мають включати функціонал перевірки форматування різних елементів документа (включно з ілюстраціями, формулами, таблицями, списками тощо) та інтеграцію з популярними текстовими редакторами, що дозволить забезпечити повну відповідність робіт формальним вимогам і підвищити загальний рівень якості підготовки дипломних кваліфікаційних робіт.

1.2.2 Використання LLM (Великі мовні моделі)

З розвитком штучного інтелекту та зокрема великих мовних моделей (LLM), таких як ChatGPT, з'явилася можливість використовувати ці інструменти для автоматичного аналізу текстових документів, зокрема дипломних робіт. Ці моделі демонструють здатність працювати з неструктурованими текстами, розуміти контекст, аналізувати логіку викладення матеріалу та навіть виявляти помилки у змісті. Такий потенціал викликає зацікавленість у дослідженні можливостей LLM як інструменту перевірки відповідності документів стандартам оформлення, зокрема ДСТУ. При правильному підході модель може:

- 1) аналізувати зміст тексту дипломної роботи;
- 2) виявляти порушення логіки структури;
- 3) знаходити відсутні елементи (наприклад, посилання на таблиці або рисунки);
- 4) пропонувати покращення оформлення;
- 5) працювати як в режимі інтерактивного асистента для перевіряючого, так і в повністю автоматичному режимі;
- 6) виявляти ознаки автоматичної генерації тексту.

Останній пункт є особливо актуальним у контексті зростання використання ШІ для створення контенту. Існують спеціалізовані нейронні мережі, здатні визначати, чи був певний текст написаний іншою мовною моделлю. Такий аналіз ґрунтується на виявленні характерних статистичних і лінгвістичних ознак: частотності слів, шаблонів побудови речень, передбачуваності вибору слів тощо [6, 7].

Хоча точність таких методів не є абсолютно точною, вони демонструють досить непогані результати і знаходять практичне застосування, зокрема у виявленні автоматично згенерованого контенту в освітньому, науковому та журналістському середовищах.

Наявні переваги:

- 1) висока швидкість аналізу (при великій кількості ресурсів);
- 2) гнучкість і здатність працювати з неструктурованим або слабоструктурованим текстом;
- 3) підтримка декількох мов;
- 4) виявлення помилок та неточностей пов'язаних зі змістовним наповненням.

Наявні недоліки:

- 1) залежність від якості промптів (інструкцій);
- 2) не завжди точне розпізнавання структури документів;
- 3) не завжди достовірні результати — через генеративну природу моделі можуть “домислювати” або узагальнювати інформацію, навіть якщо вона відсутня в тексті;
- 4) високі вимоги до ресурсів;
- 5) відсутність підтримки специфічних форматів і стандартів (наприклад, ДСТУ) — LLM не мають вбудованого розуміння українських технічних стандартів, їх потрібно адаптувати вручну через інструкції.

Далі був проведений експеримент в ході якого було здійснені спроби перевірки оформлення документа згідно ДСТУ за допомогою великої мовної моделі(ChatGPT).

Метою експерименту було оцінити, наскільки точно модель може розпізнавати формальні помилки оформлення, працюючи із реальним PDF-документом та відповідним запитом (промптом).

Для експерименту було взято дипломну роботу з минулих років, в якій було неправильно пронумерований рисунок (перший рисунок третього розділу був пронумерований як 3.14 замість 3.1). Великій мовній моделі був заданий промпт (рис. 1.1) та надано файл в форматі PDF. В результаті перевірки отримали таку відповідь (рис. 1.2).

Перевір таку вимогу в pdf-файлі:
Рисунки нумеруємо подвійною нумерацією, тобто номер рисунка складається з номера розділу та порядкового номера рисунка в цьому розділі і розташований під рисунком. На кожен рисунок має бути посилання в тексті, причому має бути спочатку посилання на рисунок, а потім - сам рисунок.

Рисунок 1.1 — Промпт заданий ChatGPT

 **Висновок:**

У дипломній роботі дотримано всіх вимог:

- подвійна нумерація рисунків;
- підпис під зображенням;
- посилання в тексті перед рисунком.

Якщо потрібно, я можу згенерувати таблицю з усіма рисунками та відповідними сторінками й прикладами посилань у тексті.

Рисунок 1.2 — Вихідна відповідь мовної моделі

Так ми бачимо, що велика мовна модель не ідентифікувала помилку, проте також зауважимо, що промпт міг бути заданий некоректно, тому проведемо повторний запит до того ж самого файлу і вимоги (рис. 1.3). Також такий результат міг бути спричинений природою метрик які використовуються в LLM, а точніше — вихідний результат узагальнюється. Тому оскільки більшість перевірених рисунків була оформлена правильно — результатом стало правильність оформлення всіх.

Знайди всі рисунки які не відповідають вимогам

Рисунок 1.3 — Повторний запит

Таким чином отримуємо більш розширену відповідь, яка більш детально перевірила вимогу (рис. 1.4). Також бачимо, що модель занурилась у проблеми правильного підписання рисунку та формальності посилання без контекстного супроводження. Проте з поставленим завданням (знаходження неправильно підписаного рисунку 3.14) ChatGPT не впорався.

Ось стисла перевірка рисунків, що не відповідають вимогам:

№	Рисунок	Проблема	Коментар
1	Рис. 1.12	Немає посилання в тексті	Є згадка "формула (2)", але не як "рисунок"
2	Рис. 1.13	Посилання формальне	Лише згадка в дужках, без опису або контексту
3	Рис. 3.9, 3.10	Невірний формат підпису	Відсутнє тире чи двокрапка
4	Рис. 4.3–4.5	Немає індивідуальних посилань	У тексті лише загальна згадка

✅ Решта рисунків відповідає вимогам (подвійна нумерація, підпис знизу, посилання в тексті перед рисунком).

Рисунок 1.4 — Результат повторного запиту

Результати проведеного експерименту вказують на обмежену ефективність застосування великих мовних моделей для перевірки саме формальних вимог до оформлення дипломних робіт. Незважаючи на високий

потенціал моделей у плані аналізу контексту та змістовної частини документа, їх здатність до точної верифікації формальних елементів, таких як нумерація рисунків, виявилася недостатньою. Навіть після уточнення промпту, модель не змогла виявити конкретну помилку у нумерації, що свідчить про її обмеження з деталізованими структурними елементами PDF-документів.

Це підтверджує важливість правильного формулювання інструкцій при роботі з LLM, а також вказує на необхідність комбінування мовних моделей із класичними алгоритмами структурного аналізу документів. Таким чином, хоча великі мовні моделі можуть бути корисними як інтелектуальні асистенти для аналізу змісту та граматики, для цілей повної автоматизованої перевірки відповідності ДСТУ вони потребують доопрацювання або інтеграції з іншими технологіями.

1.2.3 Алгоритмічний підхід

Слід зазначити що хоча існує багато загальних досліджень, пов'язаних з обробкою текстів, розпізнаванням структури документів та перевіркою граматики або плагіату, спеціалізованих робіт, статей та підходів присвячених автоматичній перевірці оформлення кваліфікаційних робіт майже немає. Одним із небагатьох винятків є публікація [8], у якій описано підхід до побудови системи автоматизованої перевірки оформлення текстових документів відповідно до вимог ДСТУ. У ній представлено архітектуру та функціональні модулі такого інструменту, а також проведено аналіз ефективності алгоритмів обробки PDF-файлів.

Алгоритмічна перевірка передбачає створення спеціалізованих програмних засобів, які автоматично аналізують структуру та оформлення

PDF-документів згідно з вимогами ДСТУ. Такий підхід базується на аналізі текстового вмісту, атрибутів форматування (шрифт, міжрядковий інтервал, відступи), розташування елементів сторінки та інших параметрів, що піддаються формалізації з допомогою технологій обробки структури PDF-документів.

Для реалізації цього підходу використовуються бібліотеки для обробки PDF-документів. Алгоритми виконують витяг тексту з координатами, аналізують відповідність шаблону структури документа, перевіряють наявність та правильність заголовків, змісту, таблиць, рисунків, списків літератури тощо.

Переваги такого підходу.

1. Висока точність у перевірці чітко визначених параметрів (розмір шрифту, міжрядковий інтервал, відступи, наявність заголовків, титульної сторінки тощо).
2. Повна автоматизація — можливість пакетної перевірки великої кількості документів без участі людини.
3. Прозорість логіки перевірки — чітко фіксовані правила і критерії.
4. Низька вимога до ресурсів.

Недоліки такого підходу.

1. Обмежена гнучкість — важко обробляти документи з нетиповими або змішаними форматами.
2. Необхідність оновлення при зміні стандартів (наприклад, перехід із ДСТУ 3008:2015 на майбутній ДСТУ).
3. Потреба у точному формалізуванні стандарту — не всі вимоги ДСТУ можуть бути чітко виражені у вигляді правил.

1.2.4 Підхід з використанням зображень

Важливим аспектом автоматичної перевірки дипломної роботи є початкова дія – ідентифікація об'єкта для подальшого аналізу, а тобто виявлення таблиць, рисунків, заголовків є критично важливим. Саме в цьому моменті для точності детектування можна використовувати OCR для таких об'єктів.

Оптичне розпізнавання символів (OCR) — технологія, що дає змогу автоматично зчитувати текстову інформацію з зображень документів, сканів чи фотографій. Вона широко застосовується для цифровізації друкованих текстів, автоматизації введення даних, обробки анкет, а також у системах моніторингу та аналізу [9].

Технологія OCR може виконувати перетворення графічного вмісту дипломної роботи у текстову форму, що дозволяє здійснювати аналіз структури документа, зокрема виявлення наявності заголовків, підрозділів, списків використаних джерел, нумерації сторінок тощо. Після розпізнавання тексту можливо автоматично перевірити відповідність параметрів оформлення — таких як тип шрифту, міжрядковий інтервал, відступи, абзацна структура — вимогам чинного стандарту. Також OCR дозволяє проводити семантичний аналіз змісту роботи, зокрема виявляти повторювані або запозичені фрагменти тексту, що є основою для виявлення плагіату.

Суттєвою перевагою OCR є здатність масштабовано обробляти великі обсяги даних, що дозволяє реалізувати ефективну систему контролю якості документів. Завдяки використанню сучасних інструментів, таких як бібліотека Tesseract, яка підтримує українську мову, система може точно розпізнавати специфічні елементи тексту, включаючи формули, нумерацію, лапки, дужки та інші символи, що часто трапляються в академічних роботах.

З технічного погляду, OCR-системи можуть використовувати як

традиційні методи шаблонного порівняння символів, так і більш сучасні нейромережеві моделі, здатні адаптуватися до якості вхідних даних, різних шрифтів, рукописного тексту чи змішаних типів форматування [10]. У контексті дипломної роботи така адаптивність дозволяє забезпечити високу точність аналізу навіть у випадках, коли робота виконана з відхиленнями від типового шаблону.

Таким чином, оптичне розпізнавання символів може бути основою у реалізації систем автоматичної перевірки дипломних робіт. Воно забезпечить базовий рівень доступу до текстового вмісту документа, без чого подальший аналіз оформлення, структури та змісту роботи був би неможливим. OCR-технологія є не лише інструментом перетворення графіки у текст, але й критичним елементом загального алгоритму інтелектуального аналізу документів у відповідності до державних стандартів.

1.2.5 Порівняння підходів

Для вирішення питання вибору підходу для реалізації системи автоматичної перевірки бакалаврських робіт було проведено експертне опитування серед фахівців в галузі нормативного контролю. Після чого отримані результати було узагальнено в таблиці 1.1 нижче.

Таблиця 1.1 — Порівняння підходів за критеріями

Критерій	LLM-моделі	Алгоритмічний підхід	Підхід з використанням зображень
Швидкість	Низька	Висока	Середня

Продовження таблиці 1.1

Критерій	LLM-моделі	Алгоритмічний підхід	Підхід з використанням зображень
Глибина аналізу змісту	Висока	Низька	Низька
Вимоги до ресурсів	Високі	Низькі	Середні
Точність формальної перевірки	Середня	Висока	Висока
Масштабованість	Висока	Висока	Висока
Гнучкість до змін	Висока	Низька	Середня
Складність реалізації	Висока	Висока	Висока

Як бачимо немає підхода, який би був найкращий по всіх представлених критеріях. Тому напевно найкращим вибором був би деякий комбінований підхід, який би міг включати в себе всі переваги кожного з них.

1.3 Висновки до розділу 1

У ході аналізу предметної області було з'ясовано, що перевірка відповідності дипломних робіт стандартам оформлення (ДСТУ) є актуальним завданням у контексті цифровізації освіти та потребує ефективних засобів автоматизації. Існуючі підходи до вирішення цієї задачі можна умовно поділити на три категорії: використання великих мовних моделей (LLM),

алгоритмічна перевірка PDF-документів, обробка зображень (OCR).

Великі мовні моделі (LLM), такі як GPT або аналогічні, демонструють високий потенціал у розумінні тексту, генерації звітів і взаємодії з користувачем. Проте їх використання у формальному аналізі дипломних робіт має низку обмежень. Насамперед, LLM мають властивість до узагальнення: вони схильні інтерпретувати вимоги не буквально, а у більш м'якій, контекстуально-орієнтованій формі. Це робить їх менш підходящими для задач, де потрібна точна відповідність формальним критеріям, таким як конкретні розміри полів, шрифт, міжрядковий інтервал тощо. Крім того, ці моделі не завжди коректно обробляють структуровані документи у форматі PDF, що ще більше ускладнює їх застосування для технічного аудиту оформлення.

Натомість алгоритмічний підхід, хоча й менш гнучкий до змін структури документів або варіантів оформлення, демонструє найвищу точність у перевірці саме формальних вимог. Він дозволяє реалізувати чітко визначені правила (наприклад, “лівий відступ — 30 мм”, “шрифт — Times New Roman, 14 pt”, “нумерація сторінок починається з другого аркуша” тощо), які перевіряються без інтерпретації або допущень.

Основним недоліком такого підходу є його залежність від структури документів — відхилення або спосіб збереження PDF можуть ускладнити аналіз. Проте при стабільному форматі вхідних файлів алгоритмічна перевірка дозволяє повністю автоматизувати процес контролю оформлення, зберігаючи при цьому прозорість та повторюваність результатів.

РОЗДІЛ 2 ТЕОРЕТИЧНИЙ АПАРАТ

У попередньому розділі було проаналізовано можливі підходи до автоматизованої перевірки оформлення документів згідно з вимогами державних стандартів. За результатами аналізу було обґрунтовано доцільність використання алгоритмічного підходу, який передбачає поетапну перевірку окремих структурних елементів документа на відповідність формальним правилам (наприклад, відступи, заголовки, зміст, нумерація тощо).

Застосування такого підходу потребує доступу до вмісту документа та його метаданих — шрифтів, розмітки, положення текстових блоків, а також можливості аналізувати сам текст на предмет відповідності певним шаблонам. Для реалізації цього було обрано два ключові інструменти.

1. Регулярні вирази відповідають за опис формальних шаблонів тексту і подальшої перевірки таких елементів, як формат заголовків, нумерація розділів, структура змісту, відповідність підписів рисунків і таблиць тощо.
2. Бібліотека `fitz` (PyMuPDF) потрібна для доступу до внутрішньої структури PDF-документів, зчитування текстових блоків разом із їхніми координатами, шрифтами, відступами та іншими атрибутами форматування.

Об'єднання цих інструментів забезпечує надійну основу для побудови системи автоматизованої перевірки, яка може працювати незалежно від конкретного шаблону документа, орієнтуючись лише на загальні правила оформлення. У цьому розділі детально розглянуто теоретичні основи регулярних виразів та функціональні можливості бібліотеки `fitz`, що забезпечують реалізацію зазначеного підходу.

2.1 Теоретичні основи регулярних виразів

2.1.1 Формальні мови, граматики та Ієрархія Хомського

Теорія формальних мов виникла як окрема дисципліна завдяки роботам лінгвіста Ноама Хомського в 1950-х роках. Його метою було створити точне математичне представлення синтаксису природних мов. У процесі цих досліджень з'ясувалося, що ті ж самі формальні методи можуть бути ефективно застосовані й до синтаксису штучних мов, зокрема мов програмування. Саме контекстно-вільні граматики, що входять до так званої ієрархії Хомського, лягли в основу опису мов програмування на кшталт Fortran, C чи Java [11, 12].

З появою сучасних комп'ютерів стало зрозуміло, що будь-яку інформацію — чи то числа, текст, зображення чи звук — можна представити у вигляді послідовностей символів, тобто строк.

Тоді дамо визначення поняттю формальна мова — це штучно створена людиною множина слів, фраз або синтаксичних конструкцій, яка дотримується суворо визначених правил. Основною метою формальних мов є забезпечення однозначної інтерпретації виразів, що надзвичайно важливо для застосування в інформатиці, математиці, лінгвістиці, біоінформатиці та суміжних галузях.

На відміну від природних мов, які допускають метафори, контекстуальність і багатозначність, формальні мови є строго визначеними: кожне слово в такій мові або допускається, або ні, залежно від встановлених синтаксичних правил. Прикладами формальних мов є математичні вирази, мови програмування, хімічні формули тощо.

Далі потрібно дати визначення базовим поняттям. Скінченна множина

символів, неподільних у даному розгляді, називається словником чи алфавітом, а символи, що входять у множину — буквами алфавіту. Словом над алфавітом A називається довільна скінченна (можливо, порожня) послідовність символів цього алфавіту A . Символи у словах можуть повторюватися. Слово називається підсловом іншого слова, якщо його можна виділити в середині, на початку або в кінці цього слова. В окремих випадках розрізняють префікси (на початку), суфікси (у кінці) та власні підслова (не рівні всьому слову) [13-15].

Довжину слова α , тобто кількість символів у ньому позначають — $|\alpha|$. Слово довжини 0 називають порожнім і позначають як ε . Самі символи алфавіту A є словами довжини 1. Множина всіх слів над алфавітом A позначається A^* , множина без $*$ порожнього слова ε — A^+ , множина усіх слів довжиною не більше k — A^k .

Таким чином формальне визначення формальної мови буде наступним. Нехай задано алфавіт A , який розбито на дві непорожні підмножини V і T , що не перетинаються: $A = V \cup T$, $V \cap T = \emptyset$. Множину V називають алфавітом нетермінальних символів (нетермінальним алфавітом), а T — алфавітом термінальних символів (термінальним алфавітом). Тоді формальною мовою або просто мовою над алфавітом T називають будь-яку підмножину L множини T^* : $L \subset T^*$ [13-15].

Над формальними мовами визначаються такі операції [13-15].

1. Об'єднання: $L_1 \cup L_2$ — усі слова з L_1 або L_2 .
2. Перетин: $L_1 \cap L_2$ — слова, що є в обох мовах.
3. Різниця: $L_1 \setminus L_2$ — слова, що є в L_1 , але не в L_2 .
4. Доповнення: $L^- = T^* \setminus L$ — усі слова, що не входять у мову L .
5. Конкатенація: $L_1 \cdot L_2 = \{\alpha \cdot \beta: \alpha \in L_1, \beta \in L_2\}$.

6. Замикання (зірочка) Кліні: L^* — множина, що включає всі можливі конкатенації довільної кількості слів з L , включно з ϵ .

7. Обернення мови: якщо w — слово, то його дзеркальне відображення w^R — це слово з тими ж літерами у зворотному порядку.

Дзеркалом мови L є мова $L^R = \{w^R : w \in L\}$ [13-16].

Формальна граматика — це сукупність правил, які визначають, як утворюються допустимі рядки (слова) у формальній мові. Формально її задають як кортеж:

$$G = \langle V, T, P, S \rangle, \quad (1.1)$$

де V — алфавіт нетермінальних символів, які не мають фіксованого значення і можуть бути замінені іншими символами згідно з правилами;

T — алфавіт термінальних символів, які мають конкретне значення й не замінюються (наприклад, букви, цифри тощо);

P — непорожня скінченна множина продукцій на алфавітом $A = V \cup T$;

$S \in V$ — фіксований нетермінальний символ або джерело.

Продукції застосовуються для поступової заміни нетерміналів у рядках до отримання рядка, що складається лише з термінальних символів (тобто слова мови).

Найпоширеніша класифікація формальних граматик була запропонована Ноамом Хомським у 1956 році для аналізу природних мов і стала фундаментом теорії формальних мов і автоматів. Ця класифікація формальних граматик (таблиця 2.1), яка розділяє їх на чотири основні типи залежно від вигляду продукцій (див. роботи Н.Хомського [11, 12], а також [13-15]).

Типи граматик у ієрархії Хомського.

1. **Тип 0** — найзагальніший тип що описує всі формальні мови, які розпізнаються машиною Тюрінга. До нього відноситься будь-яка формальна граматика.
2. **Тип 1** — контекстно-залежні граматики це ті, де множина продукцій має лише продукції вигляду $\alpha A \beta \rightarrow \alpha \gamma \beta$, де A — один нетермінальний символ, α і β — послідовності символів (включаючи порожні послідовності), а γ — послідовність символів, яка не може бути порожньою. Такі граматики описують мови, що розпізнаються лінійно обмеженими автоматами або лінійно обмеженими машинами Тюрінга.
3. **Тип 2** — контекстно-вільні граматики це такі, що множина продукцій містить лише продукції вигляду $A \rightarrow \alpha$, де A — один нетермінальний символ, а α — послідовність символів, яка може містити як термінали, так і нетермінали. Використовуються для опису синтаксису програмних мов.
4. **Тип 3** — регулярні граматики мають лише продукції вигляду $A \rightarrow \alpha B$ або $A \rightarrow \alpha$, де A і B — нетермінальні символи, α — термінальний символ (виділяють і праволінійні граматики де α — послідовність термінальних символів). Такі граматики описують регулярні мови й відповідають скінченним автоматам.

Таблиця 2.1 — Відповідність класу абстрактних автоматів до граматик.

Тип граматики	Клас формальних мов	Клас абстрактних автоматів
0	Напіврозв'язні	Мащини Тюрінга
1	Контекстно-залежні	Лінійно-обмежені МТ
2	Контекстно-вільні	Магазинні автомати
3	Регулярні	Скінченні автомати

2.1.2 Скінченні автомати, регулярні мови та вирази

Скінченні автомати є ключовою технологією в обробці природної мови. Вони застосовуються у таких завданнях, як пошук у тексті, токенізація, синтаксичний аналіз, перевірка орфографії, машинний переклад, розпізнавання сутностей та обробка мовлення. У деяких випадках рішення повністю базуються на скінченних автоматах, зокрема в морфологічній та фонологічній обробці [15].

Регулярні вирази та скінченні автомати з 1940-х років є основою обчислювальних моделей в інформатиці. У лінгвістиці використовують модифіковані варіанти автоматів — зокрема трансдуктори та ймовірнісні автомати, що менш поширені в теоретичній інформатиці [15].

Скінченний автомат — це математична модель, яка описує поведінку системи з обмеженою кількістю станів. Він читає вхідний рядок символів і переходячи між станами згідно з визначеними правилами, дивиться чи належить цей рядок певній мові. Скінченні автомати є моделлю регулярних мов — класу мов у ієрархії Хомського (тип 3). Вони лежать в основі таких завдань, як розпізнавання шаблонів, компіляція, аналіз тексту та обробка мови.

Формально скінченний автомат визначається як п'ятірка-набір

$$M = \langle Q, T, \Delta, I, F \rangle, \quad (1.2)$$

де Q та T — скінченні непорожні множини;

$$\Delta \subseteq (Q \times T \times Q), \quad I \subseteq Q, \quad F \subseteq Q;$$

Q — множиною станів;

T — алфавітом вхідних символів;

Δ — відношенням (множиною) переходів;

I — множиною початкових станів;

F — множиною допускаючих (кінцевих) станів [13-15].

Скінченний автомат працює з послідовністю символів, переходячи між станами за чітко визначеними правилами. Щоб формально описати цей процес, використовують поняття конфігурації.

Конфігурація автомата $\langle Q, T, \Delta, I, F \rangle$ — це пара:

$$(q, w), \quad (1.3)$$

де $q \in Q$ — поточний стан автомата;

$w \in T^*$ — частина вхідного слова, яку ще залишилось обробити.

Для скінченного автомата $M = \langle Q, T, \Delta, I, F \rangle$ визначено важливе поняття - такту роботи, яке формально описує один крок обробки вхідного слова. Це поняття задається спеціальним бінарним відношенням " \vdash_M " на множині всіх конфігурацій автомата. Відношення такту роботи визначається наступним чином, для двох конфігурацій (q_1, w) і (q_2, u) маємо $(q_1, w) \vdash_M (q_2, u)$ тоді і тільки тоді, коли виконуються одночасно дві умови:

- 1) вхідне слово w може бути представлене як $w = au$, де $a \in T$ (один символ вхідного алфавіту), $u \in T^*$ (залишок слова);
- 2) а також існує відповідний перехід у автоматі: $(q_1, a, q_2) \in \Delta$.

Тобто відношення " \vdash_M " містить пари конфігурацій вигляду $((q_1, au), (q_2, u))$, де $(q_1, a, q_2) \in \Delta$.

Скінченний автомат $M = \langle Q, T, \Delta, I, F \rangle$ допускає (або розпізнає) слово w з множини T^* , якщо існує такий початковий стан $q_0 \in I$ і такий допускаючий стан $q \in F$, що автомат M може перейти від конфігурації (q_0, w) до (q, ε) за деяку кількість кроків. Множина всіх слів $L[M]$, які розпізнає автомат M , називається формальною мовою, що задається цим автоматом. Якщо мову можна задати за допомогою хоча б одного скінченного автомата, то вона називається автоматною [13].

Існує два основних типи скінченних автоматів: детерміновані та

недетерміновані. Обидва вони розпізнають один і той самий клас мов — регулярні мови, проте відрізняються організацією переходів.

У детермінованому скінченному автоматі для кожного стану та вхідного символу існує лише один можливий перехід. Натомість у недетермінованому скінченному автоматі допускається кілька переходів або їх відсутність, зокрема можуть бути переходи без читання символу (ϵ -переходи). Це означає, що під час обробки рядка недетермінований автомат може розглядати кілька варіантів одночасно, і рядок вважається прийнятим, якщо хоча б один шлях призводить до допускаючого стану [16].

Регулярні вирази — це формальні записи, які описують множини рядків над деяким термінальним алфавітом T .

Формально це визначається так, нехай T — фіксований термінальний алфавіт. Множину регулярних виразів над T (або просто регулярних виразів) визначають рекурсивно трьома умовами [13]:

- 1) символи 0 , 1 та довільний символ $a \in T$ є регулярними виразами над T ;
- 2) якщо r та s — регулярні вирази, то: $(r + s)$, $(r \cdot s)$, r^* є регулярними виразами над T ;
- 3) інших регулярних виразів над T немає.

Множину всіх регулярних виразів над T позначається як $R[T]$.

Одним із ключових результатів теорії формальних мов є теорема Кліні, яка встановлює еквівалентність між регулярними виразами та скінченними автоматами. Вона стверджує, що для кожної мови, яку можна описати регулярним виразом, існує скінченний автомат, який розпізнає цю мову, і навпаки — для будь-якої мови, що розпізнається скінченним автоматом, існує регулярний вираз, який її описує [13, 16].

Цей результат є надзвичайно важливим, оскільки об'єднує два на перший погляд різні способи задання мов: логічний — через регулярні вирази, і механічний — через автомати. Таким чином, регулярні вирази та

скінченні автомати є різними формалізмами, що оперують з одними й тими самими об'єктами — регулярними мовами.

З практичної точки зору це означає, що опис мови за допомогою регулярного виразу може бути автоматично трансформований у скінченний автомат, який реалізує розпізнавання слів цієї мови, наприклад, у програмному забезпеченні. І навпаки — якщо є заданий скінченний автомат, можна вивести з нього регулярний вираз, що задає ту саму мову, яку він розпізнає. У цьому сенсі теорема Кліні дає потужний інструмент переходу між різними поданнями регулярних мов, дозволяючи обирати той, що є зручнішим у конкретній ситуації — аналітичній чи алгоритмічній [14, 16].

2.2 Регулярні вирази в Python

Регулярні вирази — це дуже вузька спеціалізована мова для опису шаблонів пошуку в тексті, яка реалізована в Python через стандартний модуль `re`. Шаблони регулярних виразів компілюються у внутрішній байт-код, який виконується високопродуктивним механізмом, написаним на мові C. Для досягнення більшої ефективності при обробці складних виразів іноді потрібно враховувати особливості цього механізму та формулювати регулярні вирази таким чином, щоб створений байт-код працював швидше. Водночас мова регулярних виразів є досить обмеженою за функціональністю, тому не всі задачі обробки рядкових даних можна ефективно розв'язати за допомогою регулярних виразів [19].

2.2.1 Огляд модуля re

Модуль re є частиною стандартної бібліотеки мови програмування Python та забезпечує засоби для роботи з регулярними виразами. Основне призначення модуля полягає у виконанні операцій пошуку, перевірки відповідності, розбиття рядків та заміни фрагментів тексту відповідно до заданих шаблонів [17].

Регулярні вирази, що використовуються в модулі re, є формальним способом опису послідовностей символів, які відповідають певному синтаксичному правилу. Модуль реалізує підтримку широкого набору синтаксичних конструкцій, зокрема метасимволів, квантифікаторів, групування, а також класів символів. Це дозволяє створювати як прості шаблони, так і складні вирази для обробки текстових даних [17].

Функціональність модуля охоплює основні операції з рядками, зокрема пошук першого входження шаблону, пошук усіх входжень, перевірку відповідності початку рядка шаблону, а також заміну фрагментів рядка. Крім того, модуль дозволяє здійснювати попереднє компілювання регулярних виразів для підвищення продуктивності при багаторазовому використанні [17].

Модуль re підтримує використання різних параметрів налаштування, які дозволяють змінювати режим обробки тексту, зокрема режим багаторядкової обробки, нечутливість до регістру символів, а також розширений режим для покращення читабельності виразів [18].

Використання модуля re є доцільним у тих випадках, коли виникає необхідність у формалізованому аналізі або трансформації текстової інформації відповідно до заздалегідь визначених шаблонів.

2.2.2 Основні функції модуля re

Модуль re мови Python надає широкий спектр функцій для роботи з регулярними виразами. До основних належать: `match()`, `fullmatch()`, `search()`, `findall()`, `finditer()`, `sub()`, `subn()`, `split()`, `compile()`, `escape()` та `purge()` [17 - 19].

Ці функції забезпечують різні операції над рядками: пошук збігів, вилучення фрагментів, заміну підрядків, розбиття тексту, а також оптимізацію роботи зі складними шаблонами. Найпоширенішими і найбільш часто використовуваними є `match()`, `search()`, `findall()`, `sub()` та `compile()` саме вони реалізують основні типи обробки текстових даних із застосуванням регулярних виразів.

Функція `match()` перевіряє, чи починається рядок із послідовності символів, що відповідає заданому шаблону. У разі успішного співпадіння повертається об'єкт, який містить інформацію про знайдений фрагмент. Вона використовується, коли необхідно переконатися у правильності початку рядка за певним шаблоном.

Функція `search()` виконує пошук першого входження шаблону в будь-якій частині рядка. Результатом є об'єкт із деталями про позицію та зміст знайденого фрагмента. Цей метод є більш гнучким за `match()`, оскільки дозволяє знаходити відповідності не тільки на початку тексту.

Функція `findall()` повертає список усіх неперекривних входжень шаблону у вигляді рядків або груп символів. Вона застосовується для вилучення множини елементів, що відповідають певному правилу, наприклад, усіх чисел або слів у тексті.

Функція `sub()` виконує заміну всіх входжень шаблону на вказаний рядок або результат виклику функції. Це корисно для автоматичного форматування або очищення тексту, замінюючи небажані елементи на потрібні.

Функція `compile()` створює скомпільований об'єкт регулярного виразу,

який можна повторно використовувати, що покращує продуктивність при багаторазовому застосуванні одного і того ж шаблону. Скомпільовані об'єкти підтримують ті ж методи, що і функції модуля `re` [17 - 19].

Додаткові функції, такі як `finditer()`, `split()`, `escape()` та `purge()`, розширюють можливості модуля:

- 1) `finditer()` повертає ітератор об'єктів збігів для детального аналізу;
- 2) `split()` здійснює розбиття рядка за шаблоном;
- 3) `escape()` автоматично екранує спеціальні символи у рядках;
- 4) `purge()` очищує кеш скомпільованих шаблонів, що може бути корисним при роботі з великою кількістю виразів для оптимізації пам'яті.

Перелічені функції формують базу для ефективної роботи з регулярними виразами в Python, дозволяючи виконувати різноманітні операції з текстом відповідно до формалізованих шаблонів.

2.2.3 Спеціальні символи та метасимволи модуля `re`

У регулярних виразах більшість літер і символів співпадають із самими собою. Наприклад, регулярний вираз `"iаса"` точно збігатиметься з рядком `"iаса"`. Водночас можна використовувати режим без урахування регістру, що дозволить знаходити збіги з `"Iаса"` або `"IASA"`.

Однак існують символи, які не співпадають із собою, а мають спеціальне значення — їх називають метасимволами. Вони вказують механізму регулярних виразів на пошук певних шаблонів або змінюють поведінку інших частин виразу, наприклад, задаючи повторення або межі пошуку.

До основних метасимволів належать наступні.

1. Крапка (.) — символ, який відповідає будь-якому символу, крім символу нового рядка (за замовчуванням). Це універсальний знак, що дозволяє описувати будь-які символи на певній позиції шаблону.
2. Зірочка (*) — вказує на повторення попереднього символу або групи нуль або більше разів. Дозволяє визначати необмежену кількість повторень.
3. Плюс (+) — означає повторення одного або більше разів.
4. Знак питання (?) — робить попередній символ або групу необов'язковими, допускаючи їхню появу нуль або один раз. Також використовується для позначення небажаного (жадібного) або ненасиченого (лінивого) пошуку.
5. Каретка (^) — маркер початку рядка або позиції початку пошуку. Використовується для обмеження пошуку шаблону початком тексту.
6. Долар (\$) — маркер кінця рядка або позиції завершення пошуку.
7. Квадратні дужки ([]) — задають клас символів, тобто набір символів, будь-який з яких допускається в позиції пошуку. У середині дужок можна визначати діапазони, наприклад, [a-z] позначає всі малі літери латиниці.
8. Фігурні дужки ({m,n}) — визначають кількість повторень попереднього символу або групи від **m** до **n** разів. Вони забезпечують точний контроль над числом повторень.
9. Вертикальна риска (|) — виконує логічну операцію «або», дозволяючи поєднувати альтернативні варіанти шаблону.
10. Круглі дужки (()) — використовуються для групування елементів шаблону, а також для створення підмасок, які можуть бути вилучені або повторно використані.

Класи символів([]).

Метасимволи [та] використовуються для визначення класів символів — наборів символів, з яких може бути обраний будь-який для відповідності.

Символи в класі можна перераховувати явно (наприклад, `[abc]` означає, що в цій позиції буде відповідати символ a, b або c), або задавати діапазони, наприклад, `[a-z]` — всі малі літери латиниці.

Варто зауважити, що всередині класу символів більшість метасимволів втрачають своє спеціальне значення. Наприклад, `$` у класі `[akm$]` буде сприйматися як звичайний символ `$`, а не як метасимвол кінця рядка.

Якщо першим символом у класі стоїть каретка `^`, це означає інверсію класу — відповідатиме будь-який символ, який не входить у цей набір. Наприклад, `[^7]` відповідатиме будь-якому символу, крім цифри 7.

Зворотна коса риска (`\`).

Мабуть, найважливіший метасимвол — це зворотна коса риска `\`. Вона служить для двох основних цілей.

1. Екранування спеціальних символів, щоб вони сприймалися буквально. Наприклад, `\[` відповідає символу `[` у тексті, а не початку класу символів.
2. Позначення спеціальних послідовностей (спеціальних класів символів), наприклад:
 - 1) `\d` — будь-яка цифра, еквівалентно `[0-9]`;
 - 2) `\D` — будь-який нецифровий символ, еквівалентно `[^0-9]`;
 - 3) `\s` — пробільний символ (пробіл, табуляція, новий рядок тощо);
 - 4) `\S` — непробільний символ;
 - 5) `\w` — буквено-цифровий символ або підкреслення (еквівалентно до `[a-zA-Z0-9_]`);
 - 6) `\W` — символ, що не є буквено-цифровим.

Ці послідовності можна також використовувати всередині класів символів, наприклад, `[\s,.]` відповідатиме пробілу або комі чи крапці.

Метасимвол крапки (`.`).

Крапка `.` відповідає будь-якому символу, окрім символу нового рядка (за

замовчуванням). Існує режим **re.DOTALL**, який дозволяє крапці збігатися навіть із символом нового рядка.

Повторення символів (квантори).

Регулярні вирази підтримують квантори — оператори, що задають кількість повторень символів або груп:

- 1) ***** — попередній елемент повторюється нуль або більше разів (жадібно);
- 2) **+** — попередній елемент повторюється один або більше разів;
- 3) **?** — попередній елемент може бути присутній або відсутній (0 або 1 раз);
- 4) **{m,n}** — повторення від **m** до **n** разів (включно).

Приведемо приклади використання кванторів:

- 1) **ca*t** відповідатиме **ct, cat, caaaat** тощо;
- 2) **ca+t** відповідатиме **cat, caaat**, але не **ct**;
- 3) **home-?brew** відповідатиме **homebrew** або **home-brew**;
- 4) **a/{1,3}b** відповідатиме **a/b, a//b** і **a///b**, але не **ab** або **a///b**.

Квантори за замовчуванням працюють жадібно, намагаючись захопити якомога більше символів. Якщо пізніші частини шаблону не співпадають, пошук відкочує збіги поступово, зменшуючи кількість повторень. Існує також «лінивий» режим повторення — при додаванні **?** після квантора (наприклад, ***?, +?, {m,n}?**) — який намагається захопити мінімальну кількість символів.

2.3 Обробка PDF-документів у Python за допомогою бібліотеки fitz

У сучасних програмних рішеннях часто виникає потреба у програмній обробці PDF-документів: читання, аналіз вмісту, редагування або перевірка структури документів. Для таких задач у Python існує низка бібліотек, і

однією з найпотужніших є PyMuPDF, що також імпортується як fitz.

Ця бібліотека забезпечує швидкий і зручний доступ до PDF-файлів, дозволяючи не лише витягувати текст і зображення, але й працювати з розміткою, шрифтами, позиціонуванням елементів та структурою сторінки. У контексті нашої системи автоматичної перевірки оформлення дипломних робіт PyMuPDF відіграє ключову роль, оскільки дозволяє зчитувати вміст документів і перевіряти відповідність до вимог ДСТУ.

2.3.1 Загальний опис бібліотеки

Бібліотека PyMuPDF, що імпортується в коді як fitz, є потужним інструментом для програмної взаємодії з PDF-документами в середовищі Python. Вона побудована на основі рушія MuPDF — надзвичайно швидкого та легкого рендера для PDF, XPS, EPUB та інших документів. Завдяки цій основі PyMuPDF поєднує високу продуктивність з широкими можливостями для аналізу та модифікації вмісту документів.

На відміну від більш простих бібліотек, які дозволяють лише послідовне зчитування тексту, PyMuPDF надає доступ до внутрішньої структури документа. Це означає, що з кожної сторінки можна витягувати не просто текст, а й точні координати його розташування, шрифти, розміри, колір, а також інформацію про інші графічні об'єкти — такі як зображення, лінії чи фігури. Така деталізація особливо корисна в задачах, пов'язаних із перевіркою точності оформлення документів, де важливими є не лише самі слова, але й їхнє місце на сторінці.

PyMuPDF дозволяє здійснювати навігацію по сторінках PDF-файлу, аналізувати розмітку кожної сторінки, зчитувати та змінювати метадані, наприклад назву документа, ім'я автора або кількість сторінок. Також

бібліотека підтримує пошук текстових фрагментів із урахуванням регістру, виключенням пробілів або з використанням регулярних виразів. Це відкриває широкі можливості для реалізації автоматизованих перевірок форматування тексту, заголовків, полів, відступів, а також структурних елементів документа, таких як зміст, таблиці, малюнки або списки.

Крім текстового вмісту, бібліотека дозволяє працювати з графічними елементами. Наприклад, можна витягати вбудовані зображення, зчитувати їхню позицію та розміри або навіть замінювати їх на нові. Це дає змогу створювати системи для аналізу візуального контенту, перевірки наявності певних зображень або оцінки їхньої якості та відповідності вимогам.

Завдяки підтримці Unicode (універсальний код символів), PyMuPDF коректно обробляє документи, написані українською мовою, що є критично важливим для задач, пов'язаних із національними стандартами, зокрема ДСТУ. Бібліотека також підтримує виділення, вставку та редагування елементів, хоча ці можливості зазвичай використовуються рідше в контексті аналізу.

Таким чином, PyMuPDF є не просто засобом для “читання PDF”, а повноцінною платформою для побудови інтелектуальних систем аналізу документів. Саме ця гнучкість і функціональність дозволяє використовувати її в нашому програмному рішенні для автоматичної перевірки правильності оформлення дипломних робіт згідно з чинними стандартами.

2.3.2 Структури, текст та об'єкти в бібліотеці fitz

Дана бібліотека надає високопродуктивні засоби для читання, обробки та аналізу документів у форматі PDF. У цьому розділі розглядається її основний функціонал, зокрема структура представлення PDF-документа,

можливості отримання тексту та виявлення вмісту сторінок, таких як зображення, таблиці, графічні елементи тощо.

Структурне представлення PDF-документа.

PDF-документ у бібліотеці `fitz` моделюється як набір об'єктів, кожен з яких відповідає певному рівню структури файлу. На верхньому рівні знаходиться об'єкт `Document`, який представляє увесь документ. `Document`, у свою чергу, складається з послідовності сторінок (`Page`), кожна з яких містить окремі блоки вмісту (`TextBlock`, `Image`, `Shape` тощо). Ці блоки можуть бути як текстовими, так і графічними. Текстові блоки складаються з рядків (`Lines`) і слів (`Spans`, `Words`), а також містять точну інформацію про координати розміщення елементів на сторінці [20].

Кожен із цих об'єктів має точну просторову прив'язку (координати), що дозволяє виконувати геометричний аналіз вмісту — наприклад, перевіряти розміщення тексту відповідно до полів, вирівнювання елементів або пошук таблиць на основі регулярності координат.

Виявлення зображень.

Бібліотека `fitz` дозволяє отримати точну інформацію про графічні елементи на сторінці, зокрема зображення, які були вставлені у файл. Це важливо для перевірки наявності рисунків, схем, діаграм, які відповідно до стандартів оформлення мають бути належним чином підписані та розміщені з дотриманням форматування.

Кожне зображення у PDF-файлі ідентифікується за унікальним посиланням (`xref`) та описується рядом характеристик, включаючи розміри, колір, розміщення тощо [20]. За допомогою цієї інформації можливо:

- 1) підрахувати загальну кількість рисунків;
- 2) оцінити якість вставлених зображень;
- 3) перевірити, чи вони не перекривають текст або поля;
- 4) співставити зображення з підписами.

У випадку, коли PDF містить векторну графіку (наприклад, побудовані діаграми або оформлені таблиці у вигляді ліній), бібліотека також дозволяє аналізувати геометричні об'єкти: відрізки, прямокутники, фігури — і встановлювати їхню позицію та форму.

Виявлення таблиць.

Бібліотека `fitz` не надає прямої підтримки об'єктів типу “таблиця”, оскільки у форматі PDF таблиці не мають спеціальної розмітки — вони можуть бути згенеровані як сітка ліній або як впорядковані текстові блоки. Проте можливо здійснити евристичне, тобто практичне, але не обов'язково найоптимальніше виявлення таблиць, базуючись на таких підходах [21, 22]:

- 1) аналіз повторюваних горизонтальних та вертикальних координат текстових блоків;
- 2) визначення однакових відстаней між рядками і стовпцями;
- 3) наявність графічних елементів (ліній), які формують сітку.

Цей підхід дозволяє з певною точністю ідентифікувати таблиці, навіть якщо вони не мають явно вираженої структури.

Виявлення таблиць, створених як графіка.

У деяких випадках таблиці не є набором текстових блоків, а представлені у вигляді графічної сітки або навіть зображення. Таке може бути у документах, створених у графічних редакторах або експортованих з інших форматів. У цих випадках фреймворк `fitz` дозволяє отримати векторну інформацію — лінії, прямокутники, криві — які утворюють геометричну сітку. Якщо ці лінії утворюють регулярну структуру, їх також можна класифікувати як табличну структуру.

Виявлення таблиць у PDF-документах є складною задачею з кількох причин. Передусім, у форматі PDF таблиці не мають чіткої логічної розмітки — вони зберігаються лише як візуальне представлення, без семантичної інформації про рядки, стовпці чи комірки. Це унеможливорює пряме визначення структури таблиці за допомогою стандартних засобів.

Крім того, таблиці можуть бути реалізовані в різних форматах: як звичайний текст, як графічна сітка або навіть як вставлене зображення. Така різноманітність ускладнює уніфікований підхід до їх виявлення. Додатковою складністю є структура самих таблиць, яка часто включає об'єднані комірки, багаторівневі заголовки, відмінності в ширині стовпців або інші нестандартні елементи оформлення. Усе це суттєво ускладнює автоматичну класифікацію й аналіз таких об'єктів.

2.4 Висновки до розділу 2

У другому розділі було розглянуто теоретичні та технологічні основи, необхідні для подальшого дослідження та реалізації задачі, пов'язаної з обробкою текстових даних. У підрозділі 2.1 проаналізовано фундаментальні поняття теорії формальних мов, включаючи класифікацію граматики за ієрархією Хомського. Особливу увагу приділено регулярним мовам, які є типом 3 у цій ієрархії, а також скінченим автоматам як формальній моделі для їх розпізнавання. Було показано, що регулярні вирази і скінченні автомати описують один і той самий клас мов, що підтверджується теоремою Кліні.

У підрозділі 2.2 розглянуто можливості використання регулярних виразів у мові програмування Python, зокрема через модуль `re`. Наведено основні функції, спеціальні символи та правила побудови шаблонів, що дозволяють ефективно виконувати пошук, перевірку та обробку текстових рядків відповідно до заданих умов.

Підрозділ 2.3 присвячено огляду бібліотеки `fitz` (PyMuPDF), яка дає змогу здійснювати гнучку обробку PDF-документів у Python. Розглянуто структуру PDF-файлів, способи доступу до текстового вмісту, об'єктів та

метаданих документа, що створює підґрунтя для інтеграції регулярних виразів у задачі аналізу PDF-документів.

Таким чином, у цьому розділі було закладено теоретичну та технологічну базу, яка у подальших розділах буде використана для розробки програмного засобу обробки PDF-файлів з використанням регулярних виразів.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

У межах даної дипломної роботи було розроблено концепцію програмної системи, яка автоматизує процес перевірки дипломних робіт на відповідність вимогам ДСТУ. Основна ідея програмної реалізації полягає в створенні системи, яка б могла бути інтегрована в наявний процес перевірки нормоконтролю.

На рисунку 3.1 схематично представлено взаємодію між учасниками процесу: студентом, системою автоматичної перевірки та спеціалістом з нормоконтролю. Студент надсилає свою роботу на автоматичну перевірку, яка дає зворотний зв'язок про виявлені помилки. Після виправлення всіх можливих недоліків студент повторно подає роботу спеціалісту для фінального затвердження або повернення на доопрацювання. Такий підхід дозволить зменшити навантаження на експертів і підвищити якість оформлення робіт завдяки попередній автоматизованій перевірці.

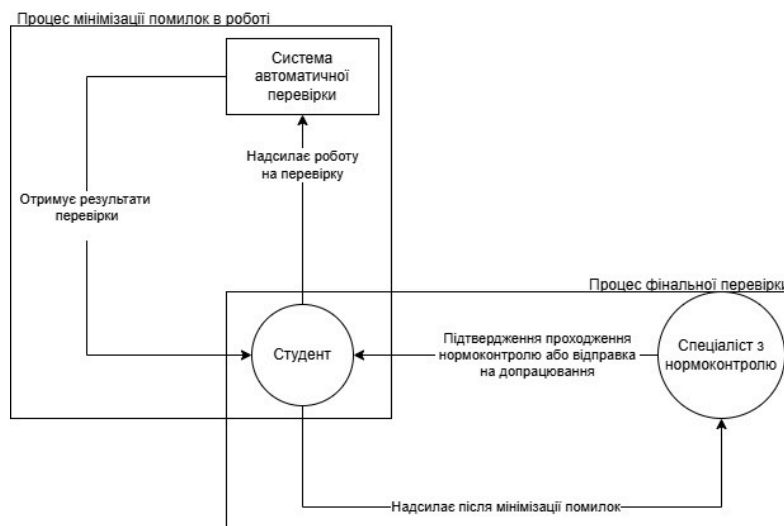


Рисунок 3.1 — Діаграма інтеграції реалізованого продукту

Далі в розділі буде розглянуто основні етапи створення програмного забезпечення, його архітектуру, вибрані технології та особливості реалізації

ключових алгоритмів.

3.1 Визначення основних вимог

В результаті системного аналізу вимог до оформлення бакалаврських робіт згідно ДСТУ [23] та провівши обговорення можливих реалізацій з фахівцями з нормативного контролю, було визначено низку правил які будуть реалізовані. Було з'ясовано основні вимоги, які будуть програмно розглядатися, виявились такі що трапляються найбільш часто при оформленні дипломної роботи студентами (табл. 3.1):

Таблиця 3.1 — Виділені основні вимоги до оформлення

Об'єкт	Вимога до об'єкта
Нумерація сторінок	Угорі праворуч, наскрізна. Титульна сторінка не нумерується.
Шрифт	Times New Roman, 14 пт.
Міжрядковий інтервал	1.5
Абзац	Відступ 1.25 см (5 символів); без відступів перед і після абзацу (0 пт).
Вирівнювання тексту	По ширині.
Відступи	Ліве — 3 см, праве — 1.5 см, верхнє — 2 см, нижнє — 2 см.

Продовження таблиці 3.1.

Об'єкт	Вимога до об'єкта
Нумерація формул	Подвійна (номер розділу.номер формули); нумеруються лише ті, на які є посилання.
Нумерація рисунків	Подвійна (номер розділу.номер рисунка); номер розташовано під рисунком. Обов'язково має бути посилання на рисунок у тексті перед самим рисунком.
Нумерація таблиць	Подвійна (номер розділу.номер таблиці); номер розташовано над таблицею. Посилання в тексті — перед таблицею. Для продовження — “Продовження таблиці” або “Кінець таблиці”.
Заголовки розділів	Великими літерами: РЕФЕРАТ, ABSTRACT, ЗМІСТ, ВСТУП, 1 ..., ВИСНОВКИ, ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ, ДОДАТОК А тощо.
Відступи перед/після заголовків	По 2 порожні рядки (Enter) над і під заголовками. Між заголовками різних рівнів можна менше. Під заголовком — щонайменше 2 рядки тексту.
Ім'я та прізвище	Розміщується під календарним планом, прізвище великими буквами

У межах розробки було здійснено спробу реалізувати перевірку відповідності оформлення бакалаврських робіт саме цим вимогам.

3.2 Обґрунтування вибору мови програмування та програмного забезпечення

Для створення програмного забезпечення системи автоматизованої перевірки бакалаврських робіт було обрано мову програмування Python. Це зумовлено її широким застосуванням у сфері розробки прикладного ПЗ для аналізу даних, обробки текстів та реалізації математичних алгоритмів. Рішення щодо вибору інструментів ґрунтується на низці переваг, які детальніше розглянуто нижче.

Зручність при розробці та читабельність коду.

Python вирізняється простим, інтуїтивно зрозумілим синтаксисом, що значно спрощує процес створення програм і зменшує кількість потенційних помилок. Завдяки цьому розробники можуть зосередитися на реалізації логіки роботи програми, а не на вирішенні проблем, пов'язаних із складністю мови. Крім того, зрозумілий код полегшує підтримку та розвиток проєкту в майбутньому. Особливо насичена стандартна бібліотека, а також розвинена інфраструктура зовнішніх модулів, роблять Python особливо зручним у розробці систем, що працюють із текстами, графами та форматами документів [24].

Наявність спеціалізованих бібліотек.

Одним із важливих чинників вибору Python є великий вибір бібліотек, що дозволяють ефективно реалізувати функціональність системи.

1. NumPy використовується для числових обчислень, що є основою при виконанні структурного аналізу та обробки масивів даних [25].
2. PyMuPDF — основна бібліотека яка забезпечує доступ до внутрішнього вмісту PDF-документів, включаючи текст, шрифти, координати елементів і структуру сторінок, що дає змогу точно аналізувати їхню відповідність вимогам оформлення [22].

3. FastAPI використовується як основа для реалізації веб-інтерфейсу системи. Цей фреймворк дозволяє швидко створювати вебсервіси з підтримкою асинхронного виконання, що забезпечує високу продуктивність та зручність у розгортанні [26].
4. Uvicorn виконує роль сервера для FastAPI-застосунку, обробляючи запити користувачів і забезпечуючи інтеграцію з сучасними протоколами обміну даними [27].

Кросплатформеність.

Програми, розроблені на Python, можуть бути запущені на різних операційних системах без необхідності значних змін у коді, що забезпечує широку доступність та простоту розгортання системи для користувачів з різними технічними умовами.

Середовище розробки.

Для написання та налагодження програмного коду використовувалося інтегроване середовище PyCharm, яке надає зручні інструменти для розробника: підсвічування синтаксису, автодоповнення, зручну навігацію по проєкту, підтримку віртуальних середовищ, а також інтеграцію з системами контролю версій. Завдяки цьому розробка програмного забезпечення була організована ефективно й структуровано.

Таким чином, обране зібрання технологій дозволить реалізувати систему автоматизованого аналізу дипломних робіт із високим ступенем точності, швидкодії та зручності використання.

3.3 Основні виклики програмної реалізації

Однією з ключових проблем під час розробки програмної частини дипломної роботи стала обробка PDF-файлів. Хоча PDF задуманий як

універсальний засіб збереження вигляду документа, різні текстові редактори (такі як Microsoft Word, LibreOffice, Google Docs, LaTeX, WPS Office тощо) застосовують власні алгоритми при експорті файлів у цей формат. У результаті отримані PDF-документи можуть мати суттєві внутрішні відмінності, які не помітні для людини, але критичні при автоматизованій обробці.

Однією з найпоширеніших проблем є те, що текст у PDF може бути розбитий на фрагменти або навіть окремі символи з власними координатами, замість звичних абзаців. Через це складно відновити логічну послідовність тексту. Іноколи текст узагалі перетворюється на зображення або векторну графіку, що унеможлиблює звичайне витягування тексту і може потребувати застосування OCR-технологій (оптичного розпізнавання символів).

Ще одна проблема — кодування символів та шрифтів. Різні редактори можуть по-різному вбудовувати або не вбудовувати шрифти, що призводить до появи символів, які неможливо розпізнати і опрацювати, особливо у випадках з кирилицею, спеціальними знаками або лапками. Навіть при однаковому візуальному вигляді, такий PDF може мати зовсім інше внутрішнє представлення тексту.

Форматування документа також відіграє значну роль. Наприклад, таблиці можуть бути реалізовані або як структуровані об'єкти, або як набір ліній і текстових блоків. Аналогічно, елементи типу заголовків, списків або колонок не завжди експортуються з дотриманням логічної структури, і ця інформація може втрачатися. У PDF відсутня семантична розмітка, тому програма не може автоматично розпізнати, де саме розміщені заголовки, а де основний текст.

Таким чином, навіть при однаковому змісті документів, які конвертовані з різних редакторів, виникає непередбачувана різниця в їхній внутрішній структурі, що суттєво ускладнює створення універсального алгоритму обробки PDF-файлів. Це вимагає розробки адаптивних підходів до

аналізу структури кожного окремого документа або попереднього етапу нормалізації перед основною обробкою.

3.4 Налаштування середовища розробки

Щоб забезпечити автоматизований та ефективний процес розгортання програмного забезпечення, було впроваджено систему безперервної інтеграції та доставки (CI/CD) [28]. Це дало змогу оновлювати сервіс без необхідності ручного втручання, зменшуючи кількість помилок під час релізів.

Для реалізації інфраструктури було використано такі ключові інструменти.

1. GitHub — платформа для керування версіями, де зберігається весь вихідний код проєкту. Репозиторій виконує роль основного джерела правди та слугує тригером для запуску процесів автоматичного оновлення після змін у головній гілці.
2. Kubernetes — система для управління контейнеризованими додатками, яка забезпечує автоматичне масштабування, оновлення та контроль над розгортанням. Сервіс перевірки дипломних робіт було упаковано в контейнери та розгорнуто в кластері Kubernetes, що гарантує стабільну роботу в різних середовищах і спрощує масштабування.
3. ArgoCD — інструмент для реалізації GitOps-підходу в середовищі Kubernetes. Завдяки ArgoCD, стан кластера автоматично синхронізується з вмістом Git-репозиторію, що дозволяє оперативно застосовувати зміни коду та конфігурацій у продуктивному середовищі без додаткових дій з боку розробників.

У результаті впровадження цих рішень вдалося налагодити сучасний DevOps-процес, скоротити час від внесення змін до їх впровадження в систему, а також досягти стабільного й передбачуваного процесу деплоюменту.

3.5 Реалізація веб-застосунку

У рамках розробки веб-застосунку було обрано архітектурний підхід REST API [29], що забезпечує ефективну та гнучку взаємодію між клієнтською та серверною частинами системи. Основою такого підходу є представлення даних у вигляді ресурсів, доступних за унікальними URL-адресами. Кожен ресурс відповідає певній сутності системи, наприклад, користувачам, замовленням чи продуктам, що значно полегшує структурування даних і роботу з ними.

Взаємодія з ресурсами відбувається через стандартні HTTP-методи, які відповідають типу операції: отримання інформації, створення, оновлення або видалення. Завдяки безстановій природі протоколу HTTP кожен запит містить всю необхідну інформацію для обробки, що дозволяє забезпечити масштабованість системи та спрощує підтримку.

Для передачі даних між клієнтом і сервером використовується формат JSON, який є зручним для обробки як на стороні браузера, так і в серверних додатках. Усі URL-адреси спроектовані логічно та відображають ієрархію ресурсів, що сприяє інтуїтивному розумінню структури API і полегшує його використання розробниками.

Застосунок дозволяє автоматично перевіряти відповідність дипломної роботи вимогам оформлення, здійснюючи аналіз PDF-документа користувача.

Для реалізації серверної частини (backend) було використано

фреймворк FastAPI, що є чудовим рішенням для невеликих проектів на Python, що потребують динамічного розширення та оновлення. Так серверна частина опрацьовує PDF-файли, виконує текстовий аналіз, перевірку структури, таблиць, рисунків тощо. Потім після виконаної перевірки повертається логуювання.

Інтерфейс користувача (frontend) створений з використанням HTML, CSS та JavaScript. Він дозволяє завантажити документ, запускати перевірки та переглядати результати. Все це реалізовано з деякою простотою та мінімалістичністю (рис. 3.2).

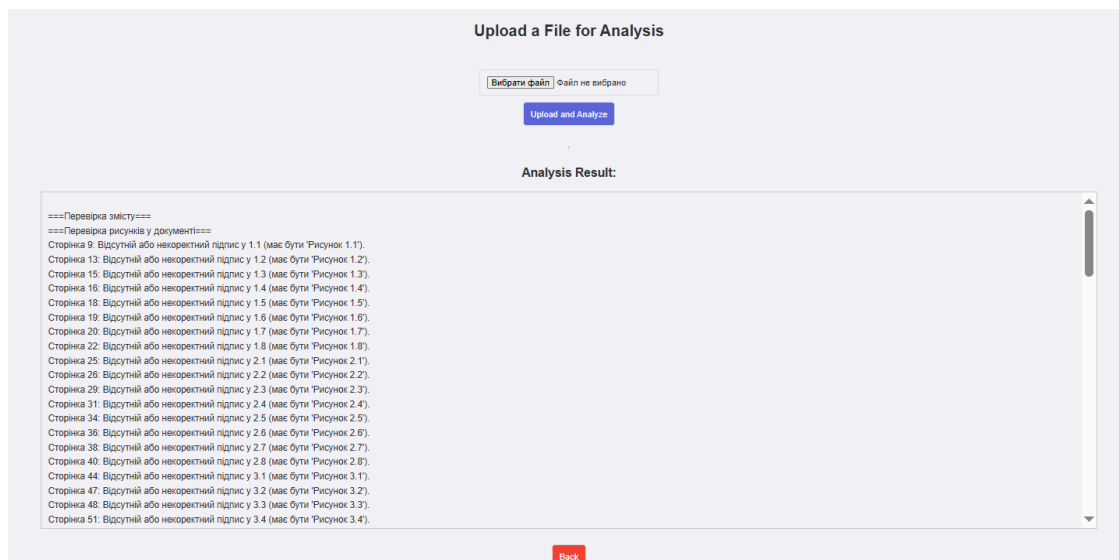


Рисунок 3.2 — Інтерфейс розробленого веб-застосунку

У майбутньому функціональність застосунку може бути розширена за рахунок додавання адміністративної частини, яка забезпечуватиме зберігання перевірених дипломних робіт, формування архіву результатів та керування процесом перевірки. Також можлива розробка окремого інтерфейсу для спеціаліста з нормоконтролю, який матиме змогу переглядати автоматично перевірені документи, залишати коментарі та підтверджувати проходження нормоконтролю або відправляти роботу на доопрацювання. Це дозволить реалізувати повний цикл перевірки (див. розділ 3, рисунок 3.1) — від процесу мінімізації помилок в документі до фінального затвердження викладачем

через адміністративну панель, підвищивши зручність і прозорість усіх етапів процесу.

3.6 Обмеження реалізованого продукту

Основним обмеженням системи є те, що реалізація основних перевірок вимог вже означає, що первинно оформлені вимоги присутні хоча б в мінімальному вигляді. Наприклад однією з початкових умов успішної роботи є правильно оформлений зміст (рис. 3.3), він повинен бути оформлений з крапками між назвами розділів та номерами сторінок, це є обов'язковою вимогою для роботи системи.

ЗМІСТ

ВСТУП.....	8
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1 АКТУАЛЬНІСТЬ ПРОБЛЕМИ.....	9
1.2 ОСОБЛИВОСТІ СТРУКТУРНОЇ ОРГАНІЗАЦІЇ СИСТЕМ РОЙОВОЇ РОБОТОТЕХНІКИ .	10
1.3 ОСОБЛИВОСТІ ПОВЕДІНКОВОЇ ОРГАНІЗАЦІЇ СИСТЕМ РОЙОВОЇ РОБОТОТЕХНІКИ	15
1.3.1 АГРЕГАЦІЯ І КЛАСТЕРИЗАЦІЯ.....	16
1.3.2 САМООРГАНІЗАЦІЯ	17
1.3.3 КООРДИНОВАНИЙ РУХ І КОЛЕКТИВНЕ СПРИЙНЯТТЯ ЯК ЕЛЕМЕНТИ КОЛЕКТИВНОГО ОБСТЕЖЕННЯ	20
Висновки	23
2 МЕТОДИ БАГАТОАГЕНТНОГО МОДЕЛЮВАННЯ.	24

Рисунок 3.3 — Правильно оформлений зміст

Для коректної перевірки робота має бути представлена у форматі PDF і містити зміст, розташований після титульних сторінок. Крім того, для забезпечення якісної перевірки рекомендується уникати вставлення у роботу зображень без попереднього обрізання або з обрізанням, зробленим

текстовим редактором (наприклад, Word може зберігати зайву інформацію, яка ускладнює автоматичний аналіз). При значному відхиленні від стандартів оформлення, наприклад, у частині змісту, може виникнути помилка сервера, що ускладнить проходження перевірки.

Особливу увагу слід приділити формату формул, якщо формули вставляються як зображення, вони автоматично класифікуються як графічні об'єкти, що вважається неправильним оформленням. Формули повинні бути представлені у вигляді текстових формул, а зображення мають містити лише графічний контент. Варто також враховувати, що малюнки з текстом можуть розпізнаватися системою як текст, що може впливати на результати перевірки. Загалом, дотримання цих рекомендацій підвищить якість оформлення роботи і допоможе уникнути технічних помилок під час автоматичної перевірки.

У випадках, коли результати автоматичної перевірки викликають сумніви або не узгоджуються з очікуваннями, робота передається на фінальну оцінку відповідальній особі — викладачу, що дозволяє гарантувати якість та точність оцінювання. Рекомендується перед поданням роботи пройти попередню перевірку автоматизованою системою, щоб мінімізувати кількість виявлених помилок і уникнути затримок.

3.7 Реалізація алгоритмів перевірки

Практична частина дипломної роботи полягає у створенні сервісу автоматичної перевірки дипломної роботи у форматі PDF на відповідність вимогам ДСТУ.

Основним елементом системи є спеціальний програмний клас `DSTUChecker`, який об'єднує в собі послідовність перевірок документа за

різними критеріями.

Система побудована за принципом ООП, тому кожна окрема перевірка реалізована як окремий метод класу, що дозволяє легко розширювати та модифікувати функціональність у майбутньому.

Організація коду.

1. Усі допоміжні функції (наприклад, обробка заголовків, класифікація розділів, витяг тексту тощо) винесено в окремий модуль допоміжних методів.
2. Основний клас виконує роль контролера, який координує окремі етапи перевірки та забезпечує послідовність виконання.

Основні етапи перевірки.

1. Зчитування документа — на першому етапі здійснюється відкриття PDF-файлу та попередня обробка: витяг заголовків розділів, визначення їхньої ієрархії, формування списку основних структурних елементів (вступ, висновки, список джерел тощо).
2. Послідовна перевірка вимог — в основному методі здійснюється виклик низки перевірок, зокрема:
 - перевірка відповідності змісту;
 - контроль правильності розмітки сторінок (нумерація, нова сторінка для розділів);
 - перевірка наявності зображень та таблиць;
 - аналіз відповідності форматування (відступи, шрифт, стиль заголовків);
 - виявлення нумерацій та структурних списків.
3. Логування результатів — після кожної перевірки результати фіксуються у внутрішній системі логів, які згодом можна вивести для перегляду або зберегти у звіт.

3.7.1 Парсинг змісту

На початку, при створенні основного класу перевірки, виконується парсинг змісту (витягування заголовків розділів зі змісту). Основна мета цього етапу — автоматично витягнути структуру документа зі сторінки змісту, щоб надалі здійснювати перевірку відповідності структури документа встановленим вимогам. Це дозволяє надалі використовувати ці дані для перевірки структури: наприклад, чи є в документі розділ “Вступ” частин, або ж для подальшого використання для перевірки заголовків першого рівня і тому подібне. Крім того, це спрощує навігацію в документі — якщо відомо, на якій сторінці починається потрібний розділ, можна швидко перейти до нього. Це економить час, знижує кількість помилок і робить перевірку стандартів більш ефективною.

Сам парсинг змісту документа відбувається наступним чином. Спочатку відкривається PDF-файл, після чого зчитується текст із тих сторінок, де зазвичай знаходиться зміст — приблизно з 5-ї по 15-ту. Далі програма намагається знайти сам заголовок “ЗМІСТ” і починає обробку тексту, який іде після нього.

Оскільки в документах ДСТУ заголовки розділів у змісті подаються у форматі (рис. 3.4, рис 3.5): “Вступ 9”, де є назва розділу, крапки і номер сторінки, — програма використовує спеціальні регулярні вирази, щоб розпізнати цей шаблон. Також враховується те, що деякі рядки можуть бути розбиті на частини через форматування PDF (рис. 3.6), тому рядки автоматично об’єднуються між собою, якщо видно, що частина заголовка пішла на наступний рядок.

ВСТУП..... 9

Рисунок 3.4 — Формат правильно оформленого змісту

1.2 Зв'язок криптовалют з фондовим ринком та акціями	12
1.3 Основні види та методи аналізу фінансових ринків	14

Рисунок 3.5 — Формат правильно оформленого змісту

1.7 Огляд сучасних платформ та інструментів для проведення торгівельних угод	26
--	----

Рисунок 3.6 — Формат правильно оформленого змісту

Після цього з кожного такого рядка програма “витягує” дві речі: текст заголовка (тобто назву розділу) та номер сторінки, на якій він починається. Весь отриманий зміст зберігається у вигляді списку (рис. 3.7), де кожен елемент містить заголовок і номер сторінки. Цей список потім використовується для перевірки структури документа: чи є всі потрібні розділи, чи правильно вони названі, і чи починаються на вказаних сторінках. Таким чином, парсинг дозволяє автоматично побудувати уявлення про структуру документа і надалі працювати з ним як з впорядкованим набором розділів.

```
[
{'title': 'ВСТУП', 'page': 9}
{'title': 'РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ МЕТОДІВ АНАЛІЗУ ФІНАНСОВИХ РИНКІВ ТА ПРОГНОЗУВАННЯ ВАРТОСТІ АКТИВІВ', 'page': 10}
{'title': '1.1 Криптовалюта як фінансовий актив та особливості криптовалютного ринку', 'page': 10}
{'title': '"1.2 Зв'язок криптовалют з фондовим ринком та акціями"', 'page': 12}
{'title': '1.3 Основні види та методи аналізу фінансових ринків', 'page': 14}
{'title': '1.3.1 Технічний аналіз', 'page': 17}
{'title': '1.3.2 Фундаментальний аналіз', 'page': 19}
{'title': '1.4 Методи машинного навчання та можливості їх застосування для інвестиційного аналізу', 'page': 20}
{'title': '1.5 Особливості поєднання методів машинного навчання з технічним аналізом', 'page': 22}
{'title': '1.6 Поняття інвестиційної стратегії, її особливості та методи розробки', 'page': 23}
{'title': '1.7 Огляд сучасних платформ та інструментів для проведення торгівельних угод', 'page': 26}
]
```

Рисунок 3.7 — Частина списку, витягнутого змісту

3.7.2 Класифікація заголовків на рівні

Після того як зі змісту документа були отримані всі назви розділів разом із номерами сторінок, наступним кроком є їхня класифікація за рівнями. Це робиться для того, щоб зрозуміти, які заголовки є головними (наприклад, “ВСТУП”), а які є підрозділами — тобто входять у склад більших розділів.

Уся логіка базується на форматі запису заголовків. Якщо назва розділу написана великими літерами (наприклад, “ВСТУП” або “ДОДАТОК А”), то вона вважається заголовком першого рівня — це найвищий рівень структури документа.

Якщо назва починається з двох чисел, розділених крапкою, наприклад “3.1 Поняття машинного навчання”, то це вже буде другий рівень. Такі заголовки зазвичай є підрозділами основного розділу.

Якщо ж у назві є три числа через крапку, наприклад “3.3.1 Загальні вимоги”, то це ще глибший рівень — третій. Такі заголовки описують ще дрібніші підпункти, які входять у склад другого рівня.

Таким чином, програма переглядає кожну назву і, в залежності від її формату, класифікує її. Така класифікація дозволяє легко будувати “основу” документа (рис. 3.8).

Заголовки 1 рівня (Основні розділи):

- ВСТУП (стор. 9)
- РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ МЕТОДІВ АНАЛІЗУ ФІНАНСОВИХ РИНКІВ ТА ПРОГНОЗУВАННЯ ВАРТОСТІ АКТИВІВ (стор. 10)
- РОЗДІЛ 2. ОГЛЯД МЕТОДІВ МАШИНОГО НАВЧАННЯ ТА ЇХ ПОЄДНАННЯ З ІНДИКАТОРАМИ ТЕХНІЧНОГО АНАЛІЗУ ДЛЯ ОПТИМІЗАЦІЇ ІНВЕСТИЦІЙНОЇ СТРАТЕГІЇ (стор. 30)
- РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ (стор. 58)
- РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ (стор. 89)
- ВИСНОВКИ (стор. 107)
- ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ (стор. 109)
- ДОДАТОК А (стор. 111)
- ДОДАТОК Б (стор. 132)

Заголовки 2 рівня (Підрозділи):

- 1.1 Криптовалюта як фінансовий актив та особливості криптовалютного ринку (стор. 10)
- 1.2 Зв'язок криптовалют з фондовим ринком та акціями (стор. 12)
- 1.3 Основні види та методи аналізу фінансових ринків (стор. 14)
- 1.4 Методи машинного навчання та можливості їх застосування для інвестиційного аналізу (стор. 20)

Рисунок 3.8 — Приклад класифікації заголовків

3.7.3 Перевірка нумерації сторінок

Однією з важливих вимог до оформлення дипломної роботи є правильне розміщення та послідовність нумерації сторінок. На цьому етапі реалізовано автоматичну перевірку відповідності нумерації цим вимогам.

На початку роботи методу до журналу логів додається відповідний заголовок про початок перевірки. Далі відкривається PDF-документ, і запускається покрокова обробка всіх сторінок.

Перша сторінка дипломної роботи, тобто титульна, не повинна містити видимого номера сторінки. Якщо на титульній сторінці виявлено цифрове значення, яке інтерпретується як номер, це фіксується як помилка (рис. 3.9).

===Перевірка нумерації===

Номер присутній на титульній сторінці - Помилка

Рисунок 3.9 — Результат перевірки нумерації сторінок

Для кожної наступної сторінки виконується перевірка:

- 1) чи присутній номер сторінки у вигляді цифри;
- 2) якщо номер відсутній, до логів додається відповідне повідомлення про помилку.

У випадку, якщо номер присутній, додатково виконується перевірка його розташування.

1. Номер має бути розміщений згори сторінки праворуч.
2. Якщо координати текстового блоку, що містить номер, не відповідають допустимим межам (наприклад, якщо номер занадто низько або зміщений ліворуч), це також фіксується як порушення.

3.7.4 Перевірка змісту на обов'язкові заголовки

На першому етапі перевірки здійснюється аналіз змісту документа на наявність обов'язкових розділів, передбачених типовою структурою дипломної роботи.

У логи сервісу вноситься повідомлення про початок перевірки змісту. Це забезпечує зручність аналізу результатів після завершення всіх етапів.

Створюється перелік базових заголовків, які повинні бути присутні в змісті дипломної роботи. До таких заголовків належать, наприклад:

- 1) “ВСТУП”;
- 2) “ВИСНОВКИ”;
- 3) “ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ”.

Ці назви є обов'язковими для більшості академічних робіт, відповідно до вимог ДСТУ. Здійснюється перевірка, чи присутні виявлені заголовки у фактичному змісті документа. Для цього використовується пошук по кожному заголовку, отриманому з попереднього етапу аналізу структури PDF.

За підсумками перевірки формується список відсутніх заголовків. Якщо один або декілька обов'язкових розділів не знайдено у змісті, в лог додається відповідне повідомлення, яке інформує користувача про необхідність перевірки структури документа вручну.

3.7.5 Перевірка відступів

Згідно з вимогами державного стандарту, дипломна робота має відповідати певним параметрам полів сторінки. На цьому етапі реалізовано

автоматичну перевірку дотримання відступів з усіх боків сторінки: зверху, знизу, зліва та справа.

На початку методу додається запис до журналу про початок перевірки. Потім відкривається PDF-документ, і для кожної сторінки виконується послідовна обробка.

Для кожної сторінки визначаються крайні текстові блоки: перший (розташований найвище) та останній (розташований найнижче). На основі їхніх координат обчислюються:

- 1) верхній відступ — повинен бути не менше 2;
- 2) уижній відступ — повинен бути не менше 3.

Ці значення переводяться в сантиметри та порівнюються з нормативами. Якщо відступи менші за допустимі (наприклад, менше 2 см), у лог записується повідомлення про помилку.

Далі для кожного текстового блоку на сторінці визначаються координати лівого та правого країв тексту. На їх основі обчислюються:

- 1) лівий відступ — повинен бути не менше 3 см;
- 2) правий відступ — повинен бути не менше 1,5 см.

Якщо виявлено порушення цих меж, відповідне повідомлення також додається до логів(рис. 3.10).

```
===Перевірка відступів===
```

```
Відступ неправильний на сторінці 1 | Верхній відступ: 1.9846314284222264 см
```

```
Відступ неправильний на сторінці 1 | Нижній відступ: 1.4602643557438408 см
```

```
Відступ неправильний на сторінці 2 | Нижній відступ: 1.3586641595001314 см
```

```
Відступ неправильний на сторінці 7 | Верхній відступ: 1.8761562464498283 см
```

Рисунок 3.10 — Частина результату перевірки на відступи

3.7.6 Перевірка стилю тексту

Важливою вимогою до оформлення дипломної роботи є дотримання єдиного стилю шрифту та встановленого розміру основного тексту. Згідно з ДСТУ, у тексті має використовуватися шрифт Times New Roman, кегль 14 пт. На цьому етапі реалізовано перевірку відповідності вказаним параметрам.

На початку роботи методу до журналу додається заголовок про початок перевірки стилю. Далі відкривається PDF-документ і визначається, з якої сторінки починаються додатки. Для цього аналізується зміст документа: сторінки з додатками виключаються з перевірки, оскільки вони можуть містити нестандартне форматування (схеми, таблиці, ілюстрації тощо).

Для кожної сторінки до розділу додатків аналізується кожен текстовий блок. Метод обробляє всі спани (фрагменти тексту з єдиним стилем), з яких зчитуються:

- 1) назва шрифту;
- 2) розмір шрифту (у пунктах);
- 3) текстове значення.

Порожні рядки, а також елементи, що містять технічні шрифти (наприклад, Cambria, Symbol, Math, LaTeX), автоматично виключаються з аналізу — це дозволяє уникнути помилкових спрацьовувань для формул, символів та математичних виразів. Для кожного текстового елемента перевіряється:

- 1) чи використовується шрифт сімейства Times;
- 2) чи дорівнює розмір шрифту 14 пунктам (з урахуванням допустимого відхилення ± 0.1).

У випадку виявлення невідповідності (наприклад, інший шрифт або шрифт меншого/більшого розміру), до логів додається детальний запис із зазначенням сторінки, тексту, шрифту та розміру (рис. 3.11).

===Перевірка стилю та розміру===

На сторінці 2: Текст: '2' | Стиль: Calibri | Розмір: 11.04

На сторінці 3: Текст: '3' | Стиль: Calibri | Розмір: 11.04

На сторінці 4: Текст: '4' | Стиль: Calibri | Розмір: 11.04

На сторінці 5: Текст: '5' | Стиль: Calibri | Розмір: 11.04

Рисунок 3.11 — Частина результату перевірки на стиль та розмір

3.7.7 Перевірка початку рядка на недопустимі символи

Відповідно до стандартів оформлення, абзаци основного тексту не повинні починатися з графічних або нестандартних символів (наприклад, маркерів списку у вигляді крапок, стрілок тощо), які не передбачені вимогами ДСТУ. Для виявлення таких порушень реалізовано перевірку наявності недопустимих символів на початку рядків.

Метод додає до журналу повідомлення про початок перевірки. Далі відкривається PDF-документ і визначається діапазон сторінок, які потрібно перевірити. Для кожної сторінки у визначеному діапазоні виконується:

- 1) зчитування повного тексту сторінки у вигляді рядків;
- 2) поелементна перевірка кожного рядка на наявність недопустимого символу на початку.

У поточній реалізації перевіряється наявність символів “●”, “?” — з яких рядок починатись не може. Було також реалізовано зі символами “!”, “.”, “,” але у зв'язку з тим що при конвертуванні в PDF формули зберігаються неконсистентно (іноді в різних рядках, іноді потрапляють символи переходу на новий рядок в місцях де цього немає), через це з програмного продукту

було прибрано перевірку на ці символи.

У випадку, якщо рядок починається з недопустимого символу, метод записує у логи відповідне повідомлення з вказанням номера сторінки та проблемного символу (рис. 3.12).

===Перевірка початку рядка===

На сторінці 44: рядок починається з недопустимого символу '●'

На сторінці 44: рядок починається з недопустимого символу '●'

На сторінці 45: рядок починається з недопустимого символу '●'

Рисунок 3.12 — Частина результату перевірки на початок рядка з недопустимих символів

3.7.8 Перевірка наявності прізвища після календарного плану

Згідно з вимогами до оформлення дипломної роботи, після розділу “Календарний план” має бути вказане прізвище та ім’я студента. При цьому прізвище має бути написано великими літерами, що є типовою вимогою до формального заповнення документів.

Для перевірки використовується лише перших 10 сторінок документа — цього зазвичай достатньо для виявлення блоку з календарним планом та підписом студента. Текст усіх цих сторінок об’єднується в один суцільний рядок для зручності пошуку.

За допомогою регулярного виразу виконується пошук шаблону. У разі успішного знаходження імені та прізвища виконується перевірка (рис. 3.13), чи написано прізвище великими літерами. Якщо ні — до журналу додається

повідомлення про порушення.

Якщо шаблон не знайдено, метод записує в логи повідомлення про відсутність прізвища та імені студента після календарного плану, що також є помилкою.

```
===Перевірка сторінок на наявність прізвища після календарного плану===
```

```
Знайдено Ім'я Прізвище студента: Віктор ГОНТАР
```

Рисунок 3.13 — Перевірка на наявність Ім'я та Прізвища після календарного плану

3.7.9 Перевірка рисунків у документі

Дипломна робота повинна містити правильно розміщені, підписані та згадані рисунки. Метод виконує повну перевірку відповідності рисунків формальним вимогам до оформлення.

Метод обмежує перевірку рисунків лише основною частиною роботи — з розділу “Вступ” до “Переліку джерел посилання”. Додатки не перевіряються.

Для кожної сторінки, яка перевіряється, визначається номер розділу за змістом (наприклад: “Розділ 2 Теоретична частина” або “2 Теоретична частина”). Це потрібно для формування правильної очікуваної нумерації рисунків у форматі 3.1, 3.2 тощо.

Перевірка кожного рисунка відбувається наступним чином.

1. Центрування рисунка — обчислюється центр рисунка та перевіряється, чи він знаходиться поблизу центру текстового блоку (допустиме відхилення — 10%).

2. Наявність підпису — нижче зображення має бути підпис виду “Рисунок Х.У”. Пошук виконується серед текстових блоків нижче зображення. Якщо підпис не знайдено — реєструється помилка.
3. Правильність нумерації — для кожної сторінки визначається номер відповідного розділу згідно зі змістом. Вона визначається на основі заздалегідь витягнутого змісту і сформованої структури (див. розділ 3.7.1). Це дозволяє створити очікувану нумерацію таблиць у форматі Х.У, де Х — номер розділу, У — порядковий номер таблиці в межах розділу. А інакше — логується повідомлення про невідповідність.
4. Центрування підпису — перевіряється, чи підпис під рисунком також знаходиться по центру (відносно основного текстового блоку).
5. Положення підпису — підпис має йти одразу після рисунка.
6. Посилання на рисунок — для кожного рисунка перевіряється, чи згадується його номер (Х.У) у тексті перед ним (на цій або попередній сторінці). Для цього витягується весь текст з цих сторінок і використовуючи шаблон регулярного виразу відбувається пошук посилання. Якщо посилання відсутнє — метод реєструє повідомлення про помилку (рис. 3.14).

===Перевірка рисунків у документі===

Сторінка 14: Відсутнє посилання на 1.3 перед рисунком на цій або попередній сторінці

Сторінка 27: Підпис 1.8 не по центру.

Сторінка 37: Відсутнє посилання на 2.4 перед рисунком на цій або попередній сторінці

Сторінка 45: Відсутній або некоректний підпис у 2.10 (має бути 'Рисунок 2.10').

Сторінка 93: Нумерація 4.3 != очікувана 4.2.

Сторінка 94: Нумерація 4.3 != очікувана 4.4.

Сторінка 95: Нумерація 4.3 != очікувана 4.5.

Рисунок 3.14 — Логінг перевірки рисунків

3.7.10 Перевірка таблиць у документі

Таблиці в дипломній роботі мають відповідати певним вимогам до оформлення, розміщення та згадування в тексті. Метод виконує автоматизовану перевірку всіх таблиць у основній частині документа. Перевірка включає в себе наступні етапи.

Обмеження діапазону перевірки. Метод обмежує перевірку сторінками від “Вступу” до “Переліку джерел посилання”. Таблиці в додатках не перевіряються.

Визначення очікуваного розділу. Для кожної сторінки визначається номер відповідного розділу згідно зі змістом. Аналогічна методологія використовується для обчислення очікуваного розділу для рисунків.

Аналіз кожної таблиці. Тоді для кожної з виявлених таблиць перевіряється:

1. Підпис таблиці:
 - має бути розміщений безпосередньо перед таблицею (не нижче неї);
 - має відповідати шаблону “Таблиця (номер розділу.номер таблиці в розділі)”;
 - перевіряється, чи підпис є відцентрований (відхилення не більше 10% від центру текстового блоку);
 - якщо підпис знаходиться не одразу перед таблицею — реєструється повідомлення.
2. Якщо знайдена нумерація не збігається з очікуваною — реєструється повідомлення про помилку.
3. Посилання в тексті:
 - перевіряється, чи згадується таблиця з відповідною нумерацією у тексті на цій або попередній сторінці (аналогічно);

— якщо посилання відсутнє — додається повідомлення про помилку.

Продовження таблиці. У випадках, коли таблиця завершується близько до нижнього краю сторінки (< 20% сторінки знизу), а також якщо на наступній сторінці з самого верху розташована таблиця (< 10% сторінки зверху) без підпису:

- 1) виконується пошук підпису виду “Продовження таблиці Х.У”;
- 2) якщо підпис відсутній — формується логінг про цю помилку;
- 3) такі продовження пропускаються для подальших перевірок (нумерація, центр, посилання тощо).

Результат перевірки може мати наступний вигляд (рис. 3.15):

===Перевірка таблиць у документі===

Сторінка 1: Відсутній або некоректний підпис у 1.1 (має починатися з 'Таблиця 1.1').

Сторінка 3: Відсутній підпис 'Продовження таблиці' перед продовженням 1.1.

Сторінка 3: Відсутній або некоректний підпис у 1.2 (має починатися з 'Таблиця 1.2').

Сторінка 54: Підпис 3.1 не знаходиться перед таблицею.

Рисунок 3.15 — Логінг перевірки таблиць

3.7.11 Перевірка формул

Аналізуючи природу формул і їх представлення в дипломних роботах за було виділено дві початкові підходи.

Алгоритмічно-аналітичний підхід.

Спочатку ідентифікувати формулу алгоритмічно використовуючи стиль тексту формул. А саме перевіряти весь текст і знаходити деяку послідовність, наприклад від 5 послідовних символів одного стилю, який відповідає

формату формул, а точніше сімей таких шрифтів. Також потрібно перевірити. Серед таких були вибрані ті що зустрічаються в дипломах минулих років, наприклад такі сім'ї шрифтів:

- 1) Math;
- 2) Latex;
- 3) Symbol;
- 4) Cambria.

Далі після ідентифікації потрібно провести перевірку вимог оформлення формул. Тому далі потрібно зробити визначення очікуваної нумерації, таким самим методом як для рисунків та таблиць. Але оскільки згідно ДСТУ потрібно нумерувати лише ті формули, на які є посилання, то наступним етапом буде перевірити чи згадується номер формули на цій або попередній сторінці (для цього можна використати підхід для знаходження посилання на рисунки та таблиці, які описані вище).

В результаті реалізації цього підходу було виявлено, що збереження формул в PDF може бути неконсистентним, тобто таким що унеможливорює створення загального правила за яким можна їх ідентифікувати.

Алгоритмічно-аналітичний підхід з використанням зображень.

Ідея цього підходу базується на ідентифікації формули на основі зображення. В основі цього підходу лежить створення моделі машинного навчання, яка буде детектувати формулу на основі зображення, для цього потрібно зібрати та розмітити зображення з формулами або ж використати готовий датасет з Kaggle тощо.

Далі кожна сторінка конвертується в зображення, ідентифікується формула та знаходяться її координати на сторінці. Наступним важливим етапом є відсіювання тих формул які знаходяться в тексті і не повинні бути оформленні як формули згідно ДСТУ. В цьому випадку потрібно алгоритмічно відібрати ті формули які знаходяться. Наступні кроки перевірки будуть аналогічні до кроків перевірки оформлення попереднього підходу.

Такий підхід дозволяє уникнути проблем з гнучкістю, тобто не прив'язується до структури PDF файлів. Проте в ході реалізації було виявлено, що наявної кількості розмічених даних для побудови моделі розпізнавання формул недостатня, що виражається в неприйнятній кількості неідентифікованих формул. Також такий метод ідентифікації потребує більшої кількості ресурсів відносно алгоритмічно-аналітичного підходу. Саме через це такий варіант був відкинтий.

3.7.12 Перевірка розміщення заголовків першого рівня

У відповідності до вимог оформлення дипломних робіт, заголовки першого рівня мають розпочинати сторінку і розміщуватись не нижче другого рядка зверху. Створений метод виконує автоматизовану перевірку цього правила.

Основні етапи перевірки.

1. Збір заголовків першого рівня — використовуються заздалегідь визначені заголовки першого рівня, які були витягнуті та класифіковані (див. розділ 3.7.1, 3.7.2), що містять назву заголовка та номер сторінки, на якій він має бути розміщений.
2. Аналіз тексту перших рядків сторінки:
 - для кожного заголовка відкривається відповідна сторінка PDF-документа;
 - зчитується текст сторінки та розбивається на окремі рядки;
 - перевіряються перші 4 рядки сторінки на наявність відповідності початку заголовка (порівнюється лише перші 20 символів).

3. Нормалізація тексту — видаляються приховані керівні символи (Unicode category Cf) як із заголовка, так і з тексту сторінки — це забезпечує стабільне порівняння при наявності непомітних символів, які можуть з'являтися при конвертації в PDF з різних едіторів.
4. Повідомлення про помилки — якщо заголовок не знайдено серед перших 4 рядків сторінки, виводиться повідомлення:
 - або заголовок має зміщення вниз більш ніж на 2 рядки;
 - або заголовок взагалі не знайдено на сторінці.

3.7 Висновки до розділу 3

У практичній частині дипломної роботи було розроблено програмний веб-застосунок, призначений для автоматизації перевірки текстових документів, зокрема бакалаврських дипломних робіт, за низкою ключових критеріїв оформлення відповідно до ДСТУ. Налаштування середовища розробки дозволило забезпечити ефективний перехід від проектування до реалізації, а також створити умови для подальшої підтримки та оновлення застосунку.

Було спроектовано чітку структуру системи, що складається із серверної частини на базі FastAPI та зручного веб-інтерфейсу, який дозволяє користувачам комфортно працювати з файлами документів.

Основна увага приділялась розробці алгоритмів, які виконують автоматичну перевірку правильності нумерації сторінок і розділів, відповідності заголовків змісту фактичним заголовкам у тексті, а також контролюють форматування відступів, шрифтів і стилів. Крім того, система визначає наявність недопустимих символів на початку рядків, перевіряє наявність підпису після календарного плану, а також правильність

розміщення та оформлення рисунків і таблиць. Для зручності роботи з документами були реалізовані допоміжні функції парсингу змісту та класифікації заголовків за рівнями.

Крім того, були розроблені допоміжні функції, такі як парсинг змісту документа та класифікація заголовків за рівнями, що значно спрощує подальшу роботу з документом і підвищує зручність використання системи.

Слід зазначити, що більшість необхідних алгоритмів перевірки було реалізовано. Однак поки що не було впроваджено автоматизовану перевірку таких параметрів, як міжрядковий інтервал, абзаци та форматування формул, що може бути предметом подальшого розвитку застосунку. В цілому можна сказати що було реалізовано близько 70% запланованих алгоритмів.

Реалізована система має знизити обсяг ручної роботи, яку доводиться виконувати фахівцям з нормоконтролю при перевірці дипломних робіт, підвищує точність і відповідність оформлення встановленим вимогам, а отже підтверджує доцільність використання автоматизованих засобів для аналізу структурованих документів у науковій та освітній сферах.

У подальшому система може бути розширена шляхом інтеграції методів комп'ютерного зору для більш точної ідентифікації таблиць, рисунків або формул, а також застосування великих мовних моделей (LLM), що дозволить реалізувати перевірки, які складно формалізуються за допомогою класичних алгоритмів. Зокрема, це стосується оцінки коректності логічних переліків, структури тексту, стилістики, узгодженості заголовків із вмістом, оформлення переліку посилання, парсинг змісту тощо. Додатково перспективними напрямками удосконалення є впровадження автоматичної перевірки на плагіат та перевірки чи був згенерований текст генеративними мережами, що може стати невіддільною частиною повного циклу контролю якості дипломних робіт.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі система автоматичної перевірки дипломних робіт на відповідність вимогам ДСТУ розглядається як програмний продукт, що включає не лише технічну реалізацію, а й економічне обґрунтування доцільності її впровадження. Така система може бути використана у вищих навчальних закладах, електронних системах документообігу або як окремий модуль в комплексах управління освітнім процесом.

Програмний продукт дозволяє суттєво скоротити витрати часу викладачів та адміністраторів за рахунок автоматизації перевірки структури, форматування та відповідності нормативним вимогам. Результати дослідження можуть бути використані для створення масштабованої, гнучкої та ефективної системи оцінки оформлення студентських робіт.

Для оцінки ефективності різних варіантів реалізації використовується функціонально-вартісний аналіз (ФВА) — інструмент, що дозволяє порівняти альтернативні рішення з погляду витрат і функціональності. Застосування ФВА дає змогу виявити найбільш доцільні технічні підходи, які забезпечують оптимальний баланс між якістю результату та витратами на розробку і впровадження.

4.1 Постановка задачі проектування

Використання методу функціонально-вартісного аналізу (ФВА) дозволило провести комплексну оцінку техніко-економічних аспектів розробки програмного продукту для автоматичної перевірки дипломних робіт

на відповідність вимогам ДСТУ. Усі рішення приймаються з урахуванням взаємозв'язку між окремими підсистемами, що впливають на загальну ефективність роботи системи. З цієї причини аналіз охоплює детальне вивчення функціональних можливостей програмного забезпечення, призначеного для обробки текстових документів у форматі PDF та перевірки їх відповідності встановленим стандартам.

Визначені такі технічні вимоги до програмного продукту:

- 1) забезпечення точного виявлення структурних та форматних невідповідностей у дипломних роботах відповідно до вимог ДСТУ;
- 2) швидка обробка документів, з мінімальною затримкою при аналізі великого обсягу тексту;
- 3) інтуїтивно зрозумілий та зручний інтерфейс для кінцевого користувача (викладача або студента);
- 4) економічна доцільність розробки системи — мінімальні витрати на впровадження, супровід та масштабування.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 полягає у створенні програмного продукту — система автоматизованої перевірки оформлення дипломних робіт на відповідність вимогам ДСТУ, що включає виявлення порушень структури, оформлення, послідовності розділів, оформлення посилань тощо. Виходячи з неї, виділяються такі ключові функції.

1. F_1 — Вибір бібліотеки обробки PDF-документів:
 - (А) використання бібліотеки PyMuPDF (fitz);
 - (В) використання бібліотеки PDFMiner;
 - (С) використання бібліотеки pdfplumber.

2. F2 — Вибір підходу реалізації логіки перевірки оформлення:
 - (A) алгоритмічно-аналітичний підхід (на основі правил, шаблонів та регулярних виразів);
 - (B) використання великих мовних моделей (LLM).
3. F3 — Вибір типу користувацького інтерфейсу:
 - (A) веб-інтерфейс на FastAPI;
 - (B) локальний застосунок на PyQt.

Варіанти реалізації кожної функції узагальнюються в морфологічній карті системи (рис. 4.1), що дозволяє порівняти їх за критеріями ефективності, вартості та складності реалізації.

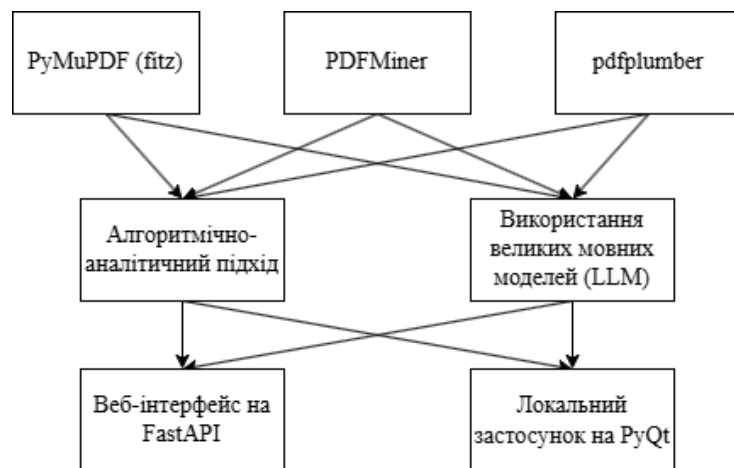


Рисунок 4.1 — Морфологічна карта

За допомогою морфологічної карти побудуємо позитивно-негативну матрицю з відображенням усіх варіантів основних функцій (табл. 4.1).

Таблиця 4.1 — Позитивно-негативна матриця

Функції	Варіанти	Переваги	Недоліки
F_1	A (PyMuPDF)	Висока швидкість роботи, підтримка зображень та тексту, зручний API	Обмежена гнучкість при роботі з табличними структурами

Продовження таблиці 4.1

F_1	В (PDFMiner)	Точна текстова розмітка, можливість отримання шрифтових метаданих	Складний API, нижча швидкість обробки
	С (pdfplumber)	Зручна робота з таблицями, гнучкий доступ до структурованого тексту	Повільніша обробка великих PDF, менша стабільність
F_2	А (Алгоритмічно-аналітичний підхід)	Простота реалізації, чітка інтерпретація результатів	Обмеженість у розумінні контексту, складність масштабування
	В (Великі мовні моделі)	Висока гнучкість, здатність працювати з неоднозначними або погано структурованими текстами	Високі обчислювальні витрати, залежність від зовнішніх API
F_3	А (Веб-інтерфейс на FastAPI)	Доступність із будь-якого пристрою, простота розгортання, масштабованість	Потребує хостингу
	В (Локальний застосунок на PyQt)	Автономність, краща інтеграція з ОС, не потребує інтернету	Обмежена доступність, складність оновлення та масштабування

Згідно з проведеним аналізом, окремі варіанти реалізації не повною мірою відповідають вимогам до точності, зручності використання та економічної ефективності.

Для функції F1 (обробка PDF) оптимальним вибором є PyMuPDF (A), оскільки бібліотека має високу швидкість роботи, підтримує як текст, так і графіку, та має простий і добре документований API. Інші варіанти (PDFMiner і pdfplumber) мають специфічні переваги, але поступаються в універсальності та продуктивності.

Щодо F2 (підхід до реалізації логіки перевірки), алгоритмічний підхід (A) дозволяє швидко створити мінімальний робочий продукт з чіткою логікою та передбачуваними результатами. Водночас використання LLM (B) забезпечує більшу гнучкість і здатність працювати з неструктурованими випадками, але потребує велику кількість ресурсів і може працювати нестабільно в залежності від промпту.

Для функції F3 (інтерфейс користувача) кращим вибором є веб-інтерфейс на FastAPI (A), який забезпечує масштабованість, легке оновлення та доступ із будь-якого пристрою. Локальний застосунок на PyQt (B) може бути резервним варіантом для офлайн-сценаріїв, але складніший в обслуговуванні.

Оптимальною комбінацією реалізації з огляду на баланс між точністю, простотою реалізації, витратами та масштабованістю є:

$$F_1^A — F_2^A — F_3^A.$$

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для оцінки технічного рівня реалізації програмного продукту.

Для характеристики розроблюваної системи автоматизованої перевірки дипломних робіт пропонується використовувати такі параметри.

1. X1 — Потенційний об'єм програмного коду (оцінює складність підтримки та масштабування).
2. X2 — Швидкодія перевірки (середній час, необхідний для аналізу одного документа).
3. X3 — Потенційний час на реалізацію (гіпотетична тривалість розробки мінімально робочої версії системи).
4. X4 — Споживання оперативної пам'яті (середній обсяг RAM, який займає програма під час виконання).

Гірші, середні і кращі значення параметрів обрано з урахуванням вимог до системи, обмежень інфраструктури та потреб замовника (табл. 4.2).

Таблиця 4.2 — Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Гірші	Середні	Кращі
Потенційний об'єм програмного коду	X1	кількість строк	5000	3000	1500
Швидкодія перевірки одного документа	X2	секунди	100	40	10

Продовження таблиці 4.2

Потенційний час розробки мінімального робочого продукту	X3	неділі	20	8	4
Споживання оперативної пам'яті	X4	МБ	600	400	100

Ці параметри дозволяють наочно порівняти ефективність різних варіантів реалізації, що розглядаються у процесі розробки системи автоматизованої перевірки дипломних робіт. Графічне відображення даних з таблиці (рис. 4.2 – рис. 4.5) сприятиме прийняттю обґрунтованого рішення щодо вибору оптимальних технологій, інструментів та архітектури програмного продукту.

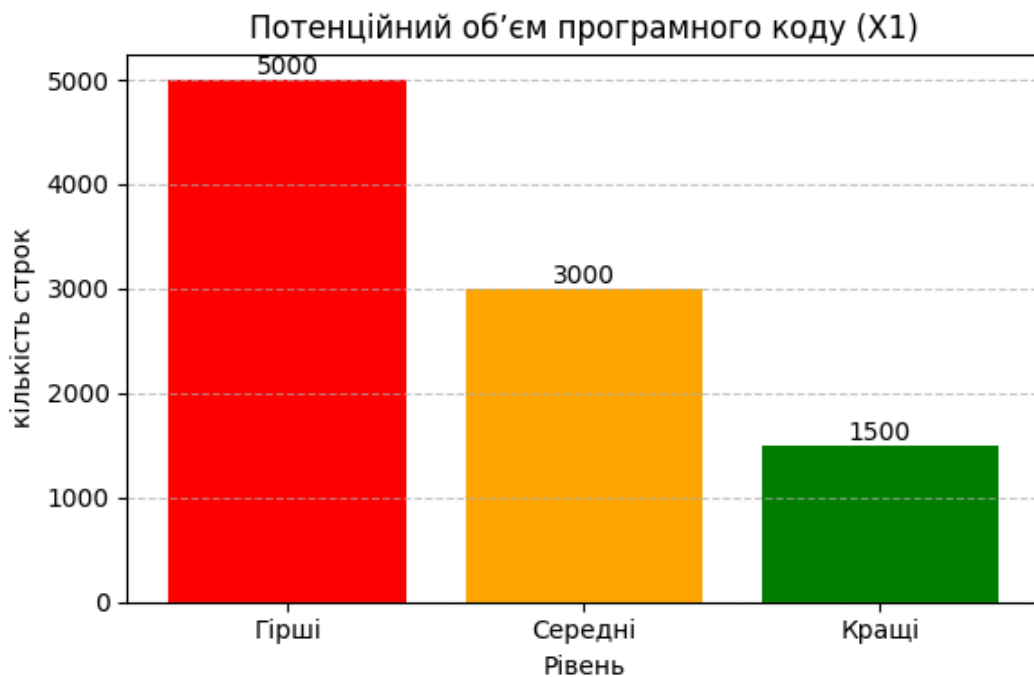


Рисунок 4.2 — X1, Потенційний об'єм програмного коду.

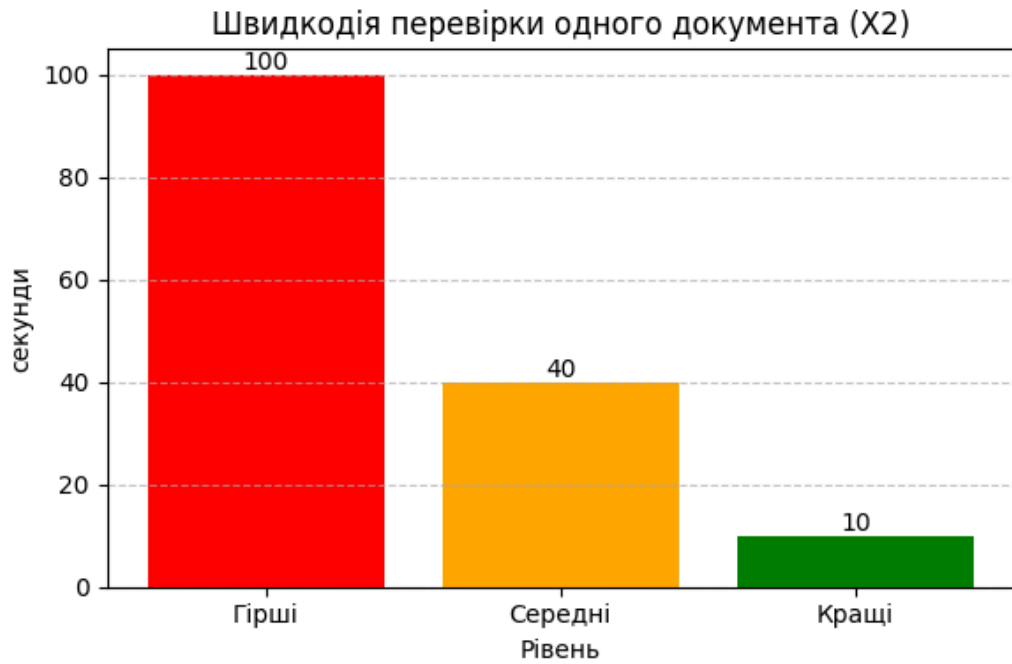


Рисунок 4.3 — X2, Швидкодія перевірки одного документа

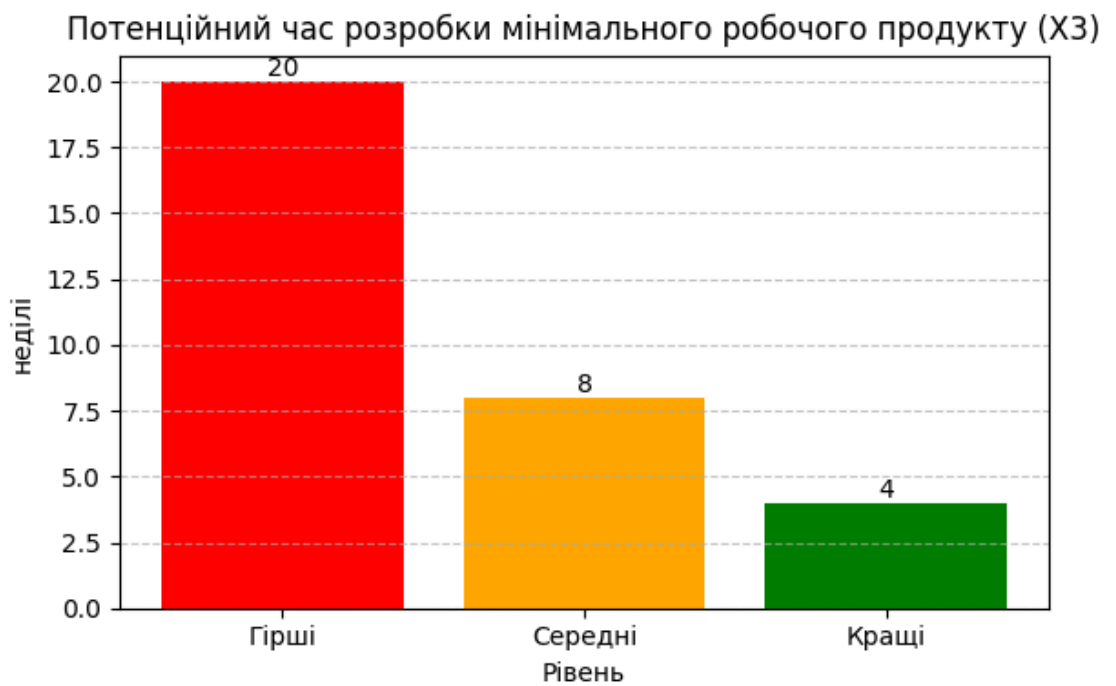


Рисунок 4.4 — X3, Потенційний час розробки мінімального робочого продукту

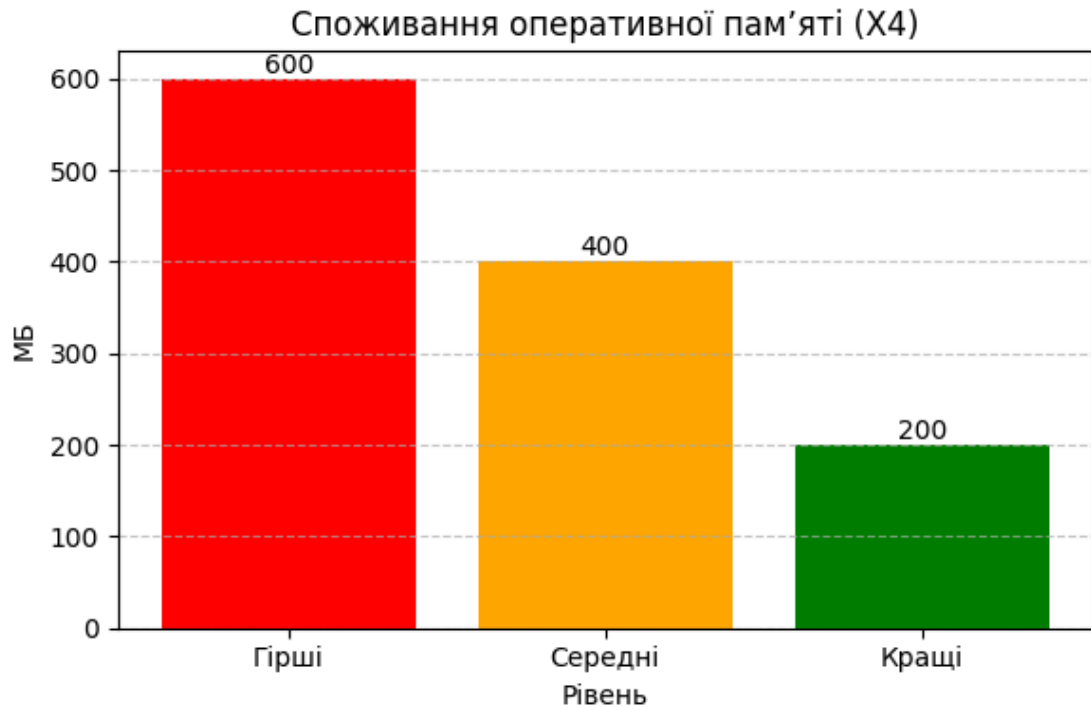


Рисунок 4.5 — X4, Споживання оперативної пам'яті

4.4 Аналіз експертного оцінювання параметрів

Після ґрунтовного аналізу та всебічного обговорення кожен з експертів надав індивідуальну оцінку значущості ключових параметрів у контексті поставленої мети — розробки програмного продукту, який забезпечує максимальну точність у визначенні параметрів моделей адаптивного прогнозування та розрахунку прогнозованих значень.

Визначення вагових коефіцієнтів параметрів здійснювалося методом попарного порівняння, проведеним групою з чотирьох експертів. Процедура включала такі етапи:

- 1) ранжування параметрів з присвоєнням їм відповідного рівня значущості;

- 2) оцінювання обґрунтованості експертних висновків для можливості подальшого застосування;
- 3) порівняння кожної пари параметрів за ступенем їх важливості;
- 4) проведення узагальненого аналізу та визначення підсумкових вагових коефіцієнтів.

Підсумкові результати рангової оцінки, отримані в результаті експертного опитування, наведено в таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів.

Пара метр	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R _i	Відхил ення Δ _i	Δ _i ²
			1	2	3	4	5	6	7			
X1	Потенційний об'єм програмного коду	кількість строк	3	1	2	1	1	1	1	10	-7,5	56,25
X2	Швидкодія перевірки одного документа	секунди	4	4	4	3	4	4	4	27	9,5	90,25
X3	Потенційний час розробки	неділі	2	2	1	2	2	2	2	13	-4,5	20,25
X4	Споживання оперативної пам'яті	МБ	1	3	3	4	3	3	3	20	2,5	6,25

Обчислимо сумарний ранг кожного параметра і загальний ранг усіх параметрів.

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N = 7 – кількість експертів;

n = 4 – число параметрів.

Обчислимо середня сума рангів кожного параметра і загальний ранг усіх параметрів.

$$T = \frac{1}{n} R_{ij} = 17.5 \quad (4.2)$$

Відхилення рангу кожного параметра від середнього, обчислюється за формулою (4.3).

$$\Delta_i = R_i - T \quad (4.3)$$

Обчислимо загальну суму квадратів відхилень.

$$S = \sum_{i=1}^N \Delta_i^2 = 173 \quad (4.4)$$

Обчислимо коефіцієнт узгодженості.

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 173}{7^2(4^3-4)} = 0,7061 > W_k = 0,67 \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Для подальшого аналізу проведено попарне порівняння параметрів, результати якого наведено у таблиці 4.4.

Таблиця 4.4 – Результати ранжування параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0.5
X1 і X3	>	<	>	<	<	<	<	<	0.5
X1 і X4	>	<	<	<	<	<	<	<	0.5
X2 і X3	>	>	>	>	>	>	>	>	1.5
X2 і X4	>	>	>	<	>	>	>	>	1.5
X3 і X4	>	<	<	<	<	<	>	<	0.5

Ступінь переваги і-го параметра над j-тим визначається числовим коефіцієнтом a_{ij} , який обчислюється за наступною формулою.

$$a_{ij} = \{1.5 \text{ при } X_i > X_j, 1.0 \text{ при } X_i = X_j, 0.5 \text{ при } X_i < X_j\} \quad (4.6)$$

З отриманих числових значень формується матриця переваг $A = \|a_{ij}\|$.

Вагомість параметрів K_{bi} обчислюється ітеративно за формулами нижче.

$$K_{bi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^N a_{ij} \quad (4.8)$$

Для точнішого визначення ваг параметрів виконується кілька ітерацій розрахунку (табл. 4.5).

$$K'_{bi} = \frac{b'_i}{\sum_{i=1}^n b'_i} \quad (4.9)$$

$$b'_i = \sum_{i=1}^N a_{ij} b'_j \quad (4.10)$$

Після другої ітерації різниця між ваговими коефіцієнтами стала меншою за 9%, тому додаткові ітерації не проводилися.

Таблиця 4.5 – Розрахунок вагомості параметрів.

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	b_i	K_{bi}	b_i	K_{bi}
X1	1	0,5	0,5	0,5	2,5	0,15625	6,25	0,09
X2	1,5	1	1,5	1,5	5,5	0,34375	30,25	0,44
X3	1,5	0,5	1	0,5	3,5	0,21875	12,25	0,18
X4	1,5	0,5	1,5	1	4,5	0,28125	20,25	0,29
					16	1	69	1

Отримані вагові коефіцієнти чітко відображають пріоритетність ключових параметрів у розробці програмного продукту. Найвищу вагу має швидкодія перевірки одного документа (X2), що свідчить про критичну

важливість оптимізації продуктивності під час автоматизованої перевірки дипломних робіт. Це узгоджується з вимогами до практичної ефективності системи.

Важливими залишаються також споживання оперативної пам'яті (X4), що впливає на придатність рішення для розгортання на обмежених за ресурсами пристроях, та потенційний час розробки MVP (X3), як фактор впровадження в стислий термін. Найменший пріоритет отримав потенційний об'єм програмного коду (X1), адже його збільшення компенсується виграшем в точності та швидкості системи.

4.5 Аналіз рівня якості реалізації основних функціональних компонентів системи

Для забезпечення ефективного функціонування програмного продукту було проведено аналіз якості реалізації основних параметрів системи. Оцінювання виконувалося для кожного параметра окремо з урахуванням їх вагомості, визначеної на попередньому етапі.

Абсолютні значення параметрів X1 (потенційний об'єм програмного коду), X2 (швидкодія перевірки одного документа) та X4 (споживання оперативної пам'яті) відповідають технічним вимогам, встановленим для умов експлуатації продукту. Параметр X3 (потенційний час розробки мінімального робочого продукту) також перебуває у допустимих межах і вибраний з урахуванням компромісу між швидкістю впровадження та складністю реалізації.

Для кожного варіанту реалізації функції обчислюється коефіцієнт технічного рівня за формулою (4.11).

$$K_K(j) = \sum_{i=1}^n K_{vi,j} B_{i,j}, \quad (4.11)$$

де n – кількість врахованих параметрів;

K_{vi} – ваговий коефіцієнт i -го параметра;

B_i – балова оцінка параметра для i -го варіанту реалізації.

Розрахунок проведено для варіантів реалізації функцій системи згідно з таблицею 4.6.

Таблиця 4.6 – Розрахунок показників рівня якості реалізації функціональних компонентів.

Основні функції	Варіант реалізації	Параметр	Абсолютне значення параметра	Бальна оцінка	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	1200	35	0,09	3,15
F2	A	X2	20	25	0,44	11
		X4	200	25	0,29	7,25
F3	A	X3	6	28	0,18	5,04

Сумарні коефіцієнти технічного рівня для обраних варіантів розраховуються за наступною формулою.

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}] \quad (4.12)$$

Визначаємо рівень для кожного з варіантів.

$$1. \quad K_{K1} = 3,15 + 11 + 5,04 = 19,19.$$

$$2. \quad K_{K2} = 3,15 + 7,25 + 5,04 = 15,44.$$

Таким чином, перевагу слід надати першому варіанту, який забезпечує вищий рівень технічної реалізації функціональних компонентів.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки програмного продукту проведемо розрахунок трудомісткості та витрат, пов'язаних з реалізацією двох основних завдань.

1. Розробка проекту системи перевірки оформлення.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи Б, завдання 2 до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Загальна трудомісткість розробки визначається за наступною формулою.

$$T_0 = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М} \quad (4.13)$$

де T_P – базова трудомісткість (люд.-днів);

K_{Π} – коефіцієнт, що враховує тип вхідної інформації;

$K_{СК}$ – коефіцієнт складності контролю вхідної та вихідної інформації;

K_M – коефіцієнт рівня мови програмування (не враховується, якщо значення 1);

$K_{СТ}$ – коефіцієнт використання стандартних модулів;

$K_{СТ.М}$ – коефіцієнт використання стандартного математичного забезпечення (не враховується, якщо значення 1).

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни Б та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 64$ людино-днів. Поправочний коефіцієнт для

першого завдання: $K_{\Pi} = 2.02$, і для другого: $K_{\Pi} = 0.81$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.6$. Тоді загальна трудомісткість програмування є наступними.

Для завдання 1:

$$T_1 = 64 \cdot 2.02 \cdot 1 \cdot 0.6 = 77.568 \text{ людино-днів.}$$

Для завдання 2:

$$T_2 = 27 \cdot 0.81 \cdot 1 \cdot 1 = 21.87 \text{ людино-днів.}$$

Загальна трудомісткість у люд.-годинах:

$$T_0 = (77.568 + 21.87) \cdot 8 = 795,48 \text{ людино-годин.}$$

В розробці бере участь один програміст з окладом 30000 грн. Визначаємо середню зарплату за годину за формулою:

$$СЧ = \frac{M}{T_m \cdot t} \text{ грн.,} \quad (4.14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_{ч} = \frac{30000}{21.8} = 178,57 \text{ грн.}$$

(4.15)

Тоді, розрахуємо заробітну плату за наступною формулою.

$$С_{зп} = С_{ч} \cdot T_i \cdot K_{д} = 178,57 \cdot 795,48 \cdot 1.125 = 159804.972, \quad (4.16)$$

де $С_{ч}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_{д}$ – норматив, який враховує додаткову заробітну плату.

Відрахування на соціальний внесок становить 22%.

$$С_{від} = С_{зп} \cdot 0.22 = 159804.972 \cdot 0.22 = 35157.094 \text{ грн.} \quad (4.17)$$

Припустимо, що програміст працює 7 годин на день та в середньому в місяці 21 робочий день, тоді середня місячна зарплата розраховується за формулою нижче.

$$M = C_{\text{ч}} \cdot T_m \cdot t = 178,57 \cdot 21 \cdot 8 = 29999.76, \quad (4.18)$$

де T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

Машинний час обслуговує один програміст із коефіцієнтом зайнятості $K_3 = 0.6$. Обчислимо річну зарплату.

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 29999,76 \cdot 0.6 = 215998.272 \text{ грн.} \quad (4.19)$$

Тепер обчислимо з урахуванням додаткової заробітної плати.

$$C_{\text{зп}} = C_{\Gamma} \cdot (1 + K_3) = 215998.272 \cdot (1 + 0.6) = 345597.235 \text{ грн.} \quad (4.20)$$

І тепер з відрахуванням на соціальний внесок.

$$C_{\text{від}} = C_{\text{зп}} \cdot 0.22 = 345597.235 \cdot 0.22 = 76031.391 \text{ грн.} \quad (4.21)$$

Обладненням для роботи є Acer Aspire 5. Придбаний він був у травні 2024-го року. Його вартість становила 700\$. Станом на зараз його ціна з урахуванням курсу НБУ (41,5285грн за долар США) становить 29069.95грн.

Розрахуємо амортизаційні витрати при амортизації 25%.

$$C_A = K_{\text{тм}} \cdot K_A \cdot C_{\text{пп}} = 1.02 \cdot 0.25 \cdot 29069.95 = 7412.837 \text{ грн.,} \quad (4.22)$$

де $K_{\text{тм}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$C_{\text{пп}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику обчислюються за наступною формулою.

$$C_P = K_{\text{тм}} \cdot C_{\text{пп}} \cdot K_P = 1.02 \cdot 29069.95 \cdot 0.03 = 889.54 \text{ грн.,} \quad (4.23)$$

де K_P – відсоток витрат на поточні ремонти.

Обчислимо ефективний річний фонд часу ПК.

$$T_{\text{ЕФ}} = (D_{\text{К}} - D_{\text{В}} - D_{\text{С}} - D_{\text{Р}}) \cdot t_3 \cdot K_{\text{В}} = (365 - 105 - 21 - 5) \cdot 8 \cdot 0.875 =$$

$$= 1638 \text{ годин,} \quad (4.24)$$

де $D_{\text{К}}$ – календарна кількість днів у році;

$D_{\text{В}}, D_{\text{С}}$ – відповідно кількість вихідних та святкових днів;

$D_{\text{Р}}$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$K_{\text{В}}$ – коефіцієнт використання приладу у часі протягом зміни.

Обчислимо витрати на електроенергію.

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_{\text{С}} \cdot K_3 \cdot C_{\text{ЕН}} = 1638 \cdot 0.03 \cdot 0.875 \cdot 9.43 =$$

$$= 405.466 \text{ грн.,} \quad (4.25)$$

де $N_{\text{С}}$ – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ – тариф за 1 кВт-годин електроенергії.

Розрахуємо накладні витрати.

$$C_{\text{Н}} = C_{\text{ПР}} \cdot 0.67 = 29069.95 \cdot 0.67 = 19476.8665 \text{ грн.} \quad (4.26)$$

Розрахуємо загальні експлуатаційні витрати.

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}$$

$$(4.27)$$

$$C_{\text{ЕКС}} = 345597.235 + 76031.391 + 7412.837 + 889.54 +$$

$$+ 405.466 + 19476.8665 = 449813.3355 \text{ грн.}$$

Обчислимо собівартість однієї машино-години.

$$C_{\text{М-Г}} = \frac{C_{\text{ЕКС}}}{T_{\text{ЕФ}}} = \frac{449813.3355}{1638} = 274.611 \text{ грн/год.}$$

$$(4.28)$$

Розрахуємо загальні витрати на машинний час.

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T = 274.611 \cdot 795,48 = 220833.998 \text{ грн.} \quad (4.29)$$

Накладні витрати складають 67% від заробітної плати.

$$C_H = C_{зп} \cdot 0,67 = 345597.235 \cdot 0,67 = 231550.147 \text{ грн.} \quad (4.30)$$

Розграхуємо загальну вартість.

$$C_{пп} = C_{зп} + C_{в\text{ИД}} + C_M + C_H \quad (4.31)$$

$$\begin{aligned} C_{пп} &= 345597.235 + 76031.391 + 220833.998 + 231550.147 = \\ &= 874012.771. \end{aligned}$$

4.7 Вибір кращого варіанту програмного продукту за техніко-економічним рівнем

Для обґрунтування ефективності впровадження гібридної рекомендаційної системи проведено оцінку техніко-економічного рівня проєкту. Розрахунок коефіцієнта техніко-економічного рівня (КТЕР) виконується за наступною формулою.

$$K_{\text{ТЕР}j} = \frac{K_{kj}}{C_{\phi j}} = \frac{19,19}{874012.771} = 2,195 \cdot 10^{-5}, \quad (4.32)$$

де K_{kj} – комплексний коефіцієнт якості програмного продукту.

$C_{\phi j}$ – собівартість функціонування або розробки програмного продукту.

У результаті комплексного техніко-економічного аналізу було розраховано трудомісткість ключових етапів розробки програмної системи для автоматизованої перевірки дипломних робіт на відповідність вимогам ДСТУ. Перше завдання — розробка ядра системи перевірки з урахуванням динамічної (перемінної) структури вхідних даних та застосування типових функціональних модулів — потребує 77,568 людино-днів. Такий показник зумовлений високим рівнем складності обробки текстових документів різної

структури та забезпечення підтримки багатьох форматів. Поправочні коефіцієнти, що враховують складність контролю даних ($KП = 2.02$) та використання стандартних модулів ($KСТ = 0.6$), забезпечують баланс між точністю реалізації та оптимізацією витрат.

Друге завдання — створення веб-інтерфейсу та зовнішньої програмної оболонки з нормативно-довідковими параметрами і середньою алгоритмічною складністю — має трудомісткість 21,87 людино-днів, що відповідає підвищенню продуктивності завдяки використанню сучасних фреймворків і шаблонів інтерфейсу.

Загальна трудомісткість реалізації програмної системи становить 795,48 людино-годин. Сукупна вартість розробки, яка включає оплату праці, соціальні внески, витрати на машинний час та накладні витрати, склала 874 012,77 грн. При цьому коефіцієнт техніко-економічного рівня ($2,195 \cdot 10^{-5}$) вказує на високу собівартість розробки стосовно технічної складності завдань, що зумовлює необхідність подальшого аналізу доцільності інвестицій і пошуку шляхів зниження витрат без зменшення функціональних можливостей системи.

4.8 Висновки до розділу 4

Проведений техніко-економічний аналіз створення програмної системи для автоматизованої перевірки дипломних робіт на відповідність вимогам ДСТУ засвідчив її актуальність та високий потенціал для впровадження в освітній процес. Розрахунок трудомісткості, витрат на розробку, експлуатаційних витрат і собівартості дозволив комплексно оцінити економічну доцільність проєкту. У результаті застосування

функціонально-вартісного підходу було забезпечено раціональне поєднання функціональності системи з витратами на її реалізацію.

Незважаючи на значну частку витрат, пов'язаних із обробкою змінної текстової інформації та використанням обчислювальних ресурсів, програмна система демонструє вагомі переваги: модульну архітектуру, можливість масштабування, високу точність перевірки структурних та змістовних вимог ДСТУ, а також зручність інтеграції в існуючу ІТ-інфраструктуру закладів освіти.

Отже, створена система відповідає сучасним технічним і нормативним вимогам, сприяє підвищенню якості контролю за оформленням наукових робіт і має значний потенціал для подальшого розвитку, оптимізації та комерційного застосування в галузі вищої освіти.

ВИСНОВКИ

У межах проведеного дослідження було здійснено аналіз предметної області в першому розділі і він виявив певну закономірність — недостатність або відсутність практичних та наукових матеріалів, що безпосередньо висвітлюють проблематику автоматичної перевірки оформлення бакалаврських робіт. Це ускладнило пошук ефективних рішень та вимагало розробки авторських підходів.

Одним із інструментів, які частково автоматизують процес оформлення, є шаблони на базі LaTeX. Вони можуть бути ефективні у технічних та природничих університетах, забезпечуючи базову відповідність структурним вимогам і допомагаючи уникнути типових помилок. Проте такі шаблони не охоплюють повний спектр вимог, передбачених державними стандартами (ДСТУ), і потребують додаткової адаптації. Варто зазначити, що LaTeX може лише створити структурний шаблон, але не може повноцінно замінити процес перевірки оформлення, оскільки не гарантує дотримання всіх специфічних норм оформлення.

У процесі дослідження було розглянуто декілька підходів до реалізації задачі перевірки оформлення: використання LLM (великих мовних моделей), зображень, а також алгоритмічний підхід. З огляду на досить великі вимоги до ресурсів першого й другого варіантів, було обрано саме алгоритмічний підхід як найбільш прийнятний та реалістичний для реалізації.

У результаті програмної реалізації було досягнуто значного прогресу, однак не всі поставлені цілі вдалося повністю реалізувати, було реалізовано 70% від всіх поставлених цілей. Зокрема, низка перевірок виявилася технічно складною або неможливою через наступні причини.

1. Невизначеність введення користувача у випадках, коли об'єкти, що підлягають перевірці (наприклад, переліки), не мають чітких

системних правил, система не може їх надійно ідентифікувати, а тому і перевірити немає можливості.

2. У ході розробки було з'ясовано, що PDF-файли, створені шляхом конвертації з інших форматів які були створені у різних текстових редакторах (наприклад, Microsoft Word, LibreOffice, Google Docs), можуть суттєво відрізнятися за структурою. Кожен редактор по-своєму форматує макети, шрифти, відступи, стилі заголовків тощо, що після конвертації у PDF призводить до появи прихованих символів або нестандартного розташування елементів. У результаті конвертації в PDF ці особливості можуть призводити до появи прихованих символів, зміщення тексту, розривів між словами або навіть порушення логічної структури документа. Це створює значні труднощі для алгоритмів автоматичної перевірки, які потребують стабільної та передбачуваної структури, щоб коректно визначати та перевіряти елементи документа. У деяких випадках це унеможлиблює реалізацію універсального підходу до перевірки та потребує врахування особливостей кожного формату окремо.

3. Неконсистентне зберігання тексту в PDF, особливо це стосується формул, які часто зберігаються в різних строках, що унеможливило реалізацію автоматизованої перевірки їх наявності та оформлення.

Незважаючи на ці обмеження, проєкт продемонстрував можливість достатньо високого рівня автоматизації перевірки оформлення, що повинно зменшити навантаження на перевіряючих осіб. Отримані результати можуть бути основою для подальшого розвитку системи з використанням гібридних підходів, які комбінують алгоритмічну обробку з елементами машинного навчання.

У перспективі система може бути доповнена шляхом інтеграції методів комп'ютерного зору для більш точної ідентифікації таблиць, рисунків та формул. Також можливе використання великих мовних моделей (LLM), що

дозволить реалізувати перевірки, які складно реалізувати класичними алгоритмами. Йдеться, зокрема, про оцінку логічності переліків, структури та стилістики тексту, відповідності заголовків змісту, правильності оформлення списку літератури, парсингу змісту тощо. Крім того, перспективними напрямками розвитку є впровадження автоматичної перевірки на плагіат, а також виявлення текстів, згенерованих за допомогою нейронних мереж, а отже це може стати невід'ємною частиною комплексної системи контролю якості дипломних робіт.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Концепція цифрової трансформації освіти і науки: громадське обговорення. URL: <https://mon.gov.ua/news/kontseptsiya-tsifrovoi-transformatsii-osviti-i-nauki-mon-zapros hue-do-gromadskogo-obgovorenn ya> (дата звернення: 13.04.2025).
2. A comprehensive LaTeX thesis template for Zhejiang University. URL: <https://www.projectscouts.com/zjuthesis-a-comprehensive-latex-thesis-template-for-zhejiang-university/> (Last accessed: 14.04.2025).
3. Zjuthesis GitHub repository. URL: <https://github.com/TheNetAdmin/zjuthesis?tab=readme-ov-file> (Last accessed: 14.04.2025).
4. Tau LaTeX thesis template Overleaf. URL: <https://www.overleaf.com/project/60d420415d85cbf48bbd830a> (Last accessed: 15.04.2025).
5. Scientific Thesis Template. URL: <https://latextemplates.github.io/scientific-thesis-template/> (Last accessed: 15.04.2025).
6. Weber-Wulff D. et al. Testing of Detection Tools for AI-Generated Text. *International Journal for Educational Integrity*. 2023. Vol. 19, No. 26. 39 p. DOI: 10.1007/s40979-023-00146-z.
7. Fraser C. K., Dawkins H., Kiritchenko S. Detecting AI-generated text: Factors influencing detectability with current methods. *Journal of Artificial Intelligence Research*. 2025. Vol. 82. P. 2233-2278. DOI: 10.48550/arXiv.2406.15583
8. Hlukhov V. S., Sydorko D. S. Algorithms and software for verification of scientific and technical text documents. *Applied Aspects of Information*

- Technology*. 2023. Vol. 6, No. 3. P. 304–317. DOI: doi.org/10.15276/aait.06.2023.21
9. Smith R. An Overview of the Tesseract OCR Engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Curitiba, Brazil. 2007. Vol. 2. P. 629-633. DOI: 10.1109/ICDAR.2007.4376991.
 10. Das T. K., Tripathy A. K., Mishra A. K. Optical character recognition using artificial neural network. *2017 International Conference on Computer Communication and Informatics (ICCCI)*. Coimbatore, India. 2017. P. 1-4. DOI: 10.1109/ICCCI.2017.8117703.
 11. Chomsky N. Three models for the description of language. *IEEE Transactions on Information Theory*. 1956. Vol. 2:3, P. 113–124. 8.
 12. Chomsky N. On Certain Formal Properties of Grammars. *Information and Control*, 1959, 2(2), P. 137–167.
 13. Спекторський І.Я., Статкевич В.М. Формальні мови та автомати: підруч. Київ, КПІ ім. Ігоря Сікорського. 2019. 167 с.
 14. Jiang T., Li M., Ravikumar B., Regan, K. W. Formal Grammars and Languages. *ResearchGate*. 2010. URL: https://www.researchgate.net/publication/234814120_Formal_grammars_and_languages. (Last accessed: 10.05.2025)
 15. *The Oxford Handbook of Computational Linguistics*. 2nd edn / R. Mitkov et. al. Oxford: Oxford University Press, 2022. 1376 p.
 16. Hopcroft J. E., Motwani R., Ullman J. D. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Boston, MA: Addison-Wesley, 2006. 750 p.
 17. Friedl J. *Mastering Regular Expressions*. 3rd ed. Sebastopol, CA: O'Reilly Media, 2006. 544 p.
 18. Regular expression operations. URL: <https://docs.python.org/3/library/re.html> (Last accessed: 29.04.2025)

19. Регулярний вираз HOWTO. URL: <https://docs.python.org/uk/3.13/howto/regex.html> (Last accessed: 29.04.2025)
20. How to Extract Images from a PDF in Python. *Medium*. URL: <https://medium.com/@nutanbhogendrasharma/how-to-extract-images-from-a-pdf-in-python-e37e75997642>. (Last accessed: 05.05.2025)
21. Table Recognition and Extraction With PyMuPDF. *Artifex*. URL: <https://artifex.com/blog/table-recognition-extraction-from-pdfs-pymupdf-python> (Last accessed: 02.04.2025)
22. PyMuPDF documentation. URL: <https://pymupdf.readthedocs.io/en/latest/> (Last accessed: 23.04.2025)
23. ДСТУ 3008:2015 «Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання» / Нац. стандарт України. Вид. офіц. Київ : ДП «УкрНДНЦ», 2016. 72 с. (Інформація та документація)
24. Browning J. B., Alchin M. Pro Python 3: Features and Tools for Professional Development. NY, New York: Apress, 2019.
25. NumPy documentation. URL: <https://numpy.org/doc/stable/> (Last accessed: 23.04.2025)
26. FastAPI documentation. URL: <https://fastapi.tiangolo.com/> (Last accessed: 23.04.2025)
27. Uvicorn documentation. URL: <https://www.uvicorn.org/> (Last accessed: 23.04.2025)
28. Ska Y. A study and analysis of continuous delivery, continuous integration in software development environment. *SSRN Electronic Journal*. 2019. Vol. 6, No. 9. P. 96-107.
29. Taleb M. Use and Design REST APIs. *ResearchGate*. URL: https://www.researchgate.net/publication/389944668_Use_and_Design_REST_APIs (Last accessed: 26.04.2025)

ДОДАТОК А. ПРОГРАМНИЙ КОД

файл requirements.txt

```
annotated-types==0.7.0
anyio==4.6.2.post1
click==8.1.7
colorama==0.4.6
fastapi==0.115.5
h11==0.14.0
idna==3.10
Jinja2==3.1.4
MarkupSafe==3.0.2
pydantic==2.9.2
pydantic_core==2.23.4
PyMuPDF==1.25.5
python-multipart==0.0.17
sniffio==1.3.1
starlette==0.41.3
typing_extensions==4.12.2
uvicorn==0.32.0
pdfplumber==0.11.6
regex==2024.11.6
```

файл main.py

```
import uvicorn
from src.app import app

if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000, reload=True)
```

файл src/app.py

```
from fastapi import FastAPI, File, UploadFile, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
from src.dstu_check.dstu_checker import DSTUChecker
import os
```

```

app = FastAPI(title="DSTU Checker",
version=os.environ.get("VERSION", "0.0.0"))

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# mount the static files and templates?
app.mount("/static", StaticFiles(directory="src/static"),
name="static")
templates = Jinja2Templates(directory="src/templates")

upload_directory = "uploads"
os.makedirs(upload_directory, exist_ok=True) # створюємо папку
uploads якщо її немає

@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request":
request, "analysis_result": None})

@app.post("/uploadfile/")
async def upload_file(request: Request, file: UploadFile =
File(...)):
    file_location = os.path.join(upload_directory, file.filename)
    with open(file_location, "wb") as f:
        f.write(await file.read())

# функція аналізу
checker = DSTUChecker(file_location)
checker.analyze_file()

return templates.TemplateResponse("index.html", {
    "request": request,
    "analysis_result": checker.logs # СПИСОК ПОМИЛОК
})

```

файл `src/static/style.css`

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f9;
  color: #333;
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 0;
  padding: 20px;
}

h2 {
  color: #333;
}

form {
  margin: 20px 0;
  display: flex;
  flex-direction: column;
  align-items: center;
}

input[type="file"], button {
  padding: 10px;
  margin: 5px;
  border-radius: 5px;
  border: 1px solid #ddd;
}

button {
  background-color: #5a67d8;
  color: white;
  cursor: pointer;
}

button:hover {
  background-color: #4c51bf;
}

/* Scrollable Errors Container */
.scrollable {
```

```
border: 1px solid #ccc;
padding: 10px;
max-height: 500px;
overflow-y: auto;
background-color: #ffffff;
margin-top: 20px;
}

ul {
  list-style-type: none;
  padding: 0;
}

li {
  font-size: 14px;
  line-height: 1.6;
}

p {
  font-size: 16px;
  color: #333;
}

/* Back Button */
form[action="/"] {
  text-align: center;
  margin-top: 20px;
}

form[action="/"] button {
  background-color: #f44336;
}

form[action="/"] button:hover {
  background-color: #e53935;
}
```

файл src/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>File Upload and Analysis</title>
    <link rel="stylesheet" href="../../static/style.css">
</head>
<body>
    <h2>Upload a File for Analysis</h2>

    <!-- Upload Form -->
    <form action="/uploadfile/" enctype="multipart/form-data"
method="post">
        <input type="file" name="file">
        <button type="submit">Upload and Analyze</button>
    </form>

    {% if analysis_result is not none %}
        <hr>
        <h3>Analysis Result:</h3>

        <!-- Scrollable Errors Container -->
        <div style="border: 1px solid #ccc; padding: 10px;
max-height: 500px; overflow-y: scroll;">
            {% if analysis_result|length == 0 %}
                <p>The file passed all checks. No errors
found!</p>
            {% else %}
                <ul>
                    {% for error in analysis_result %}
                        <li>{{ error }}</li>
                    {% endfor %}
                </ul>
            {% endif %}
        </div>

        <!-- Back Button -->
        <form action="/" method="get">
            <button type="submit">Back</button>
        </form>
    {% endif %}
</body>
</html>

```

файл `src/dstu_check/dstu_checker.py`

```

import fitz
import re
import regex
from src.dstu_check.helpers import Helpers

BASE_TITLES = ["РЕФЕРАТ", "ABSTRACT", "ЗМІСТ", "ВСТУП", "ПЕРЕЛІК
ДЖЕРЕЛ ПОСИЛАННЯ"]
BASE_CONTENT_TITLES = ["ВСТУП", "ВИСНОВКИ", "ПЕРЕЛІК ДЖЕРЕЛ
ПОСИЛАННЯ"]

class DSTUChecker:
    def __init__(self, pdf_path: str):
        self.logs = []
        self.pdf_path = pdf_path
        self.content_titles =
Helpers.extract_titles_and_pages(self.pdf_path)
        self.title_levels =
Helpers.classify_titles(self.content_titles)
        self.sections = [item for item in
self.title_levels["level_1"]
                        if any(char.isdigit() for char in
item['title'][:10])]
        self.doc = fitz.open(self.pdf_path)
        # TODO: probably put this helpers function into this
class

    def show_logs(self):
        for log in self.logs:
            print(log, '\n')

    def analyze_file(self) -> None:
        self._check_content()
        self._check_images()
        self._check_name_after_plan()
        self._check_row_start()
        self._check_page_numbers()
        self._check_indent()
        self._check_font()
        self._find_enumerations_in_pdf()
        self._check_level_1_titles()
        self._check_tables()
        self.logs.append("===Перевірка завершена===")

```

```

def _check_content(self):
    found_titles = set()
    self.logs.append("===Перевірка змісту===")
    pattern = re.compile(r"|" .join(re.escape(title) for title
in BASE_CONTENT_TITLES))

    for title in self.content_titles:
        if pattern.search(title["title"]):
            found_titles.add(title["title"])

    missing_titles = set(BASE_CONTENT_TITLES) - found_titles
    for missing in missing_titles:
        self.logs.append(f"Не знайдено в змісті: {missing}")
        self.logs.append(f"Прохання перевірити зміст перед
використанням сервісу")

def _check_page_numbers(self):
    self.logs.append("===Перевірка нумерації===")
    try:
        doc = fitz.open(self.pdf_path)

        for page_num, page in enumerate(doc, start=1):
            text = page.get_text("blocks")
            text_num = regex.sub(r'\p{Cf}+', '', text[0][4])
            text_num_lines = [line.strip() for line in
text_num.split('\n') if line.strip()]
            if page_num == 1: # перевірка титульної сторінки
                if [line.strip() for line in
text_num.split('\n')][0].strip().isdigit():
                    self.logs.append(f"Номер присутній на
титульній сторінці - Помилка")

            else: # всі інші сторінки
                if not text_num_lines[0].strip().isdigit():
# перевіряємо наявність номера
                    self.logs.append(f"Сторінка {page_num} не
має номера!")
                else:
                    if not
text_num_lines[0].strip().isdigit(): # перевіряємо розташування
щоб було зправа зверху
                        x0, y0, x1, y1 = text[0][:4]

```

```

        if y1 > Helpers.cm_to_points(3) or x1
< (page.rect.width - Helpers.cm_to_points(2)):
            self.logs.append(f"Номер сторінки
{page_num} неправильно розташований")
            doc.close()
    except Exception as e:
        print(f"Помилка обробки файлу: {e}")

def _check_indent(self):
    self.logs.append("===Перевірка відступів===")
    try:
        doc = fitz.open(self.pdf_path)

        for page_num, page in enumerate(doc, start=1):
            page_width, page_height = page.rect.width,
page.rect.height
            text_boxes = page.get_text("blocks")

            # перевіряємо верхнє та нижнє поле

            first, last =
Helpers.find_first_and_last_text(text_boxes) # знаходимо
початок тексту і кінець
            if first is None:
                continue
            upper_margin = Helpers.points_to_cm(first[1]) #
верхнє поле
            bottom_margin = Helpers.points_to_cm(page_height
- last[3]) # нижнє поле

            if upper_margin < 2:
                self.logs.append(
                    f'Відступ неправильний на сторінці
{page_num} '
                    f'| Верхній відступ: {upper_margin} см')

            if bottom_margin < 2:
                self.logs.append(
                    f'Відступ неправильний на сторінці
{page_num} '
                    f'| Нижній відступ: {bottom_margin} см')

            # перевіряємо праве та лівє поле
            for box in text_boxes:

```

```

        if "lines" in box:
            x0, _, x1, _ = box['bbox']
            left_margin = Helpers.points_to_cm(x0) #
ліве поле

            right_margin =
Helpers.points_to_cm(page_width - x1) # праве поле

            if left_margin <= 3 or right_margin <=
1.5:
                self.logs.append(
                    f'Відступ неправильний на
сторінці {page_num} '
                    f'| Лівий відступ: {left_margin}
см '
                )

                if right_margin <= 1.5:
                    self.logs.append(
                        f'Відступ неправильний на
сторінці {page_num} '
                        f'| Правий відступ:
{right_margin} см'
                    )

            doc.close()

    except Exception as e:
        print(f"Помилка обробки файлу: {e}")

    def _check_font(self):
        self.logs.append("===Перевірка стилю та розміру===")
        try:
            # відкираємо pdf
            doc = fitz.open(self.pdf_path)

            dodatok_set = set([item['page'] for item in
self.content_titles
                                if re.search(r'ДОДАТОК',
item["title"], re.IGNORECASE)])
            if not dodatok_set:
                first_dodatok_page = len(doc)
            else:
                first_dodatok_page = min(set([item['page'] for
item in self.content_titles if "ДОДАТОК" in item['title']]))

```

```

    for page_num, page in enumerate(doc, start=1):
        # ВИТЯГУЄМО ТЕКСТ
        line_count = 0
        for block in page.get_text("dict")["blocks"]:
            for line in block.get("lines", []):
                if line_count > 0 and page_num >=
first_dodatok_page:
                    break

                for span in line.get("spans", []):
                    font_name = span.get("font", "")
                    font_size = span.get("size", 0)
                    text = span.get("text", "").strip()
                    alignment = span.get("flags", 0)
                    if text == "" or
re.search(r'math|symbol|cambria|latex', font_name,
re.IGNORECASE):
                                continue # пропускаємо розмір
формул та пусті строки

                                if "Times" not in font_name and
((font_size > 14.1) or (font_size < 13.9)):
                                    self.logs.append(
                                        f"На сторінці {page_num}:
Текст: '{text}' "
                                        f"| Стилль: {font_name} |
Розмір: {font_size:.2f}")
                                    # print(
                                    #     f"На сторінці {i}: Текст:
'{text}' "
                                    #     f"| Стилль: {font_name} |
Розмір: {font_size:.2f}")
                                # if alignment != 4: # 4 -
вирівнювання по ширині
                                    #     self.logs.append(
                                    #         f"На сторінці {i}: Текст
'{text}' не вирівняний по ширині."
                                    #     )
                                line_count += 1

    doc.close()
except Exception as e:
    print(f"Помилка обробки файлу: {e}")

```

```

def _check_level_1_titles(self):
    self.logs.append("===Перевірка чи розташовані
заголовки(першого рівня) з початку сторінки===")
    try:
        level_1_titles = self.title_levels.get('level_1', [])
        doc = fitz.open(self.pdf_path)

        for title_entry in level_1_titles:
            title = title_entry['title']
            page_num = title_entry['page']
            title_found = False
            if 0 <= page_num < len(doc):
                page = doc.load_page(page_num - 1)
                text = page.get_text("text")
                lines = text.split('\n')

                for i, line in enumerate(lines[:4], start=0):
                    # ДИВИМОСЬ 4 рядка перші
                    title = regex.sub(r'\p{Cf}+', '', title)
                    line = regex.sub(r'\p{Cf}+', '', line)
                    if title[:20].strip().lower() in
line[:20].strip().lower():
                        title_found = True
                        break

                if not title_found:
                    self.logs.append(
                        f"На сторінці {page_num}: Заголовок:
                    '{title}' "
                    f"\nмає пропуск в більше ніж 2 рядки
                    зверху"
                    f"\nабо не був знайдений"
                    )

            doc.close()
    except Exception as e:
        print(f"Помилка обробки файлу: {e}")

def _check_row_start(self):
    self.logs.append("===Перевірка початку рядка===")
    wrong_symbols_to_start = ["•"]
    try:
        doc = fitz.open(self.pdf_path)

```

```

        start_page = next((item["page"] for item in
self.content_titles if item["title"] == "ВСТУП"), None)
        end_page = next(
            (item["page"] for item in self.content_titles if
item["title"] == "ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ"), None)

        if start_page is None:
            start_page = 0
        if end_page is None:
            end_page = len(doc)

        for page_num in range(start_page, end_page):
            page = doc.load_page(page_num)
            text = page.get_text("text")
            lines = text.splitlines()

            for i, line in enumerate(lines, 1):
                stripped_line = line.strip()
                if not stripped_line:
                    continue
                for symbol in wrong_symbols_to_start:
                    if stripped_line.startswith(symbol):
                        self.logs.append(
                            f"На сторінці {page_num + 1}:
рядок починається з "
                            f"{symbol}"
                            f"недопустимого символу"
                        )
            doc.close()
        except Exception as e:
            print(f"Помилка обробки файлу: {e}")

    def _find_enumerations_in_pdf(self):
        self.logs.append("===Перевірка сторінок на наявність
переліків===")
        try:
            doc = fitz.open(self.pdf_path)

            enumeration_types = [
                {"type": "1)", "regex": r":\s*\n1\) "},
                {"type": "- ", "regex": r":\s*\n- "},
                {"type": "1.", "regex": r":\s*\n1\."}
            ] # три варіанти переліків

```

```

        start_page = next((item["page"] for item in
self.content_titles if item["title"] == "ВСТУП"), None)
        end_page = next(
            (item["page"] for item in self.content_titles if
item["title"] == "ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ"), None)

        if start_page is None:
            start_page = 0
        if end_page is None:
            end_page = len(doc)

        for page_num in range(start_page, end_page):
            page = doc.load_page(page_num)
            text = page.get_text("text")

            for enum_type in enumeration_types:
                if re.search(enum_type["regex"], text):
                    self.logs.append(f"На сторінці {page_num
+ 1}: знайдено тип переліку '{enum_type['type']}'")
            doc.close()
        except Exception as e:
            print(f"Помилка обробки файлу: {e}")

    def _check_name_after_plan(self):
        self.logs.append("===Перевірка сторінок на наявність
прізвища після календарного плану===")
        doc = fitz.open(self.pdf_path)
        text = "\n".join([page.get_text("text") for page in
doc[:10]])

        # пошук календарний план та тексту студент
        match = re.search(r'календарний
план(.*)Студент\s+(\S+)\s+(\S+)', text, re.DOTALL |
re.IGNORECASE)
        if match:
            first_name = match.group(2)
            last_name = match.group(3)
            self.logs.append(f"Знайдено Ім'я Прізвище студента:
{first_name} {last_name}")

            if not last_name.isupper():
                self.logs.append("Прізвище студента не написано
великими літерами.")

```

```

else:
    self.logs.append("Ім'я Прізвище студента не знайдено
після календарного плану.")

def _get_rozdil_num(self, page_number):
    current_rozdil = 1
    for section in self.sections:
        if page_number >= section['page']:
            current_rozdil = int(re.findall(r'\d+',
section['title'])[0])
        else:
            break
    return current_rozdil

def _check_images(self):
    self.logs.append("===Перевірка рисунків у документі===")
    try:
        doc = fitz.open(self.pdf_path)
        page_texts = {page_num:
doc[page_num].get_text("text") for page_num in range(len(doc))}

        num_image_in_rozdil = 0
        start_page = next((item["page"] for item in
self.content_titles if item["title"] == "ВСТУП"), None)
        end_page = next(
            (item["page"] for item in self.content_titles
            if re.search(r'ПЕРЕЛІК', item["title"],
re.IGNORECASE)
            and re.search(r'ПОСИЛ', item["title"],
re.IGNORECASE)),
            None
        )

        dodatok_set = set([item['page'] for item in
self.content_titles
            if re.search(r'ДОДАТОК',
item["title"], re.IGNORECASE)])
        if not dodatok_set:
            first_dodatok_page = len(doc)
        else:
            first_dodatok_page = min(
                set([item['page'] for item in
self.content_titles if "ДОДАТОК" in item['title']]))

```

```

if start_page is None:
    start_page = 0
if end_page is None:
    end_page = first_dodatok_page

for page_num in range(start_page, end_page):
    page = doc[page_num]
    images = page.get_images()
    # if len(images) == 0:
    #     continue
    text_blocks = page.get_text("blocks") #
отримуємо текстові блоки з координатами
    page_width = page.rect.width

    current_rozdil = self._get_rozdil_num(page_num +
1)

    if current_rozdil >
self._get_rozdil_num(max(page_num, 0)):
        num_image_in_rozdil = 0

    for img_index, img in enumerate(images):
        num_image_in_rozdil += 1
        correct_num =
f"{current_rozdil}.{num_image_in_rozdil}"

        xref = img[0]

        # отримуємо координати зображення
        bbox = page.get_image_rects(xref)[0]
        x0, y0, x1, y1 = bbox
        img_center = (x0 + x1) / 2

        # перевіряємо, що зображення по центру
(допустиме відхилення 10%)
        centered_image = abs(img_center - page_width
/ 2) < page_width * 0.1

        # знаходимо текст нижче зображення
        caption = ""
        text_blocks_below = [block for block in
text_blocks if block[1] >= ((y0 + y1)/2)]
        text_blocks_below = sorted(text_blocks_below,
key=lambda x: x[1])

```

```

caption_found = False
centered_caption = False

left_margin = Helpers.cm_to_points(3)
right_margin = Helpers.cm_to_points(1.5)
content_left = left_margin
content_right = page_width - right_margin
content_width = content_right - content_left
content_center = (content_left +
content_right) / 2

for i, block in enumerate(text_blocks_below):
    bx0, by0, bx1, by1, text, *_ = block
    text_center = (bx0 + bx1) / 2

    if re.search(r"Рисунок (\d+\.\d+)", text,
re.IGNORECASE):

        caption = text.strip()
        num = re.search(r"Рисунок
(\d+\.\d+)", caption)

        num = num.group(1)
        caption_found = True
        centered_caption = abs(text_center -
content_center) <= content_width * 0.1

        # перевіряємо, що підпис знаходиться
в наступному рядку після рисунка
        if i <= 4:
            break # перемо перший знайдений
підпис

        else:
            self.logs.append(f"Сторінка
{page_num + 1}: "
                            f"Підпис {num}
не знаходиться одразу після рисунка.")

        # логінг перевірок
        if not caption_found:
            self.logs.append(
                f"Сторінка {page_num + 1}: Відсутній
або некоректний підпис у {correct_num} (має бути 'Рисунок
{correct_num}').")

        if not centered_image:

```



```

def _check_tables(self):
    self.logs.append("===Перевірка таблиць у документі===")
    try:
        doc = fitz.open(self.pdf_path)
        page_texts = {page_num:
doc[page_num].get_text("text") for page_num in range(len(doc))}

        num_table_in_rozdil = 0
        last_table_num = ""
        prev_table_ends_low = False
        prev_table_page = 0

        start_page = next((item["page"] for item in
self.content_titles if item["title"] == "ВСТУП"), None)
        end_page = next(
            (item["page"] for item in self.content_titles
             if re.search(r'ПЕРЕЛІК', item["title"],
re.IGNORECASE)
             and re.search(r'ПОСИЛ', item["title"],
re.IGNORECASE)),
            None
        )

        dodatok_set = set([item['page'] for item in
self.content_titles
                           if re.search(r'ДОДАТОК',
item["title"], re.IGNORECASE)])
        if not dodatok_set:
            first_dodatok_page = len(doc)
        else:
            first_dodatok_page = min(
                set([item['page'] for item in
self.content_titles if "ДОДАТОК" in item['title']]))

        if start_page is None:
            start_page = 0
        if end_page is None:
            end_page = first_dodatok_page

        for page_num in range(start_page, end_page):
            page_fitz = doc[page_num]
            text_blocks = page_fitz.get_text("blocks")
            page_width = page_fitz.rect.width

```

```

page_height = page_fitz.rect.height

table_fitz = page_fitz.find_tables()

current_rozdil = self._get_rozdil_num(page_num +
1)

    if current_rozdil >
self._get_rozdil_num(max(page_num, 0)):
        num_table_in_rozdil = 0

        for table_index, table in
enumerate(table_fitz.tables):
            bbox = table.bbox
            x0, y0, x1, y1 = bbox

            # --- ПРОДОВЖЕННЯ ТАБЛИЦІ ---
            if prev_table_ends_low and y0 < page_height *
0.15 and (page_num - prev_table_page) < 2:
                blocks_above = [block for block in
text_blocks if block[3] < y0]
                blocks_above = sorted(blocks_above,
key=lambda b: b[3], reverse=True)
                continuation_found = False
                continuation_caption_left_aligned = False
                for block in blocks_above[:3]:
                    text = block[4].strip()
                    bx0, by0, bx1, by1, t, *_ = block
                    if re.search(rf"Продовження таблиці
{re.escape(last_table_num)}", text, re.IGNORECASE):
                        continuation_found = True
                        ideal_offset =
(Helpers.cm_to_points(1.25) + Helpers.cm_to_points(3))
                        continuation_caption_left_aligned
= abs((bx0 - ideal_offset)) < 0.05 * page_width
                        break

                if not continuation_found:
                    self.logs.append(
                        f"Сторінка {page_num + 1}:
Відсутній підпис 'Продовження таблиці' перед продовженням
{last_table_num}.")
                    if continuation_found and not
continuation_caption_left_aligned:

```

```

        self.logs.append(f"Сторінка {page_num
+ 1}: Підпис 'Продовження таблиці' не вирівняний по "
                                f"лівому краю
таблиці з відступом у 1.25 см.")
        # ЦЕ ПРОДОВЖЕННЯ ТАБЛИЦІ – ПРОПУСКАЄМО
ІНШІ ПЕРЕВІРКИ
        prev_table_ends_low = y1 > page_height *
0.8
        continue

        # --- ЦЕ НОВА ТАБЛИЦЯ ---
        num_table_in_rozdil += 1
        correct_num =
f"{current_rozdil}.{num_table_in_rozdil}"

        # --- ПІДПИС ДО ТАБЛИЦІ ---
        caption = ""
        text_blocks_before = [block for block in
text_blocks if block[1] < y0 - 10]
        text_blocks_before =
sorted(text_blocks_before, key=lambda x: x[1], reverse=True)

        caption_found = False
        caption_left_aligned = False
        num = ""
        for i, block in
enumerate(text_blocks_before):
            bx0, by0, bx1, by1, text, *_ = block
            match = re.search(r"Таблиця (\d+\.\d+)",
text, re.IGNORECASE)
            if match:
                caption = text.strip()
                num = match.group(1)
                caption_found = True
                ideal_offset =
(Helpers.cm_to_points(1.25) + Helpers.cm_to_points(3))
                caption_left_aligned = abs((bx0 -
ideal_offset)) < 0.05 * page_width

            if i in [0, 1]:
                break
            else:
                self.logs.append(

```

```

f"Сторінка {page_num + 1}:
Підпис {num} не знаходиться одразу перед таблицею.")

# --- ЛОГІНГ ---
if not caption_found:
    self.logs.append(
        f"Сторінка {page_num + 1}: Відсутній
або некоректний підпис у {correct_num} "
        f"(має починатися з 'Таблиця
{correct_num}') перед таблицею.")
    if caption_found and not
caption_left_aligned:
        self.logs.append(f"Сторінка {page_num +
1}: Підпис не вирівняний по "
                           f"лівому краю таблиці з
відступом у 1.25 см.")
    if caption_found and num != correct_num:
        self.logs.append(f"Сторінка {page_num +
1}: Нумерація {num} != очікувана {correct_num}.")

# --- ПОСИЛАННЯ НА ТАБЛИЦЮ ---
if caption_found:
    fig_number = re.search(r"Таблиця
(\d+\.\d+)", caption)
    if fig_number:
        fig_number = fig_number.group(1)
        correct_reference_pattern =
fr"{fig_number}"

        found_reference = False

        if page_num > 0:
            prev_page_text =
page_texts[page_num - 1]
            if
re.search(correct_reference_pattern, prev_page_text,
re.IGNORECASE):
                found_reference = True

            if not found_reference:
                curr_page_text =
page_texts[page_num].split(caption)[0]
                if
re.search(correct_reference_pattern, curr_page_text,
re.IGNORECASE):

```

```

        found_reference = True

        if not found_reference:
            self.logs.append(
                f"Сторінка {page_num + 1}:
Відсутнє посилання на {fig_number} перед таблицею на цій або
попередній сторінці")

            last_table_num = num
        else:
            last_table_num = correct_num

        # --- ОНОВЛЮЄМО ФЛАГ ДЛЯ ПРОДОВЖЕННЯ ---
        prev_table_ends_low = y1 > page_height * 0.8
        if prev_table_ends_low:
            prev_table_page = page_num
    doc.close()
except Exception as e:
    print(f"Помилка обробки файлу: {e}")

```

файл *src/dstu_check/helpers.py*

```

import fitz
import re
import regex

class Helpers:
    def __init__(self):
        pass

    @staticmethod
    def cm_to_points(cm: float):
        return cm * 28.34645

    @staticmethod
    def points_to_cm(points: float):
        return points / 28.34645

    @staticmethod
    def find_first_and_last_text(lst: list):
        def is_normal_text(s):

```

```

        return s.strip() and not s.strip().isdigit()

    # весь валідний текст
    filtered = [item for item in lst if
is_normal_text(item[4])]
    if not filtered:
        return None, None
    # повертаємо перший блок тексту і останній
    return filtered[0], filtered[-1]

    @staticmethod
    def extract_titles_and_pages(pdf_path):

        def merge_lines(text, iterations=2):
            text = re.sub(r'[\u202c\u202d\u200e\u200f]', '',
text)

            lines = text.split('\n')
            lines = [line.strip() for line in lines]
            lines = [line for line in lines if len(line) > 2]
            merged = []
            i = 0
            while i < len(lines):
                current = lines[i]
                # Перевірка: чи повністю рядок відповідає шаблону
лише РОЗДІЛ або номер типу 1.3
                if re.fullmatch(r'(РОЗДІЛ\s+\d+|(\d+\.)+\d*)',
current):

                    if i + 1 < len(lines):
                        next_line = lines[i + 1]
                        combined = f"{current} {next_line}"
                        merged.append(combined)
                        i += 2
                    else:
                        merged.append(current)
                        i += 1
                else:
                    merged.append(current)
                    i += 1
            lines = merged
            for _ in range(iterations):
                merged = []
                i = 0
                while i < len(lines):
                    current = lines[i].rstrip()

```

```

        if i + 1 < len(lines):
            next_line = lines[i + 1].rstrip()
            # Check: current line does NOT end with
digit, next line DOES end with digit
            if not re.search(r'\d$', current) and
re.search(r'\d$', next_line):
                merged.append(current + ' ' +
next_line)
                    i += 2
                    continue
                merged.append(current)
                i += 1
            lines = merged
        lines = [line for line in lines if re.search(r'\d$',
line)]
        return '\n'.join(lines)

start_page = 2
end_page = 15

doc = fitz.open(pdf_path)

# ВИТЯГУЄМО ЗМІСТ В ПЕНОМУ ДІАПАЗОНІ ЩОБ НЕ ВІДКРИВАТИ
весь файл
text = ""
for page_number in range(start_page - 1, min(end_page,
len(doc))):
    text += doc[page_number].get_text()

# знаходимо ЗМІСТ
content_start = text.find("ЗМІСТ")
if content_start == -1:
    print("ЗМІСТ НЕ ВКАЗАНИЙ")
    return None

content = text[content_start + 5:] # +5 для того щоб не
взяти ЗМІСТ
lines = merge_lines(content)

titles = []
for line in lines.splitlines():
    line = line.strip()
    if not line or '...' not in line: # Пропускаємо,
якщо немає трьох крапок

```

```

        continue

        # Знаходимо номер сторінки (цифри в кінці)
        page_match = re.search(r'(\d+)\$', line)
        if not page_match:
            continue
        page = int(page_match.group(1))

        # Обрізаємо номер сторінки, шукаємо заголовок до
        трьох крапок
        line_without_page =
line[:page_match.start()].rstrip()
        title_match = re.match(r'^(.*?) (?=\.{3,})',
line_without_page)
        title = title_match.group(1).strip() if title_match
else line_without_page.strip()

        titles.append({"title": title, "page": int(page)})
    return titles

    @staticmethod
    def classify_titles(titles: list[dict]):
        classified_titles = {"level_1": [], "level_2": [],
"level_3": []}

        for item in titles:
            title = item['title']
            if title.isupper():
                classified_titles["level_1"].append(item)
            elif re.match(r'^\d+\.\d+\.\d+', title):
                classified_titles["level_3"].append(item)
            elif re.match(r'^\d+\.\d+', title):
                classified_titles["level_2"].append(item)
        return classified_titles

```

файл Dockerfile

```

FROM python:3.11-slim

RUN apt update && apt install -y \
    curl

```

```
COPY . .
```

```
ARG VERSION=${VERSION}
```

```
ENV VERSION=${VERSION}
```

```
#
```

```
https://github.com/Delgan/loguru/blob/master/loguru/_defaults.py
```

```
ENV LOGURU_COLORIZE="true"
```

```
# easy parse by loki
```

```
ENV LOGURU_TRACE_COLOR=""
```

```
ENV LOGURU_DEBUG_COLOR=""
```

```
ENV LOGURU_INFO_COLOR=""
```

```
ENV LOGURU_SUCCESS_COLOR=""
```

```
ENV LOGURU_WARNING_COLOR=""
```

```
ENV LOGURU_ERROR_COLOR=""
```

```
ENV LOGURU_CRITICAL_COLOR=""
```

```
RUN python3 -m pip install -r requirements.txt
```

```
ENTRYPOINT ["python3", "-m", "uvicorn", "src.app:app", "--host",  
"0.0.0.0", "--port", "8000"]
```

ДОДАТОК Б. ГРАФІЧНІ МАТЕРІАЛИ

Система автоматичної перевірки оформлення бакалаврських робіт

Виконав:
студент групи КА-13
Підберезка Артем Віталійович
Керівник:
асистент кафедри ММСА
Канцедал Георгій Олегович

Дослідження

Об'єкт дослідження: процес перевірки оформлення кваліфікаційних робіт.

Предмет дослідження: методи автоматичного виявлення помилок оформлення бакалаврських робіт у PDF-документах.

Мета роботи: створити систему для автоматичної перевірки відповідності оформлення дипломних робіт згідно державного стандарту України.

Актуальність дослідження

Ручна перевірка оформлення:

- ♦ займає багато часу;
- ♦ залежить від людського фактора.

Проблеми масштабу:

- ♦ велика кількість робіт — велике навантаження на викладачів;
- ♦ неможливо вручну перевірити всі елементи кожної сторінки (відступи, заголовки, таблиці тощо);
- ♦ важко помітити дрібні порушення (наприклад, шрифт 13.8 замість 14, неправильний відступ).



Необхідність автоматизації:

- ♦ забезпечення швидкості, точності та об'єктивності перевірки;
- ♦ допомога студентам, оскільки вони самостійно бачать і виправляють помилки;
- ♦ зменшення навантаження на фахівців з нормконтролю.

Постановка задачі

- ♦ Ознайомитися з чинними стандартами (ДСТУ) щодо оформлення дипломних робіт.
- ♦ Здійснити огляд наукових публікацій, методичних матеріалів та можливих програмних рішень, що стосуються автоматичної перевірки документів.
- ♦ Реалізувати веб-застосунок для завантаження дипломної роботи та автоматичного аналізу її оформлення.
- ♦ Розробити програмний модуль перевірки оформлення дипломних згідно ДСТУ та інтегрувати його з веб-застосунком.
- ♦ Валідувати результати системи на дипломних роботах минулих років
- ♦ Аналіз результатів роботи
- ♦ Сформулювати рекомендації щодо подальших напрямів дослідження

Методи системного аналізу в дипломній роботі

Як було застосовано методи системного аналізу:

1. **Формалізація проблеми**
 - ♦ Вимоги ДСТУ 3008:2015 перетворено у набір структурованих правил, придатних до алгоритмічної перевірки.
2. **Декомпозиція системи перевірки**
 - ♦ Процес перевірки розділено на окремі модулі (перевірка змісту, рисунків, таблиць, формул, нумерації тощо).
3. **Ієрархічна структура документа**
 - ♦ Побудовано модель документа як дерева заголовків, елементів та їхніх атрибутів (шрифт, розмір, відступи, координати).
4. **Логіко-структурний аналіз**
 - ♦ Побудовано логічні правила перевірки зв'язків між елементами (наприклад, посилання на рисунок до його появи).
5. **Аналіз альтернативних рішень**
 - ♦ Розглянуто 4 підходи (LaTeX, LLM, алгоритмічний, OCR), виконано порівняння за критеріями точності, гнучкості, ресурсоемності.

Систематизація правил оформлення згідно з ДСТУ

Виділено **основні категорії правил**:

- ♦ оформлення заголовків;
- ♦ нумерація рисунків, таблиць, формул;
- ♦ структура та відповідність змісту;
- ♦ вимоги до титульного аркуша, джерел, реферату;
- ♦ формат тексту — шрифт, міжрядкові інтервали, абзаци.

Формалізація кожного правила, наприклад:

- ♦ "Шрифт має бути Times New Roman, 14 pt" → `if font_name != "Times New Roman" or font_size != 14: error`
- ♦ "Нумерація рисунків у межах розділу" → порівняння з шаблоном розділ.номер (наприклад, 2.3)

Можливі рішення

Розглянуті варіанти:

- LaTeX-шаблони;
- LLM (великі мовні моделі);
- алгоритмічний підхід;
- використання аналізу зображень.



Порівняння можливих рішень за експертною оцінкою

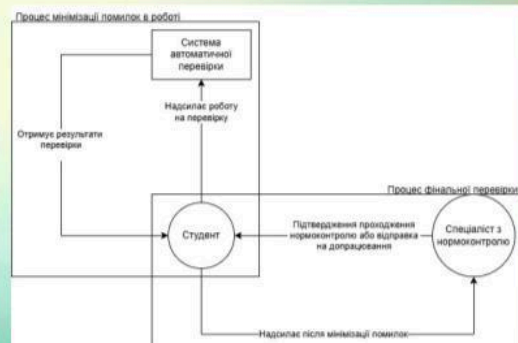
Критерій	LLM-моделі	Алгоритмічний підхід	Підхід з використанням зображень
Швидкість	Низька	Висока	Середня
Глибина аналізу змісту	Висока	Низька	Низька
Вимоги до ресурсів	Високі	Низькі	Середні
Точність формальної перевірки	Середня	Висока	Висока
Масштабованість	Висока	Висока	Висока
Гнучкість до змін	Висока	Низька	Середня
Складність реалізації	Висока	Висока	Висока

Процес проходження нормконтролю

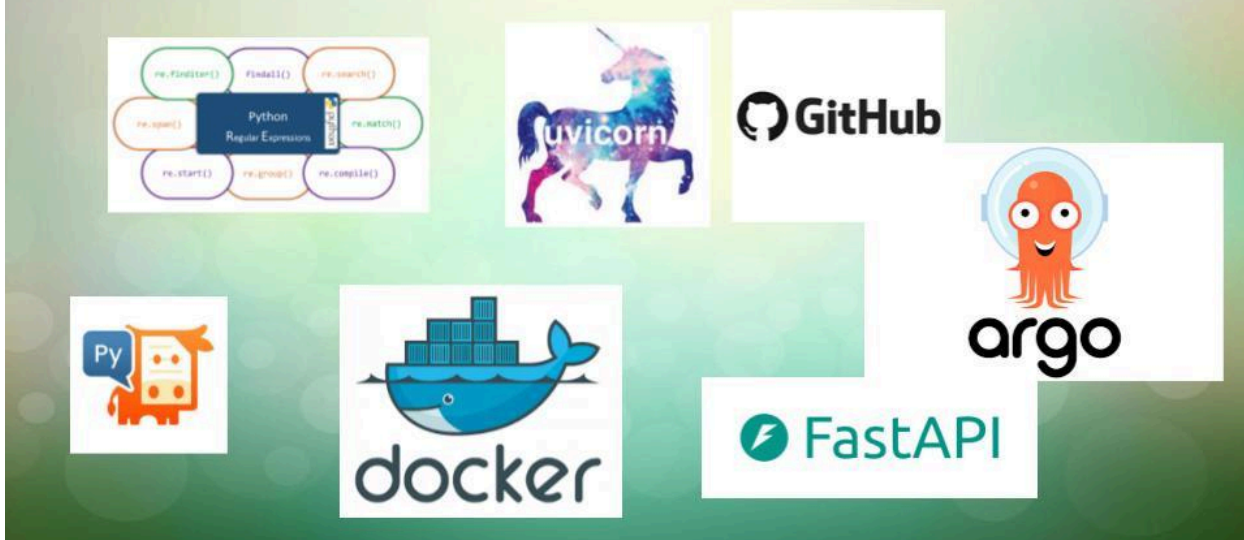
Процес передбачає попередню автоматизовану перевірку дипломної роботи, яка дозволяє виявити типові помилки у форматуванні згідно з вимогами ДСТУ ще до подачі на остаточний розгляд.

Такий підхід дозволяє:

- зменшити кількість повторних подач;
- підвищити загальну якість оформлення;
- оптимізувати навантаження на спеціалістів з нормоконтролю.



Стек використаних технологій

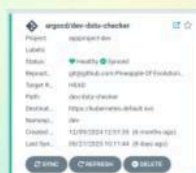


Налаштування середовища розробки

Для забезпечення стабільної, масштабованої та зручної у підтримці інфраструктури було налаштовано повноцінне CI/CD-середовище з використанням GitHub, Kubernetes та ArgoCD. Завдяки цьому можна швидко вносити зміни, тестувати їх та безпечно впроваджувати в продукт версію застосунку без ручного втручання.

Переваги:

- ♦ автоматичний деплой після змін у кодї;
- ♦ швидке внесення та перевірка змін;
- ♦ мінімізація ручного втручання;
- ♦ масштабованість і стабільність середовища;
- ♦ прозорий контроль змін (GitOps);
- ♦ швидке відновлення після збоїв;
- ♦ скорочення часу доставки оновлень.



Веб-застосунок

Архітектура:

- побудовано за принципами REST API;
- взаємодія між клієнтом і сервером через HTTP-методи (GET, POST);
- формат обміну даними в форматі JSON;
- можливість масштабування та гнучкість.



Back-end:

- фреймворк FastAPI (Python);
- обробка PDF-документів, текстовий аналіз, перевірка структури, таблиць, рисунків;
- генерація логів перевірки та передача результатів.

Front-end:

- HTML, CSS;
- завантаження PDF, запуск перевірки, перегляд результатів;
- мінімалістичний та зручний інтерфейс.

Програмна реалізація

Реалізовано сервіс автоматичної перевірки PDF-документів дипломних робіт.

- Основний клас об'єднує перевірку на відповідність вимогам ДСТУ.
- Архітектура побудована на принципах ООП.
- Кожна перевірка — окремий метод класу.
- Допоміжні функції винесено в окремий модуль.

```
def analyze_file(self) -> None:
    self._check_content()
    self._check_images()
    self._check_name_after_plan()
    self._check_row_start()
    self._check_page_numbers()
    self._check_indent()
    self._check_font()
    self._find_enumerations_in_pdf()
    self._check_level_1_titles()
    self._check_tables()
    self.logs.append("====Результат перевірки====")
```

Основні етапи перевірки.

- Зчитування документа, витяг змісту, формування структури документа.
- Послідовна перевірка вимог:
 - зміст;
 - перевірка таблиць/зображень;
 - стиль тексту;
 - відступи;
 - структурні елементи.
- Логування результатів перевірки.

Програмна реалізація

Кожна перевірка в системі DSTUChecker виконується у два етапи:

1. Ідентифікація об'єкта перевірки.

- Визначення цільових елементів у документі алгоритмічно (або OCR, ML, LLM).
- Збирання необхідних параметрів:
 - координати;
 - шрифт та розмір;
 - номер розділу на сторінці (для таблиць, формул, рисунків);
 - розташування об'єкта (зліва, по центру, зправа, зверху тощо);
 - посилання на об'єкт (для таблиць, рисунків тощо)
 - додаткові параметри (продовження таблиці тощо).

2. Формальна перевірка.

- Використання виявлених параметрів для перевірки на вимоги ДСТУ:
 - правильність формату;
 - відповідність структури;
 - дотримання стилю;
 - наявність/відсутність заборонених елементів.
- Формування логу з висновками за результатами перевірки.

Приклади

Формули були вставлені як зображення

№	Тип	Статус	Атрибути	Розмір
1	Текст	OK	Статус: Активний	11:04
2	Текст	OK	Статус: Активний	11:04
3	Текст	OK	Статус: Активний	11:04
4	Текст	OK	Статус: Активний	11:04
5	Текст	OK	Статус: Активний	11:04
6	Текст	OK	Статус: Активний	11:04
7	Текст	OK	Статус: Активний	11:04
8	Текст	OK	Статус: Активний	11:04
9	Текст	OK	Статус: Активний	11:04
10	Текст	OK	Статус: Активний	11:04
11	Текст	OK	Статус: Активний	11:04
12	Текст	OK	Статус: Активний	11:04
13	Текст	OK	Статус: Активний	11:04
14	Текст	OK	Статус: Активний	11:04
15	Текст	OK	Статус: Активний	11:04
16	Текст	OK	Статус: Активний	11:04
17	Текст	OK	Статус: Активний	11:04
18	Текст	OK	Статус: Активний	11:04

45
ориставляєм

43

Приклади

Таблиця 4.2 – Основні параметри програмного продукту

Величина параметра	Типові значення	Одиниця виміру	Грани	Середня	Крайня
Потенційний об'єм програмного коду	3.1	кількість сторінок	2000	3000	12000
Потенційна кількість функцій програмного продукту	3.2	кількість	100	40	10
Потенційний час розробки найважливіших робочих процесів	3.3	місяці	70	8	4
Складові складові системи	3.4	млн	400	400	100

====Перевірка таблиць у документі====

Сторінка 77: Відсутній або некоректний підзаголовок у 4.1 (має прочитатися з Таблиця 4.1) перед таблицею.

Сторінка 78: Відсутній підзаголовок таблиці перед підзаголовком 4.1

Сторінка 80: Підпис не вирівняний по лівому краю таблиці з відступом у 1-20 см.

Сторінка 86: Відсутнє посилання на 4.5 перед таблицею на цій або попередній сторінці

Сторінка 88: Відсутній або некоректний підзаголовок у 4.7 (має прочитатися з Таблиця 4.6) перед таблицею.

====Перевірка завершена====

Таблиця 4.1 – Пов'язано-логічний контроль

Функції	Варіанти	Параметри	Наслідки
A (PDF/MyPDF)	Висока швидкість роботи, підтримка збірників та текстів, друкований API		Обмежена гнучкість при роботі з табличними структурами

77

78

Таблиця 4.3 – Основні параметри програмного продукту

Величина параметра	Типові значення	Одиниця виміру	Грани	Середня	Крайня
Потенційний об'єм програмного коду	3.1	кількість сторінок	2000	3000	12000
Потенційна кількість функцій програмного продукту	3.2	кількість	100	40	10
Потенційний час розробки найважливіших робочих процесів	3.3	місяці	70	8	4
Складові складові системи	3.4	млн	400	400	100

79

Основні проблеми

Основні проблеми з'ясовані під час розробки.

- Різноманітність структури PDF-файлів через різні текстові редактори (Word, LibreOffice, Google Docs тощо).
- Неконсистентне зберігання тексту в PDF, зокрема формул, що ускладнює перевірку.
- Складність автоматичної перевірки логічних елементів (переліків, узгодженості заголовків) через відсутність чітких правил.
- Недостатня кількість ресурсів.
- Нестабільна та непередбачувана структура PDF-документів, приховані символи, зміщення тексту.




Висновки

- ♦ Проведене дослідження виявило відсутність ефективних готових рішень для автоматичної перевірки оформлення дипломних робіт згідно з ДСТУ, що зумовило необхідність розробки власного підходу.
- ♦ Обраний алгоритмічний підхід став оптимальним компромісом між точністю, складністю реалізації та ресурсозатратністю.
- ♦ Реалізовано приблизно 70% функціональності, що дозволяє проводити базову перевірку оформлення, зменшуючи навантаження на перевіряючих.
- ♦ Основні труднощі пов'язані зі структурною неоднорідністю PDF-документів, непередбачуваністю введених даних і складністю обробки формул та змісту.
- ♦ Незважаючи на технічні обмеження, проєкт підтвердив доцільність і перспективність автоматизації процесу перевірки оформлення.

Напрямки розвитку

- ♦ **Інтеграція комп'ютерного зору**
 - Точна ідентифікація таблиць, рисунків, формул
 - Опрацювання складних макетів документів
- ♦ **Використання великих мовних моделей (LLM)**
 - Перевірка логіки структури та змісту
 - Аналіз стилістики та узгодженості заголовків
 - Автоматична перевірка правильності бібліографії
 - Глибокий парсинг змісту
- ♦ **Комбінування підходів**
- ♦ **Контроль академічної доброчесності**
 - Автоматичне виявлення плагіату
 - Виявлення текстів, згенерованих штучним інтелектом
- ♦ **Розробка адміністративної панелі**
 - Управління користувачами та файлами
 - Перегляд історії перевірок та звітів
 - Конфігурація правил перевірки



Дякую за увагу!

ДОДАТОК В. ДОВІДКА ПРО ВПРОВАДЖЕННЯ

ДОВІДКА


про впровадження «Системи автоматичної перевірки оформлення бакалаврських робіт»
у навчальний процес кафедри математичних методів системного аналізу

Дозвольте повідомити, що дипломна робота Артема Віталійовича Підберізки (група КА-13) на здобуття ступеня бакалавра на тему «Система автоматичної перевірки оформлення бакалаврських робіт» виявилась дуже корисною. Керівником даної бакалаврської роботи є Георгій Олегович Канцедал. В рамках даної роботи була написана і ретельно відтестована на багатьох бакалаврських роботах автоматизована система, яка відслідковує багато основних моментів оформлення робіт і таким чином дозволяє суттєво зменшити навантаження на викладачів, що приймають участь у проведенні нормоконтролів. Ця система вже розгорнута на <https://dstu.labx.pp.ua> і була апробована при перевірці робіт п'яти груп КА-11...15. Апробація виявила важливі моменти.


1. Студенти завантажують свої роботи у вигляді pdf-файлів та отримують детальний звіт, згідно з яким можуть вносити правки в оформлення своєї роботи і таким чином покращують оформлення. В якості такого прикладу наведу такий: в минулих роках на перевірку викладачам надходили декілька робіт, які містили номер на титульній сторінці (згідно з ДСТУ номер на титульній сторінці не пишеться), але в даному році після впровадження системи перевірки я виявив, що робіт з номерами на титульній сторінці мені не поступало.
 2. Така перевірка зменшує навантаження на викладачів, скорочуючи час ручної перевірки, про що свідчать і Георгій Олегович Канцедал, і я.
- З початком проведення нормоконтролів в травні 2025 року систему було відтестовано та внесено деякі правки, які були виявлені доцільними.

Таким чином бакалаврська дипломна робота Артема Віталійовича Підберізки впроваджена в навчальний процес на кафедрі математичних методів системного аналізу і сприяє проведенню нормоконтролів.

Консультант з нормоконтролю,
к. ф.-м.н., доцент кафедри ММСА

 Статкевич В.М.

Завідувач кафедри ММСА

 Тимошук О.Л.

