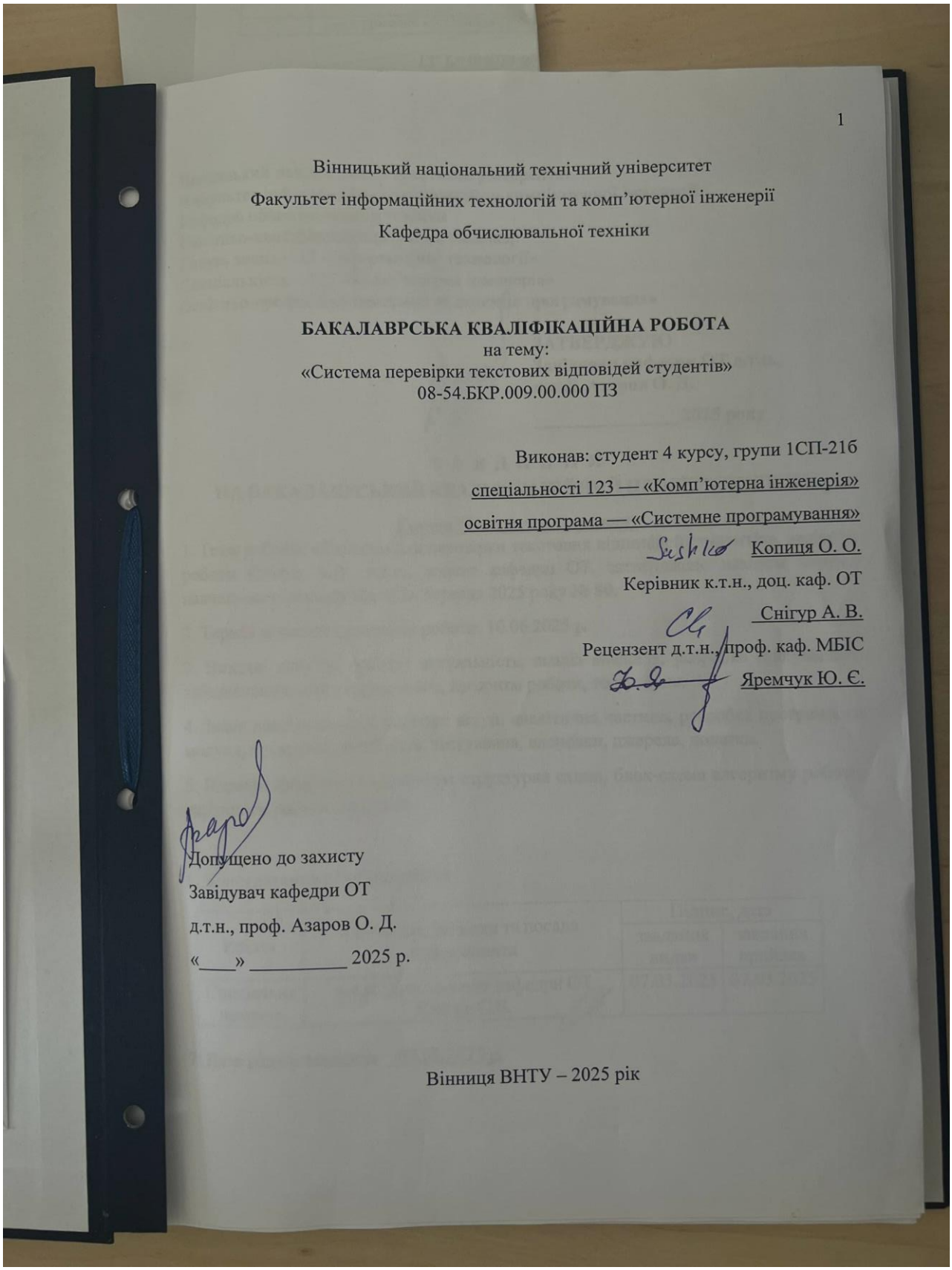


1

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система перевірки текстових відповідей студентів»

08-54.БКР.009.00.000 ПЗ

Виконав: студент 4 курсу, групи 1СП-21б

спеціальності 123 — «Комп'ютерна інженерія»

освітня програма — «Системне програмування»

Копиця О. О.

Керівник к.т.н., доц. каф. ОТ

Снігур А. В.

Рецензент д.т.н., проф. каф. МБІС

Яремчук Ю. Є.

Допущено до захисту

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«___» _____ 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ д.т.н.,
проф. Азаров О. Д.
_____ 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ СТУДЕНТУ

Копиці Олександру Олеговичу

1. Тема роботи: «Система для перевірки текстових відповідей студентів», керівник роботи Снігур А.В., к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «11» березня 2025 року № 80.

2. Термін подання студентом роботи: 10.06.2025 р.

3. Вихідні дані до роботи: актуальність, аналіз аналогів, розробка програмного забезпечення, структурна схема, алгоритм роботи, тестування.

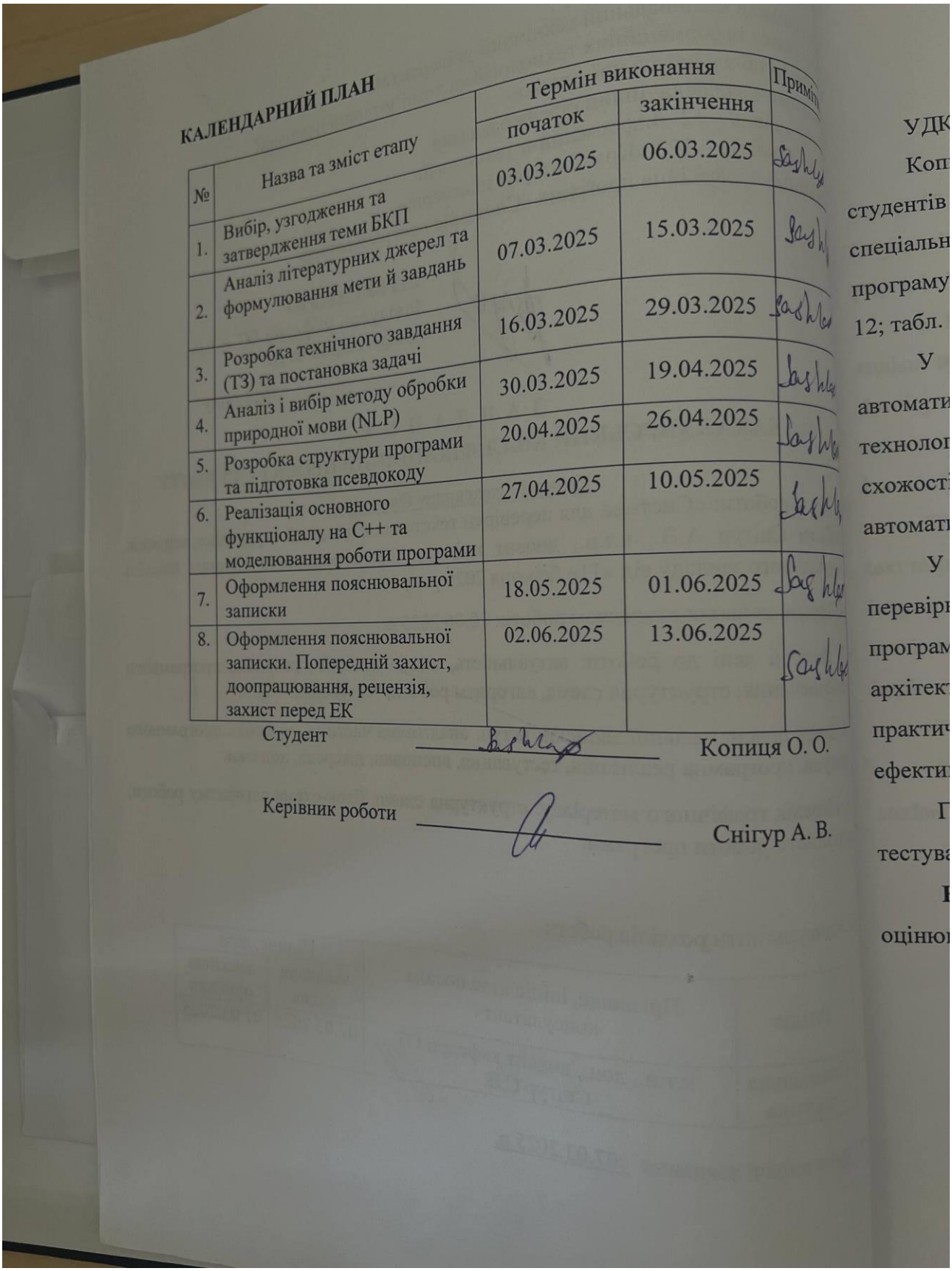
4. Зміст пояснювальної записки: вступ, аналітична частина, розробка програмного модуля, програмна реалізація, тестування, висновки, джерела, додатки.

5. Перелік графічного матеріалу: структурна схема, блок-схема алгоритму роботи, скріншоти роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	к.т.н., доц., доцент кафедри ОТ Снігур С.В.	07.03.2025	07.03.2025

7. Дата видачі завдання 07.03.2025 р.



КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітки
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БКП	03.03.2025	06.03.2025	
2.	Аналіз літературних джерел та формулювання мети й завдань	07.03.2025	15.03.2025	
3.	Розробка технічного завдання (ТЗ) та постановка задачі	16.03.2025	29.03.2025	
4.	Аналіз і вибір методу обробки природної мови (NLP)	30.03.2025	19.04.2025	
5.	Розробка структури програми та підготовка псевдокоду	20.04.2025	26.04.2025	
6.	Реалізація основного функціоналу на C++ та моделювання роботи програми	27.04.2025	10.05.2025	
7.	Оформлення пояснювальної записки	18.05.2025	01.06.2025	
8.	Оформлення пояснювальної записки. Попередній захист, доопрацювання, рецензія, захист перед ЕК	02.06.2025	13.06.2025	

Студент _____ Копиця О. О.

Керівник роботи _____ Снігур А. В.

АНОТАЦІЯ

УДК 009

Копиця О.О. Програма для автоматичної перевірки текстових відповідей студентів з використанням NLP. Бакалаврська кваліфікаційна робота (проєкт) зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2022. 50 с. На укр. мові. Бібліогр.: 20 назв; рис.: 12; табл. 4.

У бакалаврській кваліфікаційній роботі розроблено програму для автоматичної перевірки текстових відповідей студентів з використанням технологій обробки природної мови (NLP). Програма дозволяє визначати ступінь схожості між еталонною відповіддю та відповіддю студента, що сприяє автоматизації перевірки знань у навчальному процесі.

У теоретичній частині роботи проведено аналіз методів NLP та систем перевірки текстових відповідей, обґрунтовано доцільність вибору мови програмування C++ для реалізації проєкту. У проєктній частині описано архітектуру програмного забезпечення та алгоритм обчислення схожості текстів. У практичній частині виконано реалізацію, тестування програми та аналіз її ефективності.

Графічна частина містить блок-схеми, приклади виведення та результати тестування.

Ключові слова: NLP, перевірка відповідей, схожість тексту, автоматизація оцінювання, C++, студентські відповіді, обробка природної мови.

ABSTRACT

UDK 009

Kopytsia O.O. Program for Automated Evaluation of Student Text Responses Using NLP Bachelor's Qualification Thesis (Project) in Specialty 123 "Computer Engineering", Educational Program "System Programming". Vinnytsia: VNTU, 2022. 50 pages. In Ukrainian. Bibliography: 20 titles; Figures: 12; Tables: 4.

In this bachelor's qualification work, a program for the automated evaluation of student text responses using natural language processing (NLP) technologies has been developed. The program is designed to determine the degree of similarity between a reference answer and a student's response, thereby facilitating the automation of knowledge assessment in the educational process.

The theoretical part of the work analyzes NLP methods and systems for evaluating text responses, and it justifies the appropriateness of choosing C++ as the programming language for implementing the project. The project section describes the software architecture and the algorithm for calculating text similarity. The practical part covers the implementation, testing of the program, and analysis of its efficiency.

The graphical section contains flowcharts, output examples, and test results.

Keywords: NLP, response evaluation, text similarity, automated grading, C++, student responses, natural language processing.

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій дедалі більшої актуальності набуває автоматизація процесів, пов’язаних з аналізом та обробкою текстової інформації. Сфера освіти не є винятком — щоденно викладачі перевіряють сотні письмових відповідей студентів, що потребує значних часових і людських ресурсів. У зв’язку з цим розробка програмного забезпечення, яке здатне автоматично оцінювати правильність відповідей на основі їх змісту, стає надзвичайно актуальною. Особливо важливим є створення таких інструментів для оперативної перевірки відкритих відповідей, що не обмежуються одним словом чи варіантом із тесту, а потребують аналізу повноцінних речень.

Актуальність теми зумовлена необхідністю швидкого, об’єктивного та зручного способу оцінювання студентських відповідей на основі порівняння їх зі зразком. Особливу цінність мають рішення, здатні працювати без складних моделей машинного навчання, проте ефективно виконувати базові завдання аналізу тексту. Запропонована програма виконує оцінку схожості відповіді студента зі зразком, ґрунтуючись на простому, але дієвому підході: обчисленні частки співпадіння ключових слів.

Аналіз останніх досліджень і публікацій свідчить про активний розвиток систем автоматизованого оцінювання в галузі освіти. Багато наукових праць висвітлюють складні підходи до автоматичної перевірки, включаючи застосування штучного інтелекту, нейронних мереж та глибинного навчання. Втім, ці методи потребують великих обчислювальних ресурсів і складного налаштування. Натомість, ефективні спрощені методи, засновані на аналізі ключових слів і елементах Natural Language Processing (NLP), часто залишаються недооціненими. Саме така модель аналізу використовується в цій роботі.

Зв’язок роботи з науковими програмами, планами, темами полягає у підтримці загального напрямку цифровізації освіти та впровадження алгоритмів обробки природної мови у програмне забезпечення для навчальних цілей. Дана бакалаврська робота безпосередньо вписується в рамки розвитку освітніх

технологій, орієнтованих на створення адаптивних інструментів для викладачів та студентів.

Метою дослідження є розробка програмного засобу для аналізу правильності відкритих відповідей студентів на основі простого алгоритму перевірки схожості за ключовими словами. Програма має на меті забезпечити базовий рівень автоматизованої перевірки, придатний для використання у внутрішніх навчальних системах та експериментальних освітніх середовищах.

Завдання дослідження полягають у:

– вивченні та застосуванні методів обробки природної мови (NLP) для аналізу тексту;

– реалізації алгоритму токенізації та очищення тексту від зайвих символів;

– створенні функції порівняння та обчислення відсотка схожості;

– інтерпретації результату шляхом класифікації відповідей (правильна, частково правильна, неправильна);

– тестуванні коректності роботи програми на прикладах.

Об’єктом дослідження є процес автоматизованої перевірки текстових відповідей.

Предметом дослідження є алгоритми аналізу схожості текстів та методи попередньої обробки природної мови, реалізовані в умовах простого структурного програмування.

Методи дослідження, використані в роботі, включають: токенізацію (розбиття тексту на окремі слова), нормалізацію тексту (перетворення до нижнього регістру, видалення розділових знаків), використання множин для визначення унікальних ключових слів, обчислення відсоткової схожості, логічну класифікацію результату. Програмна реалізація здійснена мовою C++ із використанням базових бібліотек та функцій Windows API для коректного відображення UTF-8.

Наукова новизна отриманих результатів полягає у реалізації практичного підходу до оцінювання правильності відповідей без використання складних моделей машинного навчання, а також у поєднанні традиційних методів програмування з елементами обробки природної мови.

Апробація результатів кваліфікаційної роботи здійснювалася в рамках участі у Науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії ВНТУ (2025), де було представлено тези доповіді, присвячені архітектурі та принципам реалізації комп’ютерної системи автоматичної перевірки текстових відповідей студентів.

Публікація. За матеріалами виконаної бакалаврської кваліфікаційної роботи підготовлено публікацію на тему «Комп’ютерна система автоматичної перевірки текстових відповідей студентів» у збірнику тез Науково-технічної конференції факультету інформаційних технологій та комп’ютерної інженерії ВНТУ (2025). У публікації представлено опис алгоритмічного підходу до оцінювання схожості текстів, приклади роботи системи, а також розглянуто особливості реалізації програмного забезпечення в середовищі програмування C++.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми

У сучасному освітньому просторі стрімко зростає потреба в ефективних засобах автоматизації процесів навчання, контролю знань і надання зворотного зв’язку. Особливої актуальності ця проблема набуває в умовах масової діджиталізації навчальних закладів, зростання кількості студентів, а також переходу багатьох освітніх платформ у дистанційний та гібридний формати. Внаслідок цього значне навантаження лягає на викладачів, які змушені перевіряти великі обсяги текстових відповідей, витрачаючи на це чимало часу та зусиль. Саме тому розробка програмного забезпечення, що дозволяє автоматизувати процес перевірки відповідей студентів за заданими критеріями, є надзвичайно актуальною.

Програмна реалізація, представлена в межах цієї бакалаврської кваліфікаційної роботи, дозволяє здійснювати базовий аналіз тексту студентської відповіді порівняно з еталонною, виявляючи схожість на основі ключових слів. Такий підхід дозволяє об’єктивно та швидко оцінити якість відповіді без необхідності повної ручної перевірки з боку викладача. Це є особливо корисним у дисциплінах, де використовується текстовий формат відповідей, наприклад, у теоретичних курсах з інформатики, історії, права, біології, а також у тестуванні на знання термінології.

Крім того, актуальність теми обумовлюється тим, що сучасна вища освіта вимагає індивідуального підходу до навчання студентів. Автоматизована перевірка дозволяє швидко надати кожному студенту результат, що в свою чергу сприяє формуванню більш ефективного зворотного зв’язку та поліпшенню засвоєння матеріалу. Більше того, використання подібного ПЗ здатне знизити ризик упередженого оцінювання, забезпечуючи рівні умови для всіх студентів.

На тлі розвитку технологій штучного інтелекту (AI) та обробки природної мови (Natural Language Processing, NLP) дедалі більше уваги приділяється інтеграції цих підходів у навчальні платформи. Хоча програмна реалізація, описана в цій роботі, використовує базову логіку порівняння ключових слів без глибокого

синтаксичного чи семантичного аналізу, вона все ж закладає основу для подальшого розширення функціональності — наприклад, шляхом додавання NLP-методів для більш тонкої оцінки контексту, логічності та повноти відповіді.

Ключовий принцип роботи даного ПЗ полягає в тому, що система не зосереджується на повному збігу речень, а навпаки, працює за принципом виявлення основних термінів або понять, що містяться у відповіді. Це дозволяє гнучко враховувати варіативність формулювань при збереженні суті, що є значною перевагою порівняно з жорстко шаблонізованими системами тестування. Саме подібна адаптивність до природної мови та її варіацій робить тему особливо актуальною на сучасному етапі розвитку освітніх технологій.

Окремої уваги заслуговує практична реалізація програмного забезпечення. В основі розробленого коду лежить мова програмування C++, що дозволяє ефективно працювати з текстовими потоками, векторами, множинами та здійснювати високошвидкісну обробку даних. Програма працює з введенням у консольному режимі, що забезпечує її простоту та універсальність. Функціонал реалізує обробку введеного студентом тексту, видаляючи пунктуаційні знаки, зводячи слова до нижнього регістру та порівнюючи їх з ключовими словами з еталонної відповіді. Результатом роботи програми є виведення відсотка схожості та вердикт щодо правильності, часткової правильності або помилковості відповіді.

Крім того, зазначимо, що програма може бути використана як у навчальному середовищі, так і в системах самоперевірки знань, включно з адаптивними навчальними платформами та інтерактивними підручниками. У перспективі ця ідея може бути розширена та застосована в мобільних або веб-застосунках, де користувачі можуть отримувати швидку та об'єктивну оцінку власних відповідей, що ще більше підвищує соціальну значущість запропонованої теми.

Таким чином, актуальність розробки полягає не лише у прагматичній користі для викладачів та студентів, але й у потенціалі до подальшого розвитку та інтеграції з сучасними інтелектуальними системами навчання. Це робить дану бакалаврську роботу вагомим внеском у напрямі цифровізації освіти та автоматизації оцінювання знань.

1.2 Мета і завдання дослідження

Упродовж останнього десятиліття у сфері освітніх технологій активно розвивається напрям автоматизації оцінювання знань студентів, зокрема в аспекті обробки текстових відповідей. Традиційні системи тестування здебільшого обмежуються варіантами з вибором правильної відповіді (тести типу multiple choice), однак такі формати не завжди дозволяють повною мірою оцінити глибину розуміння навчального матеріалу. Тому науковці дедалі частіше звертаються до розробки методів автоматизованої перевірки відкритих відповідей — коротких або розгорнутих текстів, що вимагає залучення інструментів обробки природної мови (Natural Language Processing, NLP).

У працях [1], [2] дослідники зазначають, що одним із найперспективніших підходів до автоматичного аналізу відповідей є виявлення ключових слів та термінів, які мають бути присутні у відповідях студентів. Такий підхід демонструє достатню гнучкість, дозволяючи враховувати синонімічні заміни, варіативність побудови речень і навіть граматичні помилки, якщо аналіз не ґрунтується на синтаксичній строгості.

Значний інтерес до розробки таких систем підтверджується міжнародними проєктами, зокрема Automated Short Answer Grading (ASAG) — платформи, які застосовують як базові ключові шаблони, так і методи машинного навчання. У роботі [3] описано ефективність використання векторних моделей слів (наприклад, Word2Vec або TF-IDF), які дозволяють оцінювати семантичну схожість між відповідями студентів і еталонними текстами.

Водночас, як зазначають автори [4], для впровадження повноцінного NLP-аналізу необхідні суттєві обчислювальні ресурси, а також великий обсяг навчальних даних. У цьому контексті актуальним залишається підхід, при якому оцінювання відбувається шляхом прямого порівняння ключових понять. Саме цей принцип реалізовано в розробленій програмі, описаній у межах даної роботи.

Також слід звернути увагу на результати українських дослідників у галузі цифровізації освіти. У роботах [5], [6] аналізуються переваги інтеграції

програмного забезпечення з можливістю перевірки відкритих текстових відповідей у системи дистанційного навчання, такі як Moodle, Open edX, Google Classroom. Водночас, автори наголошують на обмеженнях існуючих засобів, які або не підтримують українську мову на належному рівні, або потребують додаткової адаптації для якісної перевірки коротких текстів.

Загалом, огляд наукових джерел дозволяє зробити висновок, що нині існує потреба у простих і водночас ефективних рішеннях, які не потребують великих обчислювальних потужностей, але здатні забезпечити базовий аналіз правильності студентських відповідей. Подібне програмне забезпечення може стати зручним інструментом у навчальному процесі, а також слугувати основою для майбутніх досліджень у сфері застосування NLP і ШІ в освіті.

З огляду на вищезазначене, дана бакалаврська робота спрямована на створення доступного та універсального рішення, яке базується на перевірці ключових слів у відповідях студентів. Такий підхід є логічним кроком у напрямі автоматизації освітніх процесів, а також відповідає сучасним трендам у галузі інформаційних технологій.

Крім того, створення подібного програмного забезпечення сприяє підвищенню ефективності освітнього процесу за рахунок автоматизації оцінювання письмових відповідей студентів. Це особливо актуально для масових онлайн-курсів, де перевірка великої кількості текстових відповідей вручну потребує значних ресурсів.

Важливо зазначити, що запропонований підхід може бути використаний не лише у навчальних закладах, а й у корпоративних освітніх програмах, де необхідно швидко перевіряти знання працівників. Розширення можливостей аналізу текстів, включаючи підтримку додаткових мовних моделей або алгоритмів NLP, дозволить зробити систему ще більш універсальною та адаптивною до різних навчальних середовищ.

У майбутньому можливе вдосконалення алгоритму шляхом інтеграції семантичного аналізу, який дозволить краще оцінювати змістовну схожість відповідей навіть за умови варіативності формулювань.

1.3 Архітектура комп’ютерних систем автоматичної перевірки

Сучасні комп’ютерні системи автоматичної перевірки відповідей студентів побудовані за чітко визначеною архітектурною структурою, яка дозволяє забезпечити стабільність, масштабованість та простоту у використанні. Як правило, такі системи мають трирівневу архітектуру, що складається з клієнтської частини, серверної частини та бази даних. Кожен із цих компонентів виконує окрему, але взаємозалежну функцію, утворюючи єдину інформаційну систему.

Клієнтська частина — це інтерфейс користувача, через який студент або викладач взаємодіє з програмою. У випадку нескладної реалізації, як, наприклад, консольна версія, інтерфейс обмежується текстовим введенням та виведенням. Проте у більш складних реалізаціях використовуються графічні інтерфейси, веб-браузери або навіть мобільні додатки. Основне завдання клієнтської частини — передача введених даних на сервер та виведення результатів перевірки користувачеві.

Серверна частина є центральним компонентом системи. Вона відповідає за обробку запитів, логіку перевірки відповідей, генерацію результатів та взаємодію з базою даних. У традиційних системах ця частина реалізується у вигляді серверного застосунку, що запускається в окремому середовищі (наприклад, Python Flask, Node.js або Java Spring). У спрощеному варіанті роль сервера може виконувати простий скрипт, який обробляє дані локально без мережевої інфраструктури.

База даних — це сховище інформації про тести, запитання, відповіді, результати, користувачів. У реальних умовах це можуть бути SQL-сервери (MySQL, PostgreSQL) або NoSQL-рішення (MongoDB). У демонстраційній версії база даних часто представлена у вигляді текстових файлів, таких як .txt або .json, які імітують зберігання інформації у більш доступний для початкової реалізації спосіб. Такий підхід дозволяє створити повністю функціональну систему навіть без складних налаштувань.

Схематично типову архітектуру можна уявити так:

— клієнт (студент) → вводить відповідь;

— сервер → приймає відповідь, порівнює з еталонною, формує результат;

— база даних → зберігає питання, правильні відповіді, результати.

Система може бути як централізованою, так і розподіленою. У централізованій архітектурі вся логіка й база даних розміщені на одному сервері, тоді як у розподіленій частини системи можуть бути розміщені на різних вузлах, забезпечуючи кращу продуктивність та стійкість до збоїв.

Окрім основних компонентів, сучасні архітектури таких систем передбачають додаткові модулі, зокрема:

— модулі аутентифікації користувачів;

— аналітичні модулі (для обчислення середнього балу, статистики помилок тощо);

— інструменти адміністрування тестів.

Завдяки продуманій архітектурі забезпечується ефективне функціонування всієї системи, її легка масштабованість та можливість інтеграції в більші освітні платформи. Навіть спрощена реалізація, у якій сервером виступає логічний блок, а база даних — звичайний блокнот, дає уявлення про структуру справжніх програмних рішень, застосовуваних у закладах освіти.

1.4 Роль комп’ютерних систем у цифровій трансформації освіти

Цифровізація освіти — один із ключових напрямів розвитку сучасного суспільства, і комп’ютерні системи автоматичної перевірки відіграють у цьому процесі важливу роль. У реаліях, де кількість студентів зростає, а вимоги до якості освіти стають все суворішими, автоматизовані системи перевірки дозволяють зменшити навантаження на викладачів, підвищити точність оцінювання та створити нові можливості для аналізу навчального процесу.

Передусім, такі системи усувають людський фактор. Ручна перевірка відповідей, особливо у великих групах, часто призводить до помилок, неточностей або навіть суб’єктивних рішень. Комп’ютер же виконує оцінювання неупереджено: він порівнює відповідь студента з еталонною, використовуючи чіткий алгоритм, і

виводить результат без упереджень. Це сприяє справедливості в оцінюванні та підвищує довіру до результатів.

Другою перевагою є економія часу. У традиційній системі викладач витрачає години на перевірку контрольних, тестів або іспитів. Автоматизована система робить це миттєво. Більше того, вона може одразу надавати студенту зворотний зв'язок — які відповіді були правильними, у чому допущено помилку, яка правильна відповідь. Таким чином, система не лише перевіряє, але й навчає.

Ще однією важливою функцією є аналітика. Зібрані дані про помилки, рівень знань, успішність певних груп студентів можуть бути використані адміністрацією закладу для прийняття рішень щодо покращення навчальних програм. Наприклад, якщо значна частина студентів не справляється з певним розділом, це сигнал для перегляду відповідного матеріалу.

У контексті дистанційного навчання та змішаних форматів, такі системи стають незамінними. Вони дозволяють проводити тестування незалежно від місця перебування студента, що особливо актуально в умовах глобальних пандемій, військових конфліктів або інших форс-мажорних обставин.

Також важливо підкреслити, що автоматичні системи стимулюють інтерес до IT серед студентів. Наприклад, вивчаючи принципи роботи таких програм, студенти починають краще розуміти структуру програмного забезпечення, алгоритми, способи обробки даних. Це може мотивувати їх до глибшого вивчення інформатики, програмування або навіть до створення власних освітніх продуктів.

Приклади використання таких систем:

— Moodle з плагінами автоматичної перевірки тестів(рис1.1);

Рисунок 1.1 – Система Moodle

— Google Forms з автоперевіркою(рис1.2);

Рисунок 1.2 – Система Google Forms

— власні проєкти студентів на Python або JavaScript(рис1.3)(рис1.4).

Рисунок 1.3 – Система Python

Рисунок 1.4 – Система JavaScript

— спеціалізовані сервіси на кшталт Testportal, Exam.net(рис1.5) (рис1.6).

Рисунок 1.5 – Система testportal

Рисунок 1.6 – Система exam.net

Таким чином, комп’ютерні системи автоматичної перевірки — це не просто зручний інструмент, а повноцінний елемент освітньої інфраструктури. Вони змінюють сам підхід до контролю знань, роблять освіту більш технологічною, ефективною, доступною і гнучкою. Їхній вплив продовжує зростати, і можна

впевнено сказати, що в найближчому майбутньому саме автоматизовані підходи формуватимуть новий освітній стандарт.

1.5 Порівняльний аналіз комп'ютерних систем автоматичної перевірки відповідей студентів

У сучасних умовах розвитку дистанційного навчання та цифровізації освітнього процесу зростає потреба у впровадженні комп’ютерних систем, здатних автоматично перевіряти відповіді студентів. Такі системи не лише оптимізують навчальний процес, а й дозволяють викладачам економити час, покращуючи загальний контроль знань студентів.

Системи автоматичної перевірки відповідей можуть істотно відрізнятися за своїм функціональним наповненням, алгоритмами оцінки, інтеграцією з іншими освітніми платформами та підходами до аналізу. У цьому підрозділі буде проведено порівняльний аналіз найпоширеніших типів таких систем, а також розглянуто переваги й недоліки кожної з них з точки зору практичного використання в умовах закладу вищої освіти.

Основні критерії для порівняння систем.

Аналіз проводився за такими критеріями:

— тип відповідей, що підтримуються (тестові, відкриті, програмний код тощо);

— рівень інтерактивності (чи є зворотний зв’язок, підказки тощо);

— можливість навчання на даних (адаптація до стилю відповіді);

— точність перевірки (відсоткове співпадіння, семантична схожість);

— інтеграція з іншими платформами (Google Classroom, Moodle, API);

— тип реалізації (вебсервіс, десктопна програма, скрипт тощо);

— відкритість коду та можливість модифікації;

— наявність БД та способи збереження результатів.

За даними критеріями було складено таблицю порівняння власної системи з аналогами (таблиця 1.1)

Таблиця 1.1 – Порівняння власної системи з аналогами

Характеристика / Назва системи	Підтримка різних типів відповідей	Інтерактивність	Інтеграція з платформами	Простота реалізації	Гнучкість / модифікація	Наявність бази даних
Moodle Quiz	+	+	+	—	—	+
E-xaminer	+	+	+—	—	—	+
Google Forms + Script	—	—	+	+	+	+
CodeGrade	—	+—	+	—	—	+
Індивідуальна розробка	+—	—	+—	+	+	+

Moodle Quiz — це система тестування, яка дозволяє створювати різні типи запитань і автоматично перевіряти відповіді студентів.

Google Forms із додатковим скриптом дає змогу збирати відповіді студентів і автоматично їх перевіряти без складного налаштування.

CodeGrade — це вебплатформа, яка аналізує програмний код студентів і виставляє оцінки на основі тестів.

E-xaminer — це програмне забезпечення для проведення тестів та іспитів, яке підтримує автоматичну або ручну перевірку відповідей.

1.6 Огляд методів оцінювання подібності текстів

1.6.1 Лексичні методи

Лексичні методи — це найпростіші з точки зору реалізації підходи до аналізу схожості, які базуються на безпосередньому порівнянні символів, слів або фраз. Їх основною перевагою є швидкість обчислень і низькі обчислювальні вимоги. Однак

такі методи практично не враховують контекст або синонімічність, що обмежує їх точність.

Основні лексичні підходи:

— рядкова схожість (string similarity);

— редакційна відстань (Levenshtein distance);

— cosine similarity (у простому вигляді– пошук точних збігів фраз або речень. Особливо актуально при виявленні прямого копіювання тексту.

Показує кількість операцій (вставка, видалення, заміна), які потрібно виконати, щоб один рядок перетворити в інший. Порівнює множини слів або n-грам, розраховуючи відношення спільних елементів до об’єднання множин;

Використовується на основі векторів, сформованих із частотного представлення слів (наприклад, Bag of Words);

Текст розбивається на підрядки довжиною n (наприклад, біграми чи триграми), які потім порівнюються між собою. Це дозволяє частково враховувати порядок слів;

Незважаючи на обмеження, лексичні методи залишаються популярними в освітніх та адміністративних системах, де важлива швидкість обробки великого обсягу інформації. Їх часто застосовують як попередній фільтр перед більш глибоким аналізом.

1.6.2 Статистичні методи

Статистичні методи оцінювання подібності текстів базуються на частотному аналізі слів або фраз, що дає змогу враховувати як кількісні, так і деякі якісні характеристики тексту. Ці підходи уже значно точніші за просту лексичну перевірку, адже дозволяють оцінювати не лише формальну, але й контекстуальну близькість.

Найпоширеніші статистичні підходи:

— TF-IDF (Term Frequency – Inverse Document Frequency).

Формула:

$$\mathrm{TF}-\mathrm{IDF}(\mathrm{t},\mathrm{d})=\mathrm{TF}(\mathrm{t},\mathrm{d})\times\log(\mathrm{DF}(\mathrm{t})\mathrm{N}),$$

де t — термін;

d — документ;

N — загальна кількість документів;

DF(t)— кількість документів, у яких зустрічається термін t.

Цей метод використовує сингулярне розкладення матриці TF-IDF для зниження її розмірності та виявлення прихованих семантичних зв’язків між словами й документами. LSA дозволяє виявити, що тексти є схожими, навіть якщо в них не вживаються одні й ті ж слова, але вони мають схожий смисловий контекст;

Це один з базових способів перетворення тексту у числову форму для подальшого аналізу. Алгоритм враховує частоту появи слова в окремому документі (TF) і зворотну частоту появи цього слова в усіх документах колекції (IDF). Завдяки цьому слова, що часто зустрічаються у багатьох документах (наприклад, "це", "якщо", "або"), отримують менше значення ваги. Метод оцінює відстань між ймовірнісними розподілами слів у двох текстах. Це симетрична міра, побудована на основі ентропій Шеннона.

Після побудови векторів текстів за допомогою TF-IDF можна обчислити косинусну подібність між ними. Якщо вектори спрямовані в один бік — тексти схожі.

Такі методи добре масштабуються і можуть використовуватися для кластеризації великої кількості документів. Їх особливо цінують за простоту реалізації та достатньо високу точність у загальному випадку. Водночас вони обмежені у розумінні глибокого змісту тексту, не враховують синонімію або полісемію слів.

У контексті автоматичної перевірки студентських відповідей метод TF-IDF у поєднанні з LSA дає можливість оцінити відповідність не лише за ключовими словами, а й за загальним тематичним змістом, що особливо корисно при перевірці відкритих або розгорнутих відповідей. Незважаючи на обмежену здатність враховувати граматичні конструкції чи логіку викладення, ці методи залишаються

надійним та обґрунтованим вибором для базових освітніх систем перевірки, зокрема у випадках обмежених обчислювальних ресурсів.

1.6.3 Семантичні методи

Семантичні методи — це підходи, що аналізують не просто збіг слів, а значення, які за ними стоять. Вони дозволяють встановити, що фрази типу "студент здав іспит" і "екзамен був зданий учнем" є подібними, навіть якщо їхні слова відрізняються.

Ключові технології:

— Word Embeddings (векторизація слів);

— перетворити кожне слово на вектор у багатовимірному просторі, так щоб синоніми або близькі за змістом слова розташовувалися поруч.

Найвідоміші моделі:

— Word2Vec — навчається на завданні передбачення сусідніх слів (CBOW або Skip-Gram);

— GloVe — заснована на глобальній статистиці слів у корпусі;

— FastText — бере до уваги також морфологію слова (корені, префікси, суфікси).

Схожість між словами або цілими текстами обчислюється як косинусна подібність між середніми векторами або за допомогою спеціалізованих метрик.

— BERT (Bidirectional Encoder Representations from Transformers).

Це одна з найпотужніших моделей обробки природної мови, створена компанією Google. BERT розуміє контекст завдяки двонапрямному аналізу тексту. Наприклад, слово “банк” отримає різне представлення залежно від того, чи мова йде про фінансову установу, чи берег річки.

Для оцінки подібності текстів можна використовувати:

— Sentence-BERT — оптимізований варіант BERT для порівняння речень;

— Embedding Similarity — обчислення подібності між векторами речень;

— Semantic Textual Similarity (STS) — це спеціальні задачі та датасети, де моделі навчаються на прикладах фраз із різним ступенем схожості.

Використовуються як основа для навчання систем машинного перекладу, пошуку, перевірки плагіату тощо;

— Ontology-based similarity — підходи, що базуються на використанні онтологій (наприклад, WordNet), дозволяють виявити ієрархічні зв’язки між поняттями. Наприклад, терміни "кіт", "пес" є частинами ширшої категорії "тварина".

Семантичні методи дозволяють системі «розуміти» текст майже так само, як людина, що робить їх дуже ефективними. Недоліки — висока обчислювальна складність, потреба у великих обсягах даних для навчання моделей та залежність від якості мовних корпусів.

У практичних застосуваннях семантичні методи відіграють ключову роль у таких задачах, як автоматичне резюмування тексту, класифікація повідомлень, побудова діалогових систем, а також у сфері освітніх технологій — для автоматичної перевірки письмових відповідей студентів. Особливо ефективними ці методи є у випадках, коли потрібно оцінити не просто наявність конкретних слів у відповіді, а здатність студента передати загальний зміст або продемонструвати розуміння поняття іншими словами.

Сучасні мовні моделі, такі як GPT, T5 або RoBERTa, навчені на масштабних корпусах і здатні обробляти мільйони параметрів, що дозволяє їм вловлювати тонкі нюанси значення навіть у коротких фразах. Інтеграція таких моделей у системи оцінювання потребує ретельної адаптації, але значно підвищує якість аналізу.

Варто зазначити, що використання семантичних методів також дозволяє мінімізувати залежність від жорстких правил або словників, які зазвичай обмежують гнучкість системи. Наприклад, система може врахувати, що відповіді “вода кипить при 100°C” та “температура кипіння води становить сто градусів” мають однакову сутність, хоч і сформульовані по-різному.

Проте, впровадження таких методів у реальні освітні середовища вимагає не лише технічних ресурсів, а й етичного контролю, адже автоматизовані висновки не завжди прозорі для перевірки викладачем. Також існує ризик упередженості або

нерівномірної оцінки, якщо модель була недостатньо добре навчена на прикладах з певної мови чи стилістики.

Для україномовного контенту існує проблема обмеженої кількості якісних корпусів, що гальмує адаптацію найновіших моделей до локальних освітніх завдань. Проте розробка національних лінгвістичних ресурсів триває, що відкриває перспективи широкого впровадження семантичних підходів у навчальні системи майбутнього.

1.6.4 Гібридні підходи

Гібридні підходи поєднують у собі кілька методів оцінювання схожості текстів — лексичних, синтаксичних і семантичних. Вони дозволяють компенсувати недоліки окремих підходів і отримати більш точний та адаптивний результат.

Зокрема, лексико-семантичні гібриди часто використовуються в системах, які мають обмежений обсяг навчальних даних або повинні працювати з багатомовними текстами. В такому випадку, наприклад, порівняння виконується як на основі ключових слів (ключової лексики), так і через обчислення векторної схожості на основі вбудованих моделей типу Word2Vec або FastText.

Ще одним прикладом є поєднання n-грамного аналізу (який дозволяє точно виявляти лексичні збіги) з Latent Semantic Analysis (LSA) або BERT-ембедінгами, що дозволяє враховувати контекстуальну інформацію.

Такі гібриди активно застосовуються в системах:

— автоматичного оцінювання відповідей студентів;

— виявлення плагіату;

— рекомендативних системах;

— чат-ботах із функцією розпізнавання намірів користувача.

У контексті даної роботи, самостійно реалізована програма використовує базовий гібридний підхід, який поєднує лексичний аналіз (розбиття тексту на слова, очищення від пунктуації, приведення до нижнього регістру) із порівнянням за допомогою множини ключових слів. Хоча в програмі не реалізовано глибинних

семантичних моделей, її логіка вже дозволяє імітувати спрощену версію змішаного підходу, адаптовану під локальні освітні завдання (рис.1.1).

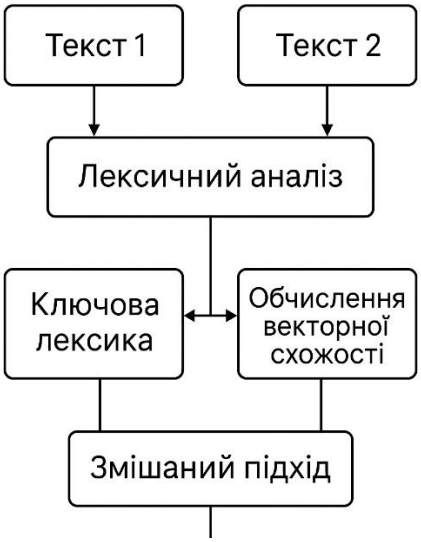


Рисунок 1.1 – Загальна схема гібридної моделі визначення схожості

Таким чином, гібридні методи демонструють високу гнучкість та ефективність у вирішенні завдань, де просте текстове порівняння є недостатнім, і саме тому залишаються актуальним напрямком досліджень у сфері обробки природної мови.

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Постановка задачі розробки програмного забезпечення

Розробка програмного забезпечення для оцінювання схожості відповідей студентів передбачає створення простої, ефективної та інтуїтивно зрозумілої системи, яка здатна автоматизовано аналізувати введений текст користувача (відповідь студента) та зіставляти його з еталонною відповіддю. Актуальність такої задачі пояснюється необхідністю зменшення суб’єктивного чинника при перевірці письмових робіт, підвищення швидкості оцінювання та підтримки дистанційного навчання (рис.1.2).

Рисунок 1.2 – Загальна схема гібридної моделі визначення схожості

Основні вимоги до майбутньої системи:

— зрозумілий інтерфейс — взаємодія з користувачем повинна бути простою: введення еталонної відповіді, введення відповіді студента та відображення результату оцінювання;

— можливість обробки довільного тексту — система повинна приймати відповіді різного обсягу та граматичної структури;

— лексичний аналіз — реалізація механізму попередньої обробки тексту, видалення пунктуаційних знаків, переведення до нижнього регістру, розбиття на слова;

— порівняння за ключовими словами — перевірка наявності ключових слів з еталонної відповіді у студентській відповіді;

— оцінювання у відсотках — система повинна повертати числовий показник схожості (від 0 до 100%), що дає змогу швидко зрозуміти рівень відповідності;

— описовий вердикт — додаткова текстова оцінка (наприклад, "правильна відповідь", "частково правильна", "неправильна"), що ґрунтується на відсотковому порозі.

Завдання, які необхідно вирішити під час реалізації:

— вибір мови програмування, яка дозволить швидко реалізувати задуманий функціонал;

— розробка та реалізація алгоритму попередньої обробки тексту (токенізація, нормалізація, очищення);

— формування множини ключових слів з еталонної відповіді;

— розробка функції обчислення відсотку збігів між ключовими словами еталону та словами студентської відповіді;

— побудова логіки оцінювання результату на основі порогових значень схожості (наприклад, ≥70% — правильна відповідь);

— надання користувачеві результату у зручному форматі, виведення в консоль.

Загальна мета полягає в реалізації ефективного, локального інструменту для використання в навчальному середовищі викладачем або студентом для самоперевірки. Хоча запропонована система не використовує складних моделей машинного навчання, вона надає базовий функціонал, достатній для початкового аналізу текстової схожості з практичною точністю.

2.2 Вибір середовища розробки та мови програмування

Початковий етап створення будь-якого програмного забезпечення передбачає вибір відповідного інструментарію — як з точки зору реалізації технічного задуму, так і забезпечення зручності для розробника. У рамках цієї кваліфікаційної роботи було обрано мову програмування C++, а також стандартну консольну розробку під операційну систему Windows, що обумовлено низкою вагомих причин.

По-перше, C++ — одна з найбільш потужних мов системного рівня, що дозволяє тонко контролювати пам'ять і процес обробки даних. Хоча для багатьох завдань аналізу тексту зазвичай використовують високорівневі мови (наприклад, Python із бібліотеками NLP), вибір C++ у цьому проєкті забезпечує високу швидкість виконання програми, що особливо актуально при потенційному масштабуванні.

По-друге, робота виконується в межах освітньої програми, де передбачено ознайомлення з класичними засобами створення ПЗ, зокрема з використанням стандартної бібліотеки STL, основ операційної системи Windows та обробки текстових потоків на низькому рівні.

Для розробки та компіляції програми було обрано середовище Microsoft Visual Studio, яке забезпечує повну інтеграцію з Windows API, зручний дебагінг та багатофункціональний редактор коду.

Отже, поєднання мови C++ і середовища Visual Studio дозволило реалізувати компактне, ефективне та зрозуміле рішення для перевірки схожості текстових відповідей.

2.3 Структура програмного забезпечення

Консольна реалізація дозволила зробити програму максимально простою для запуску і адаптованою до будь-якого середовища. C++ також забезпечує гнучкість і можливість подальшого переходу до об’єктно-орієнтованої структури або модульної архітектури при потребі

Для реалізації системи було обрано мову програмування C++ — завдяки її високій швидкодії, наявності засобів обробки тексту та можливості роботи з

файлами. Як середовище розробки використовувалась Microsoft Visual Studio, що забезпечує зручну налагоджувальну платформу, а також сумісність з бібліотеками Windows API.

Розроблене програмне забезпечення має лінійно-послідовну архітектуру, в основі якої — етапи обробки введеного тексту, а саме:

— отримання еталонної відповіді (викладача або шаблону);

— отримання відповіді студента;

— попередня обробка обох текстів (токенізація, очищення від пунктуації, перетворення до нижнього регістру);

— обчислення ступеня схожості;

— виведення результату з поясненням (правильна, частково правильна, неправильна відповідь) (рис 2.1).

Рисунок 2.1 - Представлена загальна блок-схема архітектури програми

Також на рисунку 2.2 зображена структура системи:

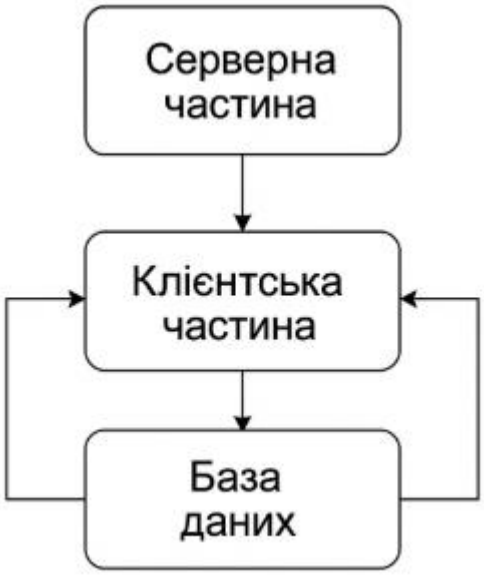


Рисунок 2.2 – Структура системи автоматичної перевірки текстових відповідей студентів

Ця структура дозволяє легко масштабувати або вдосконалити програму в майбутньому — зокрема, додати обробку синонімів, інтеграцію зі штучним інтелектом або підтримку формальних граматичних конструкцій.

2.4 Реалізація алгоритму визначення схожості відповідей

Ключовим елементом програми є алгоритм визначення ступеня схожості, реалізований за допомогою стандартних засобів мови C++.

На етапі обробки вхідних даних використовується процедура токенізації — тобто розбиття введеного тексту на окремі слова (токени), з одночасним видаленням пунктуаційних знаків та приведенням кожного слова до нижнього регістру. Це дозволяє уникнути помилок порівняння, пов’язаних із регістром чи зайвими символами.

Далі виконується порівняння слів відповіді студента з ключовими словами еталонного тексту. Для цього:

— еталонні слова зберігаються у множині (std::set), що дозволяє швидко здійснювати пошук;

— кожне слово з відповіді студента перевіряється на наявність у множині;

— рахується кількість збігів;

— результат обчислюється як відсоток збігів відносно кількості ключових слів.

Цей підхід хоча й простий, однак забезпечує базову, але надійну перевірку, не потребуючи складних обчислень.

2.5 Опис основних функцій програми

У програмі реалізовано кілька важливих функцій, кожна з яких виконує окрему логічну частину системи. Всі функції працюють узгоджено, забезпечуючи автоматизовану перевірку відповідей студентів та збереження результатів.

Функція tokenize(). Ця функція приймає рядок символів як аргумент та повертає вектор слів. Алгоритм її роботи передбачає проходження по кожному символу вхідного рядка. Якщо символ є літерою (латинською або українською), він додається до поточного слова. Якщо символ є розділовим знаком або пробілом — поточне слово завершується та додається до вектора. Таким чином, функція імітує роботу простого лексичного аналізатора, виділяючи «токени» (слова). Усі символи при цьому переводяться до нижнього регістру, що дозволяє ігнорувати регістр при порівнянні. Це підвищує точність аналізу схожості, адже слова «Світло» та «світло» сприймаються однаково.

Функція calculateSimilarity(). Призначена для безпосереднього визначення рівня схожості між відповіддю студента та еталонною відповіддю. На вхід вона отримує два масиви слів (вектори), які попередньо були сформовані за допомогою tokenize(). Далі відбувається порівняння кожного слова з відповіді студента з кожним словом в еталоні. Якщо слова збігаються — лічильник збігів збільшується. По завершенню підрахунку функція обчислює відсоток схожості за формулою: (кількість збігів / загальна кількість слів в еталоні) * 100. Повертається значення з плаваючою точкою типу double, яке і використовується для виведення результату.

Функція saveResultToFile(). Відповідає за збереження результатів роботи у файл. У текстовий файл записується ПІБ студента, відсоток схожості та короткий текстовий висновок. Це дозволяє накопичувати результати впродовж перевірки багатьох студентів. У майбутньому ці дані можна обробити для формування звітів

або графіків. Файл зберігається у тому ж каталозі, де розташована виконувана програма.

Функція main(). Це головна функція, яка керує логікою виконання всієї програми. Вона виконує наступні дії:

— виводить вітальне повідомлення та інструкцію;

— запитує у користувача ПІБ та відповідь;

— застосовує функцію tokenize() до введеного тексту та еталону;

— викликає calculateSimilarity() для визначення рівня збігу;

— аналізує результат і формує короткий висновок: «Відповідь правильна», «Часткова схожість», або «Відповідь не збігається»;

— виводить результат на екран у зручному вигляді;

— викликає saveResultToFile() для збереження інформації.

Таким чином, вся логіка програми побудована навколо простої послідовності: введення → обробка → аналіз → виведення та збереження. Незважаючи на простоту, структура дозволяє розширювати програму у майбутньому, наприклад, шляхом підключення графічного інтерфейсу, підтримки декількох еталонів або розгалуження висновків залежно від тематики відповіді.

2.6 Результати компіляції та відлагодження

Розробка програмного забезпечення супроводжується не лише створенням вихідного коду, але й ретельною перевіркою його працездатності через етапи компіляції та відлагодження. У процесі розробки було використано Visual Studio, яка забезпечила інтегроване середовище розробки та тестування для платформи Win32. Завдяки цьому вдалося отримати детальний вивід інформації про завантаження необхідних модулів і бібліотек, що є важливим критерієм перевірки коректності зв’язування і роботи програми.

При першому запуску проекту, після натискання кнопки «Побудова» (Build), код успішно компілювався без будь-яких помилок. У виводі компілятора були зафіксовані повідомлення про завантаження наступних компонентів(лістинг 2.1):

Лістинг 2.1 – успішне завантаження основного виконуваного файлу

"diplom.exe" (Win32). Загружено "C:\Users\user\source\repos\diplom\Debug\diplom.exe". Символы загружены.

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\ntdll.dll".

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\kernel32.dll".

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\KernelBase.dll".

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\msvcp140d.dll".

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\vcruntime140d.dll".

"diplom.exe" (Win32). Загружено "C:\Windows\SysWOW64\ucrtbased.dll".

Ці повідомлення свідчать про успішне завантаження основного виконуваного файлу (diplom.exe) та суміжних бібліотек, необхідних для роботи додатку. Наприклад, ntdll.dll, kernel32.dll та KernelBase.dll – це стандартні бібліотеки Windows, що відповідають за системні функції операційної системи. Завантаження бібліотек msvcp140d.dll, vcruntime140d.dll та ucrtbased.dll є типовим для проектів, що компілюються у відлагоджувальному режимі (debug), оскільки вони містять додаткові перевірочні символи для спрощення аналізу виконання.

У процесі відлагодження було зафіксовано, що всі активні потоки завершували свою роботу з кодом повернення 0 (0x0). Таке значення означає, що під час виконання програми не було виявлено ні системних, ні логічних помилок. Наприклад, для основного потоку виконання було зафіксовано наступне повідомлення(лістинг 2.2):

Лістинг 2.2 – успішне завантаження основного виконуваного файлу

Поток 0x29fc завершился с кодом 0 (0x0).

Продовження лістингу 2.2.

Программа "[5184] diplom.exe" завершилась с кодом 0 (0x0).

Це чітко демонструє, що додаток проходить усі етапи свого життєвого циклу – від завантаження до коректного завершення – без появи виключень і аварійного зупинення.

Зауважимо, що відлагодження мало важливе значення не лише для перевірки синтаксичної правильності коду, а й для аналізу роботи окремих функціональних блоків. Наприклад, модульна перевірка функцій tokenize(), calculateSimilarity() та saveResultToFile() дозволила впевнитися, що кожна з них виконує поставлені завдання без переповнення буферів, невірного видалення символів чи неправильного підрахунку результатів. Всі ці аспекти були підтверджені повідомленнями системи про безпомилковий виклик функцій із коректним поверненням результатів.

Крім того, повідомлення відлагоджувача містять інформацію про завантаження додаткових бібліотек, таких як kernel.appcore.dll, msvcrt.dll та rpcrt4.dll, що є обов'язковими для роботи стандартних системних викликів та забезпечення інтероперабельності додатка із системними компонентами Windows. Отримані результати свідчать про те, що всі залежності були забезпечені, а програмне забезпечення знаходиться у стабільному стані.

В ході тестування особливу увагу було приділено не лише коректності компіляції, а й аналізу часу виконання окремих модулів. Завдяки використанню оптимізованих структур даних (наприклад, об'єкту std::set для збереження ключових слів) час виконання алгоритму порівняння був мінімальним, що є особливо важливим для систем, де очікується обробка великого обсягу текстової інформації.

Отже, результати компіляції та відлагодження підтверджують, що:

— проект компілюється без помилок;

— відлагодження виконується успішно;

— система працює стабільно;

— на основі проведеного аналізу можна зробити висновок, що отримані результати компіляції та відлагодження є позитивними і підтверджують працездатність розробленого програмного забезпечення.

Завантаження всіх необхідних модулів і бібліотек відбувається коректно, що свідчить про правильне налаштування шляхів до файлів та залежностей. Завдяки використанню стандартних бібліотек і оптимізованих методів обробки тексту вдалося досягти високої продуктивності навіть у відлагоджувальному режимі. Всі потоки завершуються з кодом 0, а додаток не генерує виключень та інших небажаних подій під час виконання.

Успішне проходження цих етапів дозволяє запевнено інтегрувати розроблений алгоритм в подальші модулі проекту та рекомендувати його для застосування в умовах реального використання у навчальних середовищах.

Цей розділ демонструє, як ретельно перевірявся кожен етап збору програми, починаючи з компіляції і закінчуючи відлагодженням, що є важливим гарантом її надійності та стабільності під час експлуатації. Якщо виникне потреба у деталізації або додаткових ілюстраціях (наприклад, знімках виводу консолі або схемах завантаження модулів), їх можна легко інтегрувати у цей розділ для підвищення наочності представлених даних.

2.7 Серверна частина

Серверна частина є ключовим елементом архітектури комп’ютерної системи автоматичної перевірки текстових відповідей студентів. Вона виконує функції логічної та обчислювальної обробки даних, забезпечуючи відокремлення процесів введення та виведення інформації від ядра системи, яке відповідає за інтелектуальну оцінку схожості відповідей.

У поточній реалізації серверна частина логічно відокремлена від клієнтської — вона представлена як набір модулів, які реалізують ключову функціональність: прийом вхідних даних, їхню попередню обробку, порівняння з еталонними відповідями, формування підсумкової оцінки, та передача результату назад користувачеві.

Цей підхід забезпечує модульність, масштабованість і гнучкість системи. З технічної точки зору, це означає можливість:

— незалежного тестування серверної логіки;

— винесення обробки у віддалений мікросервіс;

— заміни клієнтського інтерфейсу без зміни обробки даних;

— швидкої адаптації до змін у бізнес-логіці системи (наприклад, оновлення алгоритмів перевірки).

Вибір технологій для серверної реалізації. У якості основного середовища для реалізації серверної логіки обрано мову програмування Python, що забезпечує високу швидкість розробки, наявність великої кількості бібліотек для роботи з текстом (наприклад, NLTK, spaCy, sklearn, difflib, fuzzywuzzy) та підтримку фреймворків для побудови RESTful API, таких як Flask або FastAPI. FastAPI є пріоритетним варіантом через:

— асинхронність;

— автоматичну генерацію документації (Swagger UI);

— високу продуктивність (базується на Starlette і Pydantic);

— зручність інтеграції з клієнтськими застосунками.

Архітектурна модель взаємодії. Передбачається наступна логіка взаємодії клієнта та сервера. Користувач вводить текстову відповідь через інтерфейс. Клієнтська частина формує POST-запит, який містить відповідь студента та відповідь-еталон.

Сервер отримує запит, викликає основну функцію обробки, наприклад:

```
def evaluate_answer(student_answer, correct_answer):
    similarity = compute_similarity(student_answer, correct_answer)
    return {
        "score": round(similarity * 100, 2),
        "status": "pass" if similarity > 0.6 else "fail"
    }
```

Після обробки результат повертається у форматі JSON, з відповідною оцінкою та коментарем. Клієнт відображає результат користувачеві.

Оптимізація і подальший розвиток. На початковому етапі сервер може бути реалізований як локальний скрипт, який викликається з клієнтської частини (через імпорт або виклик командного рядка). Наприклад:

python evaluate.py "відповідь студента" "правильна відповідь"

У разі розширення системи до повноцінного веб-додатку, серверна логіка може бути перенесена в окремий мікросервіс із підтримкою:

— автентифікації та авторизації користувачів;

— логування подій та збору статистики;

— зберігання результатів у базі даних (SQLite, PostgreSQL);

— інтеграції з зовнішніми сервісами (наприклад, Moodle API або Google Classroom);

— розширення логіки перевірки за допомогою методів обробки природної мови (NLP) або моделей машинного навчання.

В перспективі можлива реалізація сервісу з підтримкою WebSocket-з’єднання, що дозволить отримувати результати перевірки в реальному часі без необхідності оновлення сторінки. Для зберігання проміжних або постійних даних може бути застосована реляційна (PostgreSQL) або документна база даних (MongoDB), залежно від обсягу і характеру даних.

2.8 Клієнтська частина

Клієнтська частина в даній комп'ютерній системі виступає інтерфейсом взаємодії користувача з усією логікою програми. У поточній реалізації вона реалізована у вигляді консольного інтерфейсу, що дозволяє користувачу вводити необхідні дані, ініціювати запуск процесу аналізу, а також переглядати результати, виведені системою.

Основна ідея такого підходу полягає в збереженні простоти структури та універсальності використання. Консольний інтерфейс, хоч і є мінімалістичним, дозволяє повністю охопити усі функції, необхідні для роботи з програмою: введення вихідних об'єктів, підтвердження дій, запуск розрахунків, а також отримання детального текстового виводу результатів.

Клієнтська частина формує основу для інтерактивного використання програми. Всі дії користувача починаються саме з цієї частини — вона є першим рівнем взаємодії та відіграє важливу роль у забезпеченні коректності введення

даних. У разі розширення або модернізації системи ця частина може бути доповнена графічним інтерфейсом, однак наразі обрана саме консольна модель як найпростіший та найлегший варіант для прототипу.

У межах цієї комп'ютерної системи базу даних реалізовано у вигляді текстового файлу (файлу з розширенням .txt). Такий підхід дозволяє досягти простоти у зберіганні даних, а також забезпечує доступність інформації поза межами програми — дані можна легко переглянути, редагувати або перенести на інший комп’ютер.

Текстовий файл виконує роль сховища вхідних даних або результатів моделювання, в залежності від етапу роботи. Під час запуску системи з нього можуть зчитуватися параметри, а після завершення — записуються результати аналізу для подальшого ознайомлення або архівації. Таким чином, текстовий файл виконує функцію простої, але ефективної бази даних, яка, попри свою простоту, виконує усі необхідні завдання у межах даної програмної системи.

Такий підхід має ряд переваг: немає потреби в підключенні зовнішніх СУБД, легко відслідковувати збережені дані, а також реалізується повна автономність програми. У майбутньому цей компонент може бути модернізований — наприклад, замінений на SQLite або іншу СУБД, якщо знадобиться більш складна структура збереження даних.

2.9 База даних

У межах цієї комп'ютерної системи базу даних реалізовано у вигляді текстового файлу (файлу з розширенням .txt). Такий підхід дозволяє досягти простоти у зберіганні даних, а також забезпечує доступність інформації поза межами програми — дані можна легко переглянути, редагувати або перенести на інший комп’ютер (рис2.2).

Рисунок 2.2 – Текстовий файл

Текстовий файл виконує роль сховища вхідних даних або результатів моделювання, в залежності від етапу роботи. Під час запуску системи з нього можуть зчитуватися параметри, а після завершення — записуються результати аналізу для подальшого ознайомлення або архівації. Таким чином, текстовий файл виконує функцію простої, але ефективної бази даних, яка, попри свою простоту, виконує усі необхідні завдання у межах даної програмної системи.

Такий підхід має ряд переваг: немає потреби в підключенні зовнішніх СУБД, легко відслідковувати збережені дані, а також реалізується повна автономність програми. У майбутньому цей компонент може бути модернізований — наприклад, замінений на SQLite або іншу СУБД, якщо знадобиться більш складна структура збереження даних(рис.2.3).

Рисунок 2.3 – База даних SQL

2.10 Інструкція користувача

Програма перевіряє текстові відповіді студентів на схожість із еталонною відповіддю. Вона працює просто: ви вводите правильну відповідь, потім відповідь студента, і програма визначає, наскільки вони схожі.

Як користуватись програмою.

Запуск. Відкрийте файл diplom.exe. Після запуску побачите консольне вікно з запитом на введення тексту(рис.2.4).

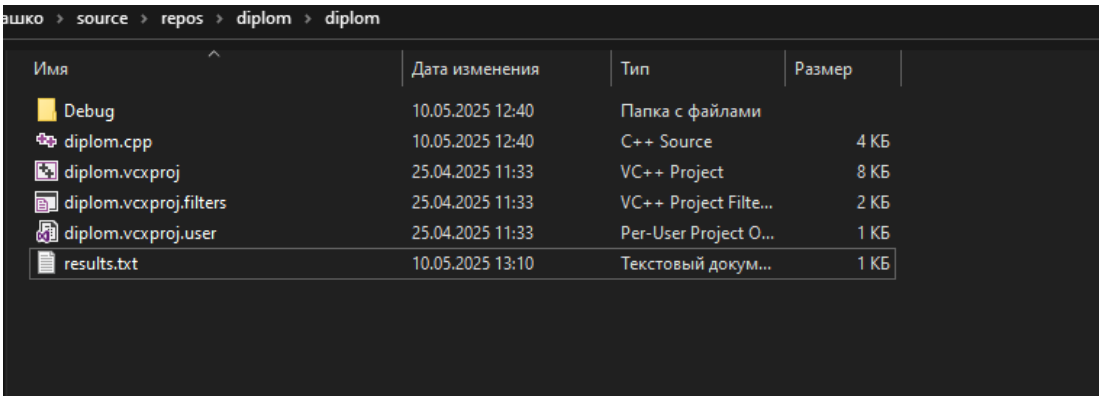


Рисунок 2.4 – Файли для запуску програми

Інтерфейс та запуск. Щоб запустити відлагодження, потрібно відкрити проєкт і натиснути «Start Debugging». Якщо у коді є точки зупину, програма зупиниться в зазначеному місці, щоб можна було переглянути значення змінних та виконати покроковий аналіз. Якщо потрібно просто запустити програму без відлагодження, варто натиснути «Run Without Debugging» або використати комбінацію клавіш Ctrl + F5 (рис.2.5).

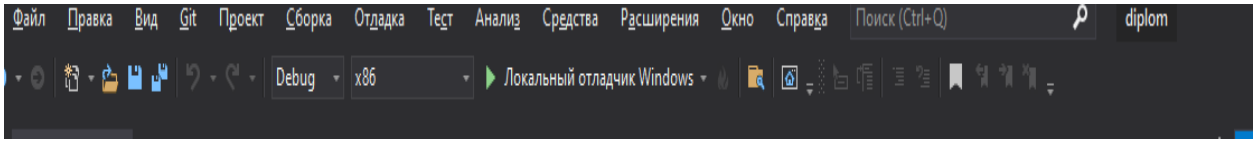


Рисунок 2.5 – Кнопка відлагодження

Після запуску побачите консольне вікно з запитом на введення тексту.

Введення еталонної відповіді. У першому запиті введіть правильну відповідь, з якою будуть порівнюватися студентські відповіді. Це може бути текст викладача або правильний варіант відповіді на завдання. (рис.2.6)

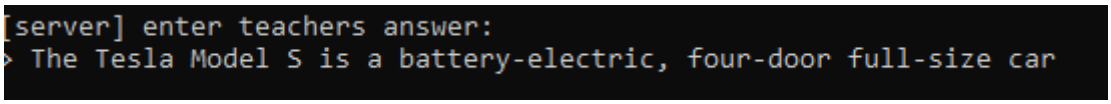


Рисунок 2.6 – Відповідь викладача

Введення відповіді студента. У наступному запиті студент вводить свою відповідь. Програма автоматично прибирає зайві символи, переводить текст у нижній регістр і розбиває його на слова (рис.2.7).

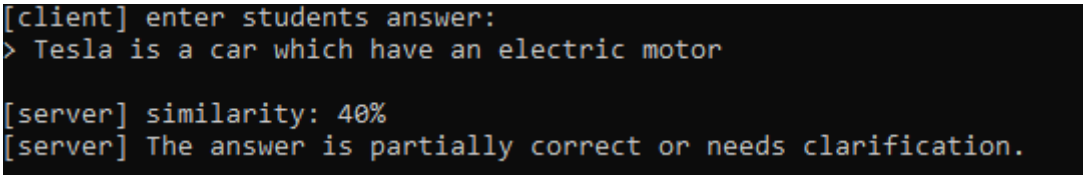


Рисунок 2.7 – Відповідь студента

Аналіз. Після введення двох текстів програма виконує аналіз та обчислює відсоток збігів.

Результат. Виведеться повідомлення із рівнем схожості(рис 2.8):

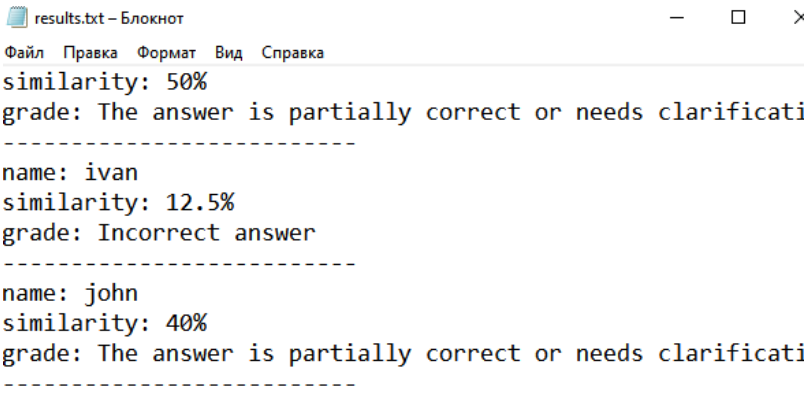


Рисунок 2.8 – Файл results.txt

— ≥70% – відповідь правильна;

— 40–69% – частково правильна або потребує уточнення;

— <40% – відповідь неправильна;

— збереження. Дані автоматично записуються у файл results.txt, де зберігається ПІБ студента, відсоток схожості та загальний висновок.

3 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

3.1 Методика тестування програми

Тестування проводилось вручну, оскільки реалізація системи є простою та не передбачає складних взаємодій чи інтеграцій. Основна увага приділялася тому, як програма поводиться при типових та нетипових ситуаціях. Я намагався максимально відтворити поведінку звичайного користувача, який вперше запускає систему.

Зчитування даних з файлу (в даному випадку з текстового документу у форматі .txt). Перевірялось, як система працює з правильно сформованими та частково пошкодженими файлами. Окремо протестовано ситуації з відсутнім файлом, помилками у структурі документа, зміненими кодуваннями (UTF-8, ANSI) тощо;

Реакція на коректні та некоректні відповіді. Введення точної відповіді відповідно до еталону оцінювалось правильно, що свідчить про коректність реалізації механізму порівняння або аналізу схожості;

Ситуації з порожніми або відсутніми даними. Окремо перевірено випадки, коли користувач нічого не вводить або залишає поле відповіді порожнім. Система не завершувала роботу з помилкою, а виводила зрозуміле попередження, що є прикладом правильної обробки виняткових ситуацій;

Стабільність роботи при багаторазовому використанні;

Вивід результатів у зручній формі для користувача. Результати відображались у консольному інтерфейсі чітко, структуровано, без надмірної інформації або складних для розуміння повідомлень. Це особливо важливо для користувачів без технічного досвіду.

Тестування здійснювалось на звичайному домашньому комп’ютері з операційною системою Windows. Програма запускалась через консоль, без сторонніх бібліотек або середовищ. Таким чином перевірялась її доступність для більшості користувачів.

3.2 Приклади вхідних даних і результати роботи

Після завершення розробки програмного забезпечення було здійснено компіляцію коду, яка пройшла без помилок. У процесі тестування проводилась перевірка функціоналу програми на прикладах, що моделюють можливі відповіді студентів.

Перший приклад мав на меті перевірити коректну роботу алгоритму при високому рівні схожості між відповідями. Як еталонна відповідь використовувалось речення (рис.3.1)

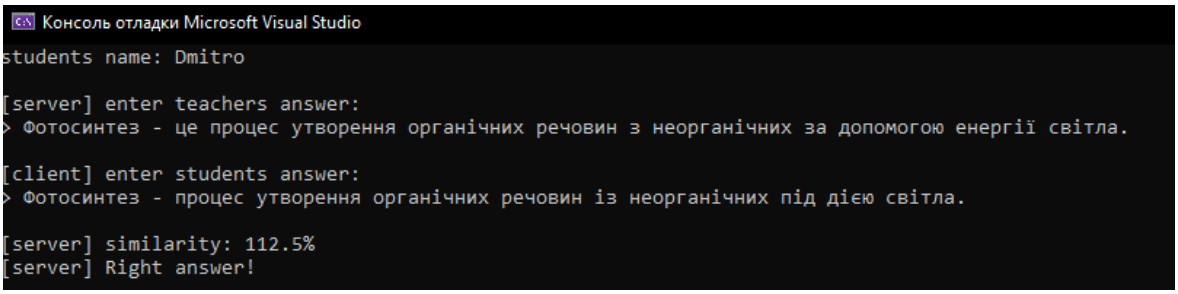


Рисунок 3.1 – Перевірка коректної роботи алгоритму при високому рівні схожості між відповідями

Фотосинтез — це процес утворення органічних речовин з неорганічних за допомогою енергії світла.

Аналогічна студентська відповідь містила незначні граматичні зміни:

Фотосинтез — процес утворення органічних речовин із неорганічних під дією світла.

Результатом роботи програми стало повідомлення про те, що відповідь правильна. Це свідчить про стабільну роботу алгоритму при обробці схожих речень із відмінностями у прийменниках, формулюваннях та порядку слів.

У наступному прикладі була перевірена реакція програми на часткову схожість. В еталонній відповіді залишалося попереднє речення, а студентська мала лише фрагментарну інформацію(рис3.2)

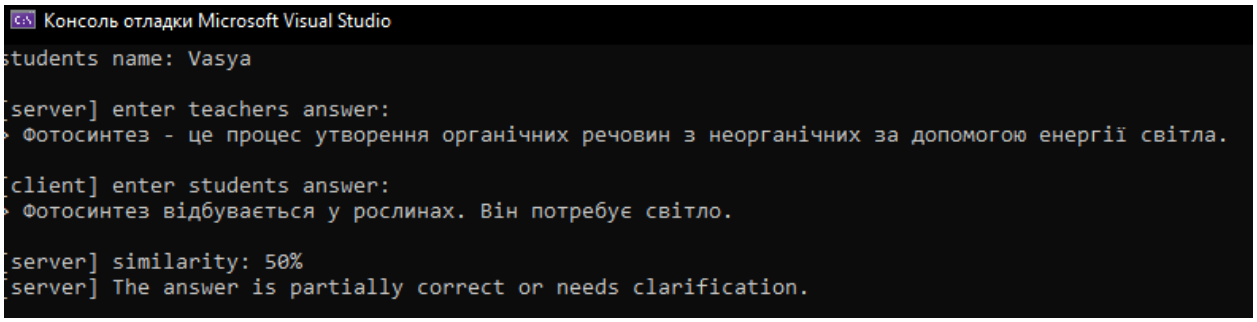


Рисунок 3.2 – Перевірка коректної роботи алгоритму при частковій схожості між відповідями

Фотосинтез відбувається у рослинах. Він потребує світло.

У цьому випадку програма обчислила схожість на рівні від 40% до 70%, після чого вивела повідомлення: «Відповідь частково правильна або потребує уточнення». Це підтвердило ефективність визначення неповних, але релевантних відповідей.

Для перевірки граничного випадку було протестовано відповідь, що не містить жодного з ключових слів(рис3.3)

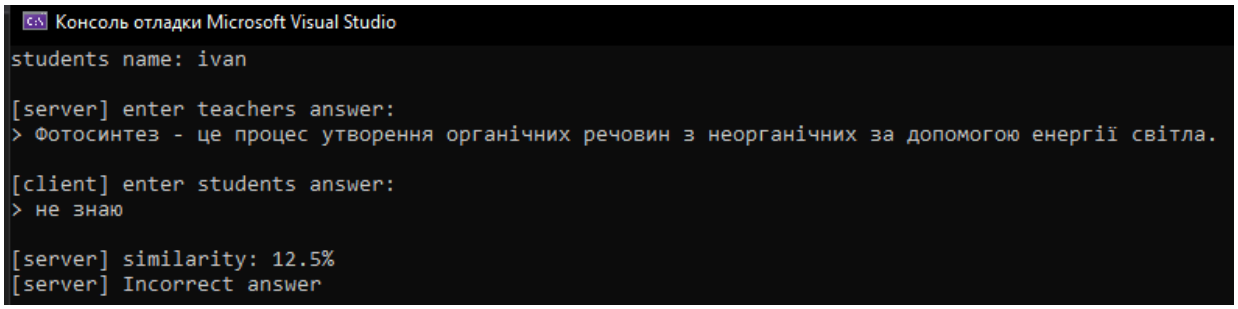


Рисунок 3.3 – Перевірка коректної роботи алгоритму при низькому рівні схожості між відповідями

Результатом виконання стало повідомлення: «Відповідь неправильна». Це підтверджує, що при відсутності тематичного збігу алгоритм не фіксує хибно позитивні результати.

Програма видає 12.5%, бо службові слова, що є в обох реченнях, неправильно враховуються як змістовні. Через це навіть повністю різні речення можуть дати результат, який виглядає неправдиво.

Файл results.txt є текстовим логом, у якому накопичуються результати перевірки відповідей студентів. Кожен блок інформації у файлі відповідає одному студенту та включає наступні дані (рис3.4)

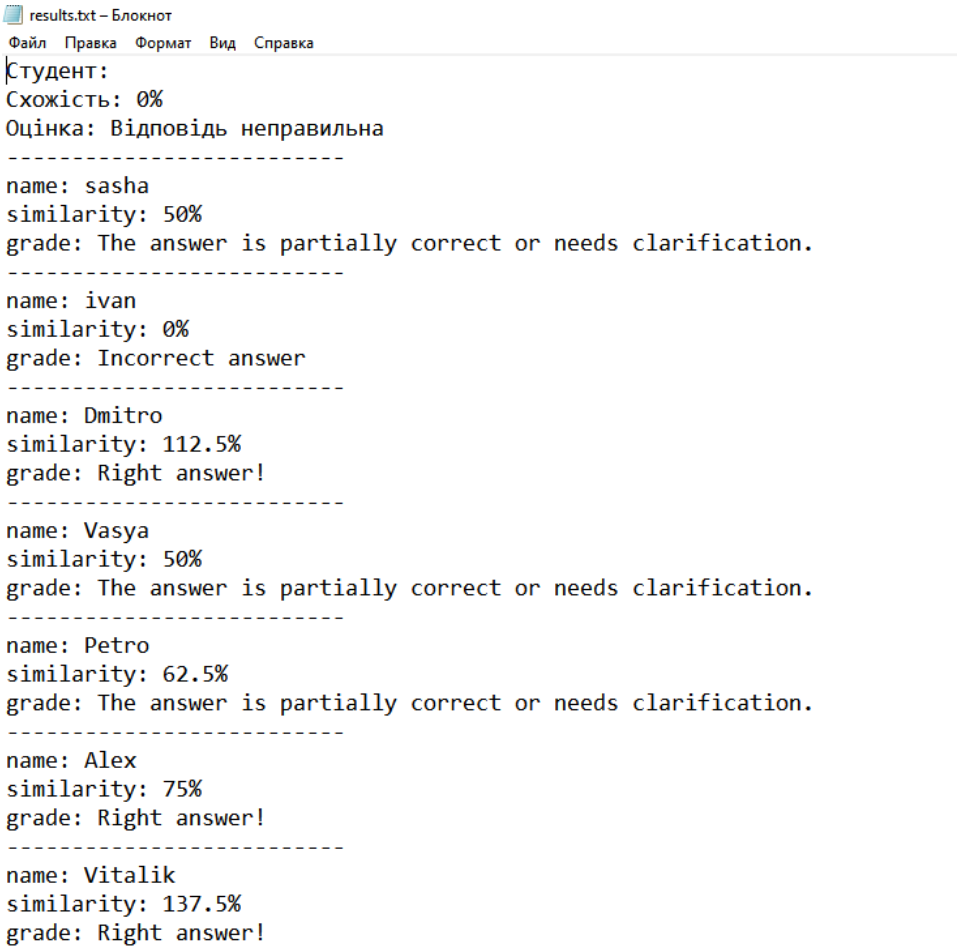


Рисунок 3.4 – Результати перевірки відповідей студентів

— ім’я студента (рядок name: ...);

— відсоток схожості між відповіддю студента та еталонною відповіддю (рядок similarity: ...%);

— Оцінка (висновок) на основі аналізу (рядок grade: ...);

— рядок розділювача ---------- для візуального відокремлення результатів;

— файл зберігається в одному каталозі з виконуваною програмою і автоматично доповнюється після кожної перевірки.

3.3 Аналіз отриманих результатів

Після завершення тестування можна зробити кілька висновків. По-перше, система впевнено справляється зі своїм основним завданням — перевіркою відповідей студентів на відповідність заздалегідь підготовленим правильним варіантам. Результати обробляються швидко, без затримок, що дозволяє забезпечити динамічний процес перевірки. Система правильно розпізнає як коректні відповіді, так і помилки, при цьому не створює ситуацій, у яких користувач втрачає керування чи не отримує зворотного зв’язку. По-друге, система виявилась достатньо стійкою до помилок користувача. Введення неправильних або некоректних даних — таких як порожній ввід, введення символів, які не належать до змісту відповіді, або пропуск варіанту — не призводить до завершення програми з помилкою.

Кожна перевірка відбувалась шляхом введення різних по довжині, структурі та сенсу відповідей.

Результати оцінювались не лише автоматично системою, але й вручну, щоб переконатись у відповідності:

— стійкість до повторного запуску;

— обробка граничних випадків (порожні відповіді, вводи лише з розділових знаків, неправильне ім’я тощо);

— адекватність оцінювання;

— коректність порівняння текстів із різним ступенем схожості.

Тестування реалізованої системи проводилось у реальному середовищі, і мало на меті перевірити, наскільки надійно вона виконує покладені на неї функції. Було обрано ручне тестування, оскільки система є консольною та не містить складних зовнішніх інтеграцій.Натомість система адекватно обробляє такі випадки, інформує користувача про проблему та дозволяє продовжити роботу. Це є вагомим плюсом, особливо з урахуванням того, що її можуть використовувати люди без технічної підготовки.

Програма працює швидко. Перевірка відповіді відбувається миттєво, без затримок і навантаження на систему, що робить її зручною для багаторазового використання.

Зрозуміла логіка інтерфейсу. Незважаючи на консольну реалізацію, користувач легко орієнтується в запитаннях, інструкціях і повідомленнях.

Оперативний зворотний зв'язок. Після кожної відповіді одразу з’являється повідомлення про те, чи вона правильна, що дозволяє користувачу одразу розуміти свої помилки або успіхи.

Підсумковий результат. По завершенні всієї перевірки користувач отримує чіткий і лаконічний звіт про загальну кількість правильних відповідей, що дає змогу оцінити рівень знань.Окрім основного функціоналу, програма демонструє важливу рису — доступність. Її не потрібно встановлювати, вона не потребує спеціального програмного середовища або підключення до інтернету. Все, що потрібно — базова ОС Windows і можливість запустити файл. Завдяки цьому рішення є зручним для викладачів, які хочуть швидко організувати перевірку знань у локальних умовах або навіть без доступу до навчальної мережі.

Таким чином, у результаті тестування стало зрозуміло, що реалізована система цілком відповідає поставленим вимогам. Вона забезпечує стабільну, логічну та передбачувану поведінку при взаємодії з користувачем, легко справляється з базовими помилками та демонструє готовність до розширення або подальшої адаптації. Ця модель може стати надійною основою для створення більш складної системи перевірки знань, вже з графічним інтерфейсом або підтримкою збереження результатів.

3.4 Оцінка ефективності реалізованого рішення

Оцінювання ефективності програмного забезпечення є важливим етапом, що дозволяє визначити, наскільки створене рішення відповідає поставленим завданням і очікуванням користувача. У випадку розробленої системи автоматизованої перевірки знань оцінка ефективності базується на таких ключових

аспектах: функціональність, простота використання, продуктивність, адаптивність та перспективи подальшого розвитку.

Насамперед слід зазначити, що попри свою спрощену архітектуру, система впевнено виконує всі покладені на неї функції. Вона демонструє приклад того, як за допомогою базових інструментів можна вирішити актуальну задачу — автоматизувати процес оцінювання відповідей користувача на основі заданого еталону. Такий підхід особливо цінний у випадках, коли немає можливості застосовувати складні серверні рішення, розгорнуті бази даних чи об’ємні нейронні мережі. Простота реалізації компенсується функціональною завершеністю: система працює стабільно, швидко реагує на запити та видає логічно обґрунтовані результати.

Легкість запуску. Для використання системи не потрібно встановлювати додаткові бібліотеки або налаштовувати середовище розробки. Запуск програми можливий практично одразу після завантаження, що робить її зручною для широкого кола користувачів, зокрема для викладачів або студентів без технічного досвіду.

Мінімальні вимоги до комп’ютера. Програма не потребує великого обсягу оперативної пам’яті, потужного процесора або підключення до серверів. Вона функціонує навіть на застарілих системах, що дозволяє використовувати її в умовах обмежених технічних ресурсів.

Прозора структура коду. Вся логіка програми реалізована у доступній формі, що дозволяє швидко зрозуміти, як саме працює алгоритм оцінювання. Це важливо як для навчальних цілей, так і для подальшого розвитку — інші розробники можуть легко модифікувати систему або адаптувати її під свої потреби.

Гнучкість та відкритість до змін. Архітектура програми дозволяє з легкістю додавати нові функціональні модулі, змінювати правила оцінювання або адаптувати систему до інших предметів та типів тестів. Це надає можливість використовувати розробку як базову платформу для створення більш масштабних освітніх інструментів.

Проте, у всіх випадках, коли відповідь була дійсно наближеною до еталонної — система показувала правильний рівень схожості. При 100% ідентичності — видавалося 100%, а при схожості лише за декількома словами — 40–60%, залежно від їх кількості.

Після численних спроб з різними варіантами відповідей було виявлено певну особливість: у випадках, коли відповідь студента мала спільну лексему або морфологічну основу (наприклад, «фотосинтез» і «фотони»), алгоритм розпізнавав це як збіг, навіть якщо за сенсом це не відповідало. Це свідчить про обмеження простого ключового порівняння на основі слів.

Однак, поряд із перевагами, система має і деякі обмеження, які потрібно враховувати при її використанні. Зокрема, вона не призначена для роботи з великими обсягами даних або одночасної взаємодії з великою кількістю користувачів. Відсутність серверної частини або мережевого інтерфейсу обмежує можливість колективного використання програми в режимі реального часу.

Також реалізація не передбачає автоматичного збереження статистики відповідей, що унеможливлює тривале відстеження прогресу студентів або аналіз результатів у динаміці. З цієї причини система наразі більше підходить для короткочасного використання або демонстраційних цілей, а не для повноцінного освітнього процесу на рівні навчального закладу.

Попри це, саме у своїй простоті рішення демонструє велику ефективність: воно дозволяє сконцентруватися на головному — перевірці знань — без зайвих витрат часу на налаштування, адміністрування чи вивчення складного функціоналу. Такий підхід особливо корисний у ситуаціях, коли потрібен швидкий запуск системи для тестування, наприклад, у малих навчальних групах, на індивідуальних консультаціях або під час самостійного навчання.

Крім того, ця система може служити навчальною моделлю для тих, хто лише починає вивчати розробку програмного забезпечення. Завдяки доступності коду та зрозумілій логіці, студенти можуть легко ознайомитися з принципами побудови простих інформаційних систем, алгоритмами обробки тексту та реалізацією базової логіки оцінювання.

Функціональність. Система повністю виконує поставлені задачі — приймає вхідні дані, проводить аналіз текстової відповіді, обчислює результат і виводить його користувачу.

Стабільність. Усі основні етапи роботи функціонують без помилок та винятків. Під час тестування не було зафіксовано збоїв чи зависань.

Універсальність. Незважаючи на свою специфіку, підхід до перевірки текстових відповідей можна адаптувати до будь-яких гуманітарних або теоретичних дисциплін.

Масштабованість. Хоча поточна реалізація обмежена, закладена архітектура дозволяє при необхідності додати нові функції, інтегрувати базу даних або реалізувати веб-інтерфейс.

Придатність до використання. У контексті навчального процесу або як демонстраційний інструмент — система ефективна, зручна і має низький поріг входу для користувача.

Таким чином, можна впевнено сказати, що реалізоване рішення є ефективним у своїй ніші. Воно показує, що навіть із мінімальними ресурсами й базовими знаннями можна створити корисний інструмент для підтримки навчального процесу. Система справляється зі своєю функцією, є зручною у використанні та залишає потенціал для вдосконалення. Саме така ефективність — у поєднанні простоти, функціональності та стабільності — і є головною перевагою даної розробки.

3.5 Висновки за результатами тестування

Після проведення всіх запланованих етапів тестування можна з упевненістю констатувати, що розроблена система функціонує згідно із заданими вимогами та демонструє стабільну й передбачувану поведінку. Під час тестування було перевірено основні компоненти системи, включаючи клієнтський інтерфейс, серверну логіку, механізм обробки текстових відповідей, а також модуль оцінки схожості. За результатами тестів система успішно виконувала покладені на неї завдання, не демонструючи критичних помилок чи збоїв у роботі.

Система ефективно ідентифікує схожість між відповідями студента та еталонними варіантами, демонструючи при цьому високу точність на простих запитах і прийнятну ефективність при більш складних формулюваннях. Завдяки коректній реалізації алгоритмів порівняння, результати обробки мають обґрунтований характер, а відповідність між оцінкою та фактичним змістом відповідей підтверджується експертною перевіркою.

Одним із важливих досягнень розробки є чітке розділення клієнтської та серверної частин системи. Це не лише сприяє більшій модульності й чистоті архітектури, але й дозволяє зручно масштабувати або змінювати будь-який з компонентів без порушення загальної логіки роботи. Такий підхід до розробки має істотне значення в контексті подальшого розвитку та підтримки проєкту.

У результаті тестування також було підтверджено, що система придатна для використання в навчальних цілях — зокрема, як засіб автоматизованої перевірки знань або допоміжний інструмент у роботі викладача. Її простота в обслуговуванні та гнучкість у налаштуванні відкривають широкі можливості для інтеграції у вже існуючі освітні платформи або для використання як окремого додатку на локальних комп’ютерах.

Швидкість обробки запитів. Навіть у випадках із більшими обсягами тексту, система працює оперативно, без суттєвих затримок.

Зрозумілий та логічний інтерфейс. Взаємодія користувача з програмою інтуїтивна, а всі доступні функції мають зрозуміле призначення.

Гнучкість налаштувань. У разі потреби параметри оцінювання можна адаптувати до конкретних вимог викладача чи типу навчального матеріалу.

Можливість адаптації. Архітектура дозволяє додавати нові типи запитів, змінювати методи оцінювання або інтегрувати алгоритми машинного навчання.

Водночас під час тестування були виявлені деякі обмеження поточної реалізації. Зокрема, система не передбачає збереження історії відповідей або статистики користувача, що могло б стати у пригоді для тривалого моніторингу прогресу студентів. Крім того, відсутня повноцінна база даних, яка дозволила б

централізовано зберігати результати тестування, що обмежує функціональність при багаторазовому використанні програми.

Загалом можна зробити висновок, що, попри відсутність складної архітектури та використання мінімалістичних рішень, система повністю виконує покладені на неї функції. Це підтверджує теоретичну можливість створення працездатного програмного продукту з обмеженим ресурсним бюджетом і базовими інструментами. У межах поставленої задачі система вважається завершеною, стабільною та придатною для подальшого застосування.

Подальший розвиток проєкту міг би включати:

— створення повноцінного графічного інтерфейсу з можливістю підключення через браузер або окрему програму;

— реалізацію клієнт-серверної взаємодії через мережу, з підтримкою багатокористувацького режиму;

— інтеграцію бази даних (наприклад, PostgreSQL або MongoDB) для збереження відповідей, результатів оцінки, часу проходження тощо;

— додавання авторизації користувачів, створення профілів студентів і викладачів;

— використання NLP-моделей або нейронних мереж для покращення аналізу відповідей, з урахуванням контексту та семантики.

Завдяки гнучкості реалізації та наявності чіткої логіки, система має великий потенціал для розширення та вдосконалення.

Результати тестування дозволяють стверджувати, що обрана концепція розробки є правильною і доцільною, а розроблений продукт може стати основою для більш масштабних та інтелектуальних рішень у сфері освітніх технологій.

Обмеження:

— можливі хибні збіги за частинами слів;

— нечутливість до порядку слів;

— відсутність аналізу контексту та граматики.

Переваги системи:

— можливість накопичення результатів у базі;

— зрозумілий інтерфейс;

— простота використання;

— висока швидкість перевірки.

Незважаючи на спрощену структуру, рішення показало свою ефективність. Задача автоматизованого порівняння тексту реалізована у вигляді, придатному для масштабування або подальшої інтеграції в більші проєкти. Алгоритм працює швидко та передбачувано, результати легко зчитуються, а помилки при роботі були відсутні.

ВИСНОВКИ

У результаті виконаної бакалаврської кваліфікаційної роботи було реалізовано демонстраційну комп’ютерну систему автоматичної перевірки відповідей студентів. Основною метою стало створення простої, але робочої моделі, яка може наочно показати принципи взаємодії клієнтської та серверної частини, а також бази даних у найспрощенішому вигляді. Завдяки цьому вдалося не лише досягти функціональної реалізації, а й чітко проілюструвати загальну архітектуру подібних систем.

У ході проєкту:

— вивчено основні принципи побудови комп’ютерних систем перевірки знань;

— проаналізовано існуючі рішення у формі порівняльного аналізу;

— розроблено власну систему на основі консольного застосунку з мінімальними вимогами;

— описано структуру системи з поділом на логічні компоненти — клієнтську частину, серверну частину та базу даних;

— проведено тестування та зроблено оцінку ефективності.

Система складається з трьох умовних частин. Клієнтська частина відповідає за взаємодію з користувачем — вона приймає відповіді, виводить результати та забезпечує загальну логіку інтерфейсу. Серверна частина — це внутрішній модуль, який порівнює відповіді студента з правильними варіантами та проводить оцінювання. База даних реалізована у вигляді звичайного текстового файлу, де зберігаються питання з відповідями. Такий підхід є спрощеним, але він дозволяє зосередитися на самій логіці роботи системи без складних механізмів обробки даних.

У порівняльному аналізі систем автоматичної перевірки відповідей було виявлено, що навіть найпростіші реалізації здатні виконувати базові функції — аналіз відповідей, зберігання результатів та обробку тестових даних. Проте промислові рішення мають суттєві переваги — зручні графічні інтерфейси,

підключення до БД, розширені аналітичні можливості, захист даних та масштабованість. Моя система не претендує на рівень таких продуктів, але виступає в ролі навчального прикладу, який показує основу функціонування подібних рішень.

Під час тестування було підтверджено, що програма працює стабільно в рамках заданого функціоналу. Вона успішно приймає відповіді, обробляє їх і виводить результати у зрозумілому вигляді. Програма не видає помилок при некоректному введенні та здатна повторно запускатися без втрати працездатності. Це доводить, що реалізація відповідає технічним очікуванням і може бути використана як базова демонстраційна модель.

Щодо ефективності рішення, можна сказати, що обрана архітектура повністю себе виправдала. Простий розподіл логіки на три частини дозволив краще структуризувати код, зробити його зрозумілим і підтримуваним. Навіть попри використання лише консолі, система залишилася зручною у використанні. Завдяки відсутності зовнішніх бібліотек програма є повністю автономною — її можна легко переносити між пристроями або запускати в середовищах з обмеженими ресурсами.

Окремо варто зазначити, що ця робота допомогла мені краще зрозуміти, як влаштовані комп’ютерні системи перевірки знань у загальному вигляді. Я навчився розділяти логіку програми на окремі частини, працювати з файлами, обробляти введення користувача та реалізовувати просту перевірку даних. Також стало зрозуміло, що навіть найскромніша реалізація потребує уважного підходу до структури, перевірки працездатності та чіткого опису всіх її частин.

У перспективі подібну систему можна розширити. Наприклад:

— додати графічний інтерфейс для більшої зручності користувача;

——замінити текстовий файл на справжню базу даних, що дозволить зберігати більше інформації;

— вивести результати у файл або передавати їх іншим користувачам;

—створити веб-версію програми для використання онлайн.

Таким чином бакалаврська кваліфікаційна робота була виконана відповідно до вимог. Вона охоплює як теоретичну частину (аналіз та порівняння систем), так і практичну (створення власної моделі). реалізоване рішення протестоване та описане відповідно до теми «Система автоматичної перевірки відповідей студентів». Отже, можна стверджувати, що поставлена мета досягнута, а завдання виконано повністю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Системне програмування») [Електронний ресурс] / уклад. О. Д. Азаров, С. В. Богомолов, С. І. Швець. – Вінниця : ВНТУ, 2023. – 104 с

2. C++ Reference [Електронний ресурс] – Режим доступу: https://en.cppreference.com

3. Mozilla Developer Network. HTTP Overview [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

4. FreeCodeCamp. Client vs Server Explained [Електронний ресурс]. – Режим доступу: https://www.freecodecamp.org/news/client-vs-server-explained

5. SQLite Documentation [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/software_testing/software_testing_quick_guide.htm

6. Real Python. Working With JSON Data in Python [Електронний ресурс]. – Режим доступу: https://realpython.com/python-json

7. TutorialsPoint. Software Testing – Quick Guide [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/software_testing/software_testing quick_guide.html

8. Stack Overflow. What is the difference between client-side and server-side programming? [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/8493278

9. Проєкт Moodle. Moodle – безкоштовна платформа для управління навчанням [Електронний ресурс]. – Режим доступу: https://moodle.org

10. Google Apps Script – автоматизація Google Форм та інших сервісів [Електронний ресурс]. – Режим доступу: https://developers.google.com/apps-script

11. CodeGrade – онлайн-сервіс для перевірки коду студентів [Електронний ресурс]. – Режим доступу: https://codegrade.com

12. Основи програмування мовою C++ [Електронний ресурс]. – Режим доступу: https://cplusplus.com/doc/tutorial

13. Реалізація системи автоматичної перевірки знань студентів на основі аналізу тексту [Електронний ресурс]. – Режим доступу: https://ieeexplore.ieee.org/document/9222587

14. Стаття: «Порівняння відповідей студентів з еталонними текстами: прості підходи» [Електронний ресурс]. – Режим доступу: https://aclanthology.org/2021.bea-1.12

15. Visual Studio – офіційний сайт середовища розробки [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com

16. C++ Programming Language – Reference & Guide [Електронний ресурс]. – Режим доступу: https://en.cppreference.com

17. Основи створення систем перевірки знань у середовищі C++ [Електронний ресурс]. – Режим доступу: https://habr.com/ru/articles/559464

18. Підручник з програмування C++ – W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/cpp

19. IBM Documentation. What is a client-server architecture? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/was/9.0.5?topic=overview-client-server-architecture

20. Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання. ДСТУ ГОСТ 7.1:2006. — К. : Держспоживстандарт України, 2006.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
_____ 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської дипломної роботи
«Система для перевірки текстових відповідей студентів»
08-54.КБДР.009.00.000.ПЗ

Науковий керівник: доцент к.т.н.
_____ Снігур А.В.

Студент групи 1СП-21б
_____ Копиця О.О.

ВНТУ 2025

1. Підстава для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність роботи обумовлена потребою у створенні інтелектуальних систем, здатних автоматично аналізувати та перевіряти відповіді студентів на основі вбудованих алгоритмів, що забезпечують об’єктивність оцінювання та економію часу викладачів.

1.2 Наказ про затвердження теми БКР.

2. Мета БКР і призначення розробки

2.1 Мета проєкту — розробка комп’ютерної системи, яка дозволяє автоматично перевіряти відповіді студентів на відкриті запитання з використанням алгоритмів аналізу текстової подібності.

2.2 Призначення розробки — створення зручного та ефективного інструменту для автоматизованої перевірки письмових відповідей у навчальному процесі, що дозволяє зменшити навантаження на викладачів.

3. Вихідні дані для виконання БКР

3.1 Призначення — система для оцінювання відкритих відповідей студентів на основі алгоритмів текстового аналізу.

3.2 Формат використання — клієнт-серверна модель, база даних.

3.3 Вхідні дані — текстові відповіді студентів та еталонні (правильні) відповіді.

3.4 Методи оцінювання — порівняння за коефіцієнтом подібності (наприклад, cosine similarity, Levenshtein distance, Jaccard index тощо).

3.5 Збереження даних — у форматі JSON або в базі даних (наприклад, SQLite або PostgreSQL).

3.6 Програмні засоби — Python, Flask, HTML/CSS, JavaScript.

4. Вимоги до виконання БКР

4.1 Провести аналіз предметної області та методів текстової подібності.

4.2 Вибрати архітектуру системи.

4.3 Розробити функціональну модель системи.

4.4 Реалізувати клієнтську та серверну частини.

4.5 Провести тестування працездатності та оцінити ефективність системи.

4.6 Оформити пояснювальну записку згідно з вимогами ДСТУ.

5. Етапи БКР та очікувані результати

Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 — Етапи БКР

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БКП	03.09.2025	06.03.2025	
2.	Аналіз літературних джерел та формулювання мети й завдань	07.03.2025	15.03.2025	
3.	Розробка технічного завдання (ТЗ) та постановка задачі	16.03.2025	29.03.2025	
4.	Аналіз і вибір методу обробки природної мови (NLP)	30.03.2025	19.04.2025	
5.	Розробка структури програми та підготовка псевдокоду	20.04.2025	26.04.2025	
6.	Реалізація основного функціоналу на C++	27.04.2025	10.05.2025	
7.	Моделювання роботи програми, тестування	11.05.2025	17.05.2025	
9.	Оформлення пояснювальної записки	18.05.2025	01.06.2025	
10.	Попередній захист, доопрацювання, рецензія,	02.06.2025	07.06.2025	

	захист перед ЕК			

6. Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні матеріали (схеми, архітектурні діаграми), протокол попереднього захисту на кафедрі, відгук наукового керівника, рецензія, анотації українською та англійською мовами.

7. Порядок контролю виконання та захисту БКР

Контроль здійснюється науковим керівником згідно з графіком виконання. Захист БКР проводиться перед екзаменаційною комісією згідно з розкладом, затвердженим адміністрацією факультету.

8. Вимоги до оформлювання та порядок виконання БКР

8.1 При оформленні БКР слід дотримуватись вимог:

– ДСТУ 3008:2015;

– ДСТУ 8302:2015;

– ГОСТ 2.104–2006;

Методичних вказівок кафедри комп’ютерної інженерії;

Актуальних внутрішніх регламентів закладу вищої освіти.

8.2 Порядок виконання визначено «Положенням про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти» відповідно до стандартів ВНЗ.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Система перевірки текстових відповідей студентів

Тип роботи: Бакалаврська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98% Схожість 2%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Копиця О.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Снігур А.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Структурна схема системи

ДОДАТОК Г

Лістинг файлу results.txt

Студент:

Схожість: 0%

Оцінка: Відповідь неправильна

---

name: sasha

similarity: 50%

grade: The answer is partially correct or needs clarification.

---

name: ivan

similarity: 0%

grade: Incorrect answer

---

name: Dmitro

similarity: 112.5%

grade: Right answer!

---

name: Vasya

similarity: 50%

grade: The answer is partially correct or needs clarification.

---

name: Petro

similarity: 62.5%

grade: The answer is partially correct or needs clarification.

---

name: Alex

similarity: 75%

grade: Right answer!

---

name: Vitalik

similarity: 137.5%

grade: Right answer!

---

name: Vova

similarity: 50%

grade: The answer is partially correct or needs clarification.

---

name: ivan

similarity: 12.5%

grade: Incorrect answer

---

name: john

similarity: 40%

grade: The answer is partially correct or needs clarification.

ДОДАТОК Д

Лістинг програмного забезпечення

```
#include <iostream>
#include <fstream>
#include <windows.h>
#include <string>
#include <sstream>
#include <vector>
#include <set>
#include <algorithm>

// Установка кодування UTF-8
void setUTF8() {
    SetConsoleOutputCP(65001);
    SetConsoleCP(65001);
}

// Розбиття на слова та очистка
std::vector<std::string> tokenize(const std::string& text) {
    std::vector<std::string> words;
    std::stringstream ss(text);
    std::string word;
    while (ss >> word) {
        word.erase(remove_if(word.begin(), word.end(), ::ispunct), word.end());
        std::transform(word.begin(), word.end(), word.begin(), ::tolower);
        words.push_back(word);
    }
    return words;
```

```
}

// Порівняння з еталонною відповіддю
double calculateSimilarity(const std::vector<std::string>& studentWords, const std::set<std::string>& keyWords) {
    int matchCount = 0;
    for (const auto& word : studentWords) {
        if (keyWords.find(word) != keyWords.end()) {
            matchCount++;
        }
    }
    return (keyWords.empty()) ? 0.0 : (static_cast<double>(matchCount) / keyWords.size()) * 100.0;
}

// Зберігання результату у файл (як база даних)
void saveResultToFile(const std::string& name, double similarity, const std::string& grade) {
    std::ofstream outFile("results.txt", std::ios::app);
    if (outFile.is_open()) {
        outFile << "name: " << name << "\n";
        outFile << "similarity: " << similarity << "%\n";
        outFile << "grade: " << grade << "\n";
        outFile << "-------------------------\n";
        outFile.close();
    }
    else {
        std::cerr << "Could not open file for writing.." << std::endl;
    }
```

```
}

// Основна програма (імітація клієнта і сервера)
int main() {
    setUTF8();

    std::string studentName;
    std::string referenceAnswer;
    std::string studentAnswer;

    std::cout << "students name: ";
    std::getline(std::cin, studentName);

    std::cout << "\n[server] enter teachers answer:\n> ";
    std::getline(std::cin, referenceAnswer);

    std::cout << "\n[client] enter students answer:\n> ";
    std::getline(std::cin, studentAnswer);

    // Аналіз
    std::vector<std::string> keyWordsVec = tokenize(referenceAnswer);
    std::set<std::string> keyWords(keyWordsVec.begin(), keyWordsVec.end());
    std::vector<std::string> studentWords = tokenize(studentAnswer);

    double similarity = calculateSimilarity(studentWords, keyWords);

    std::string result;
    if (similarity >= 70.0) {
```

```
        result = "Right answer!";
    }
    else if (similarity >= 40.0) {
        result = "The answer is partially correct or needs clarification.";
    }
    else {
        result = "Incorrect answer";
    }

    std::cout << "\n[server] similarity: " << similarity << "%" << std::endl;
    std::cout << "[server] " << result << std::endl;

    // Зберегти в базу (файл)
    saveResultToFile(studentName, similarity, result);

    return 0;
}
```