DOI: https://doi.org/10.36910/6775-2524-0560-2026-64-03
УДК 004.89:378.147:371.26
Чибіряк Яна Іванівна, к.т.н., доцент
https://orcid.org/0000-0002-0634-7609
Лавров Євгеній Анатолійович, д.т.н, професор
https://orcid.org/0000-0001-9117-5727
Неня Анна Вікторівна, к.т.н., доцент,
https://orcid.org/0000-0003-2003-3531
Павленко Андрій Володимирович, магістр
https://orcid.org/0009-0003-7267-8195
Завдов'єв Денис Олександрович, студент
https://orcid.org/0009-0004-4480-9415
Сумський державний університет, м. Суми, Україна

АВТОМАТИЗАЦІЯ ПЕРЕВІРКИ СТУДЕНТСЬКИХ РОБІТ З ІТ-ДИСЦИПЛІН: МЕТОД КЛІЄНТСЬКОЇ ВЗАЄМОДІЇ З ВЕЛИКИМИ МОВНИМИ МОДЕЛЯМИ

Чибіряк Я. І., Лавров Є. А., Неня А. В., Павленко А. В., Завдов'єв Д. О. Автоматизація перевірки студентських робіт з ІТ-дисциплін: метод клієнтської взаємодії з великими мовними моделями. Зростання кількості студентів ІТ-спеціальностей зумовлює значне навантаження на викладачів під час перевірки лабораторних робіт. Існуючі засоби автоматизації або не здатні до комплексного аналізу студентських робіт (включаючи логіку програмного коду, текстові описи та графічну частину звітів), або потребують складних серверних рішень для залучення штучного інтелекту. Метою дослідження є розробка та апробація клієнтського програмного модуля (браузерного розширення) для автоматизованої перевірки студентських робіт безпосередньо у середовищі електронних навчальних платформ. Наукова новизна роботи полягає у запропонованому методі прямої клієнтської взаємодії між вебінтерфейсом платформи та API великої мовної моделі без використання проміжних серверів, що мінімізує ризики витоку персональних даних. Розроблений програмний модуль автоматично збирає вхідні дані (текст, код та зображення), формує контекстний запит до ШІ-моделі, підтримує гнучке налаштування інструкцій та має вбудовані алгоритми відмовостійкості. Експериментальна апробація на вибірці лабораторних робіт з ІТ-дисциплін довела високу ефективність рішення: середній час перевірки однієї роботи скоротився із 14,2 до 0,67 хвилини, а точність виявлення синтаксичних та логічних помилок досягла 95%. Зафіксовані відхилення в оцінюванні творчих завдань (до 25–30%) довели практичну цінність реалізованої концепції «людина в контурі управління». Штучний інтелект виступає як асистент: згенеровані результати повертаються у вебінтерфейс навчальної платформи у формі редагованих полів, залишаючи за викладачем право коригування рецензії та ухвалення кінцевої оцінки.
Ключові слова: штучний інтелект, великі мовні моделі, автоматизація оцінювання, браузерне розширення, клієнтська інтеграція, електронні навчальні платформи, промпти, AI-агенти, API-запити.

Chybiriak Ya., Lavrov E., Nenia A., Pavlenko A., Zavdoviev D. Automating the Assessment of Student Assignments in IT Disciplines: A Method of Client-Side Interaction with Large Language Models. The growing number of IT students leads to a significant workload for instructors when grading laboratory assignments. Existing automation tools are either incapable of a comprehensive analysis of student assignments (including source code logic, text descriptions, and graphical components of reports) or require complex server-side solutions to integrate artificial intelligence. The aim of this study is the development and testing of a client software module (a browser extension) for the automated assessment of student assignments directly within e-learning platforms. The scientific novelty of the work lies in the proposed method of direct client-side interaction between the platform's web interface and a Large Language Model API without using intermediate servers, which minimizes the risks of personal data leakage. The developed software module automatically collects input data (text, code, and images), generates a contextual request for the AI model, supports flexible instruction configuration, and features built-in fault-tolerance algorithms. Experimental testing on a sample of IT laboratory assignments demonstrated the high efficiency of the solution: the average grading time per assignment was reduced from 14.2 to 0.67 minutes, while the accuracy of detecting syntax and logical errors reached 95%. The recorded deviations in evaluating creative tasks (up to 25–30%) confirmed the practical value of the implemented "human-in-the-loop" concept. Artificial intelligence acts as an assistant: the generated results are returned to the educational platform's web interface as editable fields, reserving the instructor's right to modify the review and determine the final grade.
Keywords: artificial intelligence, large language models, assessment automation, browser extension, client-side integration, e-learning platforms, prompts, AI agents, API requests.

Постановка проблеми.

Зростання попиту на ІТ-фахівців зумовлює значне збільшення кількості студентів на комп'ютерних спеціальностях у закладах вищої освіти. Це створює значне педагогічне навантаження на викладачів профільних дисциплін, більшу частину якого становить перевірка лабораторних та практичних робіт. Сьогодні цей процес здебільшого відбувається у середовищі електронних навчальних платформ. Якщо роботи студентів містять програмні коди, текстові пояснення та графічну частину, витрати часу на перевірку зростають у рази. Оскільки запуск

кожного коду в середовищі розробки здійснити практично неможливо, тексти програм аналізуються викладачами візуально безпосередньо через вебінтерфейс навчальної платформи. Така перевірка часто спричиняє пропуск помилок та появу суб'єктивізму в оцінюванні. Обмеження часового ресурсу викладача призводять до затримок у перевірці робіт, а також до відсутності розгорнутих письмових пояснень і коментарів при оцінюванні студентських звітів. Активне впровадження моделей змішаного та адаптивного навчання [1-4] у навчальний процес лише посилює когнітивне навантаження на викладача та доводить необхідність розробки автоматизованих систем оцінювання [5].

Аналіз останніх досліджень і публікацій.

У науковій літературі, присвяченій автоматизації перевірки програмних кодів, розглядається використання класичних методів, до яких належать unit-тести та статичний аналіз коду, вбудовані у більшість сучасних систем управління навчанням (Moodle, GitHub Classroom, Coursera, edX). За дослідженнями M. Messer та співавторів [6], unit-тести є ефективним інструментом для швидкої перевірки функціональної правильності коду, проте вони не виявляють логічних помилок та потребують суттєвих часових витрат на розробку [6-9]. Статичний аналіз виявляє логічні помилки у коді [9]. Однак жоден із цих методів не здатний оцінювати текстову (описову) та графічну частини студентських звітів. Необхідність їхнього комплексного аналізу зумовила перехід до використання великих мовних моделей (LLM) [6, 10]. Наприклад, такі інноваційні системи перевірки студентських робіт, як BeGrading [11] та CodEv [10], вже застосовують LLM та ШІ-агентів, демонструючи результати, співставні з оцінками викладачів. Але ці системи не здатні автоматично інтегрувати згенеровані оцінки та рецензії безпосередньо у вебінтерфейс навчальних платформ.

Попри високу ефективність, використання моделей ШІ для оцінювання стикається з проблемами. По-перше, складною задачею є автоматичне формування змістовного промпту, який об'єднує постановку задачі, критерії оцінювання, код та контент студентського звіту [12]. Аналіз понад мільйона API-запитів виявив, що якість оцінювання з використанням LLM погіршується за умови неповного контексту промптів, а надто довгі промпти створюють проблеми в роботі з моделями [13]. Автоматично згенеровані промпти підвищують точність оцінювання (у середньому на 9%), але часто спотворюють початкові інструкції викладача [14, 15]. По-друге, виникає проблема забезпечення контролю викладача за відсутністю галюцинацій з боку мовної моделі під час видачі результатів [16-18]. Для вирішення цих проблем дослідники пропонують впровадження гібридних архітектур, де класичні алгоритми (юніт-тести, статичний аналіз коду) виконують технічну перевірку кодів, а ШІ аналізує текстовий і графічний контент студентських звітів [6, 19].

Практична реалізація гібридних систем у межах навчальних платформ зазвичай спирається на серверну архітектуру (протокол LTI або прямі API-запити) з використанням проміжних серверів [20]. Прикладом є поєднання платформи Moodle, системи GitHub та зовнішніх спеціалізованих серверів для перевірки коду (автогрейдерів) [21]. Суть цього підходу полягає в тому, що після завантаження студентом коду на GitHub, автогрейдер автоматично його компілює, перевіряє через юніт-тести та передає бали назад у середовище Moodle [21]. Хоча протокол LTI забезпечує авторизовану передачу даних, він не регулює їхнє подальше збереження, використання для навчання моделей чи передачу третім сторонам, що створює загрозу витоку персональних даних студентів [22, 24]. Крім того, впровадження стандарту LTI вимагає складного управління ключами доступу та синхронізації конфігурацій між Moodle і сервером провайдера [22, 24]. При цьому LTI стандартизує лише базовий запуск зовнішнього ШІ-інструменту та передачу згенерованих оцінок до навчальної платформи, але не дозволяє контролювати якість ШІ-відповідей, безпеку промптів чи відсутність галюцинацій мовної моделі [24, 25]. Таким чином, незважаючи на те, що LTI поєднує ШІ-сервіси з Moodle, питання захисту даних [22-28] та педагогічного контролю [23, 28] залишаються відкритими.

На противагу гібридним системам, альтернативним сучасним підходом є використання автономних AI-агентів (агентів штучного інтелекту). Вони здатні аналізувати творчі та нестандартні відповіді студентів, а їхнє функціонування не залежить від специфіки користувацького інтерфейсу навчальних платформ, оскільки AI-агенти оперують безпосередньо семантикою даних. Однак їхнє практичне застосування має певні обмеження: агенти схильні до накопичення похибок, коли

неточний висновок на початковому етапі аналізу повністю спотворює підсумкову оцінку [29]. Також під час масової перевірки робіт виникають часові затримки та суттєво зростають витрати на API-токени [30], а розгортання складних серверних фреймворків для інтеграції агентів з навчальними платформами створює технічні виклики для закладів освіти [31]. Менш сучасним підходом є використання спеціальних роботів-скриптів, що імітують дії викладача у веббраузері. Цей метод розглядається в літературі як ненадійний, оскільки він є вразливим до найменших змін вебінтерфейсу навчальних платформ та порушує політики безпеки через імітацію дій користувача [32].

Таким чином, сучасні спроби автоматизації перевірки студентських робіт поділяються на дві основні категорії:

1. Використання спеціалізованих інструментів тестування (unit-тести, статичні аналізатори коду), які працюють за принципом зіставлення вхідних та вихідних даних коду. Вони є ефективними але вимагають від викладача значних витрат часу на написання перевірочних тестів. Крім того, такі системи неспроможні оцінювати текстову частину студентських звітів.

2. Застосування великих мовних моделей (LLM) та автономних AI-агентів для комплексної перевірки як програмного коду, так і текстової частини звітів. Пряме використання LLM вимагає частого написання промптів та ручного копіювання матеріалів звіту між платформою і вебінтерфейсом нейромережі, що порушує безперервність роботи викладача. Більш складні автоматизовані рішення на базі LTI або AI-агентів вимагають розгортання проміжних серверів, призводять до зростання витрат на API-токени, часових затримок, накопичення помилок, а також створюють загрози інформаційній безпеці через передавання персональних даних студентів на сторонні сервери.

З огляду на це, виникає потреба в розробці спеціалізованого програмного модуля на стороні клієнта (браузерного розширення). Такий модуль має здійснювати пряму взаємодію вебінтерфейсу навчальної платформи з API мовної моделі без участі проміжних серверів. Крім того, програмний модуль повинен бути стійким до відмов на випадок збоїв нейромережі та функціонувати за принципом «людина в контурі управління», зберігаючи за викладачем вирішальну роль в оцінюванні студентських робіт.

Метою даного дослідження є розробка та апробація клієнтського програмного модуля у вигляді браузерного розширення для автоматизації перевірки студентських робіт у середовищі навчальних вебплатформ на основі прямої взаємодії з API великих мовних моделей.

Для реалізації поставленої мети, у роботі вирішуються такі **завдання**:

– розробити архітектуру та алгоритм роботи браузерного розширення, яке забезпечує пряму взаємодію з API мовної моделі, автоматичне формування промптів та повернення результатів у вебінтерфейс платформи;

– створити функціональний прототип програмного модуля для автоматизованого аналізу програмного коду та текстових звітів студентів у середовищі навчальної платформи;

– провести експериментальну апробацію розробленого модуля в умовах реального навчального процесу (на базі платформи МІХ СумДУ) та оцінити його ефективність щодо економії часу викладача і точності перевірки.

Наукова новизна отриманих результатів полягає у розробці та обґрунтуванні методу клієнтської інтеграції великих мовних моделей у системи управління навчанням. Уперше запропоновано спосіб прямої взаємодії між вебінтерфейсом навчальної платформи та API мовної моделі, який реалізується через браузерне розширення. На відміну від існуючих серверних та LTI-рішень, запропонований підхід забезпечує:

– обробку даних та автоматичне формування промптів у фоновому режимі безпосередньо у браузері користувача під час автоматизованої перевірки студентських робіт, що виключає потребу у використанні проміжних серверів та підвищує рівень захисту персональної інформації;

– пряме повернення згенерованих оцінок і рецензій у вебінтерфейс навчальної платформи з можливістю їхнього редагування викладачем, що реалізує концепцію «людина в контурі управління».

Практичне значення одержаних результатів. Застосування клієнтського браузерного розширення для взаємодії з мовною моделлю під час роботи у навчальній платформі дозволяє суттєво скоротити час викладача на перевірку студентських робіт, мінімізуючи ризики витоку

персональних даних завдяки відсутності проміжних серверів. Практична цінність розробленого модуля полягає також у реалізації принципу «людина в контурі управління»: згенеровані штучним інтелектом рецензія та оцінка повертаються безпосередньо у вебінтерфейс платформи у формі редагованих полів, що дає змогу викладачу коригувати аналітичні висновки й бали перед їх остаточним збереженням. Ефективність запропонованого методу підтверджено під час експериментальної апробації у середовищі навчальної платформи MIX (СумДУ) на вибірці лабораторних робіт з IT-дисциплін.

Виклад основного матеріалу.

Концептуальне моделювання архітектури програмного модуля. Розробка програмного модуля для організації взаємодії між навчальною вебплатформою та мовною моделлю під час перевірки студентських звітів вимагає попередньої формалізації процесів. На цьому етапі використано методологію функціонального моделювання в нотації IDEF0.

Цільова модель «Як буде» (TO-BE) та її декомпозиція (рис. 1) показують, що викладач ініціює процес перевірки студентського звіту, прикріпленого до вебсторінки навчальної платформи.

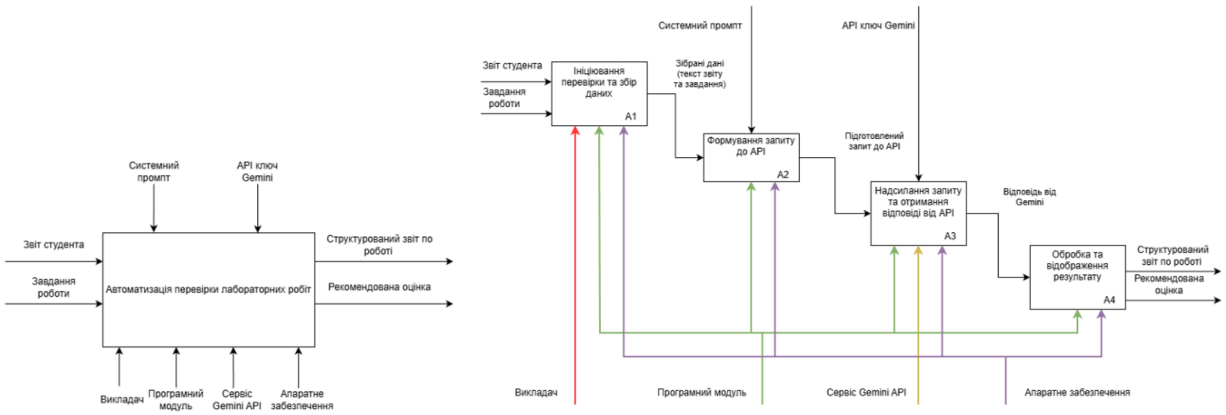


Рис. 1. Контекстна діаграма (A-0) та її декомпозиція (A0) процесу «Як буде»

Внутрішня архітектура процесу A0 декомпозується на чотири етапи:

– A1: ініціювання перевірки та збір даних. Передбачає автоматичне вилучення з вебсторінки тексту звіту студента, постановки завдання та критеріїв оцінювання для створення масиву вхідних даних;

– A2: формування запиту до API. Передбачає поєднання системного промпту з масивом вхідних даних, зібраних на етапі A1. Системний промпт зберігається в конфігурації програмного модуля. Програма автоматично інтегрує текст завдання та студентського звіту в структуру промпту, формуючи підготовлений запит до нейромережі;

– A3: надсилання запиту та отримання відповіді від API. Передача підготовленого запиту до мовної моделі Gemini. Цей етап керується обов'язковою авторизацією за допомогою API-ключа, який ідентифікує запити в системі Google та надає програмному модулю доступ до нейромережі з урахуванням персональних лімітів. Викладач самостійно генерує API-ключ у середовищі Google AI Studio та зберігає його в локальних налаштуваннях програми. Після опрацювання запиту мовна модель повертає програмному модулю результат у форматі структурованого тексту (JSON).

– A4: обробка та відображення результату. На основі отриманої JSON-відповіді програма формує структурований звіт про результати перевірки та рекомендовану оцінку. Цей результат автоматично інтегрується і відображається у текстовому вікні на вебсторінці навчальної платформи.

Діаграма послідовності (рис. 2) демонструє взаємодію основних компонентів системи під час перевірки роботи:

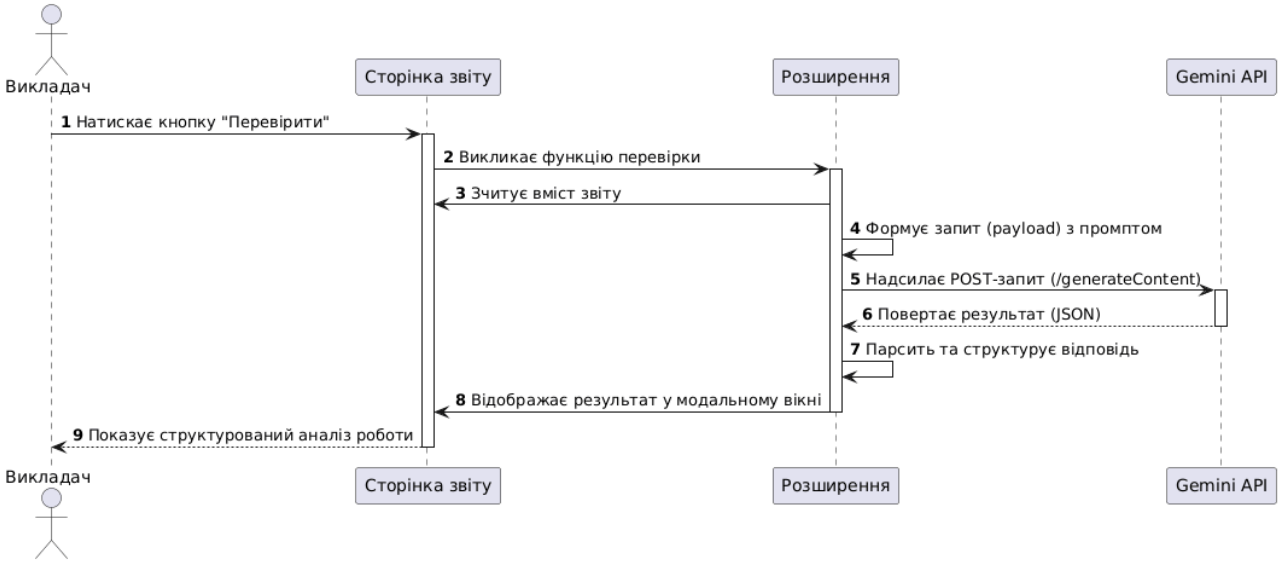


Рис. 2. Діаграма послідовності (Sequence)

– викладач ініціює процес, натискаючи кнопку перевірки на вебсторінці навчальної платформи;
– програмний модуль (розширення браузера) зчитує вміст сторінки звіту, формує масив вхідних даних (payload) із системним промптом та надсилає POST-запит до Gemini API;
– Gemini API обробляє отримані дані та повертає результат генерації у форматі JSON;
– програмний модуль парсить отриману відповідь і відображає структурований аналіз (результат перевірки) у модальному вікні безпосередньо на сторінці навчальної платформи.

Запропоновані архітектурні рішення надають такі переваги:

– асинхронна обробка даних у фоновому режимі: викладач може продовжувати роботу з іншими елементами вебсторінки навчальної платформи під час очікування відповіді від мовної моделі;

– захист даних та конфіденційність: використання персонального API-ключа та прямого з'єднання за схемою «клієнт - API» робить неможливим проходження студентських робіт через проміжні сервери, що відповідає сучасним вимогам безпеки інформаційних систем.

Сформована архітектурна модель слугує основою для подальшої програмної реалізації розроблюваного модуля.

Програмна реалізація модуля автоматизації перевірки звітів. Програмний модуль має відповідати таким вимогам:

– мінімізація споживання системних ресурсів комп'ютера;
– інтеграція з освітньою платформою (робота в межах поточної авторизованої сесії викладача);
– мультимодальна обробка даних (здатність автоматично вилучати текст, програмний код та графічні зображення зі звіту студента та формувати комплексний запит до нейромережі).

Виконання цих вимог забезпечується обраною архітектурою (рис. 3) та використанням сучасних вебтехнологій.

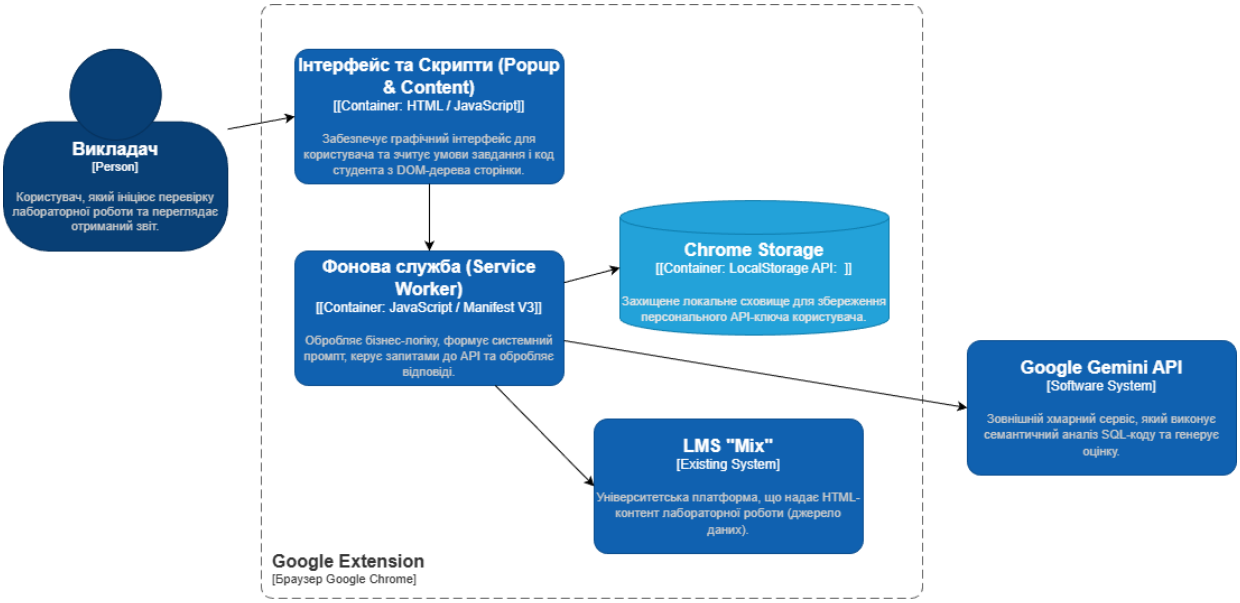


Рис. 3. Архітектура програмного модуля у форматі C4 Model

Економія системних ресурсів досягається шляхом створення програмного модуля на основі стандарту Manifest V3. Технічно це означає відмову від постійно активних фонових сторінок (Background Pages) на користь подієво-орієнтованих служб (Service Workers), які активуються лише в момент перевірки студентського звіту та автоматично вивантажуються з пам'яті після отримання відповіді від API великої мовної моделі. Завдяки цьому програмний модуль залишається неактивним під час звичайної роботи користувача у браузері. Він завантажується в оперативну пам'ять лише в момент натискання кнопки перевірки звіту. Щойно програма завершує свою роботу (збір даних, відправлення запиту до ШІ та виведення результату), модуль автоматично вивантажується. Такий підхід економить оперативну пам'ять, зменшує навантаження на процесор та знижує енергоспоживання пристрою.

Вимоги щодо роботи в межах авторизованої сесії викладача та мультимодальної обробки даних досягаються завдяки функціонуванню модуля безпосередньо у браузері користувача. Для цього модуль було реалізовано у вигляді браузерного розширення, що містить спеціальні алгоритми: вилучення даних із вебсторінки та локального парсингу документів.

Більшість сучасних навчальних платформ будуються за архітектурою односторінкових застосувань (SPA), що передбачає динамічне завантаження контенту. Суть цього підходу полягає в тому, що вміст вебсторінки (зокрема посилання на прикріплені звіти) завантажується асинхронно за допомогою JavaScript без повного перезавантаження HTML-коду вебсторінки. Оскільки такі платформи зазвичай не надають відкритого публічного API, отримання доступу до студентських звітів потребує застосування спеціальних алгоритмів відстеження змін у DOM-дереві вебсторінки. Для вирішення цієї проблеми у програмному модулі застосовано спеціальний алгоритм пошуку посилань на прикріплені до вебсторінки файли (рис. 4).

```
sendBtn.addEventListener('click', async () => {
  ensureResultPanel();
  setResult("🔄 Завантаження файлу...");
  try {
      const attachment = await findAttachment();
      console.log('🔍 Found attachment:', attachment);
      if (!attachment) {
        setResult('Не знайдено посилання на звіт (attachment).');
        return;
      }
      const name = attachment.title || attachment.filename || 'report.docx';
      const context = attachment.context || {};
      const allAttachments = attachment.allAttachments || [attachment];
      console.log('🔍 Attachment context:', context);
      console.log('🔍 All attachments:', allAttachments);
```

Рис. 4. Фрагмент коду ініціалізації збору даних та вилучення метаданих

Пошук виконує функція findAttachment(). Програма сканує структуру вебсторінки, глобальні змінні JavaScript та приховані поля форм. Результатом пошуку є JSON-об'єкт, з якого вилучається масив прямих посилань на файли – студентський звіт та постановку завдання.

Після отримання посилань, файли автоматично завантажуються в оперативну пам'ять браузера, де відбувається їх локальний парсинг і конвертація у текстовий формат, придатний для обробки нейромережею (рис. 5).

```
async function convertDocxToParts(url) {
  // Fetch via background to bypass potential CORS/cookie issues
  const fetched = await chrome.runtime.sendMessage({ type: 'mix-fetch-arraybuffer', url });
  if (!fetched?.ok) throw new Error(fetched?.error || 'FETCH_FAILED');
  const buf = base64ToArrayBuffer(fetched.base64);
  const result = await mammoth.convertToHtml({ arrayBuffer: buf }, {
    convertImage: mammoth.images.inline(async element => element.read('base64').then(b64 => ({
      src: `data:${element.contentType};base64,${b64}`
    })))
  });
  const html = result.value || '';
  const text = htmlToPlain(html);
  const imageParts = extractImagesFromHtml(html).map(d => ({ inline_data: dataUrlToInline(d) }));
  const parts = [{ text }].concat(imageParts);
  return parts;
}
```

Рис. 5. Програмна реалізація завантаження та локального парсингу документа

Процес обробки файлів виконує функція convertDocxToParts(). Алгоритм обробки документу складається з трьох етапів:

1. Автоматичне завантаження студентського звіту в пам'ять браузера. Отриманий файл конвертується у бінарний формат для подальшої обробки в оперативній пам'яті.
2. Розпізнавання та перетворення бінарного файлу на HTML-код з використанням клієнтської бібліотеки Mammoth.js. Важливою особливістю розробленого алгоритму є те, що всі графічні елементи, вбудовані у звіт (скріншоти результатів, діаграми, графіки тощо), автоматично вилучаються та конвертуються в текстовий формат Base64 без їх збереження на проміжних серверах.
3. Згенерований HTML-код розділяється: текстова частина очищується від HTML-тегів до формату звичайного тексту, а закодовані зображення виділяються в окремий масив даних. Після цього текст і графіка об'єднуються у спільну структуру даних, готову до формування запиту (промпту).

Такий підхід дозволяє сформувати масив даних, який відповідає вимогам API мовної моделі Gemini.

Формування контекстного запиту (промпту) до великої мовної моделі. Ядром розробленої програмної системи є модуль автоматичної генерації запиту (АГЗ).

На рис. 6 наведено структурну схему, яка відображає принцип формування контекстного запиту до моделі штучного інтелекту.

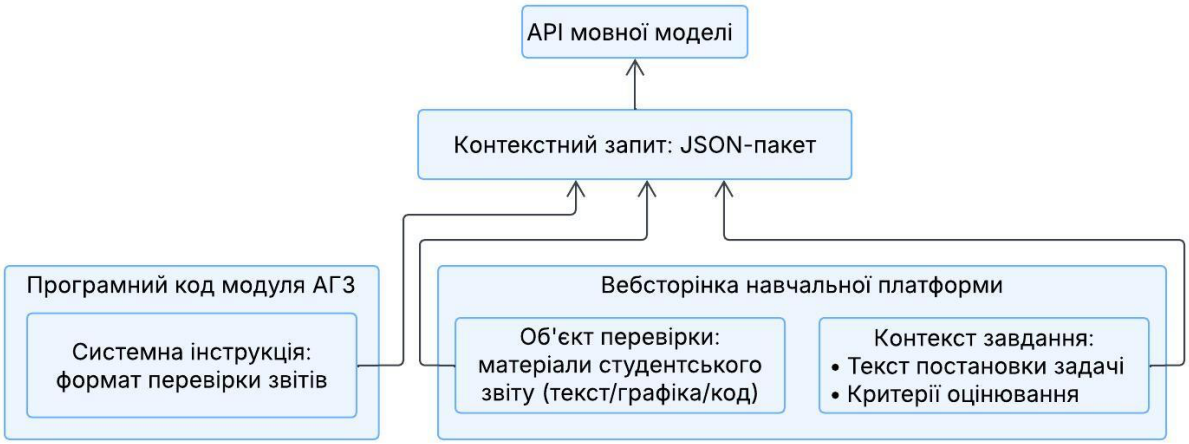


Рис. 6. Структурна схема формування контекстного запиту до мовної моделі

Як видно зі схеми, загальний контекстний запит (промпт) формується у вигляді єдиного JSON-пакета та синтезується з двох незалежних потоків інформації:

– системної інструкції, закладеної у програмний код модуля, яка визначає базові правила та формат перевірки звітів (цей шаблон може гнучко налаштовуватися викладачем);
– динамічних вхідних даних, які автоматично вилучаються модулем безпосередньо з вебсторінки навчальної платформи. Цей блок структурно поділяється на дві складові: контекст завдання (текст постановки задачі та критерії оцінювання) і безпосередній об'єкт перевірки (текст, графічні матеріали або програмний код зі звіту студента).

Такий підхід забезпечує передачу до API мовної моделі чітко структурованого масиву даних, де вимоги, постановка задачі та критерії оцінювання роботи відокремлені від матеріалів, призначених для перевірки.

Розробка цього модуля базувалася на трьох аспектах:

– дотриманні встановлених критеріїв оцінювання і постановки завдання;
– адаптивності до різних видів студентських робіт;
– забезпеченні відмовостійкості під час взаємодії з мовною моделлю.

Алгоритм АГЗ враховує критерії оцінювання таким чином: якщо в описі завдання, розміщеному в окремому файлі або на вебсторінці, знайдено ключові слова («Критерії», «Бали» тощо), система формує інструкцію з дотриманням правил розподілу балів, встановлених викладачем.

Також особливістю розробленого модуля є гнучкість налаштувань. Оскільки різні види робіт студентів (лабораторні, практичні) по різним дисциплінам можуть вимагати різних форматів звітності (наприклад, акцент на якості програмного коду або, навпаки, на глибині теоретичних висновків), було реалізовано механізм впровадження користувацької інструкції [33]. Це дозволяє викладачу через інтерфейс налаштувань задати власний шаблон структури результату перевірки. Заданий шаблон перевизначає базові системні інструкції, вбудовані у код програмного модуля за замовчуванням. На технічному рівні цей механізм забезпечується збереженням заданого шаблону у локальному сховищі браузера (localStorage) та його динамічною підстановкою під час формування запиту до API Google Gemini.

Взаємодія з API Google Gemini проєктувалася з урахуванням вимог до відмовостійкості системи [26, 27]. Оскільки під час роботи можливі перевищення лімітів кількості запитів або тимчасова недоступність окремих версій моделей ШІ, у програмі реалізовано стратегію автоматичного перебору доступних моделей. Замість жорсткої прив'язки до конкретної версії нейромережі, алгоритм послідовно звертається до альтернативних моделей ШІ у разі відмови основної. Цей підхід гарантує, що навіть за умов оновлення версій API з боку Google або тимчасового перевантаження конкретної моделі, викладач стабільно отримуватиме результат перевірки студентських звітів. Блок-схема розробленого алгоритму формування контекстного запиту показана на рис. 7.

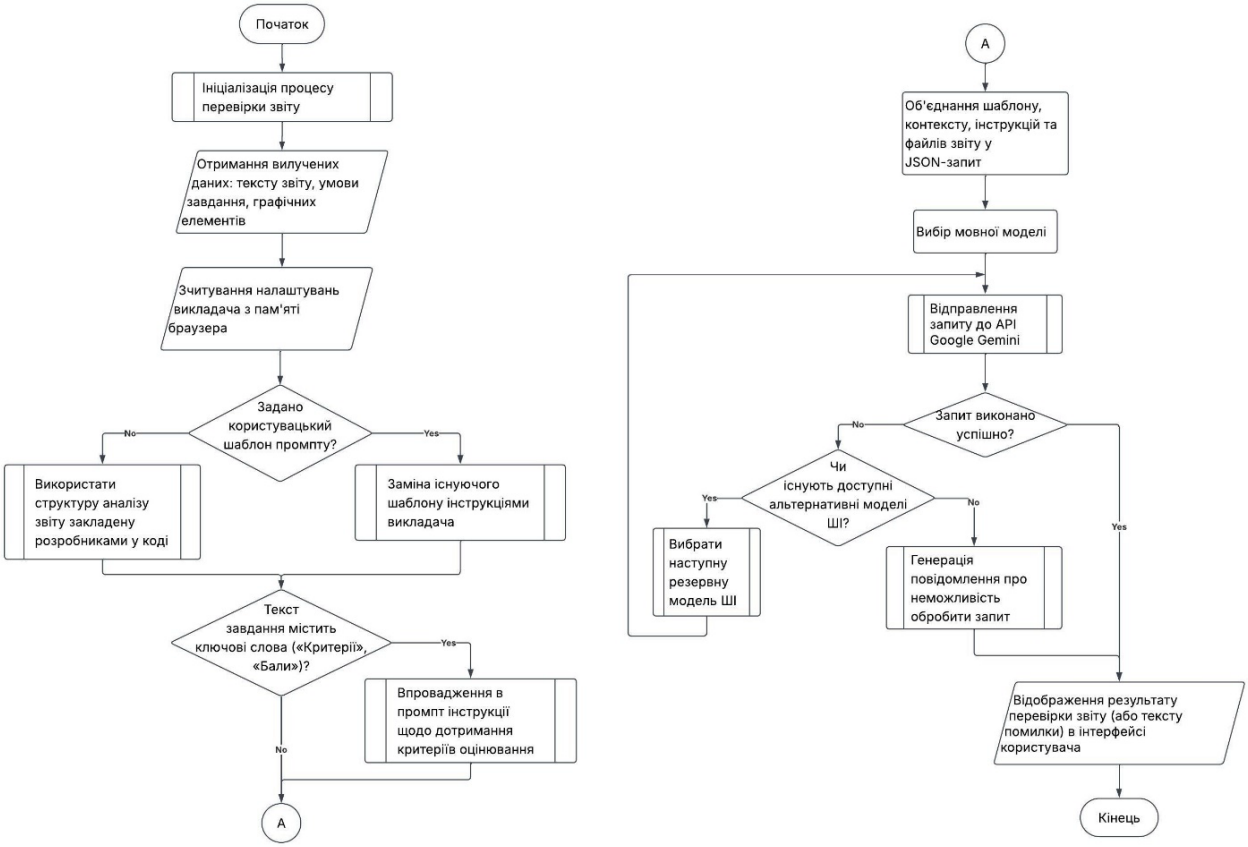


Рис. 7. Блок-схема алгоритму формування контекстного запиту до нейромережі

Запропонований алгоритм дозволяє генерувати фінальні контекстні запити до мовної моделі шляхом динамічного поєднання системних інструкцій та вхідних даних (звітів, постановки завдання, критеріїв оцінювання). Він гнучко адаптується під різні види студентських робіт та забезпечує стійкість до можливих збоїв під час функціонування сервісів ШІ.

Для підтвердження дієвості розробленого програмного модуля необхідно перевірити коректність його роботи на реальних даних.

Використання створеного браузерного розширення.

Практична апробація розробленого програмного модуля проводилася у середовищі навчальної платформи MIX (СумДУ).

Для початку роботи із системою необхідно встановити програмний модуль у вигляді розширення для браузера Google Chrome та згенерувати персональний API-ключ за допомогою сервісу Google AI Studio.

Робота починається з етапу базового налаштування параметрів модуля, який виконується одноразово. На рис. 8 зображено вікно налаштувань розширення. Обов'язковою умовою працездатності є введення ключа авторизації (персонального API-ключа).

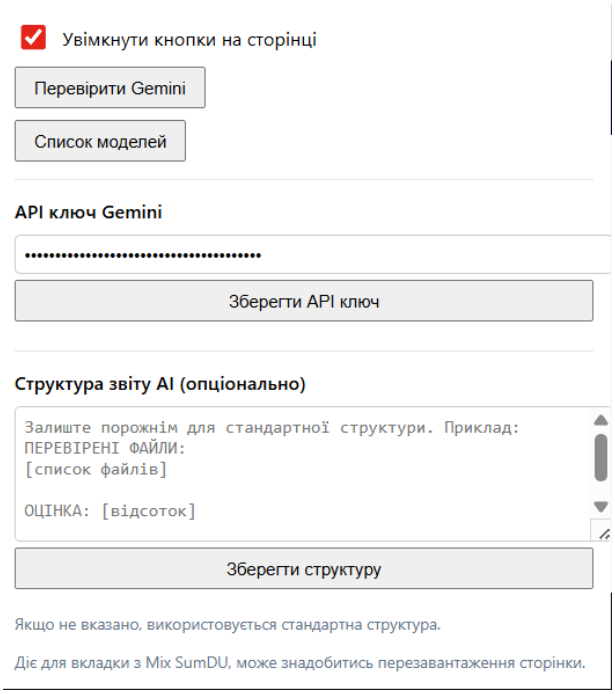


Рис. 8. Інтерфейс налаштування та перевірки з'єднання

У полі «Структура звіту AI» викладач має змогу задати власний формат результату перевірки. Перемикач «Увімкнути кнопки на сторінці» активує відображення кнопки «Відправити на перевірку» на вебсторінці навчальної платформи (рис. 9).

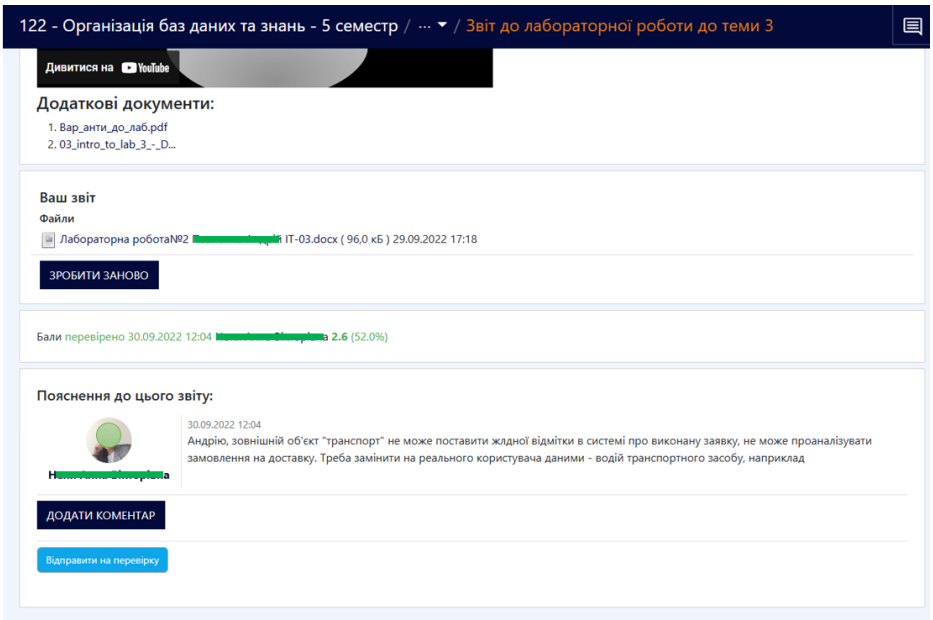


Рис. 9. Візуалізація елементів керування на сторінці MIX

Після активації перевірки звіту починається фоновий процес обробки інформації. На цьому етапі програмний модуль автоматично формує масив динамічних вхідних даних, виконуючи запит до внутрішнього API навчальної платформи. Отримавши відповідь від сервера, алгоритм вилучає структурований JSON-об'єкт із контекстом завдання (тема, постановка задачі), а також ідентифікує прямі посилання на файли звіту студента, розміщені у хмарному сховищі. Детальний перебіг цього фонового процесу можна відстежити через консоль розробника браузера (рис. 10).

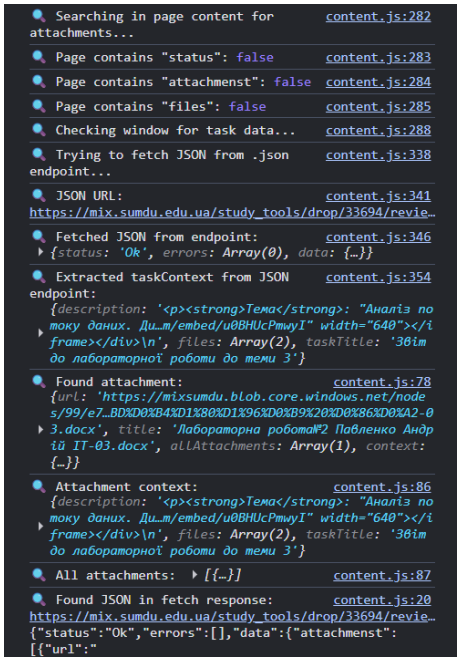


Рис. 10. Процес формування масиву динамічних вхідних даних

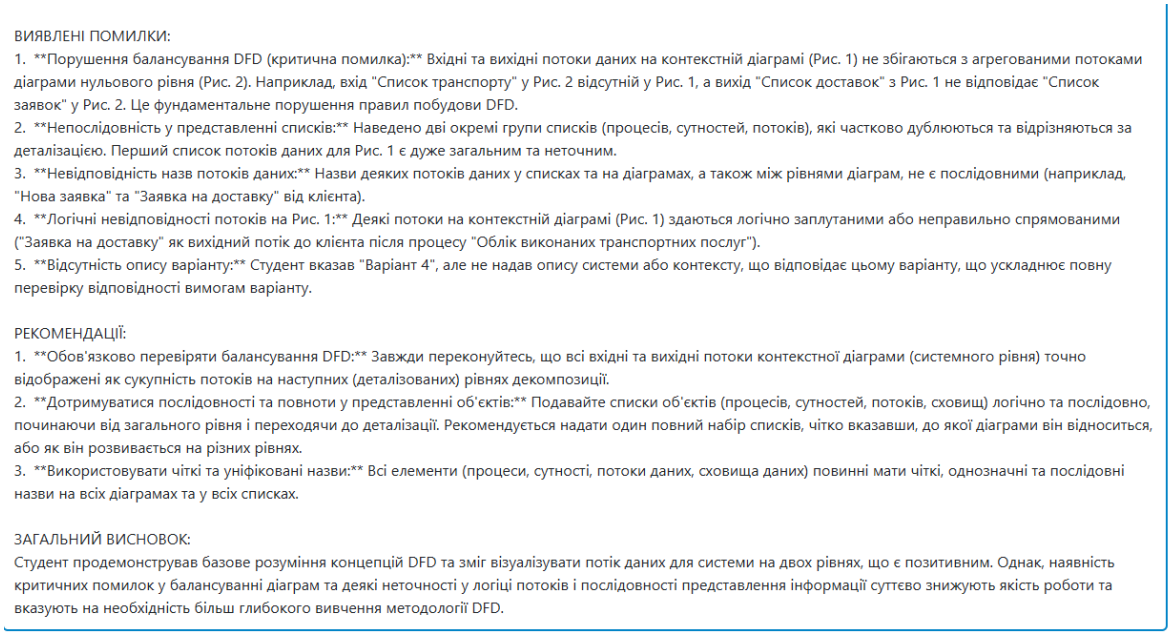
Далі триває етап підготовки зібраних даних до відправлення на сервер нейромережі, що також виконується у фоновому режимі. Текстова складова файлів студентського звіту та методичних вказівок викладача вилучається для подальшої обробки, тоді як графічні елементи файлів (схеми, діаграми) конвертуються у формат Base64, що є необхідною умовою для аналізу даних штучним інтелектом. Вилучені та перетворені дані поєднуються з текстом інструкцій для формування промпту (запиту), який відправляється до нейромережі.

Фінальним кроком є отримання відповіді від API Gemini та її виведення на вебсторінку навчальної платформи у окремому вікні (рис. 11).

Рис. 11. Візуалізація результатів автоматизованої перевірки роботи

Звіт від моделі ШІ охоплює перелік опрацьованих документів, орієнтовний бал у відсотковому співвідношенні, розбір виконаного студентом завдання та список знайдених недоліків. Важливою особливістю є те, що викладач має змогу редагувати текст згенерованого звіту перед тим, як сформувати кінцеві коментарі та виставити підсумкову оцінку. Таким чином, фінальний етап прийняття рішення щодо результату оцінювання роботи залишається за викладачем.

Розроблений програмний модуль дозволяє виявляти не лише синтаксичні помилки, а й складні логічні неточності. Наприклад, на рис. 12 показано, як модуль виявив порушення методології проєктування DFD-схеми (відсутність балансування потоків даних між різними рівнями), що підтверджує високий рівень семантичного розуміння контексту завдання.



ВИЯВЛЕНІ ПОМИЛКИ:
1. **Порушення балансування DFD (критична помилка):** Вхідні та вихідні потоки даних на контекстній діаграмі (Рис. 1) не збігаються з агрегованими потоками діаграми нульового рівня (Рис. 2). Наприклад, вхід "Список транспорту" у Рис. 2 відсутній у Рис. 1, а вихід "Список доставок" з Рис. 1 не відповідає "Список заявок" у Рис. 2. Це фундаментальне порушення правил побудови DFD.
2. **Непослідовність у представленні списків:** Наведено дві окремі групи списків (процесів, сутностей, потоків), які частково дублюються та відрізняються за деталізацією. Перший список потоків даних для Рис. 1 є дуже загальним та неточним.
3. **Невідповідність назв потоків даних:** Назви деяких потоків даних у списках та на діаграмах, а також між рівнями діаграм, не є послідовними (наприклад, "Нова заявка" та "Заявка на доставку" від клієнта).
4. **Логічні невідповідності потоків на Рис. 1:** Деякі потоки на контекстній діаграмі (Рис. 1) здаються логічно заплутаними або неправильно спрямованими ("Заявка на доставку" як вихідний потік до клієнта після процесу "Облік виконаних транспортних послуг").
5. **Відсутність опису варіанту:** Студент вказав "Варіант 4", але не надав опису системи або контексту, що відповідає цьому варіанту, що ускладнює повну перевірку відповідності вимогам варіанту.

РЕКОМЕНДАЦІЇ:
1. **Обов'язково перевіряти балансування DFD:** Завжди переконуйтесь, що всі вхідні та вихідні потоки контекстної діаграми (системного рівня) точно відображені як сукупність потоків на наступних (деталізованих) рівнях декомпозиції.
2. **Дотримуватися послідовності та повноти у представленні об'єктів:** Подавайте списки об'єктів (процесів, сутностей, потоків, сховищ) логічно та послідовно, починаючи від загального рівня і переходячи до деталізації. Рекомендується надати один повний набір списків, чітко вказавши, до якої діаграми він відноситься, або як він розвивається на різних рівнях.
3. **Використовувати чіткі та уніфіковані назви:** Всі елементи (процеси, сутності, потоки даних, сховища даних) повинні мати чіткі, однозначні та послідовні назви на всіх діаграмах та у всіх списках.

ЗАГАЛЬНИЙ ВИСНОВОК:
Студент продемонстрував базове розуміння концепцій DFD та зміг візуалізувати потік даних для системи на двох рівнях, що є позитивним. Однак, наявність критичних помилок у балансуванні діаграм та деякі неточності у логіці потоків і послідовності представлення інформації суттєво знижують якість роботи та вказують на необхідність більш глибокого вивчення методології DFD.

Рис. 12. Деталізація виявлених логічних помилок у роботі студента

Перевірка програмного модуля у середовищі навчальної платформи MIX (СумДУ) підтвердила коректність роботи ключових компонентів системи:

- вдалу валідацію даних (API-ключів) для авторизації користувачів (рис. 8);
- успішну інтеграцію необхідних елементів керування у структуру вебсторінки навчальної платформи (рис. 9);
- автоматичне вилучення динамічних даних (контексту завдання та посилань на файли) через внутрішнє API платформи (рис. 10);
- коректне вилучення тексту і візуальних матеріалів з документів звіту та їх конвертацію у формат Base64 (процес відбувається у фоновому режимі);
- вдале формування контекстного запиту шляхом інтеграції системних інструкцій та зібраних даних (процес відбувається у фоновому режимі);
- можливість автоматичного формування обґрунтованих рецензій на студентські роботи (рис. 11);
- можливість виявляти не лише синтаксичні, а й логічні помилки у тексті студентської роботи (рис. 12).

Перевага розробленого програмного модуля полягає в тому, що результат автоматизованої перевірки відкривається у текстовому вікні, безпосередньо на вебсторінці навчальної платформи, з можливістю внесення змін з боку викладач перед відправленням результату студенту. Це дозволяє викладачу виправити можливі неточності (галюцинації) з боку штучного інтелекту та прийняти остаточне рішення щодо результату оцінювання роботи.

Обговорення результатів.

Одним з ключових показників ефективності запропонованого підходу є якість автоматизованого оцінювання студентських робіт. Для перевірки ефективності роботи системи було проведено експеримент на вибірці з 10 лабораторних робіт, які попередньо були перевірені та оцінені викладачем вручну. Аналіз результатів оцінювання здійснювався за двома критеріями:

кількісним (відхилення балів штучного інтелекту від оцінки викладача) та якісним (релевантність коментарів і точність виявлення помилок).

Результати порівняльного аналізу ручної та автоматизованої перевірки наведено в табл. 1.

Таблиця 1 – Порівняльний аналіз ручного та автоматизованого оцінювання лабораторних робіт

№ роботи	**Оцінка викладача (бали)**	**Оцінка ШІ (бали)**	**Відхилення (%)**	**Відповідність варіанту**	**Час ручної перевірки (хв)**	**Час генерації рецензії системою (хв)**
1	95	90	5%	Так	12	0.6
2	80	85	6%	Так	15	0.7
3	60	45	25%	Так	18	0.8
4	100	95	5%	Так	10	0.5
5	75	80	6%	Так	14	0.6
6	90	92	2%	Так	12	0.6
7	85	80	5%	Так	15	0.7
8	50	65	30%	Так	20	0.9
9	100	100	0%	Так	10	0.5
10	70	75	7%	Так	16	0.8

Аналіз отриманих результатів дозволяє зробити такі висновки:

– у 80% випадків підсумковий бал, згенерований мовною моделлю, відрізнявся від оцінки експерта (викладача) не більше ніж на 10–15% (табл. 1). Це свідчить про те, що алгоритм здатний адекватно інтерпретувати критерії оцінювання, задані в промпті. У роботах №3 та №8 зафіксовано найбільше відхилення через специфіку творчої складової завдань, яку ШІ оцінив інакше, ніж викладач;

– штучний інтелект успішно ідентифікував не лише базові синтаксичні, а й складні логічні та методологічні помилки. Наприклад, під час перевірки SQL-запитів до баз даних згенерована рецензія містила такі вказівки: у запиті проігноровано обов'язковий оператор JOIN, використано некоректні умови з'єднання таблиць (наприклад, порівняння невідповідних полів), умову завдання виконано не повністю (рис. 13). Зафіксовані системою недоліки повністю збігалися із зауваженнями експерта (викладача);

– штучний інтелект успішно ідентифікував 95% синтаксичних помилок у програмному коді. Наприклад, під час перевірки баз даних модуль точно вказував на відсутність умови JOIN у SQL-запитах студентів (рис. 13), що повністю збігалося із зауваженнями експерта;

– у 100% тестових випадків система коректно визначала відповідність виконаної роботи заданому варіанту;

– впровадження розробленої системи дозволило оптимізувати часові витрати: середня тривалість ручної перевірки однієї роботи склала 14,2 хв, тоді як автоматизоване формування розгорнутої рецензії займало в середньому 0,67 хв (≈ 40 секунд), що підтверджує ефективність системи щодо заощадження часу викладача (табл. 1).

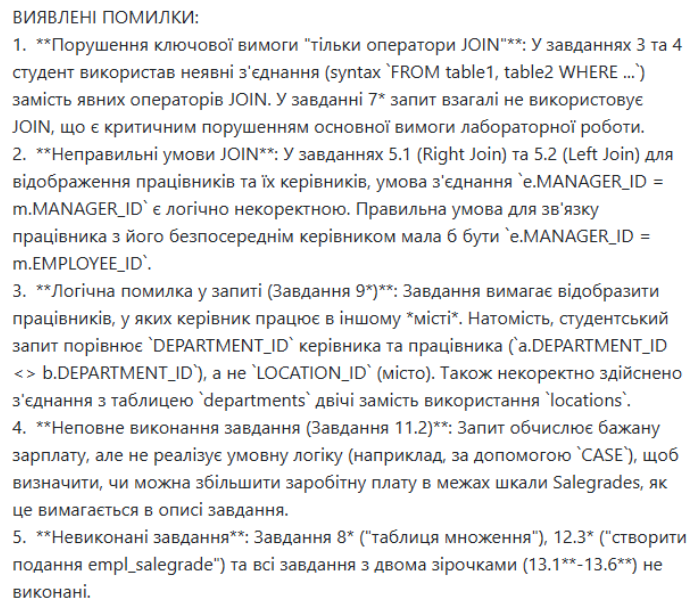

ВИЯВЛЕНІ ПОМИЛКИ:
1. **Порушення ключової вимоги "тільки оператори JOIN"**: У завданнях 3 та 4 студент використав неявні з'єднання (syntax `FROM table1, table2 WHERE ...`) замість явних операторів JOIN. У завданні 7* запит взагалі не використовує JOIN, що є критичним порушенням основної вимоги лабораторної роботи.
2. **Неправильні умови JOIN**: У завданнях 5.1 (Right Join) та 5.2 (Left Join) для відображення працівників та їх керівників, умова з'єднання `e.MANAGER_ID = m.MANAGER_ID` є логічно некоректною. Правильна умова для зв'язку працівника з його безпосереднім керівником мала б бути `e.MANAGER_ID = m.EMPLOYEE_ID`.
3. **Логічна помилка у запиті (Завдання 9*)**: Завдання вимагає відобразити працівників, у яких керівник працює в іншому *місті*. Натомість, студентський запит порівнює `DEPARTMENT_ID` керівника та працівника (`a.DEPARTMENT_ID <> b.DEPARTMENT_ID`), а не `LOCATION_ID` (місто). Також некоректно здійснено з'єднання з таблицею `departments` двічі замість використання `locations`.
4. **Неповне виконання завдання (Завдання 11.2)**: Запит обчислює бажану зарплату, але не реалізує умовну логіку (наприклад, за допомогою `CASE`), щоб визначити, чи можна збільшити заробітну плату в межах шкали Salegrades, як це вимагається в описі завдання.
5. **Невиконані завдання**: Завдання 8* ("таблиця множення"), 12.3* ("створити подання empl_salegrade") та всі завдання з двома зірочками (13.1**-13.6**) не виконані.

Рис. 13. Фрагмент згенерованої рецензії з аналізом помилок у SQL-запитах

Підсумовуючи результати експерименту, можна стверджувати, що запропонована система демонструє високу ефективність як інтелектуальний асистент викладача. Завдяки тому, що розроблений програмний модуль коректно вилучає дані та формує точні контекстні запити, залучена мовна модель успішно виконує рутинну частину роботи: пошук синтаксичних і логічних помилок, аналіз програмного коду, контроль відповідності виконаного завдання заданому варіанту. Зафіксовані розбіжності в оцінюванні творчих завдань підтверджують, що штучний інтелект не замінює викладача. Він виступає допоміжним аналітичним інструментом, який формує обґрунтовану рецензію на роботу студента, заощаджуючи час, та залишає за викладачем право коригувати згенерований відгук і ухвалювати остаточне рішення щодо підсумкової оцінки.

Висновки та перспективи подальших досліджень.

У результаті проведеного дослідження вирішено задачу автоматизації перевірки студентських робіт шляхом розробки та реалізації спеціалізованого програмного модуля, що працює як розширення браузера. Модуль у фоновому режимі забезпечує автоматичний збір вхідних даних з вебсторінки навчальної платформи, їх структурування, формування контекстного запиту та його пряме відправлення до API мовної моделі (без використання проміжних серверів). Після отримання результату від нейромережі розширення відображає його на сторінці навчальної платформи у текстовому вигляді, відкритому для подальшого коригування викладачем.

Робота з розширенням браузера передбачає гнучке налаштування системних інструкцій, що задають формат перевірки студентських робіт. Це дозволяє викладачу адаптувати роботу модуля під специфіку різних дисциплін. Також реалізовано алгоритм автоматичного перебору альтернативних моделей, що забезпечує безперебійну роботу модуля у випадку перевантаження чи недоступності основної нейромережі.

Проведений експеримент на вибірці з 10 лабораторних робіт показав, що у 80% випадків відхилення балів ШІ від оцінки викладача не перевищувало 10–15%, точність визначення відповідності варіанту склала 100%, а ефективність виявлення синтаксичних та логічних помилок у SQL-коді досягла 95%. Також доведено часову ефективність: використання модуля дозволило скоротити середній час аналізу однієї роботи приблизно у 20 разів (із 14,2 до 0,67 хвилини).

Запропонований підхід дозволив створити інтелектуального асистента викладача, який автоматизує процес перевірки студентських робіт у середовищі навчальної платформи. При цьому в системі зберігається концепція «людина в контурі управління»: штучний інтелект формує обґрунтовану аналітичну рецензію, залишаючи за викладачем вирішальне право коригувати відгук та ухвалювати остаточне рішення щодо оцінки.

Перспективи подальших науково-практичних досліджень спрямовані на адаптацію розробленого модуля під інші навчальні платформи (Moodle, Canvas, Google Classroom) та перевірку

ефективності модуля під час оцінювання дисциплін гуманітарного профілю, де критерії оцінювання є менш формалізованими.

Отримані результати мають практичну та наукову цінність для цифровізації вищої освіти. Водночас виявлені обмеження (розбіжності в оцінюванні завдань із творчою складовою до 25–30%) підтверджують, що мовні моделі вимагають обов'язкового експертного контролю з боку викладача.

Розроблений модуль рекомендовано використовувати як браузерне розширення у межах електронних навчальних платформ для автоматизованої перевірки студентських робіт з IT-дисциплін.

Список бібліографічного опису

1. Marchenko A., Antypenko V., Vashchenko S., Fedotova N., Chybiriak Y., Krasulia A. A Complex Model of Blended Learning: Using a Project Approach to Organize the Educational Process. *Communications in Computer and Information Science*. 2021. Vol. 1486. P. 265–278. DOI: 10.1007/978-3-030-88304-1_21.
2. Lavrov E., Chybiriak Y., Siryk O., Logvinenko V., Zakharova A. Training of Specialists for Adaptive management. Techniques for Teaching Computer Analysis of Automated Production Systems in the FlexSim Environment. *CEUR Workshop Proceedings*. 2022. URL: https://www.scopus.com/pages/publications/85127439894?origin=resultslist (date of access: 06.08.2026).
3. Lavrov E. A., Logvinenko V. G., Osadchyi V. V., Siryk O. Ye., Chybiriak Y. I. Adaptive learning system based on cognitive independence. *CEUR Workshop Proceedings*. 2023. URL: https://www.scopus.com/pages/publications/85177242601?origin=resultslist (date of access: 06.08.2026).
4. Чибіряк Я., Баранова І., Ніколаєнко К. Метод наскрізного навчання студентів IT-спеціальностей імітаційному моделюванню у середовищі FlexSim для пошуку резервів підвищення ефективності автоматизованих систем. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2021. Вип. 42. С. 119–129. DOI: 10.36910/6775-2524-0560-2021-42-17.
5. Lavrov E., Siryk O. Human Factor in E-learning. Challenges and Methodology for Ergonomic Support. *Springer Series in Design and Innovation*. Springer Nature, 2025. Vol. 53. P. 137–145. DOI: 10.1007/978-3-031-88134-3_18.
6. Messer M., Brown N. C. C., Kölling M., Shi M. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Transactions on Computing Education*. 2024. Vol. 24, no. 1. P. 10. DOI: 10.1145/3636515.
7. Combéfis S., de Moffarts G. Automated Generation of Computer Graded Unit Testing-Based Programming Assessments for Education. 2019. P. 91–100. DOI: 10.5121/CSIT.2019.91308. *(Примітка: у вашому вихідному тексті не вказано назву журналу чи збірника для цього джерела, тому оформлено на основі наявних даних).*
8. Challen G., Nordick B. Accelerating Accurate Assignment Authoring Using Solution-Generated Autograders. *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*. 2025. Vol. 1. P. 227–233. DOI: 10.1145/3641554.3701862.
9. Natella R. Evaluation of Systems Programming Exercises through Tailored Static Analysis. *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*. 2025. Vol. 1. P. 826–832. DOI: 10.1145/3641554.3701836.
10. Tseng E. Q., Huang P. C., Hsu C., Wu P. Y., Ku C. T., Kang Y. CodEv: An Automated Grading Framework Leveraging Large Language Models for Consistent and Constructive Feedback. *Proceedings - 2024 IEEE International Conference on Big Data, BigData 2024*. 2024. P. 5442–5449. DOI: 10.1109/BIGDATA62323.2024.10825949.
11. Yousef M., Mohamed K., Medhat W., Mohamed E. H., Khoriba G., Arafa T. BeGrading: large language models for enhanced feedback in programming education. *Neural Computing and Applications*. 2024. Vol. 37, no. 2. P. 1027–1040. DOI: 10.1007/S00521-024-10449-Y.
12. Liu Y., Iter D., Xu Y., Wang S., Xu R., Zhu C. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. *EMNLP 2023 - 2023 Conference on Empirical Methods in Natural Language Processing, Proceedings*. 2023. P. 2511–2522. DOI: 10.18653/V1/2023.EMNLP-MAIN.153.
13. Mazzone S. B., Forden J., Brylow D. Exploring the Potential of Locally Run Large Language (AI) Models for Automated Grading in Introductory Computer Science Courses. *Proceedings - Frontiers in Education Conference, FIE*. 2024. DOI: 10.1109/FIE61694.2024.10892816.
14. Xue M., Liu Y., Xiao X., Wilson M. Automatic Prompt Engineering for Automatic Scoring. *J. Educ. Meas.* 2025. Vol. 62, no. 4. P. 559–587. DOI: 10.1111/JEDM.70002.
15. Ramnath K. et al. A Systematic Survey of Automatic Prompt Optimization Techniques. *EMNLP 2025 - 2025 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. 2025. P. 33078–33110. DOI: 10.18653/V1/2025.EMNLP-MAIN.1681.
16. Frankford E. et al. A Survey on Feedback Types in Automated Programming Assessment Systems. *ACM Transactions on Computing Education*. 2025. Vol. 26, no. 1. DOI: 10.1145/3773911.
17. Silva P., Costa E. Assessing Large Language Models for Automated Feedback Generation in Learning Programming Problem Solving. *Proc. Mach. Learn. Res.* 2025. Vol. 273. P. 116–124. URL: https://arxiv.org/pdf/2503.14630 (date of access: 03.08.2026).
18. Heickal H., Lan A. Generating Feedback-Ladders for Logical Errors in Programming using Large Language Models. *Proceedings of the International Conference on Educational Data Mining*. 2024. P. 947–951. DOI: 10.5281/zenodo.12730007.
19. Paiva J. C., Leal J. P., Figueira Á. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education*. 2022. Vol. 22, no. 3. P. 34. DOI: 10.1145/3513140.
20. Learning Tools Interoperability Advantage Implementation Guide. URL: https://standards.1edtech.org/lti/guides/implementation_guide/implementation-guide (date of access: 03.08.2026).

21. Panjaitan B. M., Rukmono S. A., Perdana R. S. Integration Model for Learning Management Systems, Source Control Management, and Autograders using Service-Oriented Architecture Principle. *Proceedings of 2021 International Conference on Data and Software Engineering: Data and Software Engineering for Supporting Sustainable Development Goals, ICoDSE 2021*. 2021. DOI: 10.1109/ICODSE53690.2021.9648450.
22. Security Framework | 1EdTech. URL: https://www.1edtech.org/standards/security-framework (date of access: 04.08.2026).
23. Booth S., Peacock S., Vickers S. P. Plug and play learning application integration using IMS Learning Tools Interoperability. *ASCILITE Publications*. 2011. P. 143–147. DOI: 10.14742/APUBS.2011.1860.
24. 1EdTech Data Privacy | IMS Global Learning Consortium. URL: https://www.imsglobal.org/spec/privacy/v1p0 (date of access: 04.08.2026).
25. Kaleci D. Integration and application of artificial intelligence tools in the Moodle platform: A theoretical exploration. *Journal of Educational Technology and Online Learning*. 2025. Vol. 8, no. 1. P. 100–111. DOI: 10.31681/JETOL.1595079.
26. Configuring Feedback Studio in Moodle LTI 1.3 – Turnitin Guides. URL: https://guides.turnitin.com/hc/en-us/articles/21363693663117-Configuring-Feedback-Studio-in-Moodle-LTI-1-3 (date of access: 04.08.2026).
27. Sanfilippo M. R., Apthorpe N., Brehm K., Shvartzshnaider Y. Privacy governance not included: analysis of third parties in learning management systems. *Information and Learning Sciences*. 2023. Vol. 124, no. 9–10. P. 326–348. DOI: 10.1108/ILS-04-2023-0033.
28. Upadhyay S. Learning Tools Interoperability: AI-Driven Integration, Security Innovations, and Future Trends in Digital Education. *Future Trends in E-Learning Technologies International Journal on Science and Technology*. Vol. 16, no. 1.
29. Alvarez Rodriguez J. M., O'shaughnessy D., Vangelova A., Gancheva V. LLMs in Automated Assessment: A Role-Based Taxonomy and Framework for Controlled Educational Integration. *Applied Sciences*. 2026. Vol. 16, no. 13. P. 6617. DOI: 10.3390/APP16136617.
30. Vangelova A., Aleksieva-Petrova A. An Integrated RAG and Agent-Based Architecture for Automated Assessment in Moodle. *Mach. Learn. Knowl. Extr.* 2026. Vol. 8, no. 5. DOI: 10.3390/MAKE8050127.
31. Ekambaram L., Muni Jyothi Sai A. AIPowered Automated Quiz Creation and Assessment System with Gemini Large Language Model and LangChain. *Journal of Emerging Trends and Novel Research*. 2026. Vol. 4, no. 3. DOI: 10.56975/JETNR.V4I3.233196.
32. Del Carpio P. M. Towards more Interactive Recordings for E-learning Using Browser Automation. *Proceedings of the 2021 IEEE Engineering International Research Conference, EIRCON 2021*. 2021. DOI: 10.1109/EIRCON52903.2021.9613385.

References

1. Marchenko A., Antypenko V., Vashchenko S., Fedotova N., Chybiriak Y., Krasulia A. A Complex Model of Blended Learning: Using a Project Approach to Organize the Educational Process. *Communications in Computer and Information Science*. 2021. Vol. 1486. P. 265–278. DOI: 10.1007/978-3-030-88304-1_21.
2. Lavrov E., Chybiriak Y., Siryk O., Logvinenko V., Zakharova A. Training of Specialists for Adaptive management. Techniques for Teaching Computer Analysis of Automated Production Systems in the FlexSim Environment. *CEUR Workshop Proceedings*. 2022. URL: https://www.scopus.com/pages/publications/85127439894?origin=resultslist (date of access: 06.08.2026).
3. Lavrov E. A., Logvinenko V. G., Osadchyi V. V., Siryk O. Ye., Chybiriak Y. I. Adaptive learning system based on cognitive independence. *CEUR Workshop Proceedings*. 2023. URL: https://www.scopus.com/pages/publications/85177242601?origin=resultslist (date of access: 06.08.2026).
4. Chybiriak Ya., Baranova I., Nikolaienko K. Metod naskriznoho navchannia studentiv IT-spetsialnostei imitatsiinomu modeliuvanniu u seredovyshchi FlexSim dlia poshuku rezerviv pidvyshchennia efektyvnosti avtomatyzovanykh system [The method of end-to-end training of IT students in simulation modeling in the FlexSim environment to find reserves for increasing the efficiency of automated systems]. *Kompiuterno-intehrovani tekhnolohii: osvita, nauka, vyrobnytstvo*. 2021. No. 42. P. 119–129. DOI: 10.36910/6775-2524-0560-2021-42-17 (in Ukrainian).
5. Lavrov E., Siryk O. Human Factor in E-learning. Challenges and Methodology for Ergonomic Support. *Springer Series in Design and Innovation*. Springer Nature, 2025. Vol. 53. P. 137–145. DOI: 10.1007/978-3-031-88134-3_18.
6. Messer M., Brown N. C. C., Kölling M., Shi M. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Transactions on Computing Education*. 2024. Vol. 24, no. 1. P. 10. DOI: 10.1145/3636515.
7. Combéfis S., de Moffarts G. Automated Generation of Computer Graded Unit Testing-Based Programming Assessments for Education. 2019. P. 91–100. DOI: 10.5121/CSIT.2019.91308. *(Примітка: у вашому вихідному тексті не вказано назву журналу чи збірника для цього джерела, тому оформлено на основі наявних даних).*
8. Challen G., Nordick B. Accelerating Accurate Assignment Authoring Using Solution-Generated Autograders. *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*. 2025. Vol. 1. P. 227–233. DOI: 10.1145/3641554.3701862.
9. Natella R. Evaluation of Systems Programming Exercises through Tailored Static Analysis. *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*. 2025. Vol. 1. P. 826–832. DOI: 10.1145/3641554.3701836.
10. Tseng E. Q., Huang P. C., Hsu C., Wu P. Y., Ku C. T., Kang Y. CodEv: An Automated Grading Framework Leveraging Large Language Models for Consistent and Constructive Feedback. *Proceedings - 2024 IEEE International Conference on Big Data, BigData 2024*. 2024. P. 5442–5449. DOI: 10.1109/BIGDATA62323.2024.10825949.
11. Yousef M., Mohamed K., Medhat W., Mohamed E. H., Khoriba G., Arafa T. BeGrading: large language models for enhanced feedback in programming education. *Neural Computing and Applications*. 2024. Vol. 37, no. 2. P. 1027–1040. DOI: 10.1007/S00521-024-10449-Y.

12. Liu Y., Iter D., Xu Y., Wang S., Xu R., Zhu C. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. *EMNLP 2023 - 2023 Conference on Empirical Methods in Natural Language Processing, Proceedings*. 2023. P. 2511–2522. DOI: 10.18653/V1/2023.EMNLP-MAIN.153.
13. Mazzone S. B., Forden J., Brylow D. Exploring the Potential of Locally Run Large Language (AI) Models for Automated Grading in Introductory Computer Science Courses. *Proceedings - Frontiers in Education Conference, FIE*. 2024. DOI: 10.1109/FIE61694.2024.10892816.
14. Xue M., Liu Y., Xiao X., Wilson M. Automatic Prompt Engineering for Automatic Scoring. *J. Educ. Meas.* 2025. Vol. 62, no. 4. P. 559–587. DOI: 10.1111/JEDM.70002.
15. Ramnath K. et al. A Systematic Survey of Automatic Prompt Optimization Techniques. *EMNLP 2025 - 2025 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. 2025. P. 33078–33110. DOI: 10.18653/V1/2025.EMNLP-MAIN.1681.
16. Frankford E. et al. A Survey on Feedback Types in Automated Programming Assessment Systems. *ACM Transactions on Computing Education*. 2025. Vol. 26, no. 1. DOI: 10.1145/3773911.
17. Silva P., Costa E. Assessing Large Language Models for Automated Feedback Generation in Learning Programming Problem Solving. *Proc. Mach. Learn. Res.* 2025. Vol. 273. P. 116–124. URL: https://arxiv.org/pdf/2503.14630 (date of access: 03.08.2026).
18. Heickal H., Lan A. Generating Feedback-Ladders for Logical Errors in Programming using Large Language Models. *Proceedings of the International Conference on Educational Data Mining*. 2024. P. 947–951. DOI: 10.5281/zenodo.12730007.
19. Paiva J. C., Leal J. P., Figueira Á. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education*. 2022. Vol. 22, no. 3. P. 34. DOI: 10.1145/3513140.
20. Learning Tools Interoperability Advantage Implementation Guide. URL: https://standards.1edtech.org/lti/guides/implementation_guide/implementation-guide (date of access: 03.08.2026).
21. Panjaitan B. M., Rukmono S. A., Perdana R. S. Integration Model for Learning Management Systems, Source Control Management, and Autograders using Service-Oriented Architecture Principle. *Proceedings of 2021 International Conference on Data and Software Engineering: Data and Software Engineering for Supporting Sustainable Development Goals, ICoDSE 2021*. 2021. DOI: 10.1109/ICODSE53690.2021.9648450.
22. Security Framework | 1EdTech. URL: https://www.1edtech.org/standards/security-framework (date of access: 04.08.2026).
23. Booth S., Peacock S., Vickers S. P. Plug and play learning application integration using IMS Learning Tools Interoperability. *ASCILITE Publications*. 2011. P. 143–147. DOI: 10.14742/APUBS.2011.1860.
24. 1EdTech Data Privacy | IMS Global Learning Consortium. URL: https://www.imsglobal.org/spec/privacy/v1p0 (date of access: 04.08.2026).
25. Kaleci D. Integration and application of artificial intelligence tools in the Moodle platform: A theoretical exploration. *Journal of Educational Technology and Online Learning*. 2025. Vol. 8, no. 1. P. 100–111. DOI: 10.31681/JETOL.1595079.
26. Configuring Feedback Studio in Moodle LTI 1.3 – Turnitin Guides. URL: https://guides.turnitin.com/hc/en-us/articles/21363693663117-Configuring-Feedback-Studio-in-Moodle-LTI-1-3 (date of access: 04.08.2026).
27. Sanfilippo M. R., Apthorpe N., Brehm K., Shvartzshnaider Y. Privacy governance not included: analysis of third parties in learning management systems. *Information and Learning Sciences*. 2023. Vol. 124, no. 9–10. P. 326–348. DOI: 10.1108/ILS-04-2023-0033.
28. Upadhyay S. Learning Tools Interoperability: AI-Driven Integration, Security Innovations, and Future Trends in Digital Education. *Future Trends in E-Learning Technologies International Journal on Science and Technology*. Vol. 16, no. 1.
29. Alvarez Rodriguez J. M., O'shaughnessy D., Vangelova A., Gancheva V. LLMs in Automated Assessment: A Role-Based Taxonomy and Framework for Controlled Educational Integration. *Applied Sciences*. 2026. Vol. 16, no. 13. P. 6617. DOI: 10.3390/APP16136617.
30. Vangelova A., Aleksieva-Petrova A. An Integrated RAG and Agent-Based Architecture for Automated Assessment in Moodle. *Mach. Learn. Knowl. Extr.* 2026. Vol. 8, no. 5. DOI: 10.3390/MAKE8050127.
31. Ekambaram L., Muni Jyothi Sai A. AIPowered Automated Quiz Creation and Assessment System with Gemini Large Language Model and LangChain. *Journal of Emerging Trends and Novel Research*. 2026. Vol. 4, no. 3. DOI: 10.56975/JETNR.V4I3.233196.
32. Del Carpio P. M. Towards more Interactive Recordings for E-learning Using Browser Automation. *Proceedings of the 2021 IEEE Engineering International Research Conference, EIRCON 2021*. 2021. DOI: 10.1109/EIRCON52903.2021.9613385.

Історія статті:
Отримано: 30.05.2026 Доопрацьовано: 28.08.2026 Прийнято до друку: 23.09.2026 Опубліковано: 29.09.2026