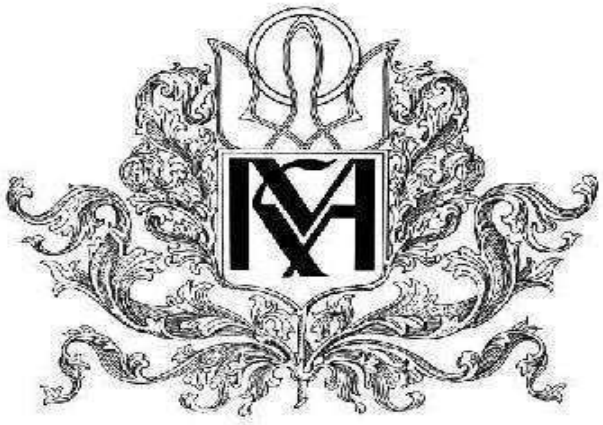
Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики



Використання фреймворку Lean для перевірки коректності математичних міркувань

Текстова частина до дипломної роботи за спеціальністю

„Комп'ютерні науки" - 122

Керівник курсової роботи

доцент, кандидат ф.-м. наук Жежерун О.П.

_____

(підпис)

"____" _____ 2026 р.

Виконав студент 2 року навчання

Спеціальності 122 Комп'ютерні науки

Кривошея О.О.

"____" _____ 2026р.

Київ 2026

Національний університет «Києво-Могилянська академія»

Факультет ___інформатики_____

Кафедра _____мультимедійних_систем_____

Освітній ступінь _____магістр_____

Спеціальність _____122_Комп'ютеоні_науки_____ (код і назва)

Освітня/освітньо-наукова програма _____Комп'ютерні_науки_____ (назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_Жежерун_Олександр_Петровмч_____

«___» _____ 20 ___ року

ЗАВДАННЯ
ДЛЯ КВАЛІФІКАЦІЙНОЇ / МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

_Кривошея_Олександр_Олександрович_____ (ПІБ)

1. Тема роботи __Використання фреймворку Lean для перевірки коректності математичних міркувань._____

керівник роботи
__Жежерун_Олександр_Петрович,_доцент,_кандидат_наук_____
_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «___» _____ 20 ___ року № ___

2. Строк подання студентом роботи _____

3. План роботи _____

Графік підготовки кваліфікаційної/магістерської роботи до захисту

№ З/П	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	жовтень			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних.	жовтень — листопад			
3.	Складання плану каліф. роботи та узгодження з науковим керівником.	до 14 грудня			
4.	Постановка експерименту, аналіз отриманих результатів.	листопад — березень			
5.	Проміжний контроль виконання роботи.	до 31 січня			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника:	грудень			
	Здійснити огляд літератури щодо автоматизованого доведення теорем та аналіз можливостей Lean 4. Дослідити поточні проблеми викладання геометрії в середній школі.	грудень — березень			
	Розробити формалізацію базових геометричних аксіом та теорем у середовищі Lean. Спроектувати архітектуру модуля перевірки учнівських рішень для рекомендаційної системи	січень — лютий			
	Провести тестування розробленого модуля на наборі типових геометричних задач. Оцінити коректність роботи алгоритму та розробити рекомендації щодо впровадження системи в навчальний процес.	лютий — березень			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання чорнової версії на відгук науковому керівнику.	до 01 квітня			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності.	квітень			
9.	Подання на зовнішню рецензію.	до 15 травня			
10.	Підготовка до захисту кваліфікаційної роботи на засіданні кафедри: написання доповіді та виготовлення ілюстративного матеріалу	травень			
11.	Попередній захист кваліфікаційної роботи на засіданні кафедри.	28 квітня			
12.	Подання кваліфікаційної роботи на кафедру з усіма супровідними документами	до 25 травня			
13.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією.	08-09 червня			

Графік узгоджено: « ___ » _____ 20 ___ р.

Науковий керівник: _Жежерун_Олександр_Петрович_____ (ПІБ)

Виконавець кваліфікаційної роботи: _Кривошея_Олександр_Олександрович_____ (ПІБ)

Зміст

Анотація

Магістерську роботу присвячено розробці та дослідженню інтелектуальної програмної системи автоматизованої перевірки розгорнутих геометричних міркувань учнів 7–9 класів на основі фреймворку Lean 4 та великих мовних моделей.

Об'єкт дослідження: процеси автоматизованої перевірки правильності математичних викладів у інтелектуальних навчальних середовищах.

Предмет дослідження: методи та програмні інструменти інтеграції середовища Lean у рекомендаційну систему для перевірки геометричних доведень.

Мета дослідження: розробити та дослідити архітектуру програмного додатка, який використовує ядро Lean для перевірки геометричних міркувань учнів, діагностує рівень їхньої підготовки та надає адаптивні рекомендації на основі виявлених прогалин.

Ключові слова: Lean 4, перевірка математичних міркувань, нейро-символічна інтеграція, формальна верифікація, великі мовні моделі, планіметрія, формувальне оцінювання.

Вступ

Стрімкий розвиток цифрових технологій в освіті зумовлює необхідність створення інтелектуальних систем, здатних не лише надавати навчальний контент, а й здійснювати якісну перевірку знань у режимі реального часу. Однією з найбільш критичних областей є навчання геометрії в середній школі, де оцінка результатів традиційно базується на фінальній відповіді, ігноруючи сам ланцюжок логічних міркувань. Це створює проблему об'єктивної діагностики знань, оскільки вчитель не завжди має змогу детально проаналізувати кожен крок доведення у великих групах учнів.

Актуальність дослідження зумовлена необхідністю впровадження методів формальної верифікації у сферу EdTech. Використання інтерактивних доводжувачів теорем, зокрема фреймворку Lean, дозволяє перевести процес перевірки геометричних завдань на рівень математично суворої верифікації. Це дозволяє уникнути неоднозначностей у трактуванні кроків доведення та забезпечує учня миттєвим, логічно обґрунтованим зворотним зв'язком. Створення системи, що інтегрує Lean для аналізу міркувань, є актуальним кроком у розвитку персоналізованого навчання та автоматизації освітнього процесу.

Робота виконується в рамках спеціальності 122 «Комп'ютерні науки» і спрямована на вдосконалення методів інтелектуального аналізу даних та розробку спеціалізованого програмного забезпечення для освітніх потреб.

Метою магістерської роботи є розробка та дослідження архітектури програмного додатка, який використовує ядро Lean для верифікації геометричних міркувань учнів, діагностує рівень їхньої підготовки та надає адаптивні рекомендації на основі виявлених прогалин.

Для досягнення поставленої мети необхідно вирішити такі наукові та практичні завдання:

1. Виконати аналіз існуючих підходів до автоматизованої перевірки математичних доведень.
2. Описати механізми формалізації геометричних аксіом та теорем засобами мови Lean.
3. Проектувати архітектуру взаємодії між клієнтською частиною (інтерфейсом учня) та серверним модулем верифікації на базі Lean.
4. Розробити алгоритм співставлення формалізованих помилок з конкретними темами навчальної програми.
5. Провести експериментальну перевірку ефективності системи на прикладі типових задач шкільного курсу геометрії.

Об'єкт дослідження — процеси автоматизованої перевірки правильності математичних викладів у інтелектуальних навчальних середовищах.

Предмет дослідження — методи та програмні інструменти інтеграції середовища Lean у рекомендаційну систему для верифікації геометричних доведень.

У роботі використано методи об'єктно-орієнтованого проектування для розробки структури системи, методи формальної логіки та теорії доведень для верифікації міркувань, а також статистичні методи для оцінки точності діагностики знань.

Наукова новизна роботи полягає у:

- запропонованому методі трансляції кроків геометричного доведення у формальні терми Lean для їх автоматизованої верифікації;
- удосконаленні алгоритму діагностики знань, що дозволяє виявляти неявні помилки в логічних переходах;
- розробці моделі адаптивного формування рекомендацій на основі графа залежностей математичних теорем.

Результати дослідження можуть бути впроваджені у формі веб-додатка для шкіл та платформ дистанційної освіти, що дозволить підвищити якість підготовки учнів з геометрії та зменшити навантаження на викладачів при перевірці творчих завдань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВЕРИФІКАЦІЇ МАТЕМАТИЧНИХ ЗНАНЬ

1.1. Нормативні вимоги та стратегічні цілі навчання геометрії в умовах НУШ

Сучасний етап розвитку базової середньої освіти в Україні орієнтований на виконання Закону України «Про повну загальну середню освіту» [1] та Державного стандарту базової середньої освіти [2]. Згідно з Модельною навчальною програмою «Геометрія. 7–9 класи» [3], вивчення цієї дисципліни є фундаментальною складовою математичної освітньої галузі, метою якої є не лише передача фактологічних знань, а й всебічний розвиток особистості через формування математичної компетентності.

1.1.1. Компетентнісна парадигма та державні пріоритети

В основу сучасного освітнього процесу покладено компетентнісний підхід, де кінцевим результатом є здатність учня застосовувати набуті знання та методи в реальних життєвих ситуаціях. Цей підхід тісно пов'язаний зі стратегічними державними пріоритетами.

Програма наголошує, що одним із головних викликів сьогодення є збереження та підвищення статусу України як провідної світової держави в наукомістких галузях, зокрема в комп'ютерних, інформаційних, авіаційних та космічних технологіях. Відповідно, підростаюче покоління повинне масово набувати компетенцій, необхідних для професійної орієнтації в цих областях, що вимагає високого рівня технічної грамотності та алгоритмічного мислення. У цьому контексті математика, і геометрія зокрема, позиціонується як універсальна мова науки та техніки, а також ефективний інструмент моделювання процесів навколишнього світу.

1.1.2. Розвиток логічного та алгоритмічного мислення

Специфіка курсу геометрії в 7–9 класах полягає у переході від наочно-інтуїтивного до формально-логічного сприйняття матеріалу. Освітні завдання програми передбачають глибокий розвиток інтелектуального апарату учня.

До ключових вимог належать:

- Здатність до доведення: Формування вміння логічно обґрунтовувати та доводити твердження, а також оцінювати правильність і раціональність розв'язування задач.
- Методологічна грамотність: Опанування математичною термінологією, розуміння сутності аксіом і теорем, володіння методами доведення (включно з індуктивними та дедуктивними міркуваннями), а також уміння формулювати й спростовувати гіпотези.
- Алгоритмізація: Формування здатності до логічних міркувань, висновків та алгоритмічного мислення.

Такі високі вимоги до дедуктивної складової роблять геометрію ідеальною предметною областю для впровадження інтелектуальних систем формальної верифікації міркувань.

1.1.3. Цифровізація та індивідуалізація освітнього процесу

Нормативні документи НУШ чітко артикулюють необхідність глибокої комп'ютеризації та інформатизації навчання. Використання комп'ютерної техніки на уроках геометрії має забезпечити:

- Формування алгоритмічного стилю мислення.

- Уміння оптимізувати свою діяльність, виокремлюючи технічну складову та перекладаючи її на засоби комп'ютерної техніки.

- Набуття навичок використання сучасних комп'ютерних засобів побудови графічних зображень.

Крім того, сучасні реалії вимагають зміщення акцентів на групові та індивідуальні форми роботи для побудови індивідуальних освітніх траєкторій. Це тісно пов'язано з впровадженням формувального оцінювання, метою якого є відстеження навчального прогресу учня, вчасне виявлення проблем та коригування методів навчання. Очевидно, що реалізація такого оцінювання в масштабах класів потребує автоматизованих систем зворотного зв'язку.

1.1.4. Етичний та культурно-історичний потенціал

Шкільний курс геометрії має значний виховний потенціал. Згідно з програмою, учні повинні усвідомити математичні знання як продукт колективної діяльності людства.

- Доказова етика: Формується потреба в доказовому неупередженому обґрунтуванні та об'єктивному оцінюванні висловлювань, рішень та дій.

- Культурна інтеграція: Учні мають усвідомлювати взаємозв'язок математики та культури на прикладах із живопису, архітектури та просторових композицій.

- Патріотичне виховання: Ознайомлення з біографіями видатних науковців, зокрема українських, стимулює потяг до наукової творчості та виховує повагу до національних цінностей.

1.2. Методичні засади реалізації програми: організаційні форми та типологія завдань

Якщо нормативна база визначає стратегічні орієнтири та компетентнісний потенціал геометричної освіти, то методичні рекомендації деталізують механізми їх досягнення в реальному освітньому процесі. Згідно з актуальними інструктивно-методичними рекомендаціями, викладання математики має базуватися на принципах діяльнісного підходу, що передбачає активну когнітивну залученість учня на кожному етапі пізнання. У контексті автоматизації та розробки інтелектуальних навчальних систем (ІНС) критично важливим є аналіз того, як саме учні взаємодіють із навчальним матеріалом та за якими критеріями оцінюються їхні знання.

1.2.1. Організаційні форми та діяльнісний підхід у вивченні геометрії

Сучасна методика викладання геометрії в 7–9 класах вимагає відходу від пасивного трансляційного навчання. Натомість акцент робиться на створенні умов, за яких учень самостійно «відкриває» геометричні закономірності.

Діяльнісний підхід у геометрії реалізується через такі етапи:

1. Емпіричне дослідження та висунення гіпотез: Учень аналізує набір геометричних фігур, виявляє інваріантні властивості та формулює припущення. На цьому етапі активно використовуються системи динамічної геометрії для візуалізації.
2. Формалізація задачі: Переклад візуальної гіпотези на строгу математичну мову.
3. Дедуктивне доведення: Побудова логічного ланцюжка міркувань, що спирається виключно на аксіоми та раніше доведені теореми.

4. Рефлексія та верифікація: Перевірка знайденого розв'язку на несуперечливість та повноту.

Організація такого процесу потребує постійного зворотного зв'язку. В умовах традиційного класно-урочного навчання викладач фізично не здатен забезпечити миттєву перевірку гіпотез та логічних кроків кожного учня індивідуально, що зумовлює гостру потребу в програмних засобах підтримки.

1.2.2. Класифікація та діагностична роль перевірочних завдань

Для проектування архітектури рекомендаційної системи необхідно чітко розмежовувати типи завдань. Кожен тип перевіряє певний рівень засвоєння матеріалу і відіграє специфічну роль у підготовці до складних доведень. Згідно з методичними вимогами, завдання класифікуються за формою очікуваної відповіді:

- Завдання з вибором однієї або кількох правильних відповідей (тестові): Спрямовані на швидку перевірку базового фактологічного матеріалу: знання означень, розпізнавання фігур на кресленнях, пам'ять на формулювання аксіом. В архітектурі навчальної системи такі завдання відіграють критичну роль первинного діагностичного фільтра. Вони дозволяють системі миттєво визначити, чи володіє учень базовим понятійним апаратом, необхідним для подальшої роботи. Комп'ютерна перевірка таких завдань є тривіальною.

- Завдання на встановлення відповідності (логічні пари): Перевіряють рівень розуміння структурних зв'язків та класифікацій (наприклад, співвіднесення виду трикутника з його властивостями або графіка з функцією). Цей тип завдань допомагає закріпити реляційні знання учня і слугує перехідним етапом від простого запам'ятовування до аналізу.

- Завдання з короткою відповіддю: Передбачають виконання конкретних обчислювальних процедур (знаходження довжини відрізка, градусної міри кута). Вони перевіряють процедурні знання та навички застосування формул. Для рекомендаційної системи цей етап є маркером того, що учень здатний виконувати елементарні математичні перетворення самостійно.

- Завдання з розгорнутою відповіддю (повне розв'язання та доведення): Є вершиною геометричної підготовки. Якщо попередні три типи завдань готують фундамент (учень знає терміни, бачить зв'язки і вміє рахувати), то цей етап вимагає синтезу — побудови несуперечливого логічного тексту. Саме тут виникає найскладніша алгоритмічна проблема: одне й те саме твердження можна довести багатьма різними, але однаково правильними шляхами. Традиційні системи оцінювання (LMS) тут виявляються неефективними, оскільки не здатні відстежити семантичну коректність довільного логічного виводу, що робить необхідним залучення доводжувачів теорем.

1.2.3. Критерії оцінювання логічних кроків та формувальне оцінювання

Сучасна освітня парадигма зміщує акцент із класичного підсумкового контролю на формувальне оцінювання [4]. Це безперервний процес, що супроводжує навчальну діяльність і спрямований на своєчасне виявлення першопричин когнітивних утруднень учня, а не лише на констатацію кінцевого результату розв'язку.

Під час перевірки розгорнутих геометричних доведень викладач — а в контексті цифровізації освіти і проектована інтелектуальна навчальна система — керується чіткою системою критеріїв:

1. Повнота обґрунтування: Здійснюється перевірка наявності всіх необхідних ланок у ланцюжку міркувань. Учень не повинен допускати логічних неточностей, коли певний неочевидний геометричний факт приймається як істинний без прямого посилання на відповідну аксіому чи раніше доведену теорему.

2. Несуперечливість (логічна валідність): Аналізується правильність застосування правил логічного висновку. Цей критерій гарантує відсутність хибних наслідків, отриманих із правильних засновків.

3. Коректність застосування термінологічного апарату: Оцінюється точність оперування математичною символікою та розрізнення фундаментальних понять.

4. Раціональність міркувань: Визначається здатність учня знаходити лаконічний та найбільш прямий шлях до доведення, уникаючи надлишкових додаткових побудов, циклічних аргументів або використання складних теорем там, де достатньо базових означень.

Для дієвої реалізації принципів формувального оцінювання комп'ютерна система повинна не просто фіксувати факт наявності помилки, а вміти локалізувати її з точністю до конкретного логічного переходу. Наприклад, система має формувати деталізований зворотний зв'язок: *«На кроці 3 неправомірно застосовано ознаку паралельності прямих, оскільки попередньо не доведено рівність внутрішніх різносторонніх кутів»*.

Забезпечення такого глибокого рівня діагностики є неможливим у рамках традиційних тестових платформ. Воно вимагає трансляції природномовної аргументації учня у строгу формальну систему, де кожна логічна залежність та предикат математично точно відстежуються ядром верифікації.

1.3. Формалізація математичних міркувань та комп'ютерне подання геометрії

Перехід від традиційного паперового формату оцінювання геометричних задач до їх автоматизованої комп'ютерної верифікації вимагає створення суворої математичної моделі предметної області. На відміну від шкільної алгебри, де перетворення виразів здебільшого підпорядковуються чітким синтаксичним правилам і легко піддаються алгоритмізації, геометрія має значну візуальну та інтуїтивну складову. Це створює низку фундаментальних науково-технічних викликів при проектуванні інтелектуальних навчальних систем.

1.3.1. Аксіоматичні базиси та проблема «інтуїтивної зрозумілості»

Традиційний шкільний курс геометрії базується на адаптованій системі аксіом, яка свідомо залишає певні просторові відношення на рівні інтуїтивного розуміння. Учні спираються на креслення як на невід'ємну частину доведення. Наприклад, твердження про те, що промінь проходить між сторонами кута, або що точка лежить на відрізку між двома іншими точками, зазвичай приймається як очевидний факт, який «видно з рисунка», і не вимагає окремого логічного виведення.

Для комп'ютерної системи формальної верифікації поняття «очевидності» не існує. Усі відношення між об'єктами мають бути описані строгою мовою логіки предикатів. Для машинного подання геометрії розглядають суворіші аксіоматичні системи:

- Система Гільберта [5]: позбавлена логічних прогалин класичних «Начал» Евкліда, містить строгі аксіоми порядку та неперервності. Проте вона є занадто громіздкою для безпосереднього використання школярами.

- Система Тарського [6]: формулюється виключно в рамках логіки першого порядку і базується лише на поняттях «лежати між» та «конгруентність». Хоча вона чудово підходить для класичного автоматичного доведення теорем машиною, використання лише предикатів першого порядку є недостатнім для повноцінної формалізації шкільної геометрії. Зокрема, аксіома неперервності вимагає квантифікації не лише за окремими точками, а й за множинами точок. Така абстракція виходить за межі логіки першого порядку і вимагає предикатів вищого порядку.

- Векторно-метричні системи (наприклад, система Вейля [7]): добре піддаються комп'ютерній обробці завдяки зведенню геометричних відношень до алгебраїчних рівнянь, однак такий підхід нівелює саму суть синтетичної геометрії, яку вивчають у 7–9 класах, і не дозволяє перевіряти саме класичні логічні міркування.

Отже, створення повноцінної навчальної системи вимагає інструментарію, здатного оперувати логікою вищих порядків. Це робить неможливим використання простих розв’язувачів першого порядку та обґрунтовує необхідність застосування сучасних систем формальної верифікації, таких як фреймворк Lean, чий теоретичний фундамент природно підтримує предикати вищих порядків та роботу з множинами.

1.3.2. Семантичний розрив між природною мовою та формальним виводом

Друга фундаментальна проблема полягає у синтаксичній та семантичній свободі, з якою учні формулюють свої думки. У шкільній практиці доведення записуються природною мовою з вкрапленням математичних символів. Учень може написати: *«Оскільки трикутники рівні за двома сторонами і кутом між ними, то їхні відповідні кути теж рівні»*.

Для викладача цей запис є абсолютно зрозумілим. Проте для комп'ютерної системи він містить низку невизначеностей:

1. Які саме два трикутники розглядаються у поточному контексті?
2. Які конкретно пари сторін є рівними і на якому попередньому кроці це було доведено?
3. Про який саме кут йдеться?

Цей феномен у комп'ютерних науках визначається як семантичний розрив [8]. Шкільне доведення — це здебільшого стислий конспект, у якому пропущено кроки, що здаються автору тривіальними. Натомість система формальної верифікації вимагає побудови повного дерева виводу, де кожен вузол є результатом застосування конкретного правила логіки до попередньо підтверджених фактів. Програмне рішення має вміти реконструювати ці «пропущені» тривіальні кроки, щоб не перевантажувати учня надлишковим формалізмом, зберігаючи при цьому строгість перевірки.

1.3.3. Виклики верифікації варіативних доведень

При проектуванні архітектури навчального додатка необхідно враховувати явище варіативності доведень. Правильний розв'язок геометричної задачі рідко буває лінійним і єдиним. Одну й ту саму теорему можна довести через рівність трикутників, через ознаки паралельних прямих або використовуючи центральну симетрію.

Класичні системи управління навчанням, які орієнтуються на порівняння тексту з жорстко заданими еталонами або регулярними виразами, не здатні впоратися з такою варіативністю. Якщо учень обирає нестандартний, але логічно бездоганний шлях, шаблонна перевірка класифікує його розв'язок як

хибний. Крім того, виникає проблема відстеження залежностей. Якщо учень робить помилку на певному етапі, система має проаналізувати орієнтований ациклічний граф його міркувань, щоб зрозуміти, які подальші висновки спираються на цю хибну передумову.

Неможливість подолати ці виклики за допомогою простих алгоритмів парсингу тексту або тестування зумовлює необхідність пошуку спеціалізованих архітектурних рішень. Перед тим як перейти до обґрунтування вибору інструментарію, доцільно розглянути, як із цими викликами справляються існуючі програмні засоби.

1.4. Аналіз існуючих програмних методів та систем інтелектуальної підтримки навчання

Проектування інтелектуальної рекомендаційної системи вимагає критичного аналізу наявних на ринку програмних рішень. Це дозволяє виявити їхні архітектурні та функціональні обмеження у контексті описаної вище задачі перевірки розгорнутих математичних міркувань. Існуючі цифрові інструменти можна класифікувати за базовим принципом дії на три основні категорії: системи динамічної геометрії, традиційні платформи тестування та автоматичні доводжувачі теорем.

1.4.1. Системи динамічної геометрії

До цього класу належать такі популярні програмні засоби, як GeoGebra, Desmos та Cabri Geometry [9]. Їхнім основним завданням є забезпечення інтерактивної візуалізації математичних об'єктів та збереження геометричних інваріантів під час маніпуляцій користувача.

- Принцип перевірки: DGS здебільшого використовують чисельні або координатно-алгебраїчні методи. Наприклад, для підтвердження

паралельності двох прямих система обчислює їхні кутові коефіцієнти в прихованій декартовій системі координат.

- Обмеження: Хоча такі системи є потужним інструментом для емпіричного дослідження та висунення гіпотез, вони принципово не здатні здійснювати верифікацію логічного виводу. DGS може підтвердити істинність кінцевого факту, але не може оцінити, чи є послідовність міркувань учня логічно несуперечливим дедуктивним ланцюжком.

1.4.2. Традиційні системи управління навчанням

Платформи на зразок Moodle, Canvas або вбудовані модулі тестування в Google Classroom є стандартом для організації дистанційного навчання.

- Принцип перевірки: Базується на співставленні введених користувачем даних із наперед визначеними еталонами або регулярними виразами.

- Обмеження: Дані системи демонструють високу ефективність при перевірці фактологічних знань, однак виявляються нерелевантними для задач із розгорнутою відповіддю. Вони не підтримують варіативності розв'язків: якщо учень обирає нестандартний, але математично правильний шлях доведення геометричної теореми, система, що спирається на жорсткі шаблони, ідентифікує його як помилковий. Крім того, LMS не мають вбудованого математичного апарату для відстеження семантичних залежностей між кроками міркувань.

1.4.3. Системи автоматичного доведення теорем

Альтернативним підходом є використання спеціалізованих геометричних солверів, що базуються на методах алгебраїчної геометрії.

- Принцип перевірки: Геометричні предикати транслюються у систему поліноміальних рівнянь, після чого алгоритм автоматично шукає розв'язок.

- Обмеження: Згенероване такими алгоритмами доведення є машинним алгебраїчним виводом, який абсолютно відірваний від синтетичної шкільної геометрії. Він є нечитабельним для учня 7–9 класів і не може бути використаний як зворотний зв'язок у рекомендаційній системі, оскільки не вказує на конкретну логічну помилку в людських міркуваннях, а лише видає власний машинний результат.

Як свідчить аналіз, жодна з класичних систем не здатна повноцінно вирішити проблему семантичного розриву та варіативності шкільних доведень. Це зумовлює потребу у використанні інтерактивних доводжувачів теорем, серед яких найдоцільнішим є середовище Lean.

1.5. Фреймворк Lean як середовище формальної верифікації

З огляду на виявлені недоліки існуючих рішень, розробка системи, здатної адекватно оцінювати розгорнуті геометричні доведення та надавати персоналізовані рекомендації, потребує залучення інтерактивних доводжувачів теорем. У межах даного дослідження як фундаментальне ядро обрано фреймворк Lean (зокрема його сучасну ітерацію — Lean 4 [10]), який унікальним чином поєднує в собі властивості суто функціональної мови програмування загального призначення та потужного ядра формальної верифікації.

1.5.1. Еволюція систем верифікації та парадигма Lean 4

Історично системи автоматизованого доведення поділялися на дві категорії: автоматичні (наприклад, Z3 [11]) та інтерактивні (Coq [12], Isabelle/HOL [13],

Agda). Автоматичні системи намагаються знайти розв'язок самостійно, перебираючи простори станів, але швидко стикаються з проблемою комбінаторного вибуху при роботі зі складними геометричними абстракціями. Інтерактивні системи залишають побудову логічного шляху людині або зовнішній програмі, виконуючи роль суворого перевіряючого.

Lean 4 представляє нову парадигму в цій галузі. На відміну від попередніх версій та багатьох аналогів, Lean 4 є повноцінною мовою програмування, що компілюється у машинний код, через проміжне представлення мовою C. Це дозволяє не лише доводити теореми, а й розробляти повноцінні програмні додатки безпосередньо в середовищі Lean, уникаючи складних і ненадійних мостів між мовою верифікації та мовою виконання.

1.5.2. Теоретико-математичний фундамент: ізоморфізм Каррі-Говарда

Фундаментом архітектури Lean є числення індуктивних конструкцій із залежними типами [14]. Математичною основою роботи фреймворку є глибокий зв'язок між комп'ютерними програмами та математичними доведеннями, відомий як ізоморфізм Каррі-Говарда [15]. Відповідно до цього принципу:

- Математичне твердження розглядається як тип.
- Доведення цього твердження розглядається як терм – конкретний об’єкт, що є мешканцем цього типу.

Таким чином, перевірка правильності геометричного доведення учня зводиться до задачі перевірки типів під час компіляції. Якщо учень генерує послідовність кроків, яка успішно компілюється в терм правильного типу без помилок типування, це математично гарантує, що його доведення є логічно правильним. Залежні типи дозволяють кодувати складні геометричні предикати.

1.5.3. Архітектура компілятора та принцип мінімального довіреного ядра

Для того щоб інтелектуальна навчальна система була надійною, вона не повинна припускатися помилок при оцінюванні робіт. Lean гарантує таку надійність завдяки архітектурі LCF [16] та принципу «мінімального довіреного ядра». Процес обробки математичного тексту у Lean проходить кілька етапів:

1. Синтаксичний аналіз: Перетворення вхідного тексту на абстрактне синтаксичне дерево.

2. Елаборація: Найскладніший етап, на якому високорівневі конструкції, тактики та неявні аргументи розгортаються у фундаментальні терми мови. Елаборатор намагається самостійно вивести типи, які учень міг пропустити, що критично важливо для шкільної геометрії.

3. Верифікація ядром: Після того як усі абстракції розгорнуто, отриманий низькорівневий терм передається до ядра. Ядро Lean — це компактний і ретельно перевірений ізольований модуль коду. Його єдине завдання — перевірити коректність типів отриманого терму згідно з базовими правилами CIC.

Завдяки такій архітектурі, навіть якщо в алгоритмах рекомендаційної системи або в механізмах парсингу відповідей учня є баги, система ніколи не визнає хибне доведення правильним. Ядро просто відхилить згенерований терм на фінальному етапі.

1.5.4. Метапрограмування та автоматизація міркувань

У шкільній геометрії існує велика кількість тривіальних тверджень, детальний розпис яких перевантажує процес навчання. Прикладом цього є розкриття дужок у рівняннях, комутативність додавання кутів, рефлексивність рівності

відрізків. Для вирішення цієї проблеми Lean 4 надає потужний апарат метапрограмування — систему тактик.

Тактики — це програми, які виконуються під час компіляції і маніпулюють станом доведення. Вони генерують низькорівневі терми доведення на основі високорівневих вказівок. Унікальність Lean 4 полягає в тому, що його тактики пишуться на самій мові Lean, що відкриває безпрецедентні можливості для створення кастомних інструментів. При розробці навчальної системи для школярів механізм тактик дозволяє запрограмувати автоматичне закриття «очевидних» кроків. Наприклад, тактика *ring* автоматично доводить алгебраїчні тотожності, а тактика *linarith* розв'язує системи лінійних нерівностей. Це дозволяє учневі фокусуватися на суто геометричних ідеях, залишаючи рутинні обчислення фреймворку.

1.5.5. Екосистема Mathlib та специфіка формалізації шкільної геометрії

Цінність Lean як інструмента верифікації підкріплюється масштабною бібліотекою *mathlib* [17], яка містить тисячі формалізованих математичних фактів. У контексті геометрії *mathlib* пропонує глибоко опрацьовані простори, що охоплюють усю необхідну базу для евклідової геометрії.

Проте інтеграція *mathlib* в архітектуру системи для середньої школи становить окреме інженерне завдання. Геометрія в *mathlib* побудована на високих рівнях абстракції, тоді як шкільна програма базується на синтетичній геометрії. Тому для забезпечення коректної роботи рекомендаційної системи необхідно розробити спеціалізовані інтерфейсні шари та визначення. Ці програмні обгортки повинні транслювати абстрактні теореми з *mathlib* у терми, зрозумілі шкільному курсу, зберігаючи при цьому строгість формальної верифікації Lean.

Використання фреймворку Lean 4 дозволяє подолати технологічний бар'єр між формальною комп'ютерною логікою та природною варіативністю учнівських міркувань, створюючи надійну основу для проектування архітектури навчальної рекомендаційної системи.

РОЗДІЛ 2. АРХІТЕКТУРНА РЕАЛІЗАЦІЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ MVP

2.1. Архітектурна модель та технологічний стек системи

Проєктування архітектурної моделі дослідницького прототипу системи автоматизованої верифікації геометричних міркувань підпорядковане концепції нейро-символічної інтеграції. Для забезпечення гнучкості дослідницького прототипу та мінімізації накладних витрат на інфраструктуру обрано тришарову клієнт-серверну архітектуру.

2.1.1. Концептуальна декомпозиція та пайплайни обробки даних

Логічна структура системи розділена на три класичні рівні, проте на рівні серверного застосунку впроваджено чітке розділення відповідальності через два паралельні програмні пайплайни (Рис. 2.1):

1. Presentation Tier: Реалізований у вигляді тонкого клієнта на базі Vanilla JS. Забезпечує асинхронне введення розв’язків та візуалізацію фідбеку.
2. Application Tier: Функціонує як оркестратор на базі мови Python 3.9+. Сервер керує багатоетапним процесом трансформації даних через такі внутрішні компоненти:
    1. *Нейро-символічний пайплайн:* Включає OpenAI Client Wrapper (для абстракції запитів до LLM), Structured Parser (для валідації та очищення JSON-відповідей моделі) та Lean Generator (для динамічного синтезу вихідного коду мовою Lean 4 на основі екстрагованих кроків).
    2. *Пайплайн конфігурації:* Включає Task Catalog та Lean API Registry для відповідності між обраною задачею та валідними методами ядра Lean 4, а

також Config Loader (In-memory кеш для системних промптів та метаданих).

3. Data & Verification Tier: Об’єднує ядро формальної верифікації Lean 4, зовнішнє API моделі Google Gemini 2.5 Flash, сховище PostgreSQL та оптимізаційний кеш SQLite.

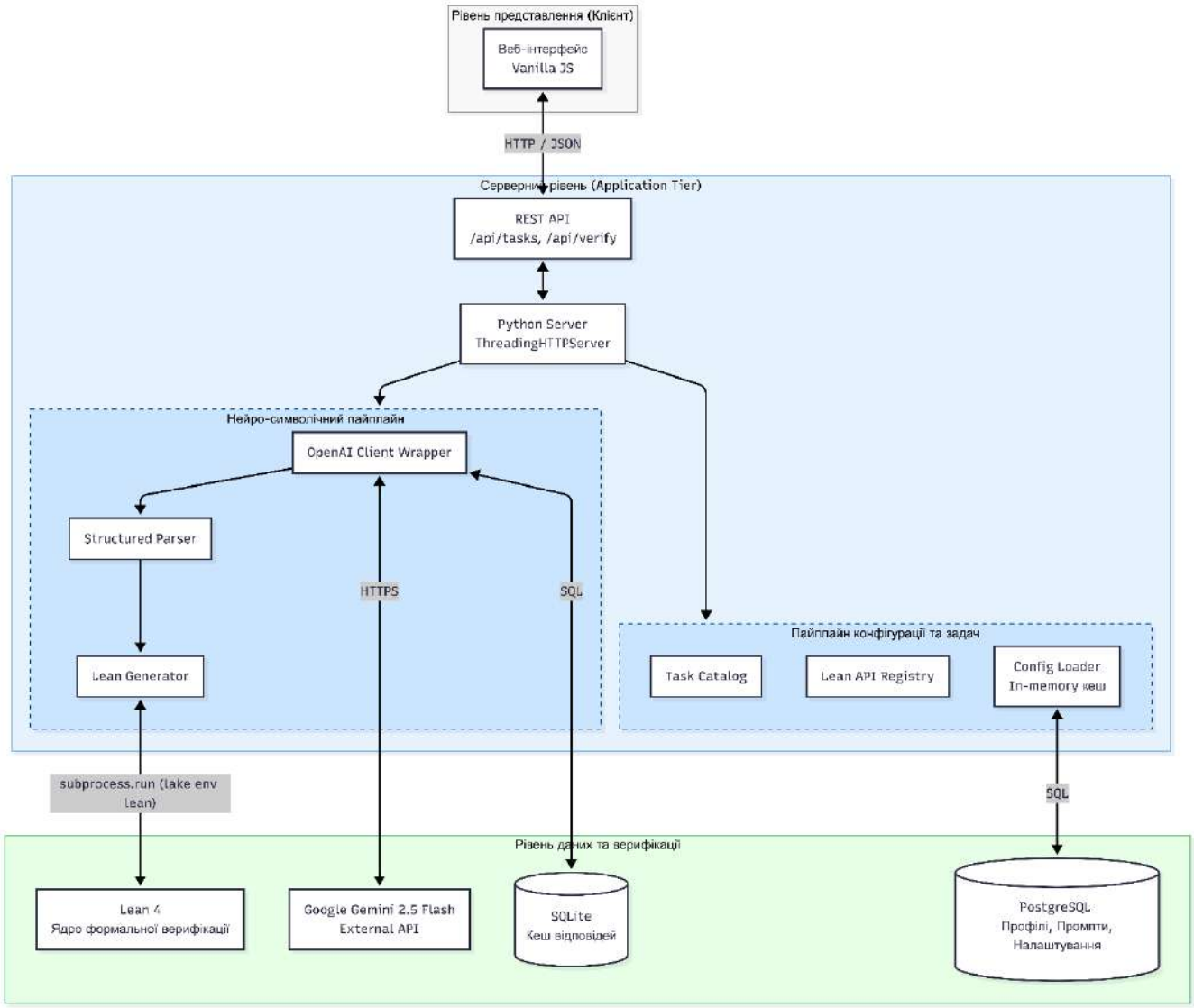


Рис. 2.1. Архітектура компонентів та внутрішніх пайплайнів обробки даних.

2.1.2. Обґрунтування вибору серверного стека

На відміну від промислових рішень, що зазвичай базуються на ASGI-фреймворках (FastAPI, Starlette), для даного дослідницького прототипу було обрано стандартну бібліотеку Python — ThreadingHTTPServer у поєднанні з BaseHTTPRequestHandler. Таке рішення є свідомим компромісом на користь простоти ізоляції процесів та зумовлене такими інженерними міркуваннями:

1. Мінімізація зовнішніх залежностей: Відсутність сторонніх фреймворків гарантує високу переносність системи та спрощує відтворення середовища.
2. Контроль життєвого циклу запиту: Модель «один потік на один HTTP-запит» дозволяє легко контролювати ресурси ОС при запуску важких зовнішніх процесів.
3. Інтеграція зовнішніх процесів: Взаємодія з ядром Lean 4 реалізована через запуск дочірніх процесів за допомогою subprocess.run. Сервер генерує тимчасовий .lean файл, після чого ініціює виклик у форматі lake env lean. Зчитування результатів відбувається безпосередньо з потоку виводу з жорстким програмним watchdog-таймаутом.

2.1.3. Модель нейро-символічної інтеграції та оптимізація

Для інтерпретації україномовних текстів розв’язань інтегровано модель Google Gemini 2.5 Flash через OpenAI-сумісний ендпоінт. Оскільки звернення до LLM є ресурсомісткою операцією, в архітектуру впроваджено рівень кешування на базі вбудованої СКБД SQLite.

Таблиця llm_responses зберігає хешовані значення промптів та відповідні відповіді моделі з часом життя 24 години. Це забезпечує:

1. Зниження латентності: Повторні перевірки ідентичних розв’язків виконуються миттєво без мережевих запитів.

2. Економічну ефективність: hit rate кешу на рівні 50–70% суттєво зменшує навантаження на зовнішнє API під час тестування.

2.1.4. Специфікація програмних інтерфейсів

Обмін даними між фронтендом та бекендом здійснюється через RESTful API.

Поле JSON	Тип	Опис
id	String	Унікальний ідентифікатор задачі
taskType	String	Категорія задачі для вибору алгоритму верифікації
grade	Integer	Цільовий клас навчання
statement	String	Повна умова задачі з підставленими параметрами
requiredSteps	Array	Перелік логічних кроків, необхідних для повного розв'язання
params	Object	Рандомізовані чисельні значення (координати, довжини)

Таблиця 2.1. Специфікація інтерфейсу отримання задач

Поле JSON (Request)	Тип	Опис
taskId	String	Ідентифікатор задачі (підтримує regex-розпізнавання варіантів)
solutionText	String	Повний текст відповіді учня
params	Object	Поточні параметри варіанту задачі для Lean-верифікатора
grade	String	Категоріальний вердикт
completionPercent	Integer	Розрахований відсоток готовності
missingSteps	Array	Перелік кроків, які не були ідентифіковані в тексті
semanticErrors	Array	Список логічних або структурних помилок
llmLean	Object	Результати виконання формального коду та трейси #eval команд

Таблиця 2.2. Специфікація інтерфейсу верифікації

2.1.5. Архітектурні артефакти: підсистема збереження стану та профілювання

Хоча основний фокус розробки в даному проєкті було свідомо зміщено на науково-технологічну складову — ядро формальної верифікації, архітектура MVP проєктувалася з урахуванням потенційного масштабування до рівня повноцінної системи управління навчанням.

Для демонстрації такої можливості було розроблено базову реляційну схему на основі PostgreSQL v15.

На етапі практичної експлуатації прототипу основна взаємодія з даними зосереджена навколо:

1. Таблиці tasks: яка слугує ключовим репозиторієм умов, параметрів, рубрик оцінювання та допустимих посилань на теореми.
2. Бази SQLite: яка забезпечує швидке кешування відповідей мовної моделі.

2.1.6. Конкурентність, серіалізація та стратегія розгортання

Специфіка роботи інтерактивних доводжувачів теорем полягає у високому споживанні системних ресурсів. Для переведення системи у продуктивне середовище задекларовано використання такого стека:

1. Nginx: Виконує роль Reverse Proxy, забезпечує термінацію SSL-з’єднань та статичну маршрутизацію.
2. Supervisor: Керує життєвим циклом Python-процесів, забезпечуючи автоматичний перезапуск сервера у разі критичних збоїв.

Окрему увагу приділено асиметричному керуванню конкурентністю. На рівні клієнтської частини впроваджено механізм локальної FIFO-черги типу single-flight. Механізм відстежує стан виконання, і у разі активної перевірки блокує відправку нових HTTP-запитів, ставлячи їх у чергу. Це повністю запобігає лавинним запитам

та перевантаженню сервера при випадкових багатократних натисканнях кнопки «Перевірити».

Такий підхід до архітектурного проєктування дозволяє створити стійкий фундамент, де кожен елемент технологічного стека відповідає за конкретний рівень абстракції, забезпечуючи при цьому високу математичну надійність фінального вердикту.

2.2. Програмна реалізація ядра формальної верифікації мовою Lean 4

Основою аналітичної спроможності системи є власна бібліотека геометричних операцій та предикатів, реалізована мовою Lean 4. У межах архітектури вона виконує роль довіреного обчислювального ядра на етапі формальної перевірки. Важливо зазначити, що система загалом залежить від точності NLP-екстракції та LLM-генерації на попередніх етапах, тому ризик хибної інтерпретації вхідного тексту залишається. Проте саме Lean-бібліотека забезпечує математичну строгість на рівні обчислень: вона коректно і детерміновано оцінює подані їй формалізовані дані, унеможливлюючи хибні позитивні вердикти через математичні помилки.

2.2.1. Модульна структура геометричної бібліотеки

Для забезпечення розширюваності та зручності підтримки, геометричне ядро поділено на п’ять вузькоспеціалізованих модулів:

- Core.lean: Визначає фундаментальні типи даних (точки, прямі) та базову арифметику.
- Predicates.lean: Містить логічні відношення між об’єктами (належність, паралельність, перпендикулярність).
- Constructors.lean: Реалізує функції для обчислення нових об’єктів (точки перетину, середини відрізків).
- Metrics.lean: Відповідає за обчислення метричних характеристик (відстані, площі).

- Verifier.lean: Виконує роль простого фасаду, що містить функції верифікації для пакетів завдань курикулуму.

У рантаймі під час серверної перевірки Python-оркестратор діє гнучко, напряму імпортуючи необхідні модулі у згенеровані файли залежно від потреб конкретної задачі.

2.2.2. Базові абстракції та прагматичний компроміс типів числення

Важливим інженерним рішенням при проєктуванні геометричного ядра став прагматичний компроміс між абсолютною точністю та обчислювальною необхідністю. У системах формальної верифікації предикат рівності вимагає уникнення похибок округлення, притаманних типам з рухомою комою (Float).

Тому базові геометричні типи, а також структурні та логічні відношення визначено виключно над полем раціональних чисел ℚ, доступним у бібліотеці Mathlib (тип ℚ або Rat).

```
structure Point where
  x : ℚ
  y : ℚ

structure Line where
  a : ℚ
  b : ℚ
  c : ℚ
Інваріант: a і b не можуть бути одночасно нулями
```

Рис. 2.2. Визначення базових типів у Core.lean над полем раціональних чисел

Використання раціональної арифметики гарантує детермінованість фундаментальних фактів. Для оптимізації метричних обчислень функція квадрата відстані також реалізована суто в ℚ.

Однак, евклідова метрика шкільної геометрії вимагає нелінійних обчислень. Тому для розрахунку фактичних відстаней або периметрів у модулі Metrics.lean здійснюється перехід до типу Float. Це створює гібридну систему числення: структурні та логічні відношення, а також квадратичні метрики строго перевіряються в ℚ, а лінійні евклідові властивості обчислюються у Float як функціональна необхідність.

2.2.3. Безпека типів у конструктивних операціях

Відмінною рисою розробленого ядра є використання системи залежних типів Lean 4 для забезпечення безпеки виконання часткових операцій. Типовим прикладом є знаходження точки перетину двох прямих: ця операція не має унікального розв'язку для паралельних прямих.

У Lean 4 ця проблема вирішується за допомогою монади `Option`.

```
def intersection (l1 l2 : Line) : Option Point :=
  let det := l1.a * l2.b - l2.a * l1.b
  if det == 0 then
    none -- Прямі паралельні, унікальної точки перетину не існує
  else
    some { x :=..., y :=... }
```

Рис 2.3. Сигнатура функції перетину з використанням Option (Constructors.lean)

Такий підхід змушує компілятор на рівні типів перевіряти коректність обробки відсутності розв'язку (наприклад, через паттерн-матчинг). Це надійно запобігає помилкам часу виконання, аналогічним NullPointerException. Хоча це не виключає ризику переривання процесу на рівні ОС через таймаут або вичерпання пам'яті OOM, логічне ядро залишається захищеним від структурних збоїв.

2.2.4. Предикати та доказова база

Логічні кроки, які формулює учень, транслюються системою у виклики відповідних функцій-предикатів. Модуль Predicates.lean перетворює геометричні абстракції на булеві результати.

Наприклад, перевірка перпендикулярності двох прямих зводиться до аналізу скалярного добутку їхніх нормальних векторів (Рис 2.4):

```
def is_perpendicular (l1 l2 : Line) : Bool :=
  l1.a * l2.a + l1.b * l2.b == 0
```

Рис. 2.4. Аналіз скалярного добутку нормальних векторів

Завдяки тому, що обчислення відбуваються в середовищі Lean під час фази елаборації, Python-серверу достатньо передати згенерований скрипт із командою #eval linesPerpendicular lineAB lineCD. Якщо Lean обчислює цей вираз як true, конкретний крок вважається успішно верифікованим на рівні формальної логіки.

2.2.5. Динамічна верифікація та оркестрація процесу

Взаємодія між Python-бекендом та ядром Lean реалізована за принципом динамічної кодогенерації (див. Рис. 2.5). Під час виклику API-ендпоінту скрипт lean_verifier.py формує тимчасовий .lean файл, у якому напряму імпортуються необхідні модулі бібліотеки, ініціалізується середовище з параметрами поточного варіанту задачі та генеруються команди #eval для логічних кроків учня. генеруються команди #eval для логічних кроків учня.

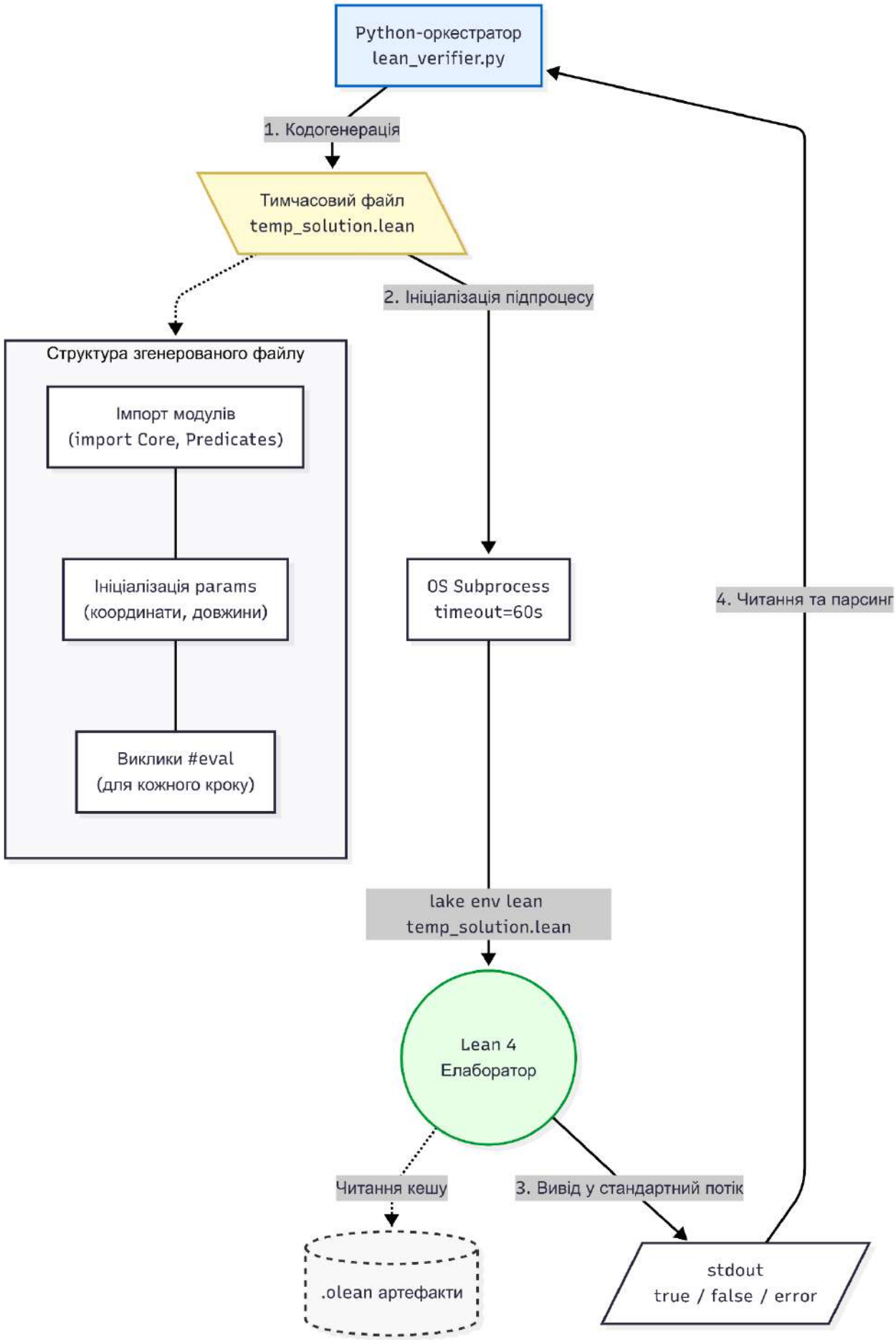

Python-оркестратор
lean_verifier.py
1. Кодогенерація
Тимчасовий файл
temp_solution.lean
2. Ініціалізація підпроцесу
Структура згенерованого файлу
Імпорт модулів
(import Core, Predicates)
Ініціалізація params
(координати, довжини)
Виклики #eval
(для кожного кроку)
OS Subprocess
timeout=60s
4. Читання та парсинг
lake env lean
temp_solution.lean
Lean 4
Елаборатор
Читання кешу
3. Вивід у стандартний потік
.olean артефакти
stdout
true / false / error

Рис. 2.5. Пайплайн динамічної генерації та виконання Lean-скриптів.

Виконання відбувається в ізольованому дочірньому процесі через виклик оболонки пакетного менеджера Lake (lake env lean <ім'я_тимчасового_файлу>.lean). Цей патерн дозволяє перевіряти довільні геометричні конфігурації в реальному часі. Використання попередньо скомпільованої бази знань дозволяє системі здійснювати перевірку в межах загального синхронного циклу HTTP-запиту. Для захисту бекенду від зависань та надмірного споживання ресурсів, на процес Lean-елаборації накладається жорсткий системний таймаут у 60 секунд. Фінальний результат формується на основі парсингу стандартного виводу Lean-процесу оркестратором Python.

2.3. Алгоритми нейро-символічної взаємодії та обробки природної мови

Подолання семантичного розриву між неструктурованим текстом учня та детермінованим середовищем перевірки типів Lean 4 здійснюється за допомогою нейро-символічного підходу. У цій архітектурі велика мовна модель (Google Gemini 2.5 Flash) виступає в ролі гнучкого інтерфейсу розпізнавання намірів, тоді як Lean 4 виконує функцію строгого логічного верифікатора.

Для трансляції тексту учня у формальні терми Lean у системі реалізовано двовекторну алгоритмічну стратегію, що дозволяє балансувати між стабільністю роботи та гнучкістю розпізнавання.

2.3.1. Стратегія А: Екстракція фактів та шаблонізація (Метод Extractor)

Перший підхід базується на делегуванні LLM виключно задач синтаксичного та семантичного парсингу. Спеціалізований підхід на основі екстракції аналізує текст розв'язку та вилучає з нього структурований JSON-об'єкт, який містить виявлені факти, числові значення, координати та перелік застосованих теорем. Слід зазначити, що загальний структурний розбір у рушії оцінювання підтримується

ширшим компонентом structured_parser.py, який відповідає за валідацію різних етапів NLP-аналізу.

Отриманий JSON-об'єкт із фактами передається до Python-оркестратора, який ін'єктує ці значення у жорстко заданий шаблон коду мовою Lean 4.

- Переваги: Максимальна стабільність процесу компіляції (Lean-код майже ніколи не містить синтаксичних помилок, оскільки каркас написаний розробником).
- Недоліки: Обмежена гнучкість. Метод ефективний лише для алгоритмічних обчислювальних задач (наприклад, метод координат), але не підходить для задач на доведення, де учень може обрати непередбачуваний логічний шлях.

2.3.2. Стратегія B: Динамічна кодогенерація (Метод Generator)

Для обробки варіативних та творчих розв'язків розроблено алгоритм динамічної генерації коду, за який відповідає компонент lean_generator.py. У цьому сценарії мовна модель генерує повноцінний скрипт мовою Lean 4, оперуючи кастомним API розробленої геометричної бібліотеки.

Якість генерації забезпечується методами промпт-інжинірингу. Системний промпт містить:

- Специфікацію типів та предикатів: Визначення структур Point, Line, функцій intersection, distance тощо.
- Few-shot приклади: Зразки правильної трансляції типових українських речень (наприклад, *"Нехай O — середина AB"*) у відповідний код Lean (let O := midpoint A B).
- Обмеження: Прямі вказівки моделі не доводити теорему самостійно, а лише перекладати ті кроки, які фактично присутні у тексті учня.

2.3.3. Механізм самокорекції та управління ретраями

Оскільки сучасні LLM схильні до галюцинацій та синтаксичних неточностей (наприклад, використання неіснуючої функції або помилки у типах Option), прямий запуск згенерованого коду часто призводить до помилок елаборації Lean 4. Для забезпечення відмовостійкості системи розроблено алгоритм автоматичної самокорекції.

Якщо дочірній процес lake env lean завершується з помилкою, оркестратор перехоплює потік помилок stderr і повертає його назад у LLM із системним повідомленням: *"Твій попередній код викликав таку помилку компілятора. Виправ її та згенеруй код знову"*. Кількість таких спроб генерації не є жорстко зашитою константою, а динамічно керується параметром середовища, що за замовчуванням дозволяє до 5 ітерацій. Якщо після вичерпання ліміту код не компілюється, процес переривається, а в логіку оцінювання передається семантична помилка llm_lean_failed. Цей механізм суттєво підвищує загальну успішність формалізації складних міркувань.

2.3.4. Семантичний аналіз та архітектура рушія оцінювання

Успішна компіляція Lean-скрипта є необхідною, але не достатньою умовою для зарахування задачі. Учень міг написати беззмістовний текст або відмову *"Не знаю як розв'язувати"*, який екстрактор хибно переклав у синтаксично правильні базові кроки.

Для захисту від хибних позитивних вердиктів результати математичної перевірки узгоджуються із результатами семантичного NLP-аналізу в модулі оцінювання. Фінальний вердикт розраховується на основі агрегації двох масивів даних:

- Математичні висновки: Результати виконання команд #eval для кожного розпізнаного кроку.

- Аналіз рубрик: Співставлення розпізнаних кроків з еталонним списком requiredSteps із каталогу задач. Якщо учень пропустив важливий етап доведення, рушій фіксує кількість пропущених кроків та генерує критичну семантичну помилку critical_step_missing.

Алгоритм формування фінального бала залежить від обраного режиму верифікації. У строгому режимі оцінювання здебільшого діє правило: тільки за умови, що всі математичні перевірки Lean пройдено успішно, а семантичний аналізатор підтверджує відсутність пропущених критичних кроків, система генерує підсумковий бал 100%. У разі часткового збігу розраховується пропорційна оцінка, а при фатальних логічних розривах завдання відхиляється з оцінкою 0%. У навчальних режимах ця логіка може адаптуватися для надання більш гнучкого формувального зворотного зв'язку.

2.4. Каталог завдань та інтелектуальна генерація варіантів

Ефективність інтелектуальної навчальної системи визначається не лише якістю формальної верифікації розв'язків, а й здатністю забезпечувати унікальність контенту для запобігання академічній недоброчесності. База даних розробленого прототипу містить 39 задач шкільного курсу планіметрії, які класифіковані за 28 унікальними алгоритмічними типами. Для забезпечення їхньої багаторазової варіативності розроблено спеціалізовану підсистему генерації та управління параметрами.

2.4.1. Декларативна специфікація каталогу завдань

У робочому режимі дані зчитуються з реляційної бази даних PostgreSQL через спеціалізований компонент каталогу. Таке архітектурне рішення забезпечує гнучкість системи, дозволяючи додавати нові задачі без втручання у вихідний код бекенду.

Кожен об'єкт задачі в каталозі містить вичерпний набір метаданих, необхідних для повного циклу обробки:

- id та класифікатори (grade, theme, topic): Забезпечують ієрархічну навігацію в користувацькому інтерфейсі та маршрутизацію API-запитів.
- Умова: Текстовий шаблон задачі, що містить маркери для динамічної підстановки згенерованих числових значень або координат.
- Еталонні кроки: Масив обов'язкових логічних або обчислювальних етапів, який слугує рубрикою для семантичного аналізатора під час розрахунку відсотка виконання.
- Параметри верифікації: Словник математичних констант та змінних, що безпосередньо підставляється у згенерований Lean-код або перевірочний шаблон під час формування тимчасового .lean файлу.

2.4.2. Спеціалізовані математичні генератори алгоритмічної рандомізації

Проста генерація випадкових чисел є неприпустимою в комп'ютерній геометрії, оскільки сліпа рандомізація координат неминуче призводить до утворення вироджених фігур (наприклад, трикутників із нульовою площею) або неіснуючих конфігурацій (порушення нерівності трикутника).

Для вирішення цієї проблеми для 18 специфічних типів задач було розроблено алгоритми математично коректної рандомізації. Замість випадкової генерації значень, ці генератори оперують інваріантами:

- Задачі на метричні співвідношення: Наприклад, для задач на обчислення периметра прямокутного трикутника генератор використовує алгоритм побудови Піфагорових трійок. Це гарантує, що всі сторони згенерованого

трикутника матимуть цілочисельні значення, що відповідає дидактичним вимогам 7–9 класів.

- Задачі на перетин об'єктів: Для генерації параметрів прямих алгоритм попередньо перевіряє умову ненульового визначника матриці коефіцієнтів, математично виключаючи ймовірність генерації паралельних прямих для задач, де вимагається знайти точку їхнього перетину.

2.4.3. Універсальний Fallback-механізм та парсинг ідентифікаторів

Створення спеціалізованого генератора для кожного з 28 алгоритмічних типів задач є економічно та алгоритмічно недоцільним на етапі прототипування. Для забезпечення стійкості та стовідсоткового технічного покриття каталогу впроваджено універсальний Fallback-механізм. Цей компонент дозволяє системі генерувати валідні параметри для будь-якої нової задачі навіть за відсутності специфічного математичного алгоритму, використовуючи базові значення як еталон.

Важливою архітектурною деталлю є механізм відстеження згенерованих варіантів. Універсальна ідентифікація здійснюється на стороні сервера за допомогою регулярних виразів. Сервер парсить вхідний taskId через патерн, що дозволяє системі динамічно відокремлювати унікальний номер варіанту від базового ідентифікатора задачі. Такий підхід усуває необхідність жорсткого кодування тисяч можливих варіантів у базі даних, дозволяючи системі правильно розпізнавати динамічні ID та ініціалізувати відповідні перевірочні скрипти Lean 4 в оперативному режимі.

2.5. Користувацький інтерфейс та сервісні модулі

Для забезпечення взаємодії учня з ядром формальної верифікації розроблено веб-інтерфейс та супутні сервісні модулі. Головною вимогою до клієнтської частини була мінімізація навантаження на браузер користувача та забезпечення плавного

користувацького досвіду при очікуванні результатів роботи ресурсомістких бекенд-процесів.

2.5.1. Архітектура та режими роботи веб-інтерфейсу

Клієнтський додаток реалізовано без використання важких реактивних фреймворків, спираючись на стандарти Vanilla JS, HTML5 та CSS3. Таке рішення дозволило створити максимально "тонкого клієнта", який відповідає виключно за рендеринг DOM-елементів та серіалізацію даних.

Інтерфейс підтримує два основні сценарії використання, які адаптують рівень деталізації зворотного зв'язку:

1. Навчальний режим: Орієнтований на формувальне оцінювання. Система надає покроковий фідбек, вказує на конкретні пропущені етапи та виводить семантичні помилки. Це дозволяє учневі ітеративно виправляти розв'язок.
2. Контрольний режим: Імітує умови тестування або іспиту. Інтерфейс приховує проміжні підказки, а розширена аналітика, включно з трейсами Lean-компілятора, стає доступною лише після остаточного завершення сесії.

2.5.2. Асинхронна взаємодія та управління станом

Взаємодія веб-додатка із сервером відбувається повністю асинхронно за допомогою технології Fetch API, що дозволяє оновлювати параметри задач та виводити результати перевірки без перезавантаження сторінки.

Для захисту від втрати даних під час можливих перебоїв з інтернет-з'єднанням реалізовано механізм локального збереження стану сесії з використанням браузерного сховища localStorage. Важливо зазначити межі цієї відмовостійкості: локально зберігаються лише параметри інтерфейсу (обраний режим, активні фільтри, sessionId) та текстові чернетки розв'язків учня. Сама ж формальна перевірка міркувань вимагає обов'язкового онлайн-доступу до бекенду і не може виконуватися офлайн.

Окремим архітектурним викликом стала проблема контролю конкурентності на боці клієнта. Оскільки перевірка у Lean 4 може займати до 60 секунд, нетерплячі користувачі могли б генерувати лавину повторних запитів, що призвело б до перевантаження сервера. Для вирішення цієї проблеми на фронтенді у контрольному режимі впроваджено патерн FIFO-черги.

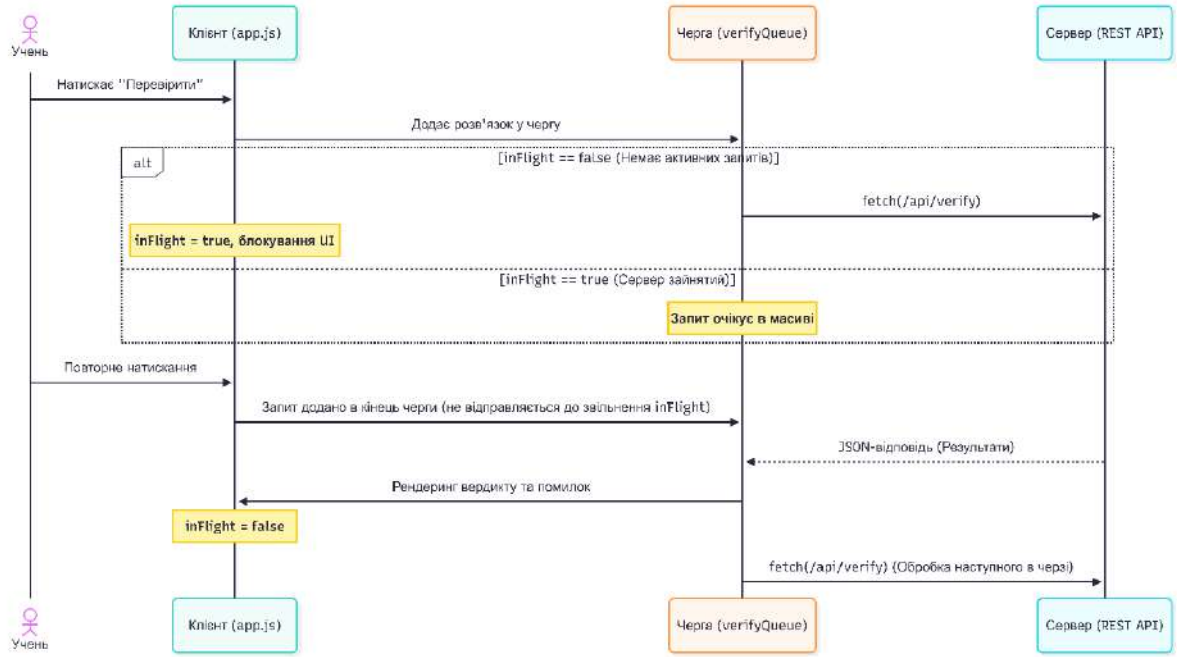


Рис. 2.6. Механізм клієнтської серіалізації запитів.

Як показано на Рис. 2.6, глобальна змінна inFlight відстежує стан мережевого запиту. Якщо сервер уже обробляє розв'язок, усі нові події відправляються в масив verifyQueue. Наступний запит ініціюється лише після отримання та рендерингу відповіді на попередній, що гарантує сувору серіалізацію навантаження. Слід зауважити, що даний механізм діє в межах однієї клієнтської сесії і не є глобальним серверним лімітером, делегуючи загальний контроль за конкурентністю серверній інфраструктурі.

2.5.3. Сервісний модуль кешування відповідей

Хоча більшість обчислень делеговано ядру Lean 4, етап семантичної трансляції тексту потребує звернення до зовнішнього API Google Gemini 2.5 Flash. Для

оптимізації роботи цього компонента на бекенді реалізовано незалежний сервісний модуль персистентного кешування.

Модуль використовує вбудовану СКБД SQLite як єдиний постійний рівень сховища пар "промпт-відповідь". Коли сервер формує новий запит до LLM, він генерує унікальний криптографічний хеш тексту та перевіряє його наявність у базі. Застосування алгоритмів кешування з часом життя запису 24 години дозволяє досягти таких показників, зафіксованих під час експериментального профілювання прототипу:

- Знизити час очікування відповіді на 80%.
- Скоротити фінансові витрати на використання комерційного API мовної моделі на 50–70% під час масового проходження однотипних завдань.
- Забезпечити стабільність системи у випадках тимчасової недоступності серверів Google.

Важливою архітектурною умовою є те, що кешування застосовується виключно до етапу NLP-екстракції та кодогенерації. Воно прискорює обробку повторних або синтаксично ідентичних запитів, але ніколи не замінює і не оминає фінальний етап строгої формальної верифікації згенерованого коду в середовищі Lean 4.

2.5.4. Сценарії взаємодії та візуалізація результатів верифікації

Розроблений клієнтський інтерфейс забезпечує динамічну візуалізацію результатів роботи нейро-символічного пайплайну. Робота із системою розпочинається з вибору задачі: інтерфейс генерує унікальний варіант умови з динамічними параметрами та надає учневі поле для введення покрокового математичного доведення (Рисунок 2.7).

Lean 4 Geometry Verifier

Клієнтський інтерфейс для перевірки формалізованих кроків доведення та отримання рекомендацій за темами 7-9 класів.

Навчальний режим | Контрольний режим

Задачі

Пошук або покроковий вибір: клас, розділ, тема.

Пошук розділу або теми

Каталог: **Задачі** / Клас: **9 клас** / Розділ: **Коло і трикутник** / Тема: **Описане коло трикутника**

Назад

Показано 1 з 1

ОПИСАНЕ КОЛО ТРИКУТНИКА

Побудова описаного кола прямокутного трикутника двома методами

9 клас | Коло і трикутник

Побудова описаного кола прямокутного трикутника двома методами

Новий варіант

Клас: 9 | Розділ: Коло і трикутник | Тема: Описане коло трикутника

Знайдіть центр O і радіус R описаного кола трикутника ABC: A(0,0), B(4,0), C(0,3). Обґрунтуйте побудову і перевірте, що відстані від центра до всіх вершин однакові.

Розв'язок

```
Наприклад:
Підставимо координати точки A(0,0) у рівняння прямої.
Отримаємо 0, тому точка A лежить на прямій AB.
```

Перевірити | Підставити шаблон

Рис. 2.7. Початковий стан інтерфейсу: згенерована умова задачі та поле введення розв'язку.

Після відправки тексту на перевірку, залежно від повноти та математичної коректності введеного учнем розв'язку, система генерує три основні сценарії зворотного зв'язку: повне зарахування, часткове зарахування та повне відхилення.

2.5.4.1 Сценарій 1: Повне та математично коректне доведення (100%)

У разі ідеального розв'язку (Рисунок 2.8) система успішно екстрагує всі необхідні факти. Модуль NLP розпізнає повний перелік еталонних кроків. Lean-ядро підтверджує логічну валідність усіх міркувань. Інтерфейс виводить зелений маркер вердикту «коректно (100%)» та повідомляє про відсутність пропущених кроків.

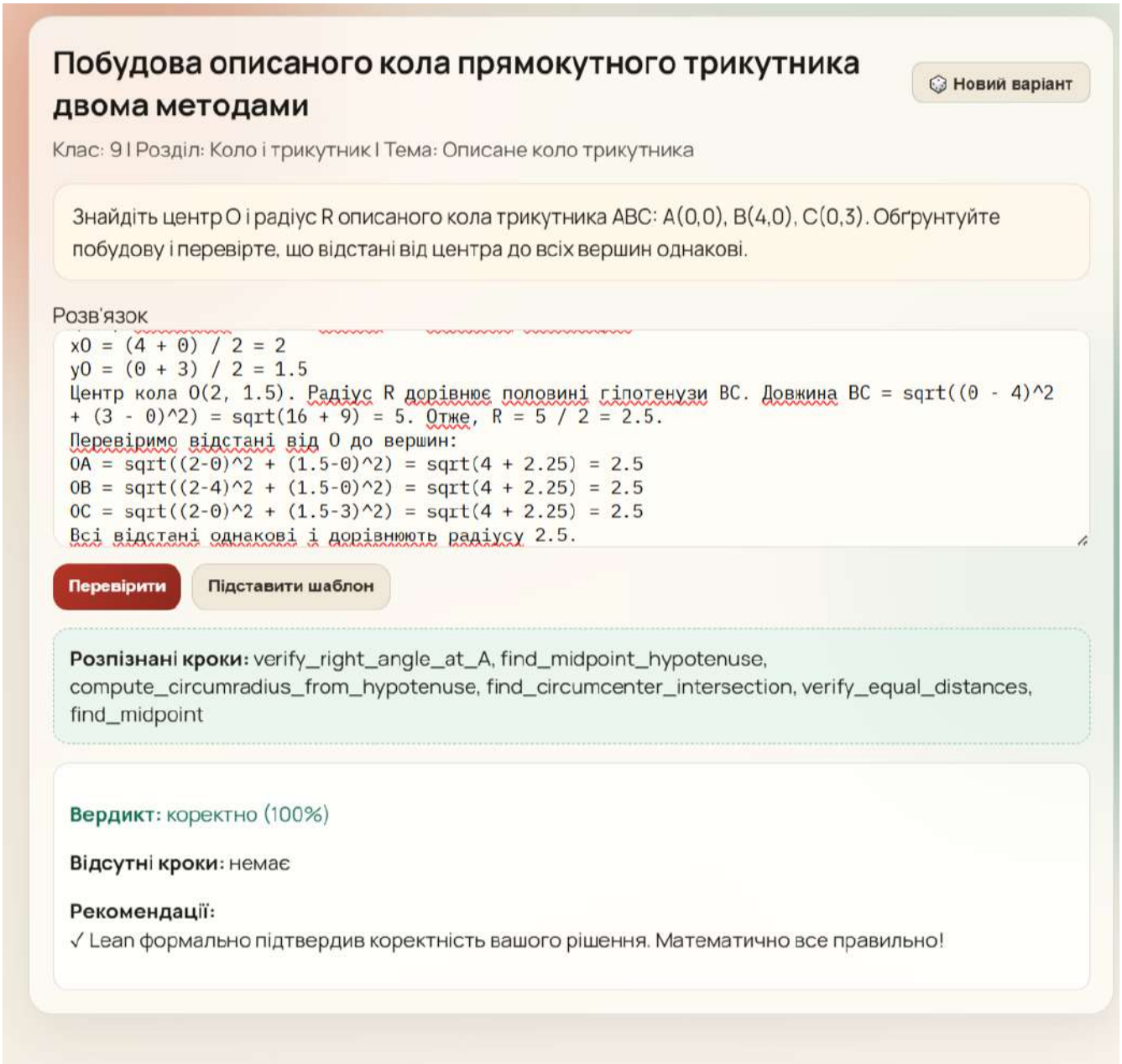


Рис. 2.8. Візуалізація успішного проходження верифікації з оцінкою 100%.

2.5.4.2 Сценарій 2: Часткове розв'язання з логічними або обчислювальними помилками

Найбільшу дидактичну цінність має сценарій обробки неповних розв'язків (Рисунок 2.9). Якщо учень правильно розпочав доведення (наприклад, знайшов координати центра кола), але припустився помилки на фінальному етапі порівняння відстаней, система не відхиляє роботу повністю. NLP-модуль фіксує частково розпізнані

кроки, однак рушій оцінювання реєструє семантичну помилку (наприклад, формальна перевірка Lean не пройдена). Інтерфейс виводить проміжний бал (у даному випадку 60%) та жовтий маркер вердикту «частково», вказуючи учневі на необхідність перевірити узгодженість тверджень.

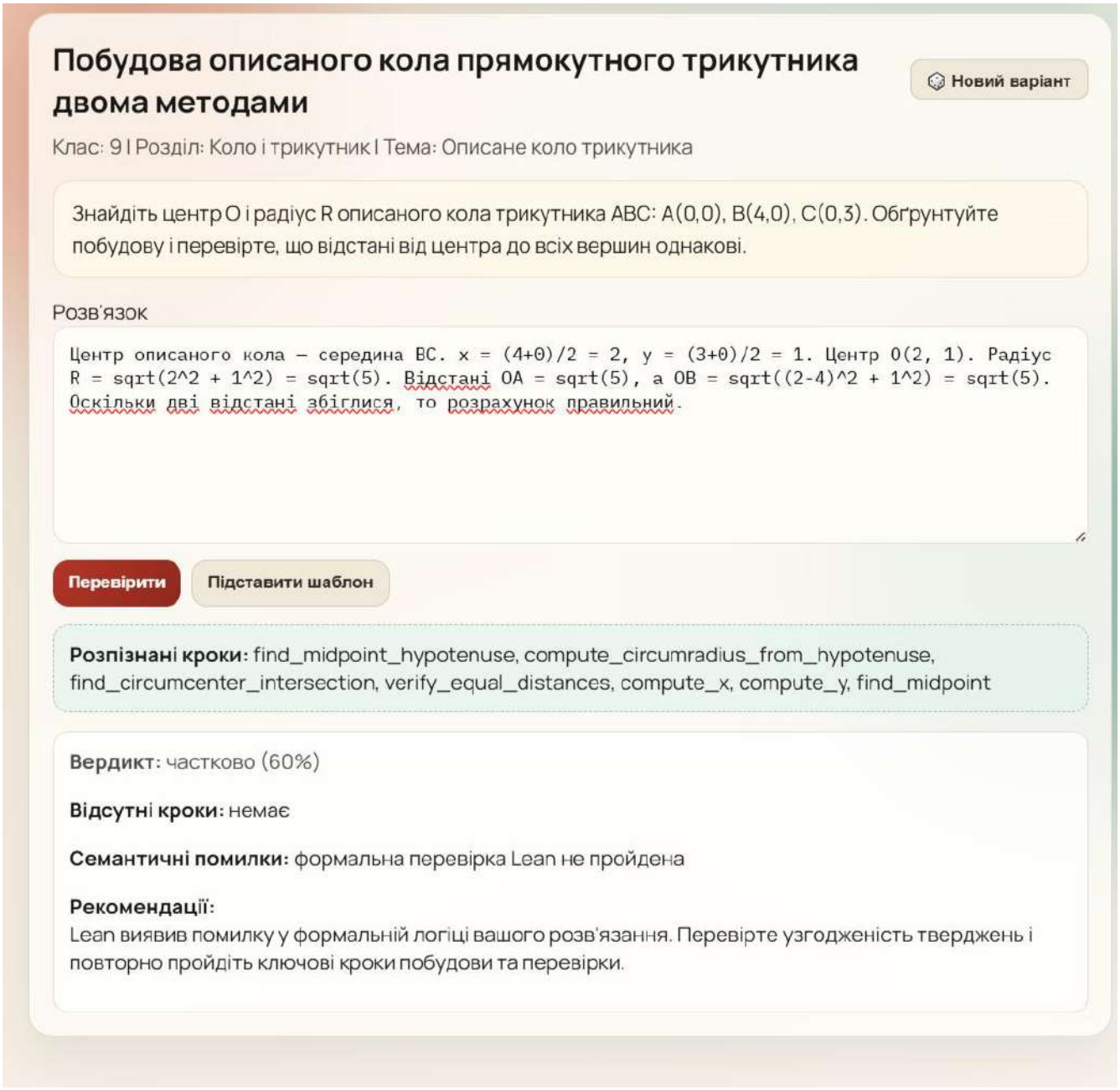


Рис. 2.9. Сценарій часткового зарахування (60%) при виявленні помилки у формальній логіці.

2.5.4.3 Сценарій 3: Відсутність логіки або некоректна відповідь (0%)

Для захисту від беззмістовних відповідей (наприклад, введення тексту *"Не знаю, як розв'язати цю задачу"*) передбачено сценарій жорсткого відхилення (Рисунок 2.10). Система коректно ідентифікує нерелевантний текст. Оскільки жоден із базових кроків не розпізнано, алгоритм генерує критичну семантичну помилку (пропущено критичний крок). Вердикт отримує червоний маркер «некоректно (0%)». Найважливішим елементом цього сценарію є блок зворотного зв'язку: система автоматично формує повний перелік відсутніх етапів та генерує рекомендацію щодо повторення конкретної теми з каталогу.

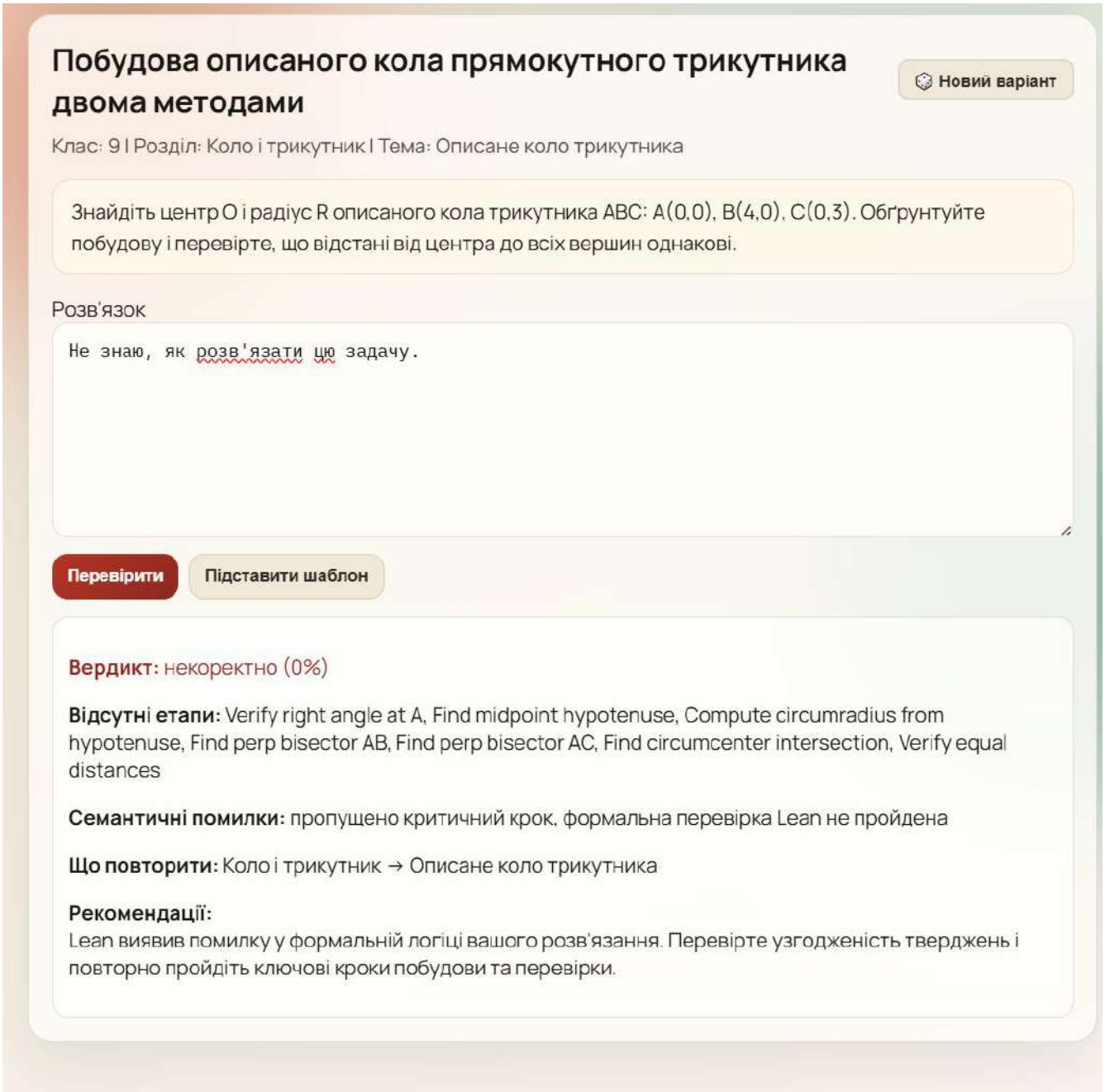


Рис. 2.10. Візуалізація результатів при повній відсутності математичних міркувань.

Така багаторівнева візуалізація вердиктів підтверджує ефективність обраної архітектури для завдань формувального оцінювання, дозволяючи системі гнучко реагувати на будь-який рівень підготовки користувача.

РОЗДІЛ 3. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОБОТИ СИСТЕМИ

3.1. Методологія тестування та інструментарій

Для оцінки надійності, відмовостійкості та математичної коректності розробленого дослідницького прототипу було застосовано комбіновану методологію тестування. Вона поєднує автоматизовані сценарії на рівні бекенду та ручну перевірку через клієнтський веб-інтерфейс і допоміжні debug-утиліти. Усі автоматизовані тестові сценарії реалізовані як незалежні CLI-скрипти, що гарантує повну відтворюваність результатів без необхідності модифікації вихідного коду.

Основу системи забезпечення якості складає набір спеціалізованих скриптів, які перевіряють різні аспекти нейро-символічного пайплайну:

- run_unit_tests.py: Відповідає за тестування базових функцій та ізольованих компонентів системи.
- run_regression.py та run_theorem_regression.py: Забезпечують регресійне тестування вільнотекстового верифікатора та логіки теорем, гарантуючи, що внесення змін до Lean-ядра або оновлення системних промптів не порушує попередньо стабільний функціонал.
- run_fuzz_regression.py: Реалізує метод fuzz-тестування для перевірки стійкості оцінювання на нейтральних, змагальних та додатково згенерованих текстових варіантах розв'язків.
- run_self_test.py: Виконує інтеграційне самотестування коректності оцінювання, запускаючи функціональні перевірки на основі наперед заданого банку кейсів та їх мутацій.

Важливим інженерним рішенням стало впровадження формалізованих профілів якості (Quality Gates) через скрипт run_quality_gate.py. Процес контролю якості є системним і передбачає три профілі інтенсивності:

1. Smoke: Використовується для швидкої валідації критичного функціоналу та включає 120 fuzz-згенерованих перевірок. Цей профіль вже зафіксовано в системних журналах тестування.
2. Medium: Передбачений для розширеного тестування на 500 згенерованих сценаріїв.
3. Heavy: Спроектований для глибокого стрес-тестування, що охоплює 3000 згенерованих перевірок.

Наявність заздалегідь спроєктованих профілів medium та heavy дозволяє гнучко масштабувати процеси тестування при подальшому розвитку проєкту.

3.2. Кількісні та якісні результати верифікації

Під час експериментальної перевірки прототипу було підтверджено його працездатність та здатність обробляти розв'язки різного рівня якості.

3.2.1 Якісні результати візуалізації та маршрутизації помилок

Як було продемонстровано під час аналізу клієнтського інтерфейсу (див. підрозділ 2.5.4), система коректно реалізує багаторівневі сценарії зворотного зв'язку. Механізми engine.py успішно класифікують математично бездоганні розв'язки та ідентифікують логічні розриви для виставлення часткового бала. Завдяки складній логіці маршрутизації, система може відносити нерелевантні або беззмістовні текстові введення до некоректних або неповних розв'язків, позначаючи їх відповідними семантичними помилками.

3.2.2 Кількісні результати автоматизованого тестування

Під час розробки та фінального налагодження системи було проведено не менше 198 автоматизованих перевірок у межах базового smoke-профілю Quality Gate. Цей набір включав:

- 61 regression-випадок для вільнотекстового верифікатора;
- 3 theorem-regression перевірки;
- 14 модульних unit-тестів;
- 120 fuzz-згенерованих текстових перевірок. Усі вказані тести були успішно пройдені, що підтверджує технічну цілісність ядра формальної верифікації та узгоджується з працездатністю Lean-компонента. Варто зазначити межі валідності цих результатів: успішність автоматизованих тестів надійно підтверджує технічну коректність реалізації прототипу, проте не може гарантувати абсолютного покриття всіх можливих семантичних варіацій та природномовних формулювань, які можуть згенерувати реальні учні.

3.2.3 Оцінка часових характеристик

Оскільки система функціонує в дослідницькому середовищі, окремі мікробенчмарки користувацької латентності не проводилися. Часові характеристики оцінювалися через операційні обмеження системи та журнали Quality Gate.

Фактичний час відповіді сервера є динамічним і залежить від складності задачі, кількості спроб LLM-кодогенерації та стану бази даних SQLite. Для забезпечення стабільності серверної інфраструктури впроваджено жорсткі часові ліміти на рівні компонентів:

- Формальна перевірка у Lean 4 запускається як окремий підпроцес із жорстким watchdog-таймаутом у 60 секунд.

- Мережевий виклик до моделі Google Gemini обмежений таймаутом у 60 секунд.
- Структурний LLM-парсер має власний HTTP-таймаут у 20 секунд.

Згідно з журналами smoke-профілю, виконання пакета з 14 unit-тестів займає всього 4.566 секунди. Проте, при повному циклі обробки нового, некешованого розв'язку час очікування обмежується виключно вказаними системними таймаутами.

3.3. Аналіз виявлених обмежень та технічного боргу

Розробка та експериментальне дослідження магістерського проєкту дозволили виявити низку системних та інфраструктурних обмежень, притаманних запропонованій архітектурі нейро-символічної верифікації. Оскільки розроблена система має статус дослідницького прототипу, детальний аналіз її технічного боргу та меж ефективності є необхідною умовою для об’єктивної оцінки практичної цінності отриманих результатів.

3.3.1. Обмеження нейро-символічної трансляції знань

Головні компроміси та обмеження на етапі перекладу математичного тексту у формальне представлення зосереджені навколо двох чинників:

- Семантична неоднозначність природної мови: Розроблений NLP-шар демонструє високу точність при обробці типових учнівських формулювань та структурованих викладів. Проте при зіткненні із надто неструктурованими відповідями, граматично розірваними реченнями або логічно заплутаними висловлюваннями точність екстракції фактів може суттєво знижуватися. Подолання цього семантичного розриву є фундаментальним викликом, оскільки система намагається побудувати детермінований граф для недетермінованого людського введення.
- Варіативність LLM-генерації формального коду: Оскільки синтез коду мовою Lean 4 делеговано зовнішній великій мовній моделі, процес генерації

є стохастичним за своєю природою. Впроваджений механізм зворотного зв’язку та параметризована система повторних спроб генерації при виявленні синтаксичних помилок Lean значно підвищують стійкість пайплайну. Проте вони не здатні повністю нівелювати ймовірність випадкових збоїв моделі на складних логічних переходах з першої спроби.

3.3.2. Архітектурні та інфраструктурні компроміси прототипу

Технічна реалізація контуру виконання та управління ресурсами операційної системи має такі специфічні особливості:

- Синхронна перевірка в межах HTTP-запиту: Виконання формальної верифікації через запуск ізольованого дочірнього процесу операційної системи реалізовано в межах синхронного циклу обробки HTTP-запиту сервером. Такий підхід є оптимальним та архітектурно виправданим для фінальної демонстраційної версії додатка через його простоту. Водночас це накладає обмеження на передбачуваність латентності: при опрацюванні граничних і складних кейсів, де процес елаборації Lean може використовувати значний обсяг часу, час очікування відповіді клієнтом стає нестабільним.
- Локальна серіалізація запитів на клієнті: Впроваджений у веб-інтерфейсі механізм FIFO-черги надійно стабілізує навантаження та виключає зайві повторні запити в межах однієї конкретної сесії користувача. Однак, на рівні бекенду системи відсутній глобальний розподілений механізм балансування навантаження або менеджер черг (на кшталт Celery або іншого брокер/воркерного контуру). Через це можливості горизонтального масштабування додатка у багатокористувацьких сценаріях із сотнями одночасних запитів залишаються обмеженими.
- Залежність від зовнішніх сервісів: Архітектурний контур системи не є повністю автономним, оскільки критична фаза NLP-інтерпретації залежить

від доступності хмарного API провайдера мовної моделі та стабільності мережевого з’єднання. Інтегрований модуль локального персистентного кешування на базі SQLite та впроваджені fallback-алгоритми суттєво знижують операційні ризики та вартість експлуатації, проте система все одно зберігає зовнішні інфраструктурні залежності.

3.4. Висновки щодо практичного застосування

Розробка та успішна апробація дослідницького прототипу системи автоматизованої верифікації геометричних міркувань дозволяє перевести акцент із суто теоретичних засад нейро-символічної інтеграції та формальних методів математичної логіки у площину їхнього безпосереднього прикладного впровадження в освітній процес. Спроєктована архітектура має виражену практичну цінність для сучасної шкільної математичної освіти, оскільки вона пропонує інструменти подолання ключових недоліків традиційних автоматизованих систем тестування.

3.4.1. Ключові вектори інтеграції системи в навчальний процес

Практичний потенціал створеного програмного комплексу реалізується за кількома основними напрямами, що охоплюють потреби як здобувачів освіти, так і педагогів:

- Організація формувального оцінювання у середній школі: Традиційні системи комп’ютерної перевірки знань зазвичай обмежуються бінарним вердиктом «правильно/неправильно» на основі фінального числового результату. На противагу цьому, розроблена система функціонує як інтерактивний тренажер із негайним розгорнутим зворотним зв’язком. Учень має можливість бачити цілісний аналіз власного ходу міркувань, оперативно фіксувати, на якому саме етапі доведення чи обчислення було допущено логічний розрив, та самостійно коригувати свій розв’язок в ітераційному режимі.
- Підготовка до стандартизованого тестування: Структура каталогу задач системи орієнтована на складні аналітичні теми шкільного курсу

планіметрії, зокрема на координатну геометрію, властивості прямих, обчислення точок перетину, середин відрізків, перевірку паралельності та перпендикулярності фігур. У контексті підготовки до державної підсумкової атестації та національного мультипредметного тесту вкрай важливо навчити учня не просто вгадувати кінцеву відповідь, а будувати несуперечливий, математично строгий ланцюжок аргументів. Система безпосередньо стимулює розвиток цієї навички, перевіряючи кожен проміжний крок.

- Інструмент автономної самопідготовки та персоналізації навчання: Прототип здатний виконувати роль «персонального цифрового тренажера». Учень може опрацьовувати геометрію у власному індивідуальному темпі, незалежно від загального ритму класу. Завдяки модулю оцінювання, який діагностує слабкі місця та автоматично формує індивідуальні рекомендації щодо повторення тем за результатами виявлених семантичних або формальних помилок, забезпечується адаптивність навчання та заповнення прогалин у знаннях.
- Інтеграція у сучасне цифрове освітнє середовище: Спроєктована REST API архітектура та модель stateless-оркестрації роблять прототип повністю готовим до інтеграції у промислові системи управління навчанням, такі як Moodle, Google Classroom або інші профільні освітні екосистеми. Програмний комплекс може вбудовуватися як зовнішній сервіс автоматичної перевірки або функціонувати як ізольований сервісний модуль через інтеграцію веб-вікна безпосередньо всередині інтерфейсу LMS. Це створює надійний інженерний місток між розробленою моделлю та реальним шкільним середовищем.
- Підтримка професійної діяльності вчителя: Впровадження системи суттєво оптимізує часовий ресурс педагога. Замість рутинної ручної перевірки сотень однотипних учнівських зошитів, вчитель отримує автоматизований

аналітичний звіт. Система дозволяє оперативно виконувати швидку перевірку типових відповідей, здійснювати автоматизоване виявлення кореляцій та специфічних помилок у межах усього класу, а також здійснювати добір дидактичних задач із каталогу за конкретною мікро-темою та рівнем складності для окремих учнів.

3.4.2. Межі дидактичної застосовності та фінальний підсумок

Визначаючи межі практичного використання розробленого продукту, необхідно чітко зафіксувати його дидактичний статус. Створена система позиціонується виключно як високоефективний допоміжний інструмент підтримки навчання, а не як повноцінна заміна професійної діяльності вчителя. Комп’ютерний верифікатор покликаний рутинізувати перевірку стандартних кроків та вивільнити час педагога для вирішення творчих, методологічних та індивідуальних психолого-педагогічних завдань у класі.

Найбільша практична та науково-методична цінність розробленої архітектури полягає у синергетичному поєднанні автоматичної формальної перевірки компілятором Lean 4 із гнучким пояснювальним зворотним зв’язком, згенерованим NLP-шаром. Такий підхід робить систему унікальним інструментом саме для тих розділів шкільної математики, де домінує жорстко формалізований хід розв'язання та критично важливою є топологічна чи алгебраїчна послідовність міркувань, а не просто констатація фінального чисельного результату. Отримані результати підтверджують життєздатність розробленої нейро-символічної моделі та відкривають перспективу для впровадження інтерактивних систем автоматизованого доведення теорем у широку педагогічну практику базової середньої освіти.

Висновки

У кваліфікаційній магістерській роботі розв’язано актуальне науково-практичне завдання — розроблено та досліджено архітектуру інтелектуальної програмної системи, яка використовує середовище Lean 4 для автоматизованої перевірки геометричних міркувань учнів та надання адаптивних рекомендацій.

За результатами проведеного дослідження зроблено такі висновки:

1. Виконано аналіз існуючих підходів до автоматизованої перевірки математичних доведень. З’ясовано, що традиційні системи управління навчанням та системи динамічної геометрії не здатні долати семантичний розрив між природномовною аргументацією учня та строгим формальним виводом. Обґрунтовано необхідність використання інтерактивних доводжувачів теорем. Вибір фреймворку Lean 4 довів свою ефективність завдяки поєднанню функціональної мови програмування загального призначення з ядром формальної верифікації на базі числення індуктивних конструкцій.
2. Спроектовано механізми формалізації геометричних аксіом та теорем засобами мови Lean. Розроблено власну модульну бібліотеку геометричних операцій та предикатів. Застосовано прагматичний компроміс у системі числення: структурні та логічні відношення визначено над полем раціональних чисел для гарантування абсолютної детермінованості, тоді як лінійні евклідові метрики обчислюються з використанням типів із рухомою комою. Для забезпечення безпеки обчислення часткових операцій успішно використано систему залежних типів та монаду Option.
3. Розроблено та імплементовано тришарову клієнт-серверну архітектуру нейро-символічної взаємодії. Запропоновано модель, де велика мовна

модель (Google Gemini 2.5 Flash) функціонує як гнучкий NLP-інтерфейс для екстракції фактів та динамічної генерації коду, а Lean 4 виступає в ролі строгого логічного ядра. Для оптимізації роботи бекенду впроваджено персистентне кешування на базі SQLite та механізм клієнтської FIFO-серіалізації запитів, що забезпечило стабільну роботу системи в умовах ресурсомістких Lean-процесів із 60-секундним таймаутом.

4. Створено алгоритм діагностики та співставлення формалізованих помилок з темами навчальної програми. Розроблено рушій оцінювання, що агрегує математичні висновки та результати семантичного NLP-аналізу. Алгоритм здатний виявляти логічні розриви, локалізувати пропущені кроки доведення та формувати трирівневі сценарії зворотного зв'язку, генеруючи індивідуальні рекомендації для учня.
5. Проведено експериментальну перевірку ефективності системи. Завдяки реалізації підсистеми генерації варіантів з механізмами уникнення вироджених конфігурацій, працездатність прототипу підтверджено серією з понад 198 автоматизованих тестових перевірок.

Розроблений дослідницький прототип підтверджує життєздатність нейро-символічної архітектури і має високу дидактичну цінність як інтерактивний тренажер для організації формувального оцінювання у 7–9 класах.

Джерела

1. Про повну загальну середню освіту : Закон України від 16 січ. 2020 р. № 463-IX // Відомості Верховної Ради України. – 2020. – № 31. – Ст. 226.
2. Про деякі питання державних стандартів повної загальної середньої освіти : постанова Кабінету Міністрів України від 30 верес. 2020 р. № 898. – URL: https://zakon.rada.gov.ua/laws/show/898-2020-п (дата звернення: 11.05.2026).
3. Геометрія. 7-9 класи : модельна навчальна програма для закладів загальної середньої освіти / М. І. Бурда, Н. А. Тарасенкова, Д. В. Васильєва. – Київ : МДУ, 2021. – 24 с.
4. Black P. Assessment and classroom learning / P. Black, D. Wiliam // Assessment in Education: principles, policy & practice. – 1998. – Vol. 5, No. 1. – P. 7-74.
5. Hilbert D. The Foundations of Geometry / D. Hilbert. – Chicago : The Open Court Publishing Company, 1902. – 132 p.
6. Tarski A. What is elementary geometry? / A. Tarski // The axiomatic method. With special reference to geometry and physics. – Amsterdam : North-Holland, 1959. – P. 16-29.
7. Weyl H. Space-Time-Matter / H. Weyl. – New York : Dover Publications, 1952. – 330 p.
8. Ganesalingam M. The Language of Mathematics: A Linguistic and Philosophical Investigation / M. Ganesalingam. – Berlin, Heidelberg : Springer-Verlag, 2013. – 248 p.
9. Hohenwarter M. Ways of linking geometry and algebra: The case of GeoGebra / M. Hohenwarter, K. Jones // Proceedings of British Society for Research into Learning Mathematics. – 2007. – Vol. 27, No. 3. – P. 126-131.

10. De Moura L. The Lean 4 theorem prover and programming language / L. de Moura, S. Ullrich // Automated Deduction – CADE 28: 28th International Conference on Automated Deduction. – Cham : Springer International Publishing, 2021. – P. 625-635.
11. De Moura L. Z3: An Efficient SMT Solver / L. de Moura, N. Bjørner // Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008). – Springer, Berlin, Heidelberg, 2008. – P. 337-340.
12. Bertot Y. Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions / Y. Bertot, P. Castéran. – Springer Science & Business Media, 2013. – 469 p.
13. Nipkow T. Isabelle/HOL: A Proof Assistant for Higher-Order Logic / T. Nipkow, L. C. Paulson, M. Wenzel. – Springer Science & Business Media, 2002. – 223 p.
14. Paulin-Mohring C. Inductive definitions in the system Coq rules and properties // International Conference on Typed Lambda Calculi and Applications. – Springer, Berlin, Heidelberg, 1993. – P. 328-345.
15. Sørensen M. H. Lectures on the Curry-Howard isomorphism / M. H. Sørensen, P. Urzyczyn. – Elsevier, 2006. – 396 p.
16. Gordon M. J. C. Edinburgh LCF: A Mechanised Logic of Computation / M. J. C. Gordon, A. J. R. G. Milner, C. P. Wadsworth. – Springer-Verlag, 1979. – 159 p.
17. The mathlib Community. The Lean mathematical library / The mathlib Community // Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020). – ACM, 2020. – P. 367-381.