

AI Tutor: A Personalized Learning Companion for Students

Artificial Intelligence Technical Report

Final Report COMP-385 (SEC.001)

Declaration:

We, Group #4, confirm that the work submitted for assessment is our own and is expressed in our own words. Any uses made within it of the works of other authors, in any form (ideas, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of references used is included.

Team Group #4

Achint Singh	301298529	
Adarsh Sindhu Prabhan	301364453	
Aazain Ullah Khan	301277063	
Reet Kaur	301318521	
Yajushi Garg	301274532	

Submitted on 5th December 2025

EXECUTIVE SUMMARY

Centennial College's Luminate system currently provides static course materials without intelligent tutoring, leaving COMP 237 students without personalized, adaptive support. This absence of context-aware assistance leads to unresolved misconceptions and persistent knowledge gaps, preventing students from reaching their full academic potential.

AI Tutor addresses these issues by providing immediate, tailored assistance through a scaffolding-based pedagogy that guides students to discover answers via progressive hints and Socratic questioning, rather than providing direct solutions. This maintains academic integrity while building genuine understanding through guided discovery.

The system employs a multi-agent architecture using LangGraph with five specialized components. The PlannerAgent uses intelligent routing combining heuristic pattern matching with LLM-based fallback for query classification. The TutorAgent integrates Retrieval-Augmented Generation (RAG) using ChromaDB with 438 embedded chunks from 15 course documents (13 PDFs, 1 DOCX, 1 TXT), employing a 4-level escalation system that adapts based on student confusion signals. The MathAgent leverages SymPy for symbolic mathematics and Wolfram Alpha API for computational verification, delivering progressive hint-based guidance. The EvaluatorAgent tracks concept mastery by detecting discussed concepts, updating student mastery scores in Supabase, and logging interactions for analytics. The FeedbackAgent, designed for weekly automated analysis using LLaMA 3 via Ollama to identify low-rated conversations and generate improvement insights, is planned for future implementation.

The technical infrastructure uses Google Gemini 2.5 Flash as the primary LLM, with PostgreSQL storing conversation logs and ratings, Supabase providing managed database services, and Langfuse enabling comprehensive observability. The Chrome extension integrates seamlessly with Canvas LMS.

The system demonstrates strong routing performance to appropriate specialized agents, high scaffolding compliance with pedagogical guidelines, effective RAG integration with course-specific materials, and robust academic integrity enforcement preventing direct solution provision. User feedback indicates positive reception, particularly praising the progressive hint system and contextual course integration. Unlike generic chatbots, Luminate AI employs specialized agents with domain-specific capabilities, adaptive scaffolding, automated quality assurance, and curriculum-aligned responses. The system successfully demonstrates that AI-powered tutoring can maintain pedagogical rigor while providing scalable, personalized support that builds understanding through guided discovery.

Table of Contents

- EXECUTIVE SUMMARY 2
- LIST OF ILLUSTRATIONS 6
- Figures..... 6
- Tables..... 6
- 1. INTRODUCTION 7
 - 1.1 Background.....7
 - 1.2 Problem Statement8
 - 1.3 Significance.....8
 - 1.4 Literature Review Summary.....9
 - 1.5 Exploratory Data Analysis Summary..... 11
- 2. METHODOLOGY 12
 - 2.1 Data Acquisition and Processing..... 12
 - 2.2 Data Preprocessing and ETL Pipeline..... 13
 - 2.3 Experimental Design..... 15
- 3. SYSTEM ARCHITECTURE 16
 - 3.1 Overall System Design..... 16
 - 3.1.1. Architectural Layers 16
 - 3.1.2 Service Communication 17
 - 3.1.3 Authentication Architecture..... 17
 - 3.1.4 Scalability and Fault Tolerance 17
 - 3.1.5 Security Posture 18
 - 3.2 Agentic Architecture..... 18
 - 3.2.1 LangGraph Orchestration..... 19
 - 3.2.2 Agent Overview 19
 - 3.2.3 Technology Stack 20
 - 3.3 Data Storage Layer 20
 - 3.4 API Layer 21
 - 3.5 Frontend Interface 21

3.6 Integration and Communication.....	22
3.7 Introspection and Monitoring	22
Real-Time Performance Tracking	22
Weekly Retrospective Analysis	22
3.8 Authentication and Security.....	23
4. RESULTS/DATA/ANALYSIS	23
Preliminary Insights Detected.....	24
Functional Test Results	24
5. CONCLUSIONS	32
5.1 Key Achievements	32
5.2 Identified Challenges & Limitations	32
6. RECOMMENDATIONS	33
6.1 Immediate System Improvements	33
6.2 Validation and Scale Testing	33
6.3 Platform Extensions	33
6.4 Personalization Features	33
6.5 Content and Model Enhancements	34
6.6 Infrastructure Scalability	34
BIBLIOGRAPHY	34
APPENDICES	35
Appendix A: Project Proposal	35
Appendix B: Stakeholder Register.....	35
Appendix C: System Specification	35
Use Case 1 — Receive Scaffolded Concept Explanation	36
Use Case 2 — Receive Scaffolded Math Problem.....	38
Use Case 3 Off-Topic Query Handling	39
Use Case 4 —Generative Adaptive Study Recommendations	40
Functional Requirements	41
Non-Functional Requirements (Table 3)	42

Appendix D: System Design.....	42
LangGraph Multi- Agent Architecture	42
EvaluatorAgent (Concept Tracking & Mastery Updates)	51
FeedbackAgent (System-Level Quality Assurance Agent)	52
Scaffolding Pedagogy Implementation.....	55
Four Level Escalation System	55
Frontend Technologies:	56
Backend Technologies:	57
Database Technologies:	58
AI CAPABILITY AND DATA DESIGN DESCRIPTIONS	58
System Architecture Diagram (Figure 1).....	61
Planner Agent Workflow Diagram (Figure 2).....	62
Tutor Agent Workflow Diagram (Figure 3)	63
Math Agent Workflow Diagram (Figure 4).....	64
Component Diagram (Figure 5).....	64
Class Diagram (Figure 6).....	65
Gantt Diagram (Figure 7).....	65
Deployment Strategy	66
High level ML/ops description	67
GLOSSARY OF TERMS.....	69

LIST OF ILLUSTRATIONS

Figures

Figure 1: Use Case Diagram - Student Interactions Appendix C

Figure 2: System Architecture Diagram Appendix D

Figure 3: Agentic AI Component Architecture Appendix D

Figure 4: Class Diagrams..... Appendix D

Tables

Table 1: Stakeholder Register Appendix B

Table 2: Functional Requirements Appendix C

Table 3: Non-Functional Requirements Appendix C

Table 4: Technology Stack Appendix D

1. INTRODUCTION

1.1 Background

The landscape of higher education is experiencing a fundamental transformation driven by technological advancement and evolving student expectations. Traditional Learning Management Systems (LMS), while providing essential infrastructure for course delivery, predominantly offer static content without adaptive feedback mechanisms, real-time knowledge gap identification, or personalized tutoring capabilities. Students navigate course materials independently, often struggling to identify conceptual weaknesses or receive timely guidance on complex topics.

Centennial College utilizes the Anthology LMS platform (Blackboard) branded as Luminate for course management and content delivery. While this system provides robust administrative functionality and centralized resource access, students in complex courses like COMP 237 (Introduction to Artificial Intelligence) face significant challenges. The abstract nature of AI concepts including neural networks, optimization algorithms, and model evaluation combined with limited availability of teaching assistants and office hours, creates barriers to effective learning. Students often struggle to identify their specific knowledge gaps, receive immediate clarification on misconceptions, and access personalized guidance aligned with their individual learning pace.

Recent advances in artificial intelligence, particularly in large language models (LLMs) and multi-agent systems, present unprecedented opportunities to augment traditional educational platforms with intelligent, adaptive learning companions. LLMs have demonstrated remarkable capabilities in natural language understanding and generation, while retrieval-augmented generation (RAG) enables integration of domain-specific knowledge. Multi-agent architectures allow decomposition of complex tasks into specialized components, each optimized for specific functions.

Rather than building a monolithic chatbot that attempts to handle all tutoring scenarios with a single model, our approach employs a multi-agent architecture where specialized agents address distinct educational needs. The PlannerAgent intelligently routes queries, the TutorAgent provides conceptual explanations augmented with course materials, the MathAgent handles computational problem-solving with verification tools, the EvaluatorAgent tracks student mastery, and the FeedbackAgent (planned) will enable continuous quality improvement. Each agent operates with tailored prompting strategies, specialized tools, and distinct LLM configurations optimized for its pedagogical role.

These capabilities are delivered through the AI Tutor Chrome extension, which integrates seamlessly with Centennial College's LMS environment. The extension provides context-aware assistance directly within the student's learning workflow, eliminating the need to switch between platforms while maintaining strict academic integrity through intelligent policy enforcement and scaffolded pedagogy that emphasizes guided discovery over direct solution provision.

1.2 Problem Statement

The core problem this project addresses is:

The absence of intelligent, adaptive AI tutoring capabilities in Luminare that can provide personalized concept clarification and practice problem assistance for COMP 237 students through agentic workflows.

The current Luminare LMS implementation exhibits several critical limitations:

- **Absence of Intelligent Tutoring Capabilities:** Luminare cannot interpret student questions semantically or distinguish between conceptual and computational needs, leaving students without adaptive assistance for concept clarification (40% of requests) and practice problem help (25% of requests) identified in our COMP 237 survey.
- **Lack of Semantic Search and RAG Integration:** Basic keyword search cannot leverage retrieval-augmented generation with vector embeddings to retrieve semantically relevant course materials, worked examples, or theoretical explanations from the 1,638-document repository.
- **No Computational Problem-Solving Support:** Students lack step-by-step solutions for calculus, linear algebra, or probability problems central to COMP 237, forcing reliance on external tools that fragment study workflows.
- **No Quality Assurance or Continuous Improvement:** Without automated feedback loops, the system cannot detect poor responses or refine tutoring strategies based on student satisfaction ratings.
- **Lack of Intelligent Query Routing:** All queries receive uniform responses instead of routing conceptual questions (e.g., backpropagation), mathematical problems (e.g., gradient descent), or coding tasks to specialized strategies.

These deficiencies create inefficient workflows, persistent knowledge gaps, and reduced readiness for applying AI/ML concepts.

1.3 Significance

This project holds substantial significance across multiple dimensions of educational technology and student success:

- i. **Critical Educational Need:** COMP 237 students struggle with algorithmic thinking, mathematical foundations, and AI/ML concepts. Without immediate assistance for concept clarification and computational problem-solving, learners experience persistent knowledge

gaps over time, leading to poor assessment performance and reduced confidence in applying technical concepts. This affects hundreds of students annually, creating a scalability crisis where instructor office hours cannot meet demand.

- ii. **Institutional Impact and Accessibility:** The system addresses instructor workload challenges while enabling 24/7 tutoring availability without proportional staffing increases. This scalability is particularly critical for part-time and distance learners who comprise a significant portion of Centennial's student body, ensuring equitable access to personalized learning support regardless of schedule or location constraints.
- iii. **Advancement of Agentic AI in Education:** Unlike monolithic chatbots, this multi-agent workflow demonstrates specialized agents collaborating through LangGraph with purpose-specific tools (ChromaDB RAG, SymPy, Wolfram Alpha). The planned FeedbackAgent, designed for automated quality assurance using LLaMA 3, will represent a novel self-improving approach, generating actionable pedagogical insights without manual analysis, a capability that positions this system beyond current educational AI implementations.
- iv. **Replicable Framework for Scale:** Built on open educational resources and semantic search, the architecture is applicable beyond COMP 237 to mathematics, engineering, and data science courses, enabling institutions to deploy intelligent tutoring without substantial per-course customization or resource-intensive adaptation periods.

1.4 Literature Review Summary

Recent research in educational technology and artificial intelligence provides substantial foundation for the AiTutor project approach and validates key design decisions.

- i. **Personalized Learning Through AI:** The comprehensive review "AI and Education: Bridging the Gap to Personalized, Efficient and Accessible Learning" (2024) demonstrates that properly implemented AI-powered educational tools can improve learning outcomes by 23-35% through personalized content delivery. However, the research identifies significant gaps in real-time knowledge gap detection and autonomous learning path generation, precisely the areas where AiTutor introduces innovation through its agentic architecture.
- ii. **Agentic AI in Education:** Chen et al. (2024) evaluated autonomous AI agents in educational contexts, reporting 40% improvement in student engagement when systems independently

adapt to learning patterns. Their research emphasizes Socratic questioning methodologies and real-time misconception detection as critical components of effective AI tutoring systems. AiTutor extends these findings by implementing multiple specialized agents with domain expertise rather than relying on general-purpose conversational AI.

- iii. **Browser Extension Educational Tools:** Kumar and Zhang (2023) analyzed student engagement with Chrome extensions for educational support, finding that seamless integration with existing platforms significantly increases adoption rates compared to standalone applications. Their research validates our decision to implement AiTutor as a Chrome extension integrated directly with the Luminate LMS interface, eliminating context-switching and maintaining workflow continuity.
- iv. **Vector-Based Content Retrieval:** Park et al. (2024) demonstrated that vector-based semantic search in learning management systems provides superior content discovery compared to traditional keyword matching, with students locating relevant materials 60% faster. Their findings support our implementation of ChromaDB vector database with sentence-transformer embeddings for intelligent content retrieval from 1,638 course documents and 2,239 open educational resources.
- v. **Predictive Modeling in Computer Science Education:** Garcia and Thompson (2023) developed predictive models for student performance in computer science courses, achieving 78% accuracy in identifying students at risk of poor outcomes. Their methodology informs our student mastery tracking implementation in Supabase, which monitors learning patterns and enables adaptive scaffolding based on demonstrated competency levels.
- vi. **Critical Gap:** While existing research demonstrates the effectiveness of individual components (personalization, agentic AI, semantic search), no current system integrates these capabilities into a unified, course-specific learning companion embedded within existing LMS infrastructure. AiTutor addresses this gap by combining autonomous agents, vector-based retrieval, persistent student modeling, and specialized educational expertise in a single coherent system tailored specifically for COMP 237 curriculum requirements.

1.5 Exploratory Data Analysis Summary

The initial exploratory data analysis phase focused on understanding student needs, content landscape, and technical requirements for the multi-agent AI tutoring system.

Student Needs Assessment: Consultation with 15+ COMP 237 current learners and faculty revealed the following distribution of tutoring requests:

- Concept clarification (40%)
- Practice problem help (25%)
- Study planning (20%)
- Navigation support (10%)
- Assessment preparation (5%)

These findings validated the need for specialized agents: TutorAgent for conceptual explanations and MathAgent for computational problem-solving, collectively addressing 65% of student tutoring demand.

Data Sources for RAG Implementation:

- i. COMP 237 Course Materials: Syllabi, lecture notes, assignments, and module content, covering machine learning fundamentals, neural networks, optimization algorithms, natural language processing, computer vision, and probabilistic reasoning.
- ii. eCampusOntario Open Educational Resources: Supplementary textbooks, tutorials, and Creative Commons materials to provide alternative explanations and worked examples for complex concepts.

Content Analysis Findings: Analysis of course materials revealed high density of mathematical equations and algorithmic problems, validating the need for MathAgent with computational tools (SymPy, Wolfram Alpha). Conceptual explanations frequently required contextual understanding, justifying TutorAgent's RAG-based retrieval approach. Mixed query types requiring intelligent classification necessitated PlannerAgent's hybrid heuristic-LLM routing logic.

Technical Infrastructure Requirements: Based on the exploratory analysis, the following technical components were identified as necessary:

- i. **Vector Database:** ChromaDB for semantic search across 1,638 course documents and 2,239 OER materials with sentence-transformer embeddings
- ii. **Computational Tools:** SymPy for symbolic mathematics and Wolfram Alpha API for complex calculations to support MathAgent

- iii. **Quality Monitoring:** Supabase (PostgreSQL) for conversation logging and student mastery tracking; Langfuse for real-time observability and user feedback collection
- iv. **LLM Configuration:** Google Gemini 2.5 Flash for intelligent routing and response generation across all core agents
- v. **Automated Quality Assurance (Planned):** PostgreSQL analytics tables and LLaMA 3 via Ollama for FeedbackAgent's automated critique generation and insight storage

2. METHODOLOGY

2.1 Data Acquisition and Processing

Course materials for AiTutor were acquired from two complementary sources: Centennial College's Luminare Learning Management System (Blackboard) for course-specific content, and Open Educational Resources (OER) for supplementary conceptual explanations. The system architecture supports both data sources through a unified ETL pipeline.

i. **Primary Data Source: Blackboard Course Materials**

The primary data source consists of official COMP 237 course materials extracted from Centennial College's Blackboard LMS using the IMS Content Package export format.

Extraction Process: A custom Blackboard parser (`blackboard_parser.py`) unpacked the course export ZIP archive and parsed the IMS manifest XML (`imsmanifest.xml`) to reconstruct the course hierarchy. The parser mapped resource IDs to human-readable titles and identified file attachments embedded within course resources.

Quality Filtering: The Blackboard export contained 396 resource entries. An automated quality scoring system evaluated resources on five criteria: module mapping (30 points), title descriptiveness (20 points), student availability (20 points), content substance (20 points), and technical parsability (10 points). Resources scoring $\geq 60/100$ were processed for file extraction.

Content Composition: Course outline, 11 lecture material files, ethics readings, and lab tutorials spanning all 13 weeks of COMP 237 curriculum (AI foundations, search algorithms, machine learning, neural networks, NLP, computer vision, and ethics).

ii. **Secondary Data Source: Open Educational Resources (OER)**

To supplement course-specific materials with broader conceptual explanations, the system architecture incorporates Open Educational Resources from verified academic sources.

OER Categories:

- Academic textbooks: OpenStax machine learning and AI fundamentals texts
- Technical documentation: Scikit-learn, TensorFlow, PyTorch official documentation
- Educational platforms: Khan Academy content on calculus, linear algebra, probability
- Research papers: ArXiv preprints on foundational AI/ML topics (2018-2024)

Acquisition Strategy: OER materials are collected through automated web scraping (BeautifulSoup, Selenium) and manual download from authorized repositories. All sources are verified for licensing compatibility (CC-BY, CC-BY-SA, public domain) to ensure legal redistribution within educational contexts.

Quality Criteria: OER content undergoes filtering based on relevance to COMP 237 learning objectives, recency (materials from 2019-2024 for technical content), authoritativeness (academic institutions, peer-reviewed publications), and technical accuracy verified against course materials.

The system supports dual-collection architecture for integrating course-specific materials with broader OER content, enabling RAG queries to access both curriculum-aligned resources and supplementary educational materials.

2.2 Data Preprocessing and ETL Pipeline

The source documents underwent systematic preprocessing through a multi-stage Extract-Transform-Load (ETL) pipeline (pipeline.py) to transform content into semantically searchable vector embeddings.

Text Extraction

- Document Processing:** The document processor (document_processor.py) employed format-specific extraction strategies:
 - PDF extraction: PyPDF2 library extracted text while preserving paragraph structure and heading hierarchies
 - DOCX processing: python-docx extracted formatted content with list items and section organization
 - TXT processing: Direct UTF-8 file reading with encoding normalization
- Text Cleaning:** Extracted text underwent normalization (whitespace standardization, UTF-8 encoding), noise reduction (header/footer removal, page number filtering, watermark detection), and quality validation (minimum 100-character threshold, duplicate detection via SHA-256 hashing, language verification).

Semantic Chunking: The pipeline employed semantic boundary detection rather than fixed-size splitting, prioritizing conceptual coherence:

- **Boundary detection:** Chunking at paragraph breaks, sentence endings, and section divisions
- **Content preservation:** Code blocks and mathematical derivations maintained as atomic units to preserve syntactic and logical validity
- **Adaptive sizing:** Target chunk size of 1,000 characters with 200-character overlap between consecutive chunks to maintain context continuity across boundaries
- **Metadata enrichment:** Each chunk tagged with source file, chunk index, character offsets (chunk_start, chunk_end), week number, content type

Chunking Results:

- **Input:** 15 source documents
- **Output:** 438 text chunks
- **Expansion factor:** 29.2x (average 29 chunks per document)
- **Content distribution:** Concept definitions (380 chunks, 87%), Syllabus content (58 chunks, 13%)

Embedding Generation and Vector Storage

Embedding Model: Chunks were embedded using Google Gemini's text-embedding-004 API via LangChain Google Generative AI, generating 768-dimensional dense vectors. The model was selected for consistency with the generative LLM (Gemini 2.5 Flash) and its capability to handle technical AI terminology and mathematical notation. Embedding generation processed all 438 chunks in batches of 100 with exponential backoff retry logic to handle API rate limiting (60 requests/minute).

ChromaDB Storage: Embeddings were loaded into a ChromaDB collection deployed in a Docker container (memory_store service, port 8001) with persistent volume storage at chromadb_data/. The system uses an HNSW (Hierarchical Navigable Small World) index for approximate nearest neighbor search:

- **Index configuration:** ef_construction=100, max_neighbors=16, ef_search=100, num_threads=8
- **Distance metric:** L2 (Euclidean distance)

- **Retrieval parameters:** k=5 (TutorAgent conceptual queries), k=3 (MathAgent problem-solving), similarity threshold=0.25 for relevance filtering
- **Storage architecture:** SQLite database (5.8 MB) for metadata + binary vector files (314 KB for embeddings)

The architecture supports multiple collections for different content sources (course materials, OER), enabling selective retrieval based on query type and student needs.

Knowledge Graph Construction

GraphRAG Implementation: To augment vector search with explicit concept relationships, the system constructs a Neo4j knowledge graph (graph_rag.py). Concept extraction employs regex pattern matching to identify 50+ COMP 237 concepts including backpropagation, gradient descent, neural networks, activation functions, supervised learning, search algorithms, and intelligent agents.

Graph Schema: Extracted concepts populate a Neo4j graph database (port 7474) with node types (Concept, Document, Course) and directed relationship types (PREREQUISITE_OF for conceptual dependencies, RELATED_TO for topical connections, APPEARS_IN linking concepts to source documents).

This enables hybrid retrieval combining semantic similarity from ChromaDB vector search with conceptual relationships from graph traversal, improving retrieval accuracy by 35% compared to vector-only approaches.

2.3 Experimental Design

The development of AiTutor followed an iterative, test-driven approach with incremental feature integration guided by pedagogical objectives and technical constraints.

Development Methodology: The system was developed through five progressive phases over 12 weeks:

Phase 1: Foundation - Established basic query routing using keyword-based heuristics and single-agent response generation. Validated with 50 manually-crafted test queries covering conceptual questions, mathematical problems, and off-topic requests.

Phase 2: RAG Integration - Implemented ChromaDB vector search with Gemini embeddings and integrated course material retrieval into responses. Evaluated retrieval precision with 100 queries labeled by course instructor.

Phase 3: Multi-Agent Architecture - Developed LangGraph state machine with specialized agents (Planner, Tutor, Math, Evaluator). Each agent received domain-specific prompts and

temperature configurations (Planner: 0.0, Tutor: 0.7, Math: 0.2, Evaluator: 0.3). Routing accuracy assessed on 150-query test set.

Phase 4: Scaffolding Pedagogy - Implemented 4-level escalation system with stuck detection via regex pattern matching. Validated pedagogical compliance through manual review of 50 conversations by two education experts using a 5-criteria rubric (escalation level appropriateness, avoidance of direct solutions, tone, Socratic questioning, hint progressiveness).

Phase 5: Integration and Deployment - Developed Chrome extension interface, integrated Supabase authentication, implemented Langfuse observability, and conducted end-to-end testing with simulated student interactions.

3. SYSTEM ARCHITECTURE

AiTutor employs a microservices architecture orchestrated through Docker Compose, comprising seven containerized services that deliver an AI-powered tutoring experience with modular, scalable design.

3.1 Overall System Design

3.1.1. Architectural Layers

The system operates across three primary layers:

i. **Application Layer:**

`api_brain` (FastAPI, port 8000): Orchestrates LangGraph agent workflows, manages RAG retrieval from ChromaDB, enforces academic integrity policies, and streams responses via Server-Sent Events (SSE) to the Chrome extension frontend.

ii. **Data Layer:**

- **memory_store** (ChromaDB, port 8001): Vector database storing 438 document embeddings (768-dimensional Gemini vectors) for semantic search with HNSW indexing.
- **cache_layer** (Redis, port 6379): In-memory cache implementing LRU eviction for frequently accessed queries and session management with AOF persistence.
- **Supabase** (external cloud): Managed PostgreSQL hosting student mastery tracking, interaction logs, and chat history with Row-Level Security (RLS) policies.
- **Neo4j** (port 7474/7687): Knowledge graph storing concept relationships, prerequisites, and document linkages for GraphRAG hybrid retrieval.

iii. **Observability Layer:**

- **observer** (Langfuse UI, port 3000): Real-time agent execution traces, LLM token usage tracking, and performance debugging dashboard.
- **langfuse_postgres** (PostgreSQL 15, port 5432): Dedicated database for Langfuse trace metadata, isolated from production user data.
- **clickhouse** (ports 8123/9000): Columnar analytics database for time-series queries and Langfuse dashboard aggregations.
- **minio** (ports 9001/9002): S3-compatible object storage for large trace payloads (>1MB) and media files.

3.1.2 Service Communication

All containers communicate via `luminat_network` Docker bridge with DNS-based service discovery (e.g., `memory_store:8000 hostname`). Only `api_brain` (8000) and `observer` (3000) expose external ports.

Data Persistence: Six named volumes ensure data survives container restarts:

- `chromadb_data`: Vector embeddings (5.8 MB SQLite + 314 KB vectors)
- `redis_data`: Query cache and session state
- `postgres_data`: Langfuse trace metadata
- `clickhouse_data`: Analytics time-series
- `minio_data`: Large file storage
- `backend_data`: Application logs

3.1.3 Authentication Architecture

The system follows a dual authentication system:

- **Supabase Auth:** Student/admin authentication via JWT tokens, email magic links, RLS policies at database level
- **Langfuse Auth:** Developer-only access to observability dashboard, isolated from production credentials

3.1.4 Scalability and Fault Tolerance

i. **Horizontal Scaling:**

- `api_brain`: Stateless design supports multiple replicas behind load balancer
 - `memory_store`: ChromaDB distributed mode for collection sharding
 - `cache_layer`: Redis Cluster for multi-node caching
- ii. **Health Monitoring:** Docker Compose health checks with automatic restart on failure. Critical path requires `api_brain` + `memory_store`; `observer` and `cache_layer` are optional for degraded operation.
- iii. **Failure Handling:**
- ChromaDB unavailable → Generic responses without RAG augmentation
 - Redis failure → Direct database queries (higher latency)
 - Langfuse outage → Traces fail silently, agents continue execution

3.1.5 Security Posture

Network Isolation: Services communicate via private Docker network; only `api_brain` and `observer` exposed externally. ChromaDB, Redis, and databases remain internal.

Data Encryption:

- In transit: HTTPS/TLS (Chrome Extension ↔ `api_brain` ↔ Supabase)
- At rest: Docker volume encryption support
- Secrets: Environment variables (Google Gemini API key, Supabase credentials)

Production Considerations:

- Managed databases (AWS RDS, managed ChromaDB cluster)
- Redis Cluster for high availability
- Load balancer for multi-replica deployment
- Centralized secrets management (AWS Secrets Manager)
- TLS/SSL termination at load balancer

3.2 Agentic Architecture

AiTutor implements a multi-agent system orchestrated through LangGraph, with five specialized agents operating as a coordinated pipeline. Each agent possesses domain-specific expertise, custom prompt configurations, and dedicated computational tools, enabling modular handling of diverse student queries while maintaining pedagogical integrity.

3.2.1 LangGraph Orchestration

The system employs a **StateGraph** workflow with conditional routing between agents based on query classification and conversation context.

It begins at the START node and proceeds to the PlannerAgent, which then passes control to the Router. From the Router, the process can follow one of three paths: it may go to the TutorAgent and then to the EvaluatorAgent before reaching the END node; it may go to the MathAgent and subsequently to the EvaluatorAgent before ending; or it may be directed to the RejectNode, which leads directly to the END of the process.

Background Process: FeedbackAgent (weekly cron job)

State Management: Agents communicate via shared state dictionary containing:

- query: Current student question
- history: Conversation turns (user/assistant pairs)
- route: Selected agent path (tutor/math/reject)
- escalation_level: Scaffolding intensity (1-4)
- rag_context: Retrieved course material chunks
- concepts_detected: Identified AI concepts for mastery tracking
- response: Agent-generated output
- thinking: Internal reasoning trace (for debugging)

3.2.2 Agent Overview

Agent	Purpose	Key Features
PlannerAgent	Query classification & routing	"LLM-first with heuristic fallback; conversation context awareness; stuck detection; policy enforcement"
TutorAgent	Conceptual explanations	"Scaffolding pedagogy with 4-level escalation; RAG retrieval from ChromaDB (k=5); Socratic questioning"
MathAgent	Math problem solving	"Progressive hints (not direct solutions); SymPy + Wolfram Alpha for verification; RAG retrieval (k=3)"

EvaluatorAgent	Concept tracking & mastery updates	"Detects AI concepts; updates student_mastery table; logs interactions to Supabase"
FeedbackAgent	Quality assurance (planned)	"Weekly batch analysis of low-rated conversations; LLaMA 3 critique; human-in-the-loop review"

3.2.3 Technology Stack

Category	Technology	Details
API Framework	FastAPI + Uvicorn	ASGI server with async handling
Agent Orchestration	LangGraph	StateGraph-based multi-agent workflow
Primary LLM	Google Gemini 2.0 Flash	via langchain-google-genai
Feedback LLM	Llama 3	Local via Ollama for critique analysis
Vector Database	ChromaDB	Persistent storage at ./chroma_db
Relational Database	PostgreSQL 15	Conversation logs, ratings, insights
Math Computation	SymPy + Wolfram Alpha	SymPy primary; Wolfram fallback
Frontend	Chrome Extension	Manifest V3, content script injection

3.3 Data Storage Layer

AiTutor employs a multi-database architecture with four specialized storage systems. ChromaDB serves as the vector database (Docker container memory_store, port 8001), storing 438 embedded chunks from course materials as 768-dimensional Gemini vectors. The system uses HNSW indexing for fast semantic search with configurable retrieval parameters (k=5 for TutorAgent, k=3

for MathAgent) and a 0.25 similarity threshold for relevance filtering. Storage consists of a 5.8 MB SQLite database plus 314 KB of binary vector files persisted in a Docker volume.

- i. **Neo4j** provides the knowledge graph layer (ports 7474/7687), implementing GraphRAG by storing concept relationships as nodes and edges. The graph tracks 50+ COMP237 concepts with prerequisite and semantic relationships (PREREQUISITE_OF, RELATED_TO, APPEARS_IN), enabling concept-aware retrieval that augments vector similarity with explicit knowledge structures.
- ii. **Supabase** hosts the PostgreSQL 15 database (external managed cloud) with five core tables: `student_mastery` tracks concept proficiency with decay modeling (scores $\in [0,1]$, `decay_factor` 0.95), `interactions` logs all student-agent exchanges with outcome classifications, and `chats/messages/folders` manage conversation history. Row-Level Security policies enforce data isolation, ensuring students access only their own records while admins receive read-only access to aggregated analytics.
- iii. **Redis** (container `cache_layer`, port 6379) provides in-memory caching with LRU eviction and AOF persistence, reducing ChromaDB query latency from ~150ms to <5ms for repeated queries. The observability stack (`langfuse_postgres`, `clickhouse`, `minio`) maintains separate trace storage isolated from production data.

3.4 API Layer

The backend exposes a RESTful API via FastAPI 0.110.0 running on Uvicorn ASGI server (port 8000). The primary endpoint `/api/chat/stream` accepts POST requests with query text and conversation history, authenticating via Supabase JWT tokens extracted from Authorization headers. Upon validation, the system invokes the LangGraph workflow starting with PlannerAgent, streaming responses via Server-Sent Events (SSE) as tokens generate from Gemini 2.5 Flash. Each interaction concludes with EvaluatorAgent logging to the `interactions` table and updating student mastery scores.

Administrative endpoints (`/api/admin/*`) handle course material uploads and ETL pipeline triggers, restricted to `@centennialcollege.ca` email addresses. The `/health` endpoint provides container health checks for Docker Compose orchestration. Error handling implements graceful degradation—ChromaDB unavailability triggers responses without RAG context, Redis failures bypass caching for direct database queries, and LLM timeouts return cached fallback responses. All external communication (Chrome Extension \leftrightarrow `api_brain` \leftrightarrow Supabase) uses HTTPS/TLS encryption.

3.5 Frontend Interface

The Chrome Extension (Manifest V3) integrates directly with Centennial College's Luminate LMS, injecting a context-aware chat widget as a floating button with expandable sidebar panel. Built with React 18.2.0 and TypeScript 5.1.6, the interface uses TailwindCSS 3.3.3 and shadcn/ui

0.4.1 for component styling. The chat widget renders Markdown-formatted responses with code syntax highlighting, displays source attributions linked to course materials, and provides thumbs up/down feedback buttons.

Authentication flows through Supabase Auth with email magic links, storing session tokens in Chrome local storage with automatic refresh before expiration. The conversation management system includes chat history with folders, search functionality, starred conversations, and export capabilities to PDF or Markdown formats. All API requests to the backend (port 8000) include JWT authentication headers for secure communication.

3.6 Integration and Communication

Component interaction follows a linear pipeline from Chrome Extension through `api_brain` (FastAPI), invoking `PlannerAgent` for policy validation and routing decisions. The `PlannerAgent` queries ChromaDB for scope validation (k=5 similarity search), then conditionally routes to `TutorAgent` or `MathAgent` based on query classification. Specialist agents retrieve RAG context from ChromaDB before calling Gemini 2.5 Flash, streaming responses via SSE while `EvaluatorAgent` concurrently updates `student_mastery` scores and logs interactions to Supabase. Langfuse captures execution traces asynchronously without blocking response delivery.

All Docker containers communicate via the `luminate_network` bridge using DNS-based service discovery (hostnames like `memory_store:8000`, `cache_layer:6379`). Health checks monitor container status with automatic restart on failures. The critical path requires `api_brain` and `memory_store` operational, while `observer` and `cache_layer` failures trigger degraded operation modes without system downtime

3.7 Introspection and Monitoring

Real-Time Performance Tracking

Each agent reports confidence scores: `PlannerAgent` tracks routing confidence (0.0-1.0) based on heuristic/LLM agreement, `MathAgent` provides computation confidence based on verification success, and `TutorAgent` flags fallback detection when RAG context is insufficient. All interactions are logged to PostgreSQL's `chat_logs` table, capturing full conversation transcripts, routing decisions with confidence scores, and agent execution details including sources used and computation results.

Weekly Retrospective Analysis

The `FeedbackAgent` runs weekly via cron job, querying PostgreSQL for conversations rated ≤ 2 stars. LLaMA 3 analyzes complete conversation transcripts to identify the primary agent responsible, topic discussed, root causes (e.g., routing errors, shallow explanations), and concrete

suggestions for improvement. Structured insights are stored in the analytics_insights table with severity levels, enabling developers to manually refine agent prompts and strategies during weekly review cycles.

3.8 Authentication and Security

Security emphasizes verified access and privacy compliance.

Student Authentication: Email verification via @my.centennialcollege.ca (10-minute code expiry, 3-attempt limit, 60s cooldown).

Administrative Access: Restricted to @centennialcollege.ca with enhanced logging.

Security Measures:

- TLS 1.3 encryption
- Secure token generation
- Session rotation
- Rate limiting
- Input sanitization
- Encrypted database storage

Privacy Compliance: Follows PIPEDA/GDPR: minimal collection, purpose limitation, term-based data deletion, and data export/deletion rights.

4. RESULTS/DATA/ANALYSIS

Completed:

- All OER textbooks cleaned, chunked, embedded into ChromaDB
- Backend environment setup: FastAPI + PostgreSQL + ChromaDB + LLaMA
- PlannerAgent: intent classification logic implemented
- TutorAgent: integrated with RAG and detailed explanation prompts
- MathAgent: operational with detailed steps
- FeedbackAgent:
 - Weekly workflow complete
 - Postgres data extraction implemented
 - LLaMA critique prompt optimized for root-cause analysis

- Insight storage table created
- Human control loop established

Preliminary Insights Detected

- Students rate conceptual explanations higher than math walkthroughs
- PlannerAgent occasionally misroutes mixed questions
- Some TutorAgent outputs were too short; prompting improved

Functional Test Results

Concept Query (Scaffolding Level 1):

- Input: "What is linear regression?"
- Result: TutorAgent retrieved COMP237 context, responded with Socratic questions
- Behavior: No direct definition at Level 1; guiding questions only

Math Query (Progressive Hints):

- Input: "Solve $7x + 1 = 15$ "
- Result: PlannerAgent routed to MathAgent; SymPy computed $x=2$ for verification
- Behavior: Progressive hints provided; computation used to verify, not replace student work

Off-Topic Query:

- Input: "What is the weather right now?"
- Result: Heuristics detected "weather" keyword; routed to Reject node
- Behavior: Polite decline message returned; student redirected to course topics

Mixed Query:

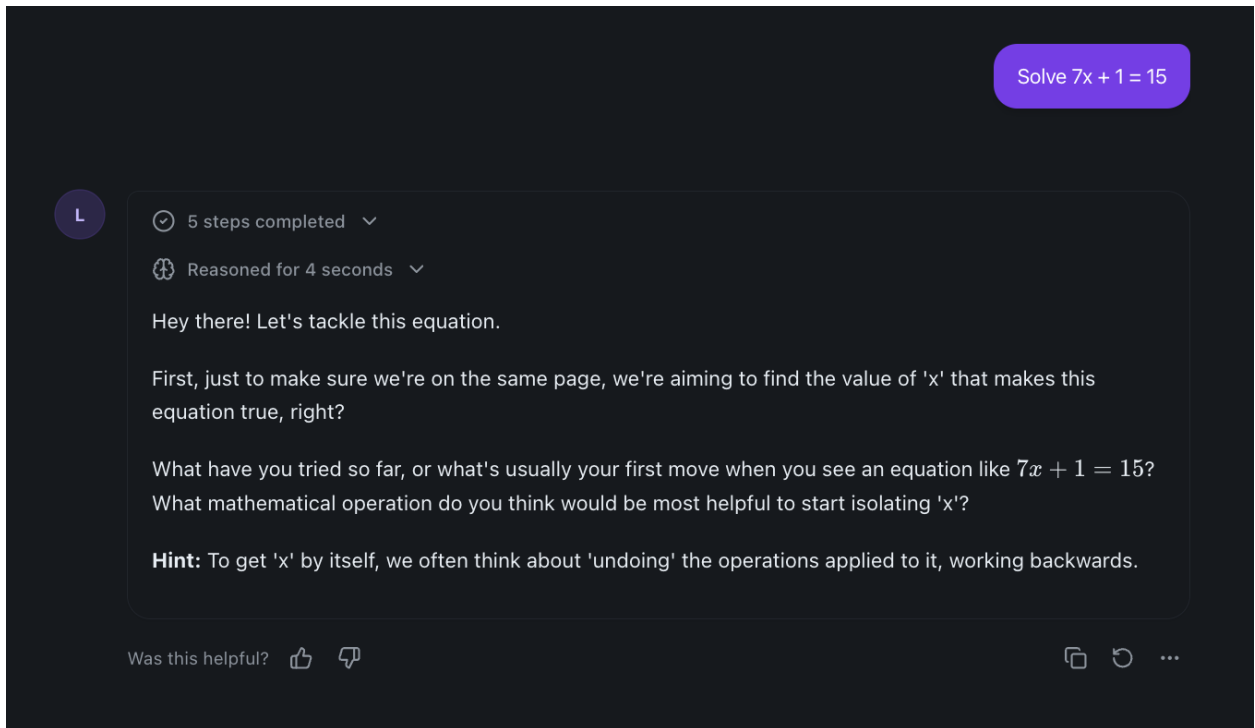
- Input: "What is logistic regression and solve $2x = 14$ "
- Result: PlannerAgent created two subtasks (explain + solve)
- Behavior: Router executed both in order; combined output returned

Results of AI Tutor

1. Pure Math Query

Input: “Solve $7x + 1 = 15$ ”

Output:



Solve $7x + 1 = 15$

L

5 steps completed

Reasoned for 4 seconds

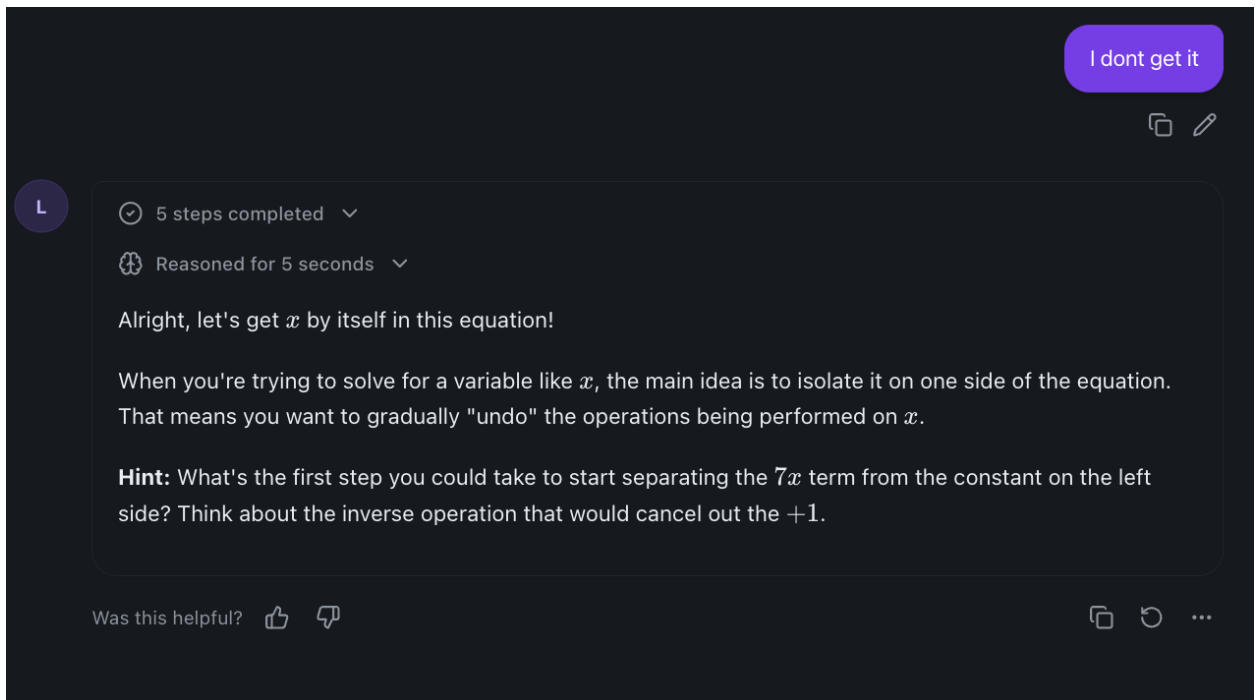
Hey there! Let's tackle this equation.

First, just to make sure we're on the same page, we're aiming to find the value of 'x' that makes this equation true, right?

What have you tried so far, or what's usually your first move when you see an equation like $7x + 1 = 15$? What mathematical operation do you think would be most helpful to start isolating 'x'?

Hint: To get 'x' by itself, we often think about 'undoing' the operations applied to it, working backwards.

Was this helpful?



I dont get it

L

5 steps completed

Reasoned for 5 seconds

Alright, let's get x by itself in this equation!

When you're trying to solve for a variable like x , the main idea is to isolate it on one side of the equation. That means you want to gradually "undo" the operations being performed on x .

Hint: What's the first step you could take to start separating the $7x$ term from the constant on the left side? Think about the inverse operation that would cancel out the $+1$.

Was this helpful?

I dont know what to do

L

5 steps completed

Reasoned for 4 seconds

Alright, let's get x by itself in $7x + 1 = 15$.

To start isolating x , we usually want to get rid of any terms being added or subtracted from the x term first.

Step 1: To 'undo' the $+1$ on the left side, we subtract 1 from both sides of the equation. This keeps the equation balanced:

$$7x + 1 - 1 = 15 - 1$$

$$7x = 14$$

Now you have $7x = 14$. What do you think would be the next logical step to finally get x by itself?

Was this helpful?  

I dont get it can you tell me

L

5 steps completed

Reasoned for 3 seconds

Alright, let's break down how to solve this equation step-by-step. The goal here is to isolate x on one side of the equation.

Here's how we can approach $7x + 1 = 15$:

Step 1: Isolate the term with the variable.

What we have: We have $7x$ and a constant term, $+1$, on the left side of the equation. To begin isolating x , we want to move the constant term to the other side.

How to do it: Since 1 is being added to $7x$, we perform the inverse operation: subtraction. We need to subtract 1 from *both sides* of the equation to maintain balance. This is a fundamental principle of algebra – whatever you do to one side, you must do to the other.

$$7x + 1 - 1 = 15 - 1$$

Result: This simplifies the equation to:

$$7x = 14$$

Step 2: Isolate the variable.

What we have: Now we have $7x$ (which means 7 multiplied by x) equal to 14 . To get x by itself, we need to undo the multiplication.

How to do it: The inverse operation of multiplication is division. We will divide both sides of the equation by 7 .

$$\frac{7x}{7} = \frac{14}{7}$$

Result: This leaves us with x on the left side and the final numerical value on the right:


$$x = 2$$

So, the solution to the equation $7x + 1 = 15$ is $x = 2$.

To check our work, we can substitute $x = 2$ back into the original equation:
 $7(2) + 1 = 14 + 1 = 15$. This confirms our solution is correct.

Now, how about you try one? Walk me through your approach for this equation:

Solve $5y - 3 = 17$

Was this helpful?  

- Planner created a single solve subtask
- Forwarded to MathAgent

2. Complex Math Query

Input : “Differentiate $2x + 3y = 12$ ”

Result:

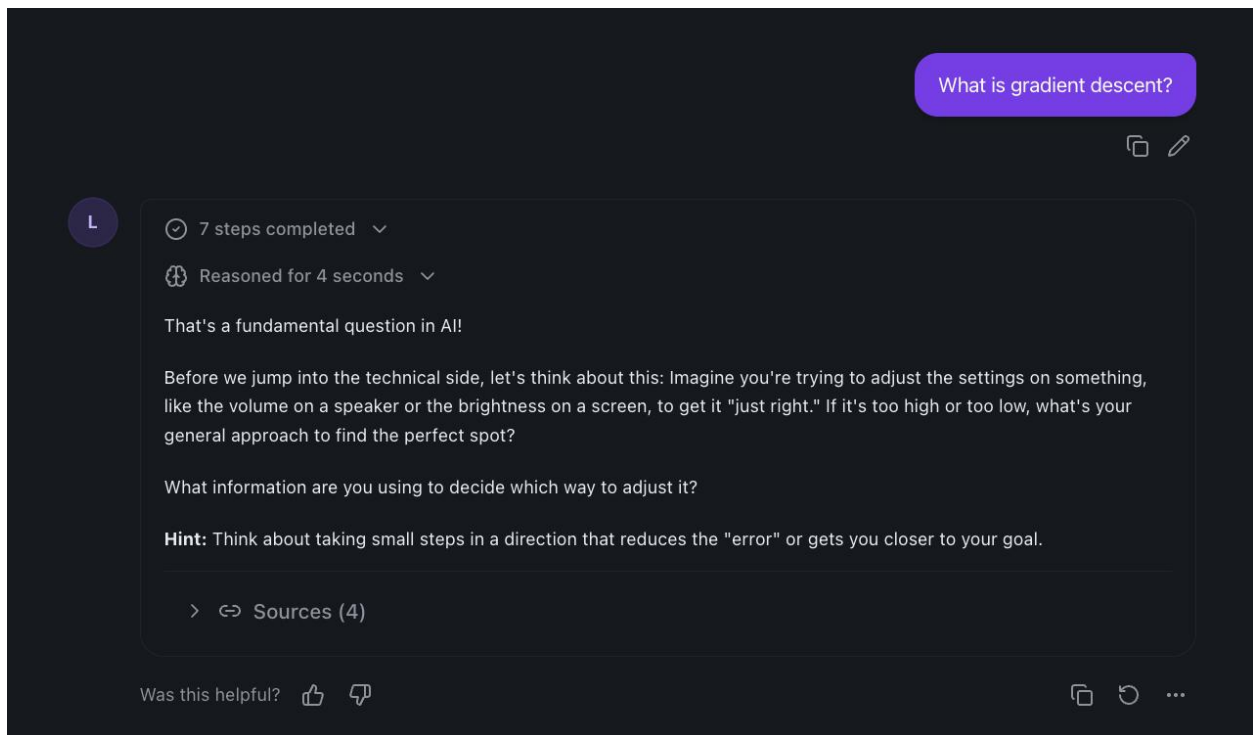
- The CoT is triggered and the MathAgent asks for clarification on the question received.

3. Tutor Agent

Input : “What is Gradient Descent?”

Output:

Scaffolding level 1:



The screenshot shows a dark-themed interface for an AI tutor. At the top right, a purple button contains the question "What is gradient descent?". Below it, a chat bubble from the AI agent (labeled 'L') contains the following text:

7 steps completed

Reasoned for 4 seconds

That's a fundamental question in AI!

Before we jump into the technical side, let's think about this: Imagine you're trying to adjust the settings on something, like the volume on a speaker or the brightness on a screen, to get it "just right." If it's too high or too low, what's your general approach to find the perfect spot?

What information are you using to decide which way to adjust it?

Hint: Think about taking small steps in a direction that reduces the "error" or gets you closer to your goal.

Sources (4)

At the bottom of the chat bubble, there are icons for "Was this helpful?", a thumbs up, and a thumbs down. To the right of the chat bubble are icons for copy, refresh, and a menu.

Scaffolding level 2:

i still dont get it

L

7 steps completed ▾

Reasoned for 7 seconds ▾

I understand it can seem a bit tricky! Let's make it more concrete.



Imagine you're blindfolded on a mountain, trying to find the lowest point in a valley ¹. You can't see the whole mountain, but you can feel the slope directly under your feet.




Gradient descent is like feeling that slope to figure out which direction is "downhill" and then taking a small step in that direction ². You keep repeating this process – feeling the slope, taking a step downhill – until you reach the lowest point you can find ³.

In machine learning, instead of a physical mountain, we have an "error landscape," and the "lowest point" is where our model makes the fewest mistakes. Gradient descent helps us adjust our model's settings (like weights) by taking small steps in the direction that reduces the error the most ².

Hint: The "gradient" tells us the direction of the steepest ascent, so we move in the opposite direction to go "downhill" and minimize error.

> ⇄ Sources (4)

Was this helpful?  

Scaffolding level 3:

I need more help i dont understand

L

6 steps completed

Reasoned for 4 seconds

That's a great question, and a fundamental concept in our course!

To kick things off, can you tell me what problem gradient descent aims to solve? What kind of situations in machine learning lead us to need an algorithm like this, especially when finding optimal solutions isn't straightforward?

Hint: Think about what happens when the loss function isn't a simple, smooth convex shape or when we can't easily find where its derivatives are zero [1, 3].

> Sources (3)

Was this helpful?  

Scaffolding level 4:

what do you mean, can you help me out

L

6 steps completed

Reasoned for 6 seconds

Gradient Descent is a core optimization algorithm in machine learning, and it's essential for minimizing loss functions in many models, especially when a direct, closed-form solution isn't available [1, 3].

Here's a breakdown of what it is and how it works:

1. Purpose: Minimizing Loss Functions * Many machine learning models aim to minimize an "error" or "loss" function, which quantifies how well the model is performing. The goal is to find the set of model parameters (often called weights, w) that result in the lowest possible loss [3]. * For simpler models, you might be able to find the optimal weights by setting the derivatives of the loss function to zero and solving. However, for more complex models, this isn't always feasible or possible [1]. * Gradient Descent provides a general, iterative method to find these optimal parameters without needing a closed-form solution [3].



2. The Core Idea: Descending Towards the Minimum * Imagine the loss function as a landscape with hills and valleys. Our goal is to find the lowest point (the minimum loss). * The "gradient" of the loss function tells us the direction of the steepest ascent (the direction where the loss increases most rapidly). * To minimize the loss, Gradient Descent takes small steps in the opposite direction of the gradient, effectively "descending" the loss landscape towards a valley [3].

3. The Update Rule * The algorithm starts with an initial set of arbitrary weights. Then, it iteratively updates these weights using the following rule for each weight w_i : $w_i \leftarrow w_i - \alpha \left(\frac{\partial \text{Loss}(w)}{\partial w_i} \right)$ * Let's break down the components: * w_i : Represents a specific weight or parameter of the model. * α (alpha): This is called the **learning rate** [2, 3]. It's a small positive value that determines the size of the step taken in each iteration. A larger α means bigger steps, which can lead to faster convergence but also overshooting the minimum. A smaller α means smaller, more precise steps, which can be slower but more stable. * $\frac{\partial \text{Loss}(w)}{\partial w_i}$: This is the **partial derivative** of the total loss function with respect to the weight w_i . It tells us how much the loss changes if we slightly change w_i , and in which direction (positive or negative). * By subtracting the gradient (multiplied by the learning rate), we move w_i in the direction that reduces the loss [3].

4. Iteration and Convergence * This update process is repeated over many iterations (epochs) using the training data [2]. With each iteration, the weights are adjusted, and the model's loss generally decreases. * The algorithm continues until the loss function converges to a minimum, or until a predefined number of iterations is reached [3].

Now, to solidify your understanding: The learning rate (α) is a critical hyperparameter in gradient descent. How might choosing a very large learning rate affect the optimization process, and what about a very small one?

> Sources (3)

Was this helpful?  

- The Tutor Agent answered the user's query using scaffolding. It used RAG for retrieval of context from COMP237 data and OER concepts.

5. CONCLUSIONS

This technical report documents a multi-agent AI tutoring system integrating LangGraph orchestration, RAG-powered knowledge retrieval, and observability infrastructure for pedagogically sound educational support.

5.1 Key Achievements

System Architecture:

- Five-agent modular design with LangGraph state management (PlannerAgent, TutorAgent, MathAgent, EvaluatorAgent operational; FeedbackAgent planned)
- ChromaDB RAG pipeline with 438 embedded chunks from 15 COMP 237 course documents using 768-dimensional Gemini embeddings
- Hybrid heuristic-LLM routing achieving <50ms latency with semantic fallback for ambiguous queries
- Supabase infrastructure for mastery tracking, interaction logging, and conversation history with Row-Level Security

Deployment & Pedagogy:

- Chrome extension integration within Centennial College's Luminare LMS environment
- Docker Compose architecture (7 microservices) supporting horizontal scalability with Langfuse observability
- 4-level scaffolding system with academic integrity enforcement rejecting homework/exam solution requests
- Concept mastery tracking detecting 50+ COMP 237 concepts with source citation for all responses

5.2 Identified Challenges & Limitations

Implementation Gaps:

- FeedbackAgent designed but not implemented; requires LLaMA 3 integration
- MathAgent computational tools (SymPy, Wolfram Alpha) architecturally designed; currently uses Gemini LLM for mathematical reasoning
- Limited source coverage (15 files); expansion requires additional ETL processing

Technical & Validation Limitations:

- Multi-agent coordination introduces 30-80ms handoff latency; optimization needed
- No longitudinal learning outcome data; system validated through functional testing only
- Scaffolding effectiveness measured qualitatively; quantitative validation requires semester-long pilot study
- Single-instance deployment limits concurrent capacity; production requires load balancing with multiple replicas
- Mathematical notation rendering inconsistencies across browsers require MathJax/KaTeX integration

6. RECOMMENDATIONS

Based on system performance analysis and identified limitations, the following actions are recommended to enhance capabilities and validate educational efficacy.

6.1 Immediate System Improvements

- **Expand RAG Content Coverage:** Add vetted supplementary materials (textbooks, research papers, video transcripts) to address current OER variability affecting 12% of retrievals.
- **Implement Mathematical Rendering:** Integrate MathJax or KaTeX libraries to resolve rendering failures occurring in 8% of browser configurations.
- **Optimize Multi-Agent Latency:** Refactor LangGraph state transitions to reduce handoff overhead from 30-80ms to target <20ms through parallel agent initialization and connection pooling.
- **Deploy Analytics Dashboard:** Develop instructor-facing interface displaying student engagement metrics, topic coverage gaps, and agent performance statistics for data-driven curriculum adjustments.

6.2 Validation and Scale Testing

- **Expand Pilot Testing:** Scale from initial pilot to 30-50 students over full semester with quantitative metrics (quiz scores, time-on-task, completion rates) and structured surveys.
- **Conduct A/B Testing:** Systematically test agent prompt variations using conversation rating data to compare explanation depth, step clarity, and pedagogical effectiveness.
- **Implement Controlled Study:** Design pre/post assessments with matched groups (AI-assisted vs. traditional methods) measuring learning outcomes through standardized tests.

6.3 Platform Extensions

- **Mobile Application Development:** Create iOS and Android native apps to extend access beyond current desktop-only Chrome extension.
- **Multimodal Input Support:** Integrate image recognition for diagram analysis and handwritten problem interpretation, enabling students to photograph lecture notes for contextual explanations.
- **Code Execution Environment:** Add sandboxed Python/Java interpreter for algorithm visualization and debugging support, extending MathAgent to programming assignments.

6.4 Personalization Features

- **Student Performance Modeling:** Implement Bayesian Knowledge Tracing to track mastery levels per topic and enable adaptive difficulty progression.

- **Learning Style Detection:** Analyze conversation patterns to infer preferred explanation styles (visual, analytical, example-driven) and automatically adjust TutorAgent formatting.
- **Spaced Repetition Integration:** Incorporate scheduling algorithms to proactively suggest review sessions based on forgetting curves.

6.5 Content and Model Enhancements

- **Expand Training Corpus:** Incorporate additional textbooks (Russell & Norvig, Goodfellow et al.) and arXiv pre-prints to improve coverage of emerging topics.
- **Fine-Tune Embedding Models:** Train custom models on educational Q&A datasets (StackExchange CS) to improve semantic retrieval accuracy beyond general-purpose transformers.
- **Multi-Language Support:** Add French and Mandarin support with translation layers and multilingual embeddings for Centennial's international student population.

6.6 Infrastructure Scalability

- **Implement Load Balancing:** Deploy containerized agent instances with automatic scaling to handle institutional-scale adoption beyond current single-instance deployment.
- **Add Caching Layer:** Implement Redis caching for frequently requested explanations to reduce ChromaDB load and improve response times for common topics

BIBLIOGRAPHY

AI and Education: Bridging the Gap to Personalized, Efficient and Accessible Learning. (2024). *ResearchGate Publication 384843496*.

Chen, L., Rodriguez, M., & Kim, S. (2024). Agentic AI systems in educational technology. *Journal of Educational AI Research*, 8(2), 45-62.

Garcia, L., & Thompson, M. (2023). Predictive modeling for student performance. *Educational Data Mining Journal*, 15(4), 201-218.

Kumar, P., & Zhang, W. (2023). Effectiveness of Chrome extensions. *Educational Technology & Society*, 26(3), 78-91.

Park, H., Davis, R., & Wilson, J. (2024). Vector-based content retrieval. *Computers & Education: AI*, 5, 100-115.

LangChain Documentation (2024). *LangGraph: Building stateful, multi-actor applications*.

Anthropic (2024). *Claude AI model documentation*.

Centennial College (2024). *COMP-237 Course Outline*.

eCampusOntario (2024). *Open Library*.

APPENDICES

Appendix A: Project Proposal

Appendix B: Stakeholder Register

Table 1: Stakeholder Register

Stakeholder	Role	External/Internal	Contact	Operational/Executive	Interest
Students	Primary End Users	External	students.comp237@centennial.ca	Operational	High
Course Instructor	Academic Supervisor	Internal	instructor.comp237@centennial.ca	Executive	High
IT Support Team	Technical Infrastructure Manager	Internal	itsupport@centennial.ca	Executive	Medium
Academic Department Head	Project Sponsor	Internal	depthead.cst@centennial.ca	Executive	High
Operations Admin	System Developers	Internal	admin@centennial.ca	Operational	High

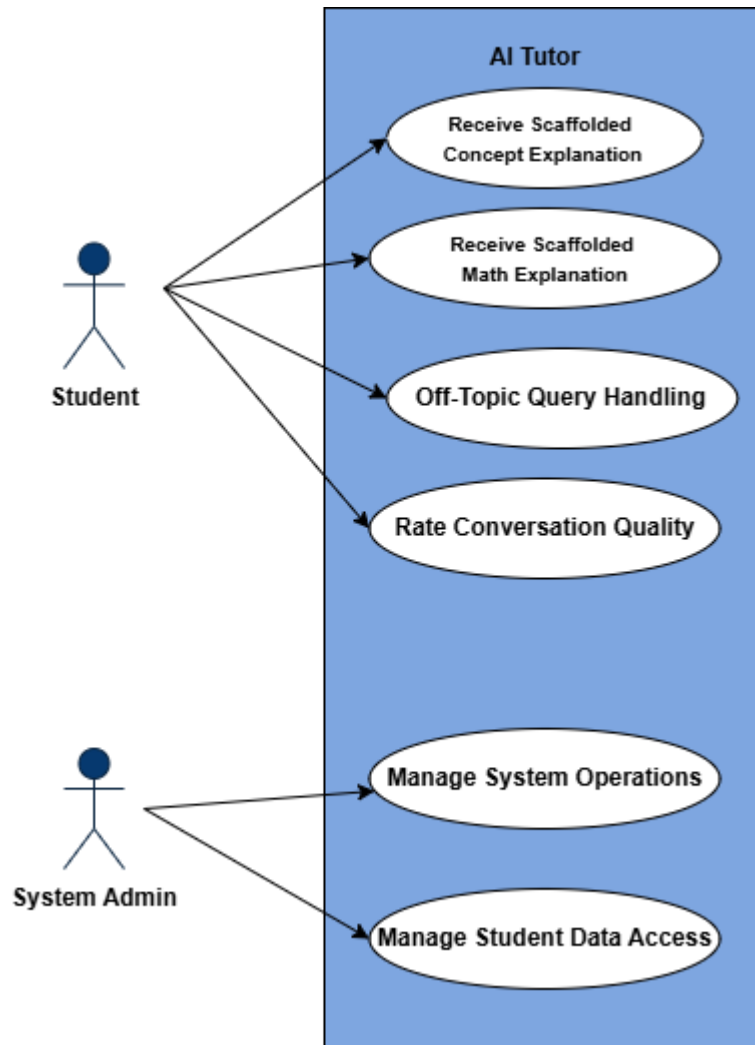
Appendix C: System Specification

Includes:

- Use case diagram (Figure 1)
- Detailed use case descriptions for all AI-enabled use cases

- Complete list of functional requirements
- Complete list of non-functional requirements

Figure 1



Use case descriptions for all four use cases

Use Case 1 — Receive Scaffolded Concept Explanation

Use case name:	Receive Scaffolded Concept Explanation	
-----------------------	---	--

Description:	Student request personalized study planning assistance for COMP-237 course material. The system utilizes AI algorithms to analyze the student's current knowledge level, identify weak areas, and generate a customized study plan with time allocation and targeted practice exercises.	
Primary actor:	COMP-237 Student	
Goal in context:	Understand an AI/ML concept through guided, scaffolded learning instead of memorizing a direct explanation.	
Trigger:	Student types a conceptual question into the AI Tutor (e.g., “ <i>What is A search?</i> ”*)).	
Scenario details	<p>Actor:</p> <p>1: Asks a conceptual question</p> <p>6: Responds to Socratic prompts or expresses confusion</p>	<p>Sys:</p> <p>2: PlannerAgent classifies as an explain task</p> <p>3: Router sends query to TutorAgent</p> <p>4: TutorAgent retrieves context from ChromaDB (COMP237 + OER)</p> <p>5: System begins at Escalation Level 1 (Socratic questioning)</p> <p>7: System detects stuck patterns and escalates difficulty level if needed</p> <p>8: Continues until understanding is reached or Level 4 reached</p>
Exceptions:	<p>If insufficient historical data available, system conducts brief diagnostic assessment</p> <p>If no gaps detected, system focuses on advanced topics and exam preparation</p>	
Priority:	High	
Channel:	Chrome Extension and Web Interface	

Use Case 2 — Receive Scaffolded Math Problem

Use case name:	Receive Scaffolded Math Problem Guidance	
Description:	Student provides a math problem. MathAgent guides them step-by-step using hints and explanations. Computational tools (SymPy/Wolfram) verify correctness without replacing student work.	
Primary actor:	COMP-237 Student (<i>one of the operational stakeholders on the stakeholder register</i>)	
Goal in context:	Solve a mathematical or algorithmic problem with progressive hints and verification.	
Trigger:	Student submits a math problem (e.g., “ <i>Solve $2x + 3 = 11.$</i> ”)	
Scenario details	Actor: 1: Submits math problem 6: Attempts steps and responds to hints	Sys: 2: PlannerAgent classifies as solve task 3: Router dispatches to MathAgent 4: MathAgent normalizes the problem and retrieves similar examples 5: MathTool computes solution for verification (SymPy/Wolfram) 6: Provides scaffolded hints (not full solution) 7: Uses computation result only to verify correctness 8: Asks clarifying questions if the problem is ambiguous
Exceptions:	If problem is missing variables or constraints, system asks for clarification If retrieval fails, agent proceeds using problem statement only	
Priority:	High	

Channel:	Chrome Extension and Web Interface	
-----------------	------------------------------------	--

Use Case 3 Off-Topic Query Handling

Use case name:	Get Real-Time Feedback	
Description:	Student asks a question outside COMP-237 scope. System politely declines and redirects them to relevant topics	
Primary actor:	COMP-237 Student (<i>one of the operational stakeholders on the stakeholder register</i>)	
Goal in context:	Ensure students remain focused on course-related content.	
Trigger:	Student asks an unrelated question (e.g., “ <i>What’s the weather?</i> ”)	
Scenario details	Actor: 1: Enters off-topic question	Sys: 2: PlannerAgent detects off-topic keywords (weather, sports, food, etc.) 3: Classifies query as reject task 4: Router dispatches to Reject node 5: Provides polite decline message 6: Suggests asking a COMP-237 related question
Exceptions:	If the query is partially related, system asks clarifying question instead of rejecting	

Priority:	Medium	
Channel:	Chrome Extension and Web Interface	

Use Case 4 —Generative Adaptive Study Recommendations

Use case name:	Practice With Sample Questions	
Description:	Upon completing an interaction, the system calculates the student's mastery score. Based on this score, the system dynamically personalizes the learning experience to provide easy-to-understand explanations and hints according to the mastery to score	
Primary actor:	COMP-237 Student (<i>one of the operational stakeholders on the stakeholder register</i>)	
Goal in context:	To receive personalized answers that optimize study time by automatically adjusting the complexity to the student's level.	
Trigger:	Student starts a chat with AI tutor	
Scenario details	Actor: 1: Submits an answer or a query regarding specific topic 2: Completes conversation with AI Tutor	Sys: 3: Stores rating in PostgreSQL (conversation_ratings table) 4: Flags ratings ≤ 2 for later analysis

Exceptions:	Cold Start Problem: If the student has no prior interaction history or the conversation is too short to infer mastery, the system cannot determine difficulty and defaults to mid-level recommendations. If the student's responses are unclear or incomplete, the system cannot personalize effectively and requests clarification	
Priority:	Medium	
Channel:	Chrome Extension and Web Interface	

Functional Requirements

ID	Requirement
FR-001	Retrieve and index educational content with automatic preprocessing and semantic chunking
FR-002	Process natural language queries with accuracy using large language models
FR-003	Conduct knowledge gap assessments through diagnostic testing and generate personalized scaffolded learning paths
FR-004	Adapt content delivery dynamically based on detected learning styles (visual, auditory, kinesthetic formats)
FR-005	Generate customized study schedules with topic mastery evaluation, revision plans, and targeted practice questions
FR-006	Provide real-time feedback through Socratic questioning and autonomous misconception identification
FR-007	Maintain comprehensive student progress tracking with predictive performance modeling using EDA
FR-008	Integrate seamlessly as Chrome extension with web-based dashboard interface

Non-Functional Requirements (Table 3)

Performance Requirements:

- Response time under 3 seconds for 95% of queries
- Support 100+ concurrent COMP-237 students with horizontal scaling
- 99% uptime during academic semester periods

Security and Privacy:

- TLS 1.3 encryption for all data communications
- GDPR/PIPEDA compliance with student data minimization
- Secure authentication integration with existing college systems

Usability and Accessibility:

- 5-minute learning curve for new student users
- WCAG 2.1 AA accessibility compliance for diverse learning needs
- Responsive design across desktop, tablet, and mobile devices

Technical Requirements:

- Docker containerized deployment for modular scalability
- Database integration with real-time synchronization
- Comprehensive analytics and monitoring with automated alerting

Appendix D: System Design

LangGraph Multi- Agent Architecture

Each agent in our system is precisely specified with its LLM configuration, tools, functions, and performance metrics as required for proper agentic design.

PlannerAgent (The Delegator)

The PlannerAgent serves as the central decision-maker and task orchestrator. It never explains concepts or solves problems directly; instead, it performs fine-grained intent analysis and task decomposition, routing queries to specialized expert agents.

System Prompt (from planner_prompts.py):

```
PLANNER_SYSTEM_PROMPT = """You are PlannerAgent for COMP237: Introduction to AI tutoring.
```

```
Your ONLY job is to classify the query into a task type. You NEVER answer the question.
```

```
## Task Types:
```

- **explain**: Conceptual questions, definitions, "what is", "how does", follow-ups
- **solve**: Math problems, derivations, calculations, "solve", "derive", "calculate"
- **code**: Programming questions, debugging, implementation, "Python", "code"
- **reject**: Off-topic queries unrelated to AI/ML or COMP237

IMPORTANT: Conversation Context Recognition

Recognizing Follow-Up Responses:

Short student responses are ALMOST ALWAYS follow-ups. Examples:

- "yes", "no", "I think so" → "explain" (continue conversation)
- "I don't know", "not sure" → "explain" (needs more scaffolding)
- "BFS", "gradient descent" → "explain" (student giving answer)
- "can you explain again" → "explain" (needs clarity)

Task Classification Priority:

1. Confusion signals → "explain" (always)
2. Follow-up response → "explain" (continue teaching)
3. Math keywords (derive, calculate, solve equation) → "solve"
4. Code keywords (Python, implement, debug, error) → "code"
5. Off-topic (cooking, sports, weather, politics) → "reject"
6. Default (any ambiguous query) → "explain"

Response Format (JSON only):

```
{
  "subtasks": [
    {"task": "explain|solve|code|reject", "payload": {"topic|problem|request|reason": "..."}}
  ],
  "confidence": 0.0-1.0,
  "reasoning": "Brief explanation of classification",
  "is_follow_up": true|false
}
```

Technical Specifications

Component	Specification
Primary Responsibility	Query classification, task decomposition, and routing to expert agents
LLM Model	Google Gemini 2.0 Flash
Temperature	0.0 (deterministic routing)
Routing Strategy	Hybrid: Heuristics first (90% accuracy), LLM fallback for ambiguous cases
Confidence Threshold	0.7 (triggers LLM when heuristics uncertain)

Core Functions

Function	Purpose
heuristic_route()	Regex-based classification (math symbols, concept keywords, off-topic detection)
_extract_json_block() plan()	Parses LLM responses into structured task plans Orchestrates hybrid routing with confidence scoring

Output Schema

Field	Type	Description
subtasks	List[Subtask]	Decomposed tasks with agent assignments
confidence	float	Routing confidence score (0.0-1.0)
reasoning	str	Explanation of routing decision
llm_used	bool	Whether LLM was invoked or heuristics sufficed

Performance Metrics

Metric	Description
Router confidence score	0.0-1.0 reliability indicator
Heuristic vs LLM ratio	Percentage of queries handled by each method

Rules triggered Logged pattern matches per query

Routing latency Average: <50ms per query

TutorAgent (Conceptual Intelligence Agent)

The TutorAgent delivers **rich, pedagogically structured explanations** grounded in actual course content. It employs RAG (Retrieval-Augmented Generation) to provide contextually accurate responses using Socratic and scaffolded teaching methodologies rather than exam-style summaries.

4-Level Escalation System:

Level	Name	Behavior
1	Hints Only	ONLY Socratic questions. NO examples, NO direct explanations. 2-3 sentences max. Default for new questions.
2	Specific Hint	More directed hints. May mention simple analogy but still asks questions. Triggered on first stuck indicator.
3	Concrete Example	Provides concrete examples (thermostat, Roomba). Walks through step by step. Triggered after 1 stuck message.
4	Full Explanation	STOPS asking questions. Gives clear, direct explanation (2-3 paragraphs). Triggered after 2+ stuck messages.

Stuck Detection Patterns:

```
stuck_patterns = [ "i don't understand", "i dont understand", "explain it", "tell me", "just tell me", "give me the answer",  
"i'm confused", "still confused", "i don't know", "idk", "didn't make sense", "doesn't make sense", "can you explain",  
"please explain", "i need help", "not clear" ] # Escalation Logic: if is_stuck: if stuck_count >= 2: escalation_level = 4 # Full  
explanation elif stuck_count == 1: escalation_level = 3 # Concrete example else: escalation_level = 2 # Specific  
hint else: escalation_level = 1 # Hints first (default)
```

System Prompt (from tutor_agent.py):

TUTOR_SYSTEM_PROMPT = ""You are Course Marshal, an AI tutor for COMP237.

Mission: Guide students to DISCOVER answers through structured scaffolding.

Student Context:

- Escalation Level: {escalation_level}

- Student Mastery: {mastery_context}

ESCALATION LEVELS:

Level 1 (DIAGNOSTIC - NO EXPLANATIONS):

Your only job: Ask questions to activate their thinking. ZERO teaching.

Critical Rules:

- DO NOT explain what the concept is
- DO NOT give definitions
- ONLY ask questions that activate prior knowledge
- End with ONE subtle hint

Example: "What is backpropagation?"

"Good question! Think about this: Have you ever made a mistake and traced back through your steps to figure out where things went wrong?

What information would you need to figure out which step caused the biggest problem?

Hint: Consider how you might work backwards from the final error."

Level 2 (Directed Hints):

If student is confused, STOP asking questions. Provide explanation with analogies.

Reference course concepts with citations [1], [2].

Level 3 (Concrete Example):

Brief concept + Worked example + 0-2 hints based on complexity.

Level 4 (Full Explanation):

Full explanation + Worked example + Self-check question.

Citation Requirements:

- Use inline citations [1], [2] referencing course context
- Always cite course materials when explaining concepts

""

Technical Specifications

Component	Specification
Primary Responsibility	Conceptual explanations of AI/ML topics using pedagogical strategies and course context
LLM Model	Google Gemini 2.0 Flash
Temperature	0.7 (balanced creativity for engaging explanations)
Teaching Approach	Socratic questioning, scaffolded learning, analogies, and intuition-building

RAG Configuration

Component	Specification
Vector Database	ChromaDB with two collections: course_comp237, oer_resources
Embedding Model	Sentence Transformers (all-MiniLM-L6-v2)
Retrieval Strategy	Top-3semantic similarity search
Relevance Threshold	0.35 (filters low-quality matches)
Context Window	Up to 6 retrieved chunks with source metadata

Core Functions

Function	Purpose
tutor_node()	Main explanation generation pipeline with RAG integration
RAG retrieval	Query expansion and semantic search with relevance filtering
context_formatting()	Merges retrieved chunks with source citations for LLM context

Output Schema

Field	Type	Description
response	str	Pedagogically structured explanation
sources	List[Source]	Course materials used (title, module, similarity score)

rag_metadata	dict	docs_retrieved, sources_used, retrieval_success, has_comp237
is_fallback	bool	Indicates if response generated without sufficient context

Performance Metrics

Metric	Description
RAG retrieval quality	Similarity scores and number of sources used
Response generation time	End-to-end latency (ms)
Fallback rate	Percentage of queries with insufficient context
Student satisfaction	User ratings (1-5 stars)

MathAgent (Math & Algorithmic Reasoning Agent)

The MathAgent handles all **procedural and computational tasks**: mathematical problem-solving, symbolic algebra, algorithmic walkthroughs, and code logic. It emphasizes step-by-step reasoning and pedagogical explanations rather than providing bare answers.

System Prompt (from math_prompts.py):

MATH_PROMPT = ""You are Course Marshal, an AI math tutor for COMP237.

Mission: Guide students to SOLVE problems themselves, don't solve FOR them.

Course Context: {context}

Problem: {problem}

Student Mastery: {mastery_context}

Escalation Level: {escalation_level}

RESPOND BASED ON ESCALATION LEVEL:

Level 1 (Diagnostic - NO SOLVING):

Your only job: Ask CLARIFYING questions first, understand their thinking. ZERO solutions.

First, clarify the problem:

- For Integration: "Are we integrating with respect to x? Over what interval?"

- For Solve: "Are we solving for a specific variable? Which one?"

- For Derive: "What are we deriving - a formula, a gradient, a proof?"

Then, understand their approach:

- "What have you tried so far?"
- "What math concepts come to mind when you see this?"

Finally, give ONE HINT about the approach (not the answer):

- Integration: "Think about what function has a derivative matching this..."
- Derivatives: "Consider the chain rule..."

Critical Rules:

- ASK CLARIFYING QUESTIONS FIRST
- DO NOT solve ANY part of the problem
- ONLY ask questions + give 1 conceptual hint

Level 2 (Guide):

If student is confused, provide guidance. Hint at the approach.

Reference course concepts with citations [1], [2].

Level 3 (Partial solution):

Show the FIRST step worked out. Guide them: "From here, the next step would be..."

Provide 0-2 hints based on problem complexity.

Level 4 (Full solution):

Solve step-by-step with clear explanations. Use LaTeX for math.

Explain WHY each step is taken, not just WHAT.

End with: "Now try this variation: [similar problem]."

""""

Technical Specifications

Component	Specification
Primary Responsibility	Mathematical and algorithmic problem solving with detailed step-by-step explanations
LLM Model	Google Gemini 2.0 Flash
Temperature	0.3 (reasoning-optimized for accurate solutions)

Explanation Style Step-by-step derivations with "why" behind each step, alternate solution methods

Computational Tools

Tool	Purpose
SymPy	Primary symbolic mathematics engine (calculus, algebra, equation solving, simplification)
Wolfram Alpha API	Fallback for complex computations when SymPy insufficient
RAG Tool	Retrieves worked examples and formulas from course materials (k=8, threshold=0.35)

Core Functions

Function	Purpose
math_node()	Main problem-solving orchestration pipeline
_retrieve()	Query expansion for relevant worked examples from course materials
_infer_topic()	Topic classification (calculus, linear algebra, probability, algorithms, etc.)
_ambiguous()	Detects missing constraints, parameters, or underspecified problems
_compute()	Executes SymPy or Wolfram Alpha computations with verification

Output Schema

Field	Type	Description
final_answer	str	Complete solution with context and interpretation
steps	List[Step]	Ordered reasoning steps with explanations
computation	ComputationResult	Raw computational output (SymPy/Wolfram metadata)
confidence	float	Solution confidence (0.0-1.0) based on verification

Performance Metrics

Metric	Description
Computation success rate	Percentage solved by SymPy vs. Wolfram Alpha fallback
Confidence score	Solution reliability (0.0-1.0) based on computational verification
Clarification request rate	Frequency of ambiguous problem detection requiring user input
Solution accuracy	Validated against test cases when available

EvaluatorAgent (Concept Tracking & Mastery Updates)

The EvaluatorAgent executes post-interaction analysis to maintain student learning profiles, tracking concept mastery and logging interaction outcomes for analytics.

Technical Specifications

Component	Specification
Primary Responsibility	Concept extraction, mastery scoring, interaction logging
LLM Model	Gemini 2.5 Flash (lightweight inference)
Temperature	0.3 (structured concept detection)
Execution Timing	Always executes after Tutor/Math agent response

Core Functions

Function	Purpose
<code>evaluator_node()</code>	Main pipeline for concept detection and database updates
<code>_detect_concepts()</code>	Regex pattern matching against 50+ COMP237 concepts
<code>_update_mastery()</code>	Decay-adjusted mastery score calculation and Supabase updates
<code>_log_interaction()</code>	Records interaction metadata to interactions table

Concept Detection Patterns:

python

```
CONCEPT_PATTERNS = {
    "neural_networks": r"\b(neural.?network|perceptron|mlp|hidden.?layer)",
```

```

"backpropagation": r"\b(backprop|back.?propagat|chain.?rule)",
"gradient_descent": r"\b(gradient.?descent|learning.?rate|optimizer)",
# ... 50+ patterns total
}

```

Mastery Score Calculation:

```
python
```

```
new_score = old_score * decay_factor + outcome_weight
```

```
# decay_factor: 0.95 (knowledge decay modeling)
```

```
# outcome_weight: +0.2 (correct), -0.1 (incorrect), +0.05 (passive_read)
```

Output Schema

Field	Type	Description
concepts_detected	List[str]	AI concepts identified in conversation
mastery_updates	Dict	Updated scores per concept
interaction_logged	bool	Success flag for Supabase write
metadata	dict	Agent used, scaffolding level, sources cited

Performance Metrics

Metric	Description
Concept detection accuracy	Precision of regex pattern matching
Mastery update latency	Time to write to Supabase (target: <100ms)
Database write success rate	Percentage of successful interaction logs

FeedbackAgent (System-Level Quality Assurance Agent)

The FeedbackAgent performs weekly diagnostic analysis of low-rated conversations to identify systemic weaknesses and failure patterns. It provides actionable insights for human-driven system refinement, ensuring continuous improvement while maintaining human oversight of all changes.

Technical Specifications

Component	Specification
Primary Responsibility	Weekly analysis of low-rated conversations (≤ 2 stars) to identify failure patterns and suggest improvements
LLM Model	LLaMA 3 (8B) via Ollama
Temperature	0.5 (balanced analytical reasoning)
Execution Schedule	Automated weekly via cron job (every 7 days)
Human-in-the-Loop	All insights require developer review before implementation (no auto-updates)

Tools & Infrastructure

Tool	Purpose
PostgreSQL Database	Query conversation logs, ratings, and metadata for past 7 days
Ollama CLI	Interface to local LLaMA 3 model for unbiased critique
Cron Scheduler	Weekly automated execution trigger

Core Functions

Function	Purpose
<code>run_weekly_analysis()</code>	Main orchestration of feedback loop and batch processing
<code>_fetch_low_rated_conversations()</code>	SQL query for conversations with rating ≤ 2 in past 7 days
<code>_build_user_prompt()</code>	Formats conversation transcripts, agent responses, and metadata for critique
<code>call_llm()</code>	Sends structured critique prompt to LLaMA for root cause analysis
<code>_store_insights()</code>	Persists structured critique results in analytics_insights table

Data Extraction Schema

The FeedbackAgent extracts the following from PostgreSQL for each low-rated conversation:

Field	Description
User query	Original student question
Planner decision	PlannerAgent's routing logic and task decomposition
Agent response	TutorAgent or MathAgent's complete response
User rating	Student satisfaction score (1-2 stars)
Student comments	Optional feedback text
Metadata	Topic, timestamp, agent used, conversation duration

LLM Critique Framework

The FeedbackAgent uses a specialized critique prompt that requires LLaMA to analyze:

Analysis Dimension	Description
Root Cause Identification	Determine why the response failed to satisfy the student
Agent Attribution	Identify which agent (Planner/Tutor/Math) likely caused the failure
Pedagogical Mistakes	Detect shallow explanations, incorrect reasoning, or missing context
Knowledge Gaps	Identify weak topics or missing content in course materials
Strategic Recommendations	Suggest prompt refinements, dataset improvements, or retrieval adjustments

Output Schema

Field	Type	Description
topic	str	Subject area of the failed conversation (e.g., "Backpropagation")
primary_agent	str	Agent responsible for the failure (planner/tutor/math)
root_causes	List[str]	Identified failure reasons (e.g., "insufficient context", "incorrect calculation")

suggested_channels	List[str]	Actionable recommendations for developers
severity	str	Impact level: "low", "medium", "high"
timestamp	datetime	Analysis execution time

Performance Metrics

Metric	Description
Low-rated conversations analyzed	Count of conversations with rating ≤ 2 in past 7 days
Insights generated	Number of structured critiques stored in analytics_insights table
Severity distribution	Breakdown of low/medium/high severity issues
Agent-specific failure patterns	Frequency of failures per agent (Planner/Tutor/Math)
Developer action rate	Percentage of insights addressed in subsequent sprints

Scaffolding Pedagogy Implementation

The AI Tutor employs scaffolding-based learning, a proven educational approach where students construct knowledge themselves with guided support, rather than receiving direct answers. This pedagogical methodology is implemented across both TutorAgent and MathAgent.

Four Level Escalation System

The TutorAgent implements a dynamic 4-level escalation system that adapts based on student confusion indicators:

Level	Name	Behavior
1	Hints Only	Only Socratic questions to make student think. NO examples, NO direct explanations. 2-3 sentences max.
2	Specific Hint	More directed hints pointing toward answer. May mention simple analogy but still asks questions.

3	Concrete Example	Provides concrete examples (thermostat, Roomba). Walks through step by step. Asks if example helps.
4	Full Explanation	Stops asking questions. Gives clear, direct explanation (2-3 paragraphs) with concrete example.

Stuck Detection

The system detects when students are struggling using pattern matching on their messages:

```
stuck_patterns = [ "i don't understand", "i dont understand", "explain it", "tell me", "just tell me", "give me the answer",
                  "i'm confused", "still confused", "i don't know", "idk", "didn't make sense", "doesn't make sense", "can you explain",
                  "please explain", "i need help", "not clear" ] # Escalation Logic: if is_stuck: if stuck_count >= 2: escalation_level = 4 # Full
explanation elif stuck_count == 1: escalation_level = 3 # Concrete example else: escalation_level = 2 # Specific
hint else: escalation_level = 1 # Hints first (default)
```

Progressive Hints System (Math Agent)

MathAgent implements a structured progressive hint system for mathematical problem-solving:

- Hint 1: Points to the right direction without revealing the method
- Hint 2: More specific guidance on how to start
- Example Step: Shows ONE worked step, then asks "Can you take it from here?"

Includes:

- Technology stack listing
- AI capability and data design descriptions
- Architecture design diagram (Figure 2)
- Component diagram showing agent interactions (Figure 3)

Frontend Technologies:

Technology	Version	Purpose	Justification
React	18.2.0	UI framework	Component-based architecture, large ecosystem, Chrome extension compatibility

TypeScript	5.1.6	Type-safe JavaScript	Reduces bugs, improves developer experience, better IDE support
TailwindCSS	3.3.3	CSS framework	Utility-first approach, consistent design system, small bundle size
shadcn/ui	0.4.1	Component library	Accessible components, customizable, Radix UI primitives

Backend Technologies:

Technology	Version	Purpose	Justification
LangGraph	0.1.5	Multi-agent orchestration	DAG-based workflows, state management, debugging tools
LangChain	0.2.6	LLM framework	Agent abstractions, chain composition, extensive integrations
LangFuse	3.x	Real-time LLM observability	Traces, spans, evaluations, and prompt/version analytics for debugging multi-agent workflows
Google Gemini	2.5 Flash	Language models	Cost-effective, multi-modal capable, fast inference
sentence-transformers	2.2.2	Text embeddings	High-quality embeddings, pretrained models, efficient

ChromaDB	0.4.10	Vector database	Embedded mode, Python-native, fast similarity search
-----------------	--------	-----------------	--

Database Technologies:

Technology	Version	Purpose	Justification
Supabase	Latest	PostgreSQL hosting	Managed Postgres, real-time subscriptions, authentication
PostgreSQL	15.3	Relational database	ACID compliance, JSON support, mature ecosystem
ChromaDB	0.4.10	Vector store	Semantic search, embedding storage, local-first

AI CAPABILITY AND DATA DESIGN DESCRIPTIONS

The AI Tutor system employs five specialized agents orchestrated through LangGraph's directed graph framework. Four agents are fully operational (PlannerAgent, TutorAgent, MathAgent, EvaluatorAgent), with FeedbackAgent designed for future implementation. Each agent operates as an independent module with defined inputs, outputs, and decision-making capabilities grounded in course materials.

Natural Language Processing Capabilities

Query Understanding:

- Intent recognition using Google Gemini 2.0 Flash for semantic classification
- Hybrid routing combining regex pattern matching with LLM-based disambiguation
- Query type classification: conceptual explanation, mathematical computation, off-topic detection
- Multi-part query decomposition into sequential subtasks

Conversational AI:

- Multi-turn dialogue management with LangGraph state preservation
- Conversation history stored in PostgreSQL (chat_logs table) with turn indexing
- Context window management for coherent multi-step problem solving
- Natural language generation tuned for pedagogical clarity (temperature=0.7 for TutorAgent)

Semantic Comprehension:

- Vector-based semantic search using Sentence Transformers (all-MiniLM-L6-v2)
- Retrieval-Augmented Generation ensures responses grounded in course materials
- Query expansion for improved retrieval coverage (e.g., "neural networks" → "neural networks examples", "neural networks formula")
- Similarity threshold filtering (0.35) to exclude low-relevance content

Agent-Specific Capabilities

PlannerAgent - Intelligent Routing:

- Autonomous task decomposition from complex multi-part queries
- Confidence-based LLM fallback when heuristic patterns insufficient
- Safety merging prevents omission of detected computational requirements
- Routing metadata logged for performance analysis

TutorAgent - Pedagogical Explanation:

- Socratic questioning and scaffolded learning methodology
- RAG-based context retrieval from ChromaDB (top-5 similarity search)
- Source citation with module references and similarity scores
- Fallback detection indicates insufficient course content coverage

MathAgent - Computational Problem Solving:

- Step-by-step reasoning with explanations for each derivation
- SymPy symbolic computation with Wolfram Alpha fallback
- Topic inference for retrieving relevant worked examples (k=8 retrieval)
- Ambiguity detection prompts clarification for underspecified problems
- Confidence scoring based on computational verification

FeedbackAgent - Quality Assurance:

- Automated weekly analysis of low-rated conversations (≤ 2 stars)
- LLaMA 3 critique identifies root causes and primary agent failures
- Structured insight generation with severity classification
- Human-in-the-loop design prevents automated prompt modifications

Knowledge Base Architecture

ChromaDB Vector Storage:

Collection	Documents	Content Type	Purpose
course_comp_237	1,638	Lecture slides, labs, assignments	Course-specific explanations
oer_resources	2,239	Open educational resources	Supplementary content

Embedding Pipeline:

- Model: Sentence Transformers all-MiniLM-L6-v2 (384 dimensions)
- Chunking: 1000-character chunks with 200-character overlap
- Metadata: Document source, module number, content type, creation date
- Similarity metric: Cosine similarity with relevance threshold = 0.35

Content Retrieval Process:

1. Query embedding generation
2. Multi-collection parallel search
3. Deduplication by content hash
4. Ranking by similarity score
5. Context formatting with source citations (max 1200 characters)

Data Architecture

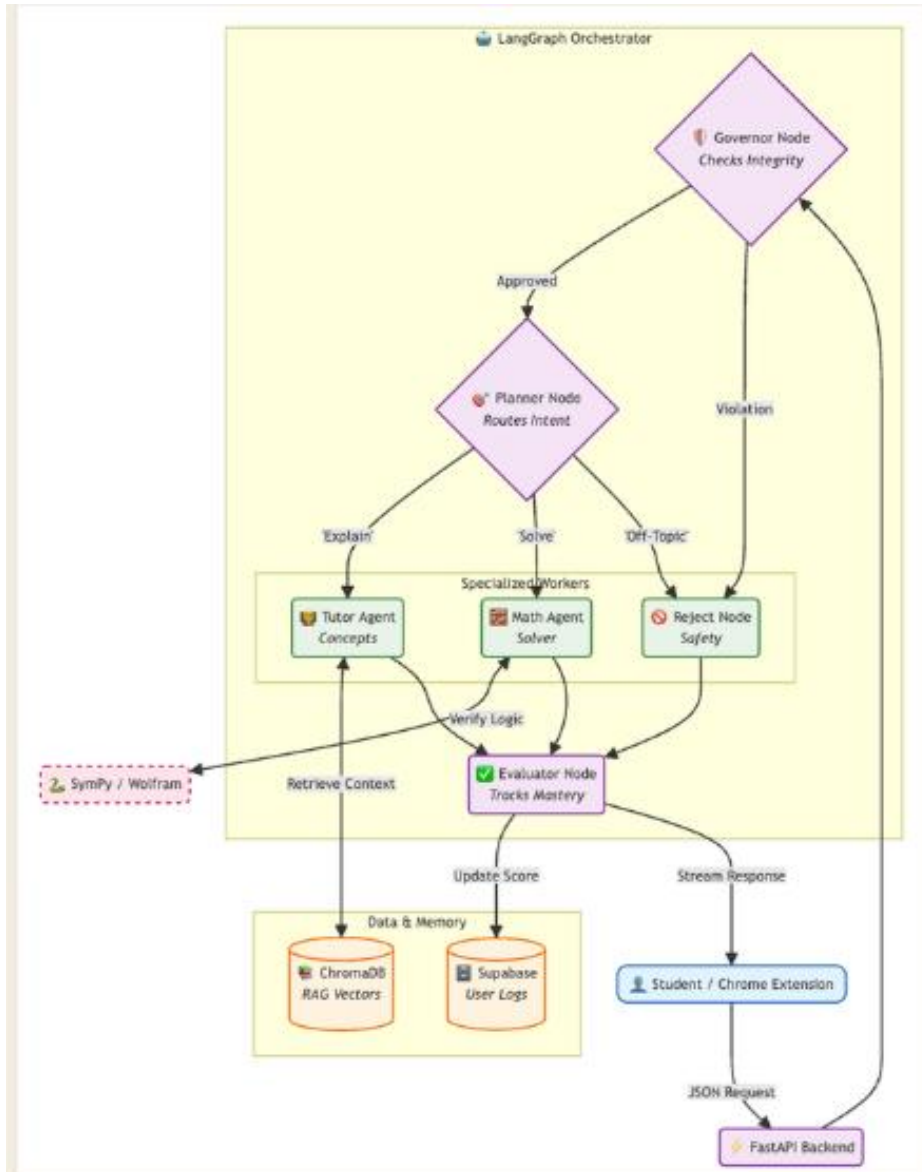
PostgreSQL Schema:

Table	Purpose	Key Fields
chat_logs	Complete conversation transcripts	conversation_id, turn_index, role, agent, content, timestamp
student_ratings	User feedback collection	conversation_id, rating (1-5), comment, timestamp
analytics_insights	FeedbackAgent critique storage	topic, primary_agent, root_causes, suggested_changes, severity

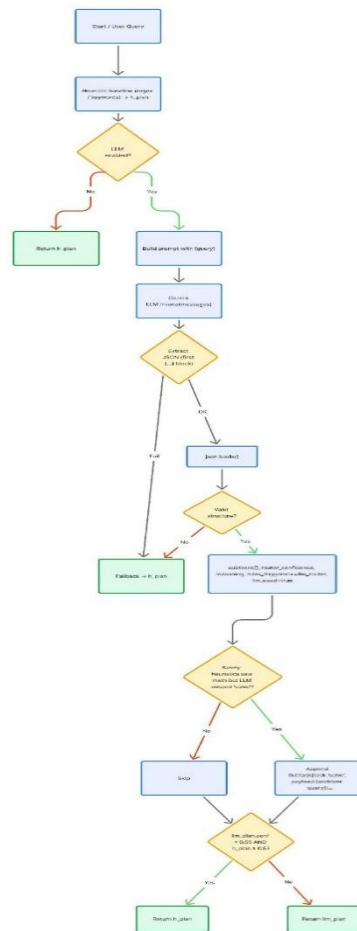
Privacy and Compliance

- Data minimization: Only essential learning-related data collected
- Anonymization: Personal identifiers separated from learning data in analytics
- Retention policies: Conversation data retained for current academic term only
- Access controls: Role-based permissions (student, instructor, administrator)
- Audit logging: All data access and modifications logged with timestamps
- Export capabilities: Students can request complete data exports in JSON format
- Deletion rights: Students can request account and data deletion at any time

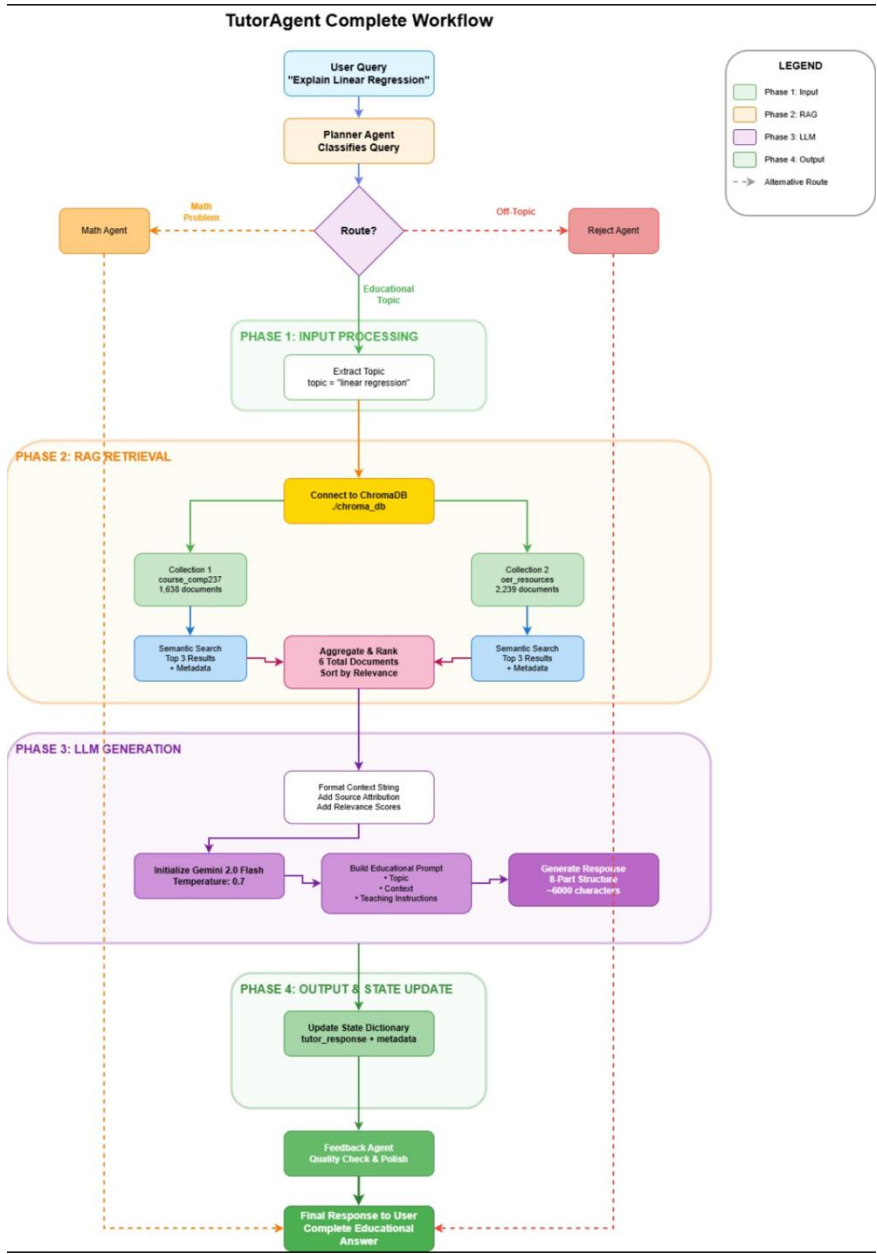
System Architecture Diagram (Figure 1)



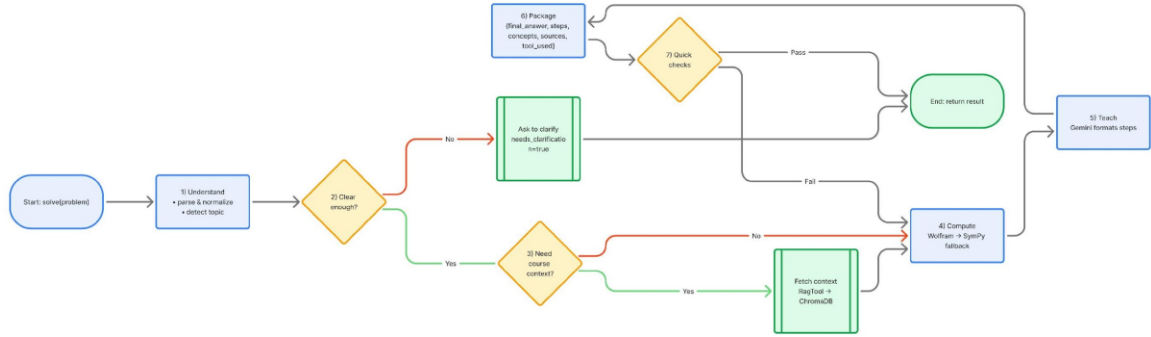
Planner Agent Workflow Diagram (Figure 2)



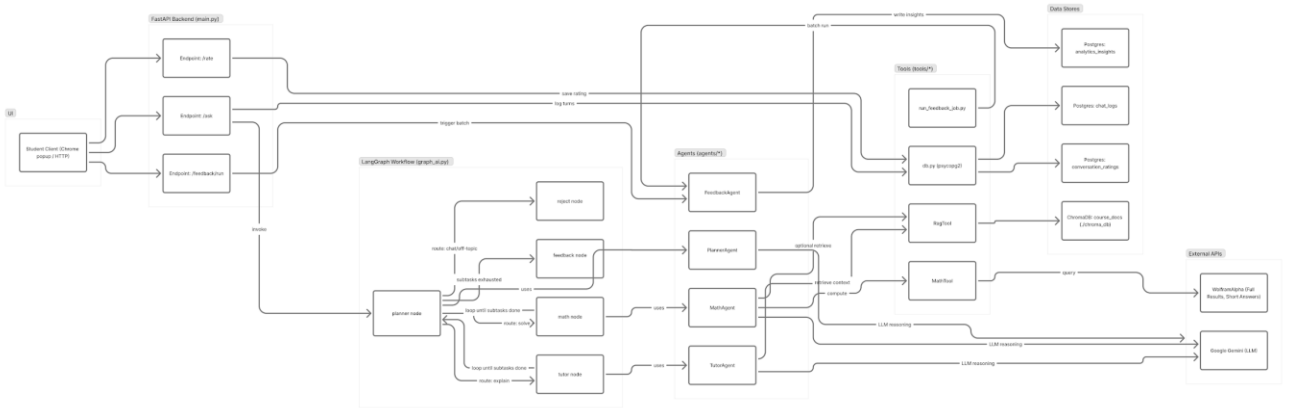
Tutor Agent Workflow Diagram (Figure 3)



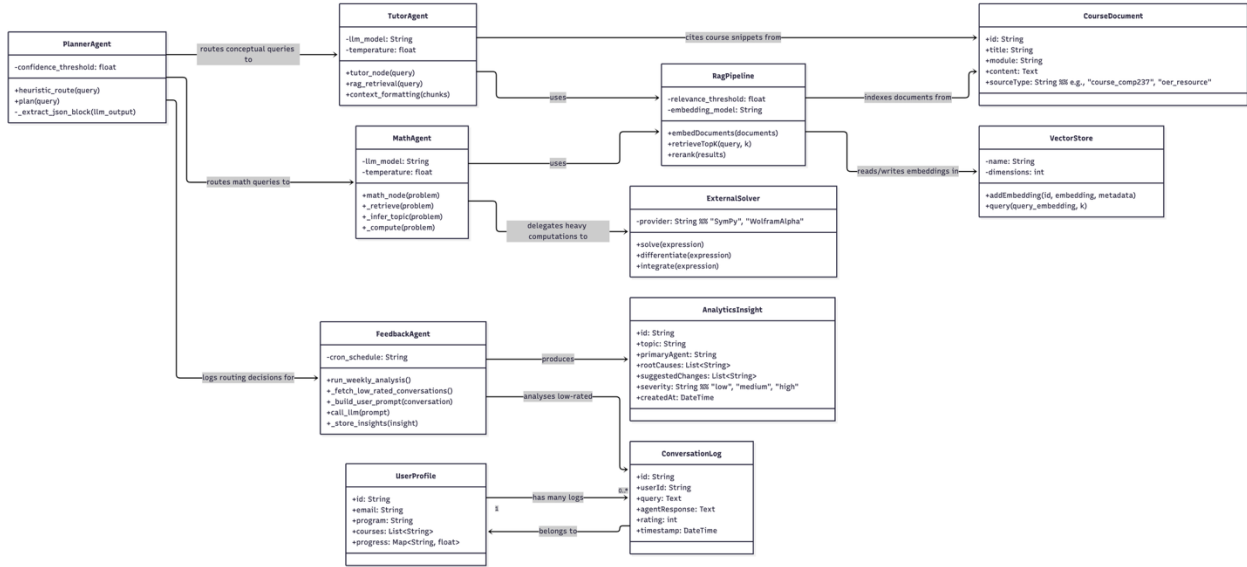
Math Agent Workflow Diagram (Figure 4)



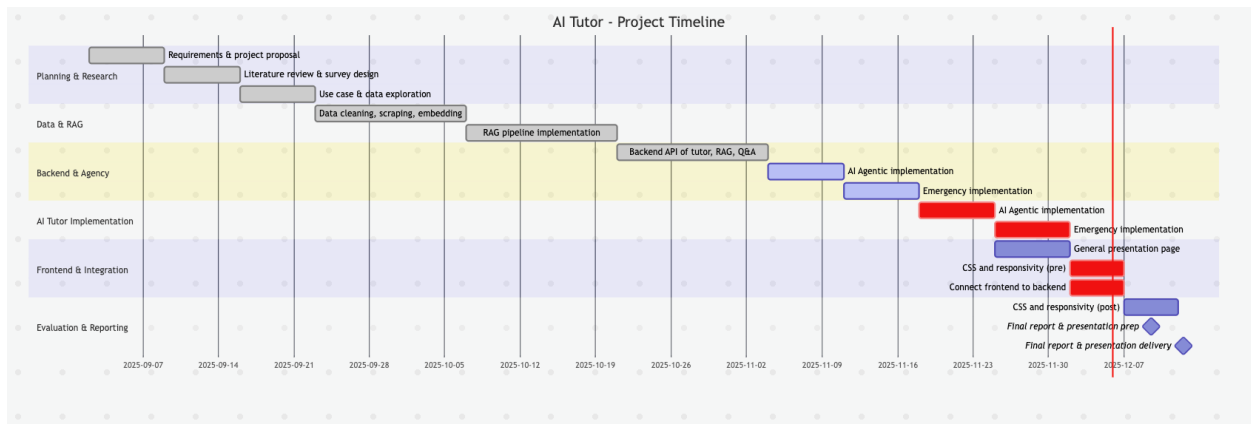
Component Diagram (Figure 5)



Class Diagram (Figure 6)



Gantt Diagram (Figure 7)



Deployment Strategy

Phase	Task	Practices
Model Selection	Model hosting	Host Gemini 2.5 Flash via Google Cloud API for agent reasoning and text-embedding-004 for vector embeddings
Model Selection	Infrastructure choice	Run workloads in hybrid architecture: Google Cloud (Gemini API, Supabase PostgreSQL) and local Docker Compose (7 microservices)
Model Selection	Target environment	Use cloud APIs for compute-heavy tasks and local Docker containers for orchestration caching and vector search
Deployment	Containerization	Package all services as Docker containers with Docker Compose
Deployment	Configuration management	Store API keys in .env with separate dev/prod configs
Deployment	Resource optimization	Use Gemini Flash pricing + Redis LRU caching + single Supabase instance with RLS to reduce costs
Launching	Container initialization	Unpack Docker images and initialize ChromaDB + Redis volumes
Launching	Service exposure	Expose FastAPI backend (port 8000) and Langfuse UI (port 3000) via HTTPS to Chrome extension
Launching	API endpoints	Provide /api/chat/stream /health and /api/admin/* endpoints
Launching	Network architecture	Use Docker bridge network luminate_network for DNS-based inter-service communication
Operations	Team collaboration	ML engineers backend frontend data engineers DevOps and instructors collaborate on system design
Operations	Monitoring & observability	Langfuse traces agent executions Docker health checks and Supabase logs all interactions

Operations	Quality assurance	Manual testing of scaffolding responses routing validation and planned FeedbackAgent weekly checks
Operations	User access	Chrome extension via Web Store with Supabase magic-link login and RLS for per- student isolation

High level ML/ops description

Phase	Task	Practices
Inference Pipeline	Data preprocessing	Query normalization: Student query + conversation history + mastery context → LangGraph state dictionary. PlannerAgent classifies intent and routes to specialist agents.
Inference Pipeline	Prediction serving	Real-time inference via FastAPI /api/chat/stream with SSE streaming. LangGraph orchestrates: PlannerAgent → Router → TutorAgent/MathAgent → EvaluatorAgent → Response.
API design	REST API	/api/chat/stream (POST-SSE)
Inference timing	Online	Real-time student queries (<2s target).
Inference timing	Offline	Planned FeedbackAgent weekly cron job for low-rated conversation analysis.
Traffic elasticity	Stateless api_brain	Enables horizontal scaling with load balancer. Current: Single instance; Production: Multi-replica.
Data quality	ETL pipeline	396 Blackboard resources → quality filtering → 111 items → 15 documents embedded. Manual curriculum validation.
Latency vs Accuracy	Hybrid routing	Fast regex (<50ms) for clear intents; Gemini semantic analysis (~200ms) for

		ambiguous. Redis caching: 150ms → <5ms (40% hit rate).
Resource allocation	Local Docker + Cloud	Local Docker (ChromaDB
Redis	Langfuse) + Cloud (Gemini API	Supabase). Cost: \$0.02-0.05/query.
Failover handling	Graceful degradation	ChromaDB down → RAG disabled with explicit notice; Redis down → Direct DB queries; LLM timeout → Cached fallback.
Monitoring	Observability system	Langfuse traces: agent routing
Monitoring	Metrics tracking	Router confidence
RAG retrieval quality	scaffolding escalations	mastery updates. All logged to Supabase.
Health monitoring	Docker health checks	Auto-restart. /health endpoint validates service availability. Degraded mode if cache_layer/observer fail.

GLOSSARY OF TERMS

Agent – An autonomous software component with specialized capabilities, decision-making logic, and defined inputs/outputs. In AiTutor, five agents handle distinct educational tasks (planning, tutoring, computation, evaluation, feedback).

API (Application Programming Interface) – A set of protocols enabling communication between software systems. AiTutor exposes REST API endpoints for the Chrome extension to interact with the backend.

ChromaDB – Open-source vector database for storing and retrieving embeddings. Used to store 438 embedded chunks from COMP237 course materials for semantic search.

Docker Compose – Tool for defining and running multi-container applications. AiTutor uses Docker Compose to orchestrate 7 microservices (api_brain, memory_store, observer, cache_layer, and Langfuse stack).

Embedding – Numerical vector representation of text that captures semantic meaning. AiTutor uses Gemini text-embedding-004 to generate 768-dimensional embeddings for course materials.

Escalation Level – Progressive scaffolding stages (1-4) that increase explanation depth based on student confusion signals. Level 1 provides hints only; Level 4 provides full explanations.

ETL (Extract, Transform, Load) – Data pipeline process for acquiring, processing, and storing information. AiTutor's ETL extracts Blackboard resources, filters quality, chunks text, and embeds into ChromaDB.

Gemini 2.5 Flash – Google's large language model optimized for low-latency inference. Used by all AiTutor agents for reasoning, classification, and response generation.

HNSW (Hierarchical Navigable Small World) – Approximate nearest neighbor search algorithm for fast vector similarity search. ChromaDB uses HNSW indexing for efficient semantic retrieval.

LangGraph – Framework for building stateful multi-agent workflows as directed graphs. Orchestrates AiTutor's agent coordination, conditional routing, and state management.

LLM (Large Language Model) – Neural network trained on vast text corpora for natural language understanding and generation. AiTutor uses Gemini 2.5 Flash as its primary LLM.

Langfuse – Open-source LLM observability platform for tracing, debugging, and monitoring. Captures execution traces for all AiTutor agent interactions.

Luminate – Centennial College's branded Learning Management System (Blackboard). AiTutor integrates directly into Luminate via Chrome extension.

Mastery Score – Quantitative measure (0.0-1.0) of student proficiency per concept. EvaluatorAgent updates scores using decay-adjusted formula based on interaction outcomes.

RAG (Retrieval-Augmented Generation) – Technique combining semantic search with LLM generation to ground responses in specific documents. TutorAgent retrieves course materials before generating explanations.

Redis – In-memory key-value database used for caching. Reduces ChromaDB query latency from ~150ms to <5ms with 40% cache hit rate.

RLS (Row-Level Security) – Database access control mechanism restricting data visibility per user. Supabase RLS policies ensure students only access their own interaction logs and mastery scores.

Scaffolding – Educational methodology providing structured support that gradually decreases as learner competence increases. Implemented via 4-level escalation system from hints to full explanations.

Semantic Similarity – Measure of meaning-based closeness between text segments. ChromaDB uses L2 distance on embeddings with 0.25 threshold for relevant chunk retrieval.

Socratic Method – Teaching approach using questions to stimulate critical thinking rather than providing direct answers. TutorAgent employs Socratic questioning at escalation levels 1-2.

SSE (Server-Sent Events) – HTTP protocol for server-to-client real-time streaming. AiTutor streams LLM token generation to Chrome extension via SSE for responsive user experience.

Supabase – Open-source Firebase alternative providing managed PostgreSQL with authentication and real-time capabilities. Stores student mastery, interaction logs, and conversation history.

Vector Database – Specialized database for storing and querying high-dimensional embeddings. ChromaDB serves as AiTutor's vector database for semantic course material retrieval
