Software Requirements Specification

for WriteLoop Adaptive Writing Coach System

Team Members:

Xin Gao (2251356)

Zhuobing Zhao (2252750)

Jialu Wang (2351890)

Yining Sun (2352736)

School of Computer Science and Technology

Tongji University

December 15, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to specify the software requirements for the WriteLoop Adaptive Writing Coach System. In the context of the growing demand for globalized education, writing quality serves as a pivotal channel for linguistic output, directly influencing learners' academic performance and cross-cultural communication efficiency. However, current learning scenarios face significant barriers: traditional instruction suffers from high feedback latency and a lack of personalization, while existing digital tools are often limited to basic grammatical corrections, failing to diagnose deep logical structures or academic expression standards. Furthermore, learners often struggle with the ”disconnect between input and output,” unable to transfer reading knowledge into writing practice, and existing tools frequently neglect the learner's physiological state, leading to inefficient ”junk time” caused by unmonitored fatigue or distraction.

To address these challenges, the WriteLoop system integrates Retrieval-Augmented Generation (RAG), Large Language Models (LLM), and Edge-side Computer Vision. The primary objective of this project is to build an intelligent tutoring system that understands both the ”text” and the ”learner.” By bridging the gaps in logical diagnosis, knowledge transfer, and cognitive state management, the system aims to facilitate a comprehensive progression for learners from simply ”knowing how to write” to ”writing efficiently” and maintaining an optimal cognitive state.

1.2 Scope

The WriteLoop Adaptive Writing Coach is a specialized intelligent support module integrated within the larger LearnLoop platform. The software is designed to provide full-process personalized writing guidance through four core capabilities: Instant Assistance, which offers context-aware vocabulary and syntactic recommendations based on the user's reading history; Logic Insight, which utilizes structural parsing and coherence heatmaps to visualize argumentative frameworks; Deep Profiling, which tracks long-term writing metrics such as vocabulary richness and syntactic complexity to generate targeted training tasks; and Intelligent State Monitoring, which uses privacy-preserving multimodal sensing to detect user fatigue and attention levels, triggering adaptive interventions when necessary.

The scope of this system is strictly limited to the upper-layer application of AI technologies for writing education. It explicitly excludes the development of underlying foundation models (such as training base LLMs from scratch). Furthermore, the system does not include general Learning Management System (LMS) functionalities of the LearnLoop platform (such as course scheduling or grade management) that are unrelated to the writing

module, nor does it involve the implementation of any payment, subscription, or commercial billing modules.

1.3 Definitions

The following terms and acronyms are used throughout this document to define specific technologies and metrics employed by the system:

- **RAG**: A technique that combines information retrieval with generative models to enhance the accuracy and reliability of AI-generated content by referencing external knowledge bases.
- **LLM**: Large-scale pre-trained models (e.g., GPT series) capable of advanced natural language understanding and generation tasks.
- **TTR**: A linguistic metric used to measure vocabulary diversity by calculating the ratio of unique words to the total number of words in a text.
- **MLU**: A metric indicating linguistic complexity, calculated by the average number of words per sentence.
- **Logic Score**: A composite indicator reflecting the logical coherence of a text, derived from calculating the average cosine similarity between embeddings of adjacent sentences.
- **Embedding**: The process of converting text into dense numerical vectors that capture semantic information for machine processing.
- **PERCLOS**: Percentage of Eyelid Closure over the Pupil, a standard physiological metric used to quantify fatigue levels based on eye closure duration.

2 Overall Description

2.1 Product Perspective

The WriteLoop Adaptive Writing Coach operates as a specialized, independent module integrated within the broader LearnLoop educational ecosystem, rather than functioning as a standalone product. It interfaces directly with the LearnLoop User Database to retrieve historical reading logs, which serve as the retrieval corpus for the Retrieval-Augmented Generation (RAG) engine, and utilizes the platform's existing OAuth 2.0 service for secure authentication. A defining architectural feature of the system is its "Edge-Cloud" hybrid design. While computational-heavy text generation tasks rely on cloud-based Large Language Models (LLMs), all video data streams required for intelligent state monitoring are

processed exclusively on the client device using Edge AI technologies. This ensures that raw video data is never transmitted to the server, strictly preserving user privacy while enabling real-time responsiveness.

2.2 Product Functions

he system provides intelligent support across the writing lifecycle through four core functional modules:

- **Instant Writing Assistance**: Delivers context-aware vocabulary recommendations and syntactic restructuring suggestions by leveraging RAG and LLMs to bridge the gap between reading inputs and writing outputs.
- **Logic Insight**: Automatically parses the argumentative structure (Thesis, Arguments, Evidence) of the text and visualizes logical coherence using heatmaps to identify structural weaknesses.
- **Deep Profiling & Adaptive Tasks**: Tracks long-term writing metrics (e.g., TTR, MLU) and employs clustering algorithms to generate personalized micro-tasks targeting specific user weaknesses.
- **Intelligent State Monitoring**: Utilizes multimodal sensing to detect real-time fatigue (PERCLOS) and distraction levels, triggering adaptive interventions to maintain optimal cognitive engagement

2.3 User Story

The system is designed to serve three distinct user classes based on proficiency levels and learning objectives.

User Story 1:Foundation Learners As a freshman preparing for exams like CET-4/6, I want real-time vocabulary expansion and syntactic optimization so that I can improve my limited vocabulary and fix monotonous sentence structures.

Description:This class is typified by undergraduate freshmen who are establishing their academic English base. They generally possess a limited vocabulary range and rely heavily on monotonous or simple sentence structures. Consequently, their primary need is the real-time expansion of academic vocabulary and syntactic optimization to elevate the baseline quality of their writing.

User Story 2:Advanced Learners As an IELTS or TOEFL candidate, I want visualization tools to diagnose structural coherence so that I can resolve issues with paragraph organization and logical transitions in rigorous academic argumentation.

Description: This class focuses on producing rigorous academic argumentation but often struggles with the macro-level organization of paragraphs and logical transitions between

ideas. They require sophisticated visualization tools (such as logic trees or heatmaps) to diagnose structural coherence and ensure their writing meets high academic standards.

User Story 3:Long-term Learners As a postgraduate applicant facing high-intensity preparation, I want data-driven performance tracking and active fatigue management so that I can visualize my progress and prevent inefficient practice caused by exhaustion.

Description: This class is characterized by high-intensity, long-duration preparation periods. They are particularly susceptible to fatigue and often face difficulty in tracking their progress over time ("blind practice"). Their critical needs are data-driven performance tracking to visualize growth and active fatigue management to intervene before efficiency drops.

2.4 Project Constraints

The design, implementation, and deployment of the WriteLoop system are governed by the following constraints:

2.4.1 Technological Environment Constraints

- **Server-Side Configuration**: The backend must be deployed on Ubuntu 22.04 LTS or CentOS Stream 9+. It requires high-performance computing resources, specifically an Intel Xeon Silver 4314+ CPU and an NVIDIA A10 (or equivalent) GPU to support concurrent LLM inference.
- **Software Stack**: The development stack is mandated to use Python 3.9+, FastAPI, PyTorch 2.0+, PostgreSQL 14+, and Redis 6.0.
- **Client-Side Compatibility**: The application must support modern operating systems (Windows 10/11, macOS 12+) and mainstream browsers (Chrome 90+, Edge 90+, Firefox 88+).
- **Hardware Requirement**: A functional webcam is strictly required on the client device to enable the computer vision features of the Intelligent State Monitoring module.
- **Technical Constraints & Phasing:**
  - *Latency:* The primary inference endpoint (`suggest`) must respond within 1.5s (P95) in the final release.
  - **MVP Scope Constraint:** Due to the 1-month development window, the MVP release prioritizes **functional completeness** and **algorithm accuracy** over high-concurrency performance optimization. Edge-side model inference (MediaPipe) may run in "Debug Mode" with reduced frame rates.

2.4.2 Regulatory and Ethical Constraints

- **Data Sovereignty**: Data storage and processing must strictly comply with local regulations, such as the Data Security Law, ensuring user data is stored within the required jurisdiction.
- **Academic Integrity**: To prevent plagiarism, the system is prohibited from generating full essays or complete paragraphs. All AI-generated suggestions must function as scaffolding (e.g., vocabulary hints) and must explicitly cite their sources.
- **Privacy Protection**: The system must employ a local-first processing mechanism for video data, ensuring that raw video footage is processed at the edge and never transmitted to cloud servers.

2.4.3 Implementation and Delivery Constraints

- **Performance Targets**:To ensure a seamless user experience, the primary inference endpoint (`suggest`) is mandated to maintain a P95 latency of ≤ 1.5 seconds under standard load9.
- **Development Timeline**: The project follows a rigid schedule, with the Minimum Viable Product (MVP) restricted to a three-month development window from November 2025 to January 2026.
- **User Support**: The delivery must include comprehensive documentation, including an Online Help Center for complex features, interactive onboarding tutorials, and a transparent Privacy Statement.

2.4.4 External Dependencies and Assumptions

- **System Dependencies**: The system functionality depends on the continued availability and stability of the LearnLoop user database and external LLM API services.
- **Network Assumption**: It is assumed that users possess a stable internet connection with a recommended bandwidth of 50 Mbps or higher to support real-time RAG retrieval.
- **Permission Assumption**: It is assumed that users will grant browser permissions for camera access; if denied, the system is designed to automatically disable state monitoring features while keeping other functionalities active.

3 System Features

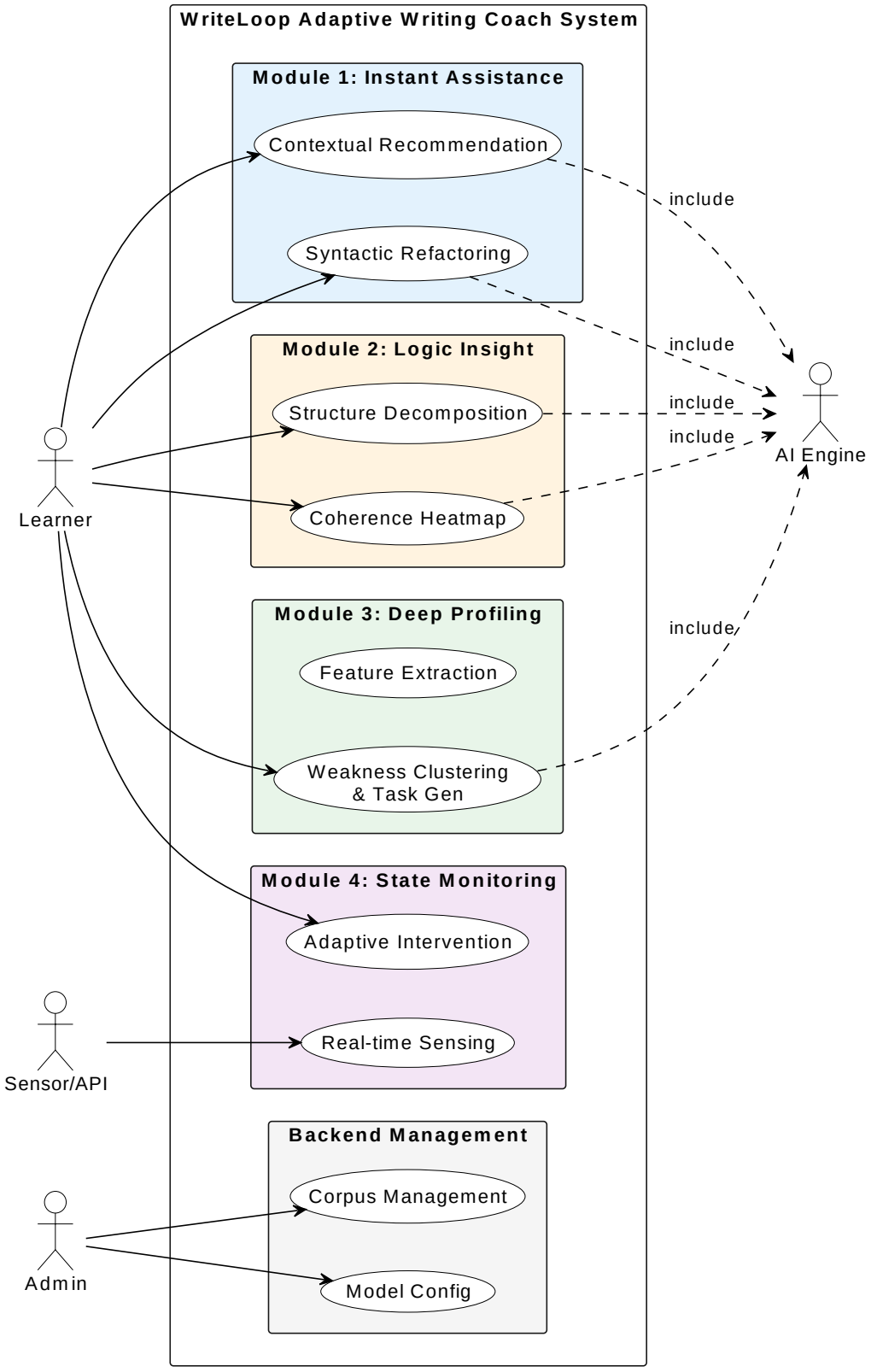

Figure 1: Overall Use Case Diagram of WriteLoop System

The system features are categorized as follows:

1. **Module 1: Instant Writing Assistance**: Focuses on real-time linguistic support using RAG.
2. **Module 2: Logic Insight**: Focuses on structural analysis and coherence visualization.
3. **Module 3: Deep Profiling & Adaptive Tasks**: Focuses on long-term data tracking and personalized training.
4. **Module 4: Intelligent State Monitoring**: Focuses on physiological state detection and intervention.

3.1 Module 1: Instant Writing Assistance

3.1.1 Description

The Instant Writing Assistance module shall fulfill the following functional requirements:

1. **Real-time Input Monitoring:**The system shall continuously monitor the user's input within the editor and support context capture through two triggering mechanisms:

- Automatic detection of writing pauses
- Explicit user invocation via a designated keyboard shortcut.

In either case, the system shall extract the text context preceding the current cursor position as the query input.

2. **Context-Aware Recommendation via RAG:**Using Retrieval-Augmented Generation (RAG), the system shall retrieve relevant passages from the user's personal reading corpus (falling back to a system-curated knowledge base during cold start) and invoke a Large Language Model (LLM) to generate three contextually appropriate, advanced lexical collocations. Each suggestion must include a citation indicating its source to ensure academic integrity and traceability.

3. **Syntactic Restructuring with Visual Highlighting:** For any user-selected sentence, the system shall provide at least two rewritten variants—"Concise" and "Academic"—by restructuring its syntactic form. Differences such as non-finite verb phrases, subordinate clauses, and other grammatical transformations shall be visually highlighted using color coding to clearly illustrate the nature of the proposed improvements.

4. **One-Click Application and Edit History:**The system shall provide a one-click interface to apply either lexical recommendations or syntactic rewrites. Upon user confirmation, the original text in the editor shall be automatically replaced. All such operations shall be recorded in an edit history stack, enabling users to undo or revert changes at any time, thereby ensuring a seamless and reversible writing experience.

3.1.2 Functional Requirements

1. **FR-1.1: Contextual Collocation Recommendation**

    - **Input:** The system shall continuously monitor the user's text input stream in the editor.
    - **Trigger Condition:** A recommendation cycle shall be triggered automatically when a user pause exceeds the threshold of **800ms**, or manually via a shortcut (e.g., `Alt+/`).
    - **Processing (RAG):** Upon triggering, the system shall:
        - Capture the preceding 512 tokens as the "Context Query."
        - Retrieve the top-3 most semantically relevant snippets from the user's personal *Reading Corpus* using vector similarity search.
        - If the personal corpus is sparse (cold start), automatically degrade to the system's *Global Academic Corpus*.
    - **Output:** The system shall display a floating "Suggestion Panel" containing 3 academic collocation options, each accompanied by its source citation (e.g., *"Source: The Economist"*).

2. **FR-1.2: Syntactic Structure Refactoring**

    - **Input:** The system shall accept a text selection (sentence or paragraph) made by the user.
    - **Processing (LLM):** The system shall invoke the LLM to generate two distinct rewritten versions of the selected text:
        - *Mode A (Concise):* Eliminates redundancy and simplifies sentence structure.
        - *Mode B (Academic):* Enhances formality, utilizes nominalization, and improves lexical density.
    - **Visualization:** The system shall use **diff-highlighting** (e.g., red for deletion, green for insertion) to visually contrast the original text with the rewritten versions.
    - **Interaction:** The system shall support "One-click Replace" and maintain an operation stack to allow `Undo/Redo` actions for data safety.

3.1.3 Use Case Analysis

The functional scope of the Instant Assistance module is visualized in Figure 2. This diagram illustrates the dependencies between user actions and internal system processes such as RAG retrieval and LLM generation.

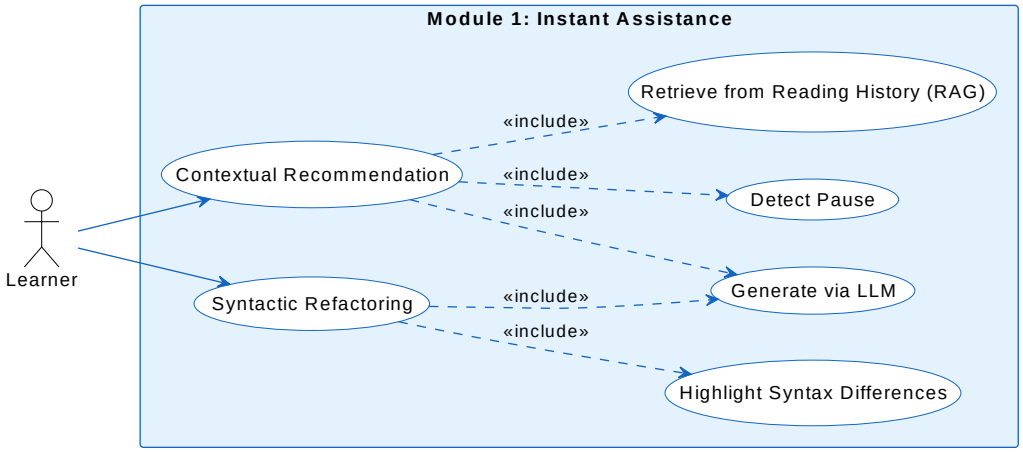

Figure 2: Detailed Use Case Diagram: Instant Writing Assistance

The following tables detail the interaction flows for the two primary use cases identified in this module.

Use Case Name	**Contextual Collocation Recommendation**
Brief Description	The system provides real-time recommendations for advanced collocations based on the context preceding the cursor and the user's reading history.
Primary Actor	Learner
Preconditions	1. User is logged into the system. 2. User is typing in the editor. 3. User has recent reading history records.
Basic Flow	1. User inputs text in the editor and pauses (duration $\geq$ 800ms). 2. System detects the pause event. 3. System captures the current context and queries the reading history index. 4. System retrieves relevant collocations via the RAG engine. 5. LLM filters results and generates 3 advanced collocations with definitions. 6. System displays the "Recommendation Panel" next to the cursor. 7. User selects "Accept" or "Reject" for the recommendation.
Alternative Flow	**2a.** If the pause is short ($<$ 800ms), no recommendation is triggered. **7a.** If User selects "Accept," the system automatically replaces the text and logs the operation. **7b.** If RAG/LLM services fail, the system downgrades to BM25-based template recommendations.
Postconditions	Recommendation interaction is logged; user profile is updated.

Table 1: Use Case: Contextual Collocation Recommendation

Use Case Name	**Syntactic Structure Refactoring**
Brief Description	Upon user selection, the system provides diverse, academic rewritten versions of the sentence.
Primary Actor	Learner
Preconditions	User is logged in and is editing text.
Basic Flow	1. User selects one or more sentences in the editor. 2. User clicks the ”Enhance Structure” button. 3. System analyzes the original syntax and context. 4. LLM generates at least two versions (e.g., Concise, Complex). 5. System highlights key syntactic changes (e.g., non-finite verbs). 6. System displays candidate sentences. 7. User chooses to ”Apply” or ”Abandon” the rewrite.
Alternative Flow	**2a.** User may trigger the function via right-click menu or shortcut key. **7a.** If User selects ”Apply,” the system replaces the original sentence. **7b.** If User selects ”Abandon,” the text remains unchanged.
Postconditions	Rewrite record is saved; User can perform Undo operation.

Table 2: Use Case: Syntactic Structure Refactoring

3.1.4 Process & Interaction Modeling

To provide a comprehensive view of the system's runtime behavior, this section models the ”Instant Assistance” module from two distinct perspectives: the business logic flow and the component interaction timing.

1. Business Process Modeling (Activity Diagram) The Activity Diagram (Figure 3) illustrates the decision-making logic of the Contextual Recommendation engine. It details how the system handles user pauses and determines whether to trigger a RAG retrieval or degrade to a rule-based approach.

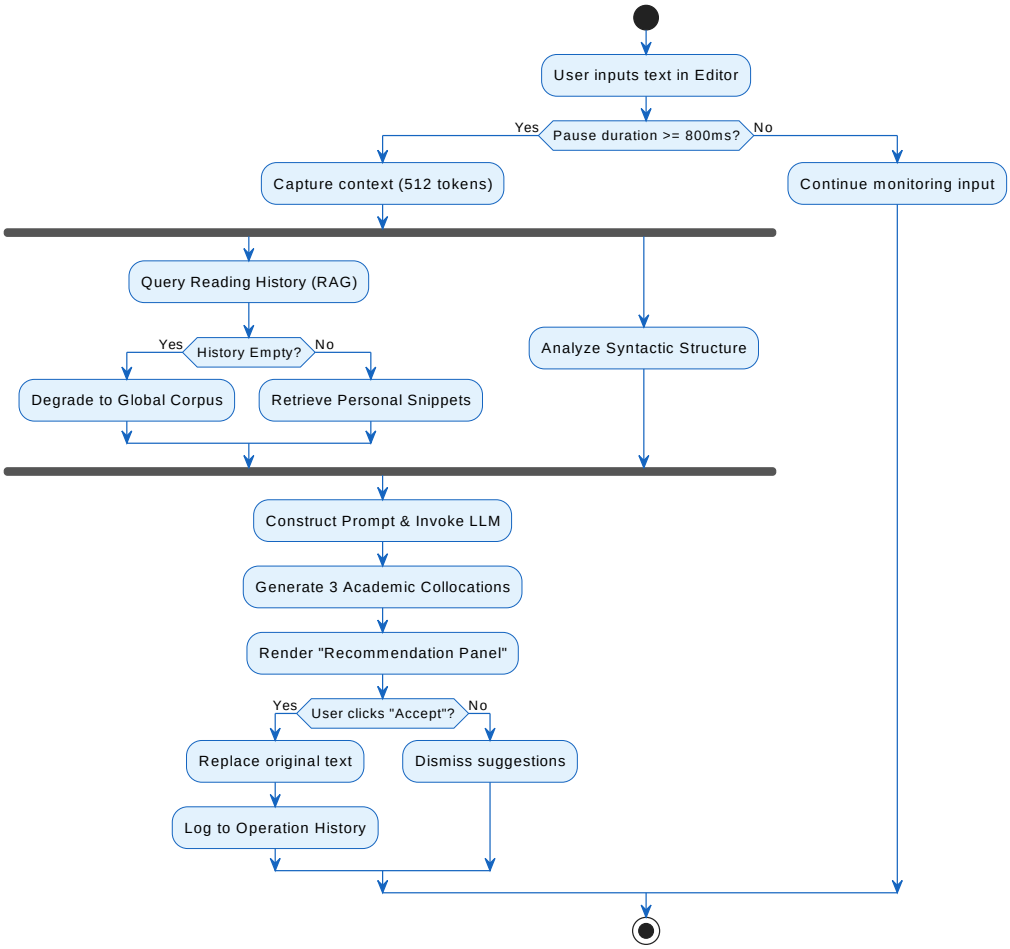

Figure 3: Activity Diagram: Recommendation Trigger Flow

2. System Interaction Modeling (Sequence Diagram) The Sequence Diagram (Figure 4) depicts the real-time interaction between the Frontend Editor, Backend API, RAG Engine, and LLM Service during a single recommendation cycle. It explicitly shows the asynchronous nature of the ”Context Capture → Vector Retrieval → Generative Inference” pipeline.

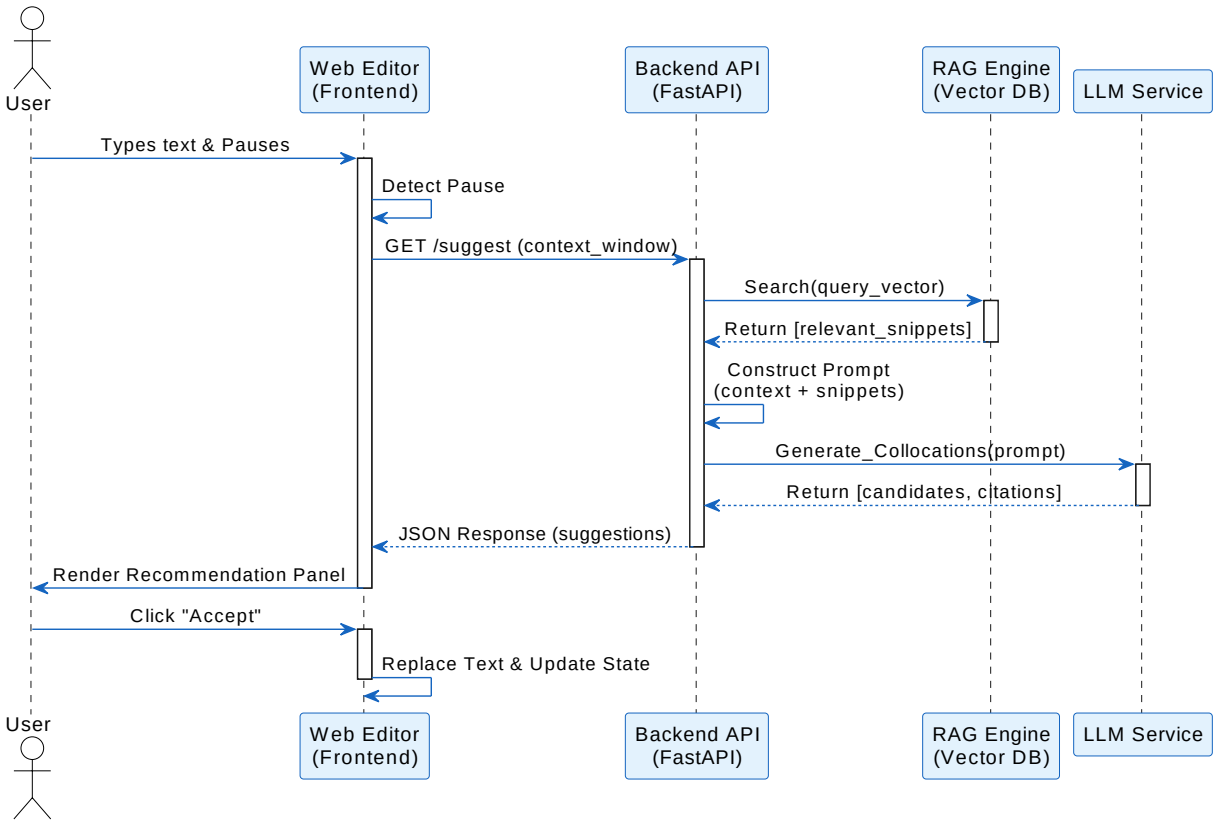

Figure 4: Sequence Diagram: Instant Recommendation Interaction

3.2 Module 2: Logic Insight

3.2.1 Description

The Logic Insight module shall fulfill the following functional requirements:

1. Deep Structural Parsing: The system shall perform end-to-end structural analysis of argumentative essays by leveraging a Large Language Model (LLM) to identify and extract the central Thesis, supporting Arguments, and their corresponding Evidence. It shall generate a visual "Argument Skeleton Tree" and automatically flag logical gaps (e.g., unsupported claims) as potential reasoning vulnerabilities.

2. Coherence Heatmap: The system shall compute cosine similarities between embeddings of adjacent sentences and paragraphs using a vector space model. A logic coherence heatmap shall be rendered, using a color gradient (e.g., green to red) to intuitively highlight regions of semantic discontinuity or abrupt transitions.

3. Context-Aware Transition Recommendation: For coherence weaknesses detected by the heatmap, the system shall analyze local and global context to intelligently recommend appropriate transitional phrases or connective sentence templates (e.g., "Conversely...", "This is further supported by...") to improve textual flow.

4. Interactive Navigation & Anchoring: The system shall provide a sidebar panel displaying the Argument Skeleton Tree and Coherence Heatmap. Upon user click on any node

or heatmap segment, the main editor view shall auto-scroll and highlight the corresponding text span, enabling seamless navigation from macro-level diagnosis to micro-level revision.

3.2.2 Functional Requirements

1. **FR-2.1: Argumentative Structure Decomposition**

    - **Structure Parsing:** The system shall utilize an LLM to perform a deep scan of the full text, identifying the hierarchical structure including the central Thesis, Arguments, and Supporting Evidence.
    - **Visual Feedback:** The system shall generate a "Logic Skeleton Tree" in the sidebar. Users can interact with tree nodes to navigate to specific paragraphs.
    - **Loophole Detection:** The system shall automatically detect and flag structural flaws, such as arguments that lack supporting evidence (marked as "Evidence Missing").

2. **FR-2.2: Logical Coherence Heatmap**

    - **Similarity Calculation:** The system shall compute the Cosine Similarity between the embeddings of adjacent sentences to measure semantic consistency.
    - **Heatmap Rendering:** The system shall visualize these similarity scores using a color-coded heatmap (e.g., Red indicating low coherence).
    - **Repair Suggestions:** For detected "Logic Break Points," the system shall recommend context-appropriate transitional phrases to improve flow.

3.2.3 Use Case Analysis

The functional scope of the Logic Insight module is visualized in Figure 5. This diagram highlights the dependency between structure parsing and coherence analysis.

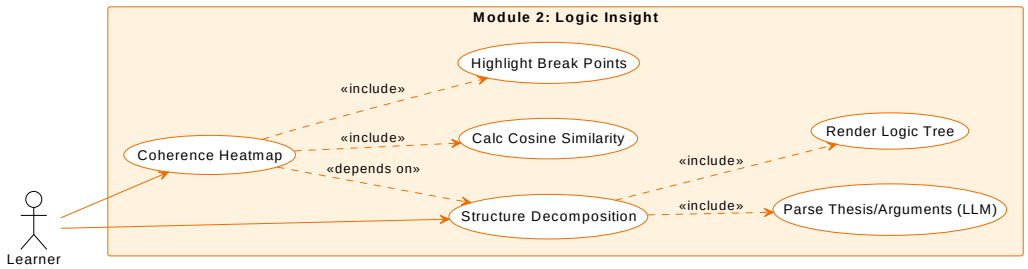

Figure 5: Detailed Use Case Diagram: Logic Insight Module

The following tables detail the interaction flows for the two primary use cases: Argumentative Structure Decomposition and Logical Coherence Heatmap.

Use Case Name	**Argumentative Structure Decomposition**
Brief Description	The system automatically parses the essay to generate a visualized hierarchical tree of the argumentation structure (Thesis, Arguments, Evidence).
Primary Actor	Learner
Preconditions	User is logged in and has inputted an argumentative essay.
Basic Flow	1. User clicks the ”Generate Structure Tree” button. 2. System extracts the full text content. 3. LLM identifies the Thesis Statement, Sub-arguments, and Evidence. 4. System generates a JSON-based structure tree. 5. System renders the visual ”Logic Skeleton Tree” in the sidebar. 6. User clicks a node to jump to the corresponding paragraph. 7. System highlights missing evidence (if any).
Alternative Flow	**1a.** The function triggers automatically when the document is saved. **3a.** (Unclear Thesis) If no thesis is found, the system prompts the user to clarify the main argument. **7a.** (Missing Evidence) System flags specific arguments as ”Evidence Missing” in red.
Postconditions	Structure data is saved and cached for subsequent analysis.

Table 3: Use Case: Argumentative Structure Decomposition

Use Case Name	**Logical Coherence Heatmap**
Brief Description	The system analyzes semantic consistency between sentences and generates a heatmap to identify logical breaks.
Primary Actor	Learner
Preconditions	Structure decomposition (UC-1) is complete.
Basic Flow	1. System triggers analysis automatically after structure generation. 2. System splits the text into independent sentences. 3. System calculates vector embeddings and computes Cosine Similarity for adjacent sentences. 4. System identifies ”Logic Break Points” (Low Similarity). 5. System renders a color-coded heatmap sidebar. 6. User clicks a ”Red” area to view details. 7. System recommends transitional phrases to repair the flow.
Alternative Flow	**1a.** User manually clicks the ”Analyze Coherence” button. **4a.** If overall coherence is high, the heatmap displays all green. **7a.** User chooses to ignore the suggestion.
Postconditions	Coherence scores are recorded for the user’s ”Logic Score” metric.

Table 4: Use Case: Logical Coherence Heatmap

3.2.4 Process & Interaction Modeling

To illustrate the computational logic behind the ”Deep Diagnosis” feature, this section models the workflow from both the business process and system interaction perspectives.

1. Business Process Modeling (Activity Diagram) The Activity Diagram (Figure 6) depicts the sequential workflow of the analysis pipeline. It demonstrates how the system processes the raw text to generate two distinct visualizations: the Logic Skeleton Tree and the Coherence Heatmap.

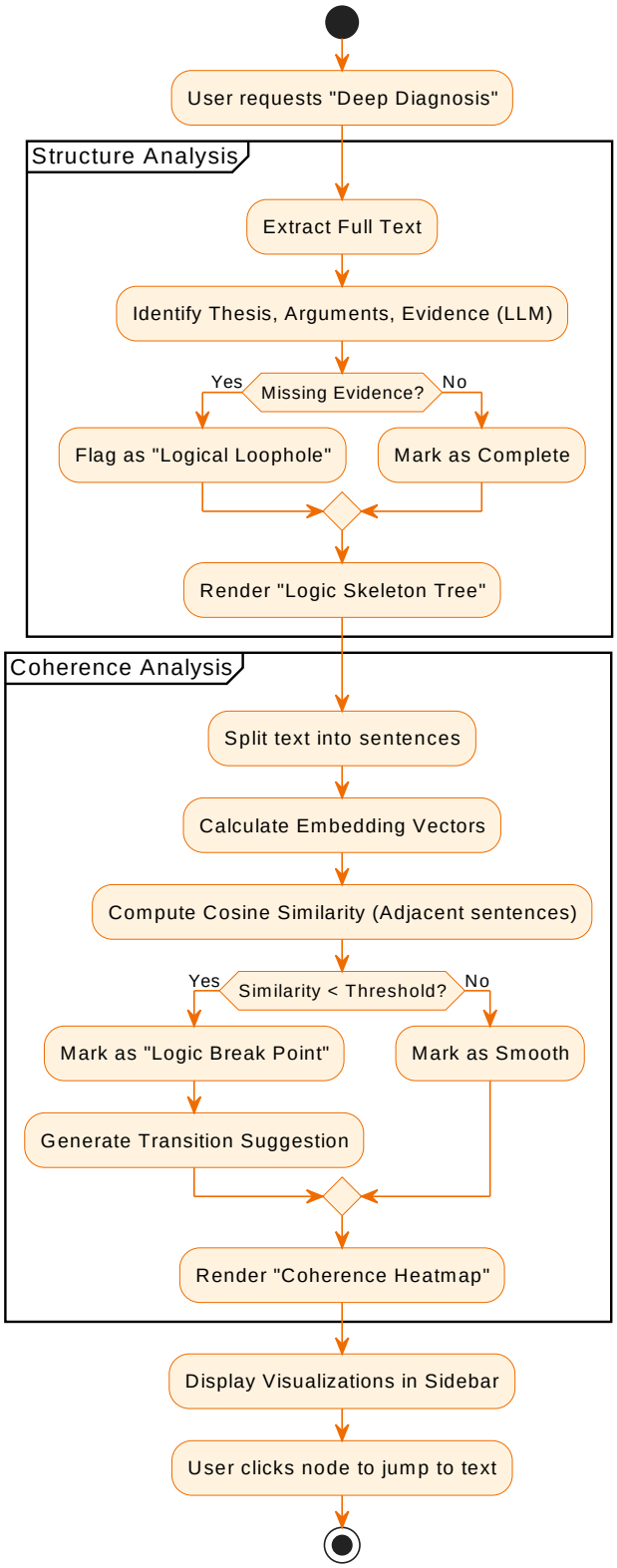

Figure 6: Activity Diagram: Logic Analysis Workflow

2. System Interaction Modeling (Sequence Diagram) The Sequence Diagram (Figure 7) details the backend orchestration required for deep diagnosis. Notably, it highlights the **parallel processing** architecture, where LLM-based structure parsing and Vector-based co-

herence calculation are executed concurrently to minimize user waiting time.

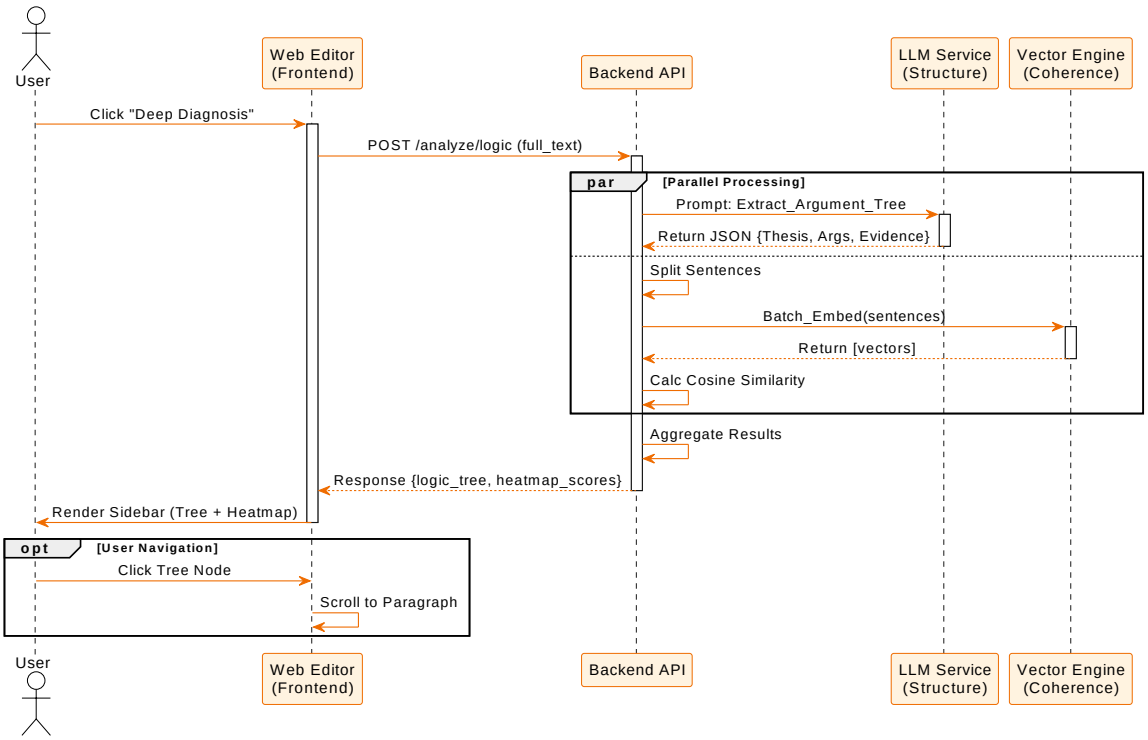

Figure 7: Sequence Diagram: Logic Insight Interaction

3.3 Module 3: Deep Profiling & Adaptive Tasks

3.3.1 Description

The Deep Profiling & Adaptive Tasks module shall fulfill the following functional requirements:

1. Multi-dimensional Feature Extraction: Upon each writing submission, the system shall automatically trigger a feature extraction pipeline to compute quantitative metrics including Type-Token Ratio (TTR, measuring lexical diversity), Mean Length of Utterance (MLU, indicating syntactic complexity), and Logic Score (assessing argumentative coherence). These metrics shall be archived in a time-series database to enable longitudinal tracking of the user's writing proficiency.

2. Weakness Pattern Clustering: The system shall periodically execute unsupervised clustering algorithms on the user's historical error logs and low-scoring paragraphs to identify statistically significant weakness patterns—such as a "Tense Confusion Cluster" or a "Weak Argumentation Cluster"—that reflect persistent skill gaps.

3. Personalized Micro-Task Generation: Based on the identified clusters, the system shall automatically generate personalized micro-tasks. Each task must specify a clear training objective, provide canonical exemplars, and define transparent scoring criteria. Generated tasks shall be pushed to the user's learning dashboard for engagement.

4. Adaptive Task Sequencing via A/B Testing: The system shall monitor user performance on assigned micro-tasks, including completion rates and metric improvements (e.g., TTR increase, Logic Score gain). Using A/B testing and reinforcement learning principles, it shall dynamically adjust the difficulty level and task type for future recommendations, thereby establishing a closed-loop, data-driven cycle of "Diagnose $\rightarrow$ Train $\rightarrow$ Improve".

3.3.2 Functional Requirements

1. **FR-3.1: Multi-dimensional Feature Extraction**

    - **Trigger Mechanism:** Upon every essay submission, the system shall automatically trigger the feature extraction pipeline.
    - **Metric Calculation:** The system shall quantify writing quality using three core metrics:
        - *TTR (Type-Token Ratio):* To measure vocabulary diversity.
        - *MLU (Mean Length of Utterance):* To measure syntactic complexity.
        - *Logic Consistency Score:* Derived from the embedding analysis in Module 2.
    - **Data Archiving:** All quantitative metrics shall be time-stamped and archived in a **Time-series Database** to enable longitudinal trend analysis (e.g., "Vocabulary Growth Curve").

2. **FR-3.2: Weakness Clustering & Task Generation**

    - **Pattern Recognition:** The system shall periodically (e.g., weekly) execute clustering algorithms on the user's historical error logs and low-score paragraphs to identify high-frequency error patterns (e.g., a "Loose Argumentation Cluster").
    - **Task Generation:** Based on the identified clusters, the system shall automatically generate targeted **Micro-tasks**. Each task must include a specific training goal, standard examples, and automated scoring rules.
    - **Feedback Loop:** The system shall track the user's performance in these tasks and use A/B testing logic to dynamically adjust the difficulty of subsequent tasks.

3.3.3 Use Case Analysis

The functional scope of the Deep Profiling module is visualized in Figure 8. Note that most actions in this module are triggered automatically by the system.

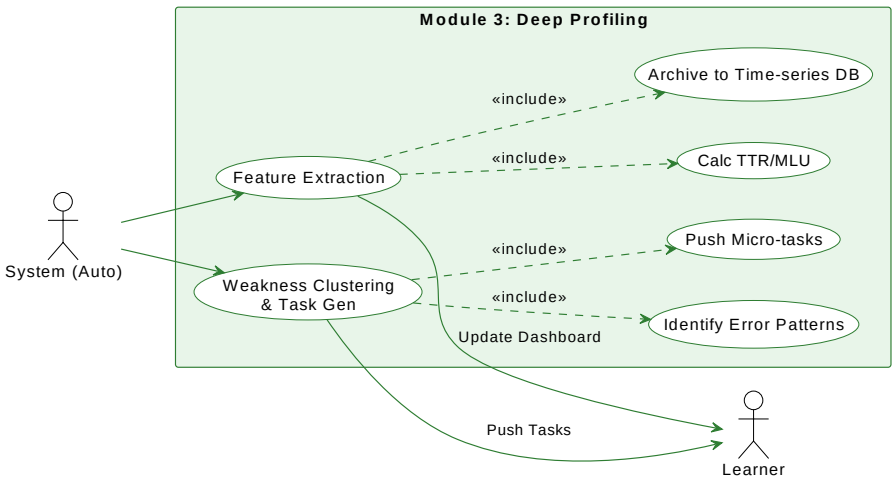

Figure 8: Use Case Diagram: Deep Profiling Module

The following tables detail the automation flows for Feature Extraction and Task Generation.

Use Case Name	**Multi-dimensional Feature Extraction**
Brief Description	The system automatically extracts quantitative metrics from the user's submission to update their longitudinal profile.
Primary Actor	System (Automatic Trigger)
Preconditions	User has successfully submitted an essay.
Basic Flow	1. System detects a new submission event. 2. System calculates **TTR** (Vocabulary Diversity). 3. System calculates **MLU** (Syntactic Complexity). 4. System retrieves the Logic Score from Module 2. 5. System archives all metrics into the Time-series Database. 6. System updates the visualization on the user's Dashboard. 7. User views the updated trend charts.
Alternative Flow	**2a.** If text length is insufficient (< 50 words), metrics calculation is skipped, and a warning is logged. **6a.** If this is the first submission, the system establishes a baseline profile.
Postconditions	User profile is updated with a new time-stamped record.

Table 5: Use Case: Multi-dimensional Feature Extraction

Use Case Name	**Weakness Clustering & Task Generation**
Brief Description	The system periodically analyzes historical error patterns to generate personalized training tasks.
Primary Actor	System (Periodic Trigger)
Preconditions	User has accumulated at least one week of writing history.
Basic Flow	1. System triggers the weekly analysis job. 2. System clusters historical errors and low-score paragraphs. 3. System identifies the primary weakness type (e.g., "Tense Errors"). 4. System generates targeted Micro-tasks (with goals and examples). 5. System pushes the task list to the user. 6. User completes the tasks. 7. System tracks completion and evaluates improvement.
Alternative Flow	**2a.** If data is insufficient, the analysis is postponed to the next cycle. **6a.** User chooses to skip or delay the task.
Postconditions	Task records are saved for A/B testing and efficacy validation.

Table 6: Use Case: Weakness Clustering & Task Generation

3.3.4 Data & Object Modeling

To address the requirement for static structural analysis, an "Object Diagram" is provided below (Figure 9). This diagram illustrates a snapshot of a 'UserProfile' object at runtime, showing how it relates to 'MetricLog' (historical data), 'ClusterResult' (analysis output), and 'MicroTask' (generated content).

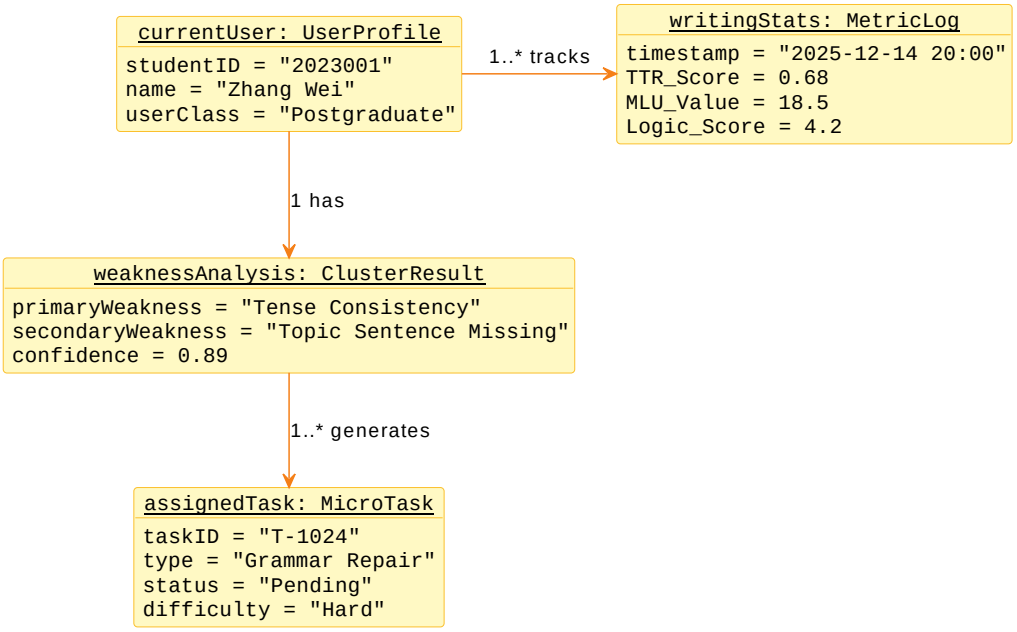

Figure 9: Object Diagram: User Profile Snapshot at Runtime

3.4 Module 4: Intelligent State Monitoring

3.4.1 Description

The Intelligent State Monitoring module shall fulfill the following functional requirements:

1. Local-Only State Monitoring: The system shall utilize browser APIs and on-device computer vision algorithms to monitor user interface activity (e.g., tab-switching frequency) and facial fatigue indicators, ensuring that all video streams are processed exclusively on the client device to preserve user privacy.

2. Real-Time Fatigue and Distraction Detection: The system shall compute PERCLOS (Percentage of Eyelid Closure over the Pupil) and attention-level metrics in real time, and automatically flag the current learning state as abnormal when these metrics continuously exceed predefined fatigue or distraction thresholds.

3. Adaptive Intervention Triggers: Upon detecting an abnormal state, the system shall automatically trigger adaptive interventions—such as subtle UI cues, mandatory rest suggestion modals, or temporary suspension of RAG-based recommendation services—to prevent ineffective learning under fatigue.

4. Intervention Logging and Feedback: The system shall log all intervention events and user responses, and generate a cognitive energy management log to provide data-driven recommendations for optimizing the user's study rhythm.

3.4.2 Functional Requirements

1. **FR-4.1: Privacy-Preserving Real-time Sensing**

- **Local Processing:** The system shall access the client's webcam stream and execute lightweight inference models (e.g., MediaPipe) directly in the browser environment. No raw video data shall be uploaded to the server.
- **Fatigue Detection (PERCLOS):** The system shall calculate the **PERCLOS** (Percentage of Eyelid Closure) index every 30 seconds. A PERCLOS value exceeding the threshold (e.g., 0.4) shall be flagged as "High Fatigue."
- **Distraction Detection:** The system shall monitor gaze direction and head pose. Sustained gaze deviation ($>$ 10s) or absence from the camera view shall be recorded as "Distracted."

2. **FR-4.2: Adaptive Intervention Mechanism**

- **Level 1 Intervention (Soft):** Upon detecting mild distraction, the system shall trigger a subtle "Breathing Light" effect on the editor borders to gently redirect attention.
- **Level 2 Intervention (Hard):** Upon detecting high fatigue or prolonged absence ($>$ 5 mins), the system shall trigger a modal dialog suggesting a "5-minute Pomodoro Break" and temporarily suspend the RAG recommendation service to prevent inefficient learning.
- **Intervention Logging:** The system shall record the frequency of interventions and the user's subsequent response (e.g., "Accepted Break" or "Dismissed") to optimize future intervention strategies.

3.4.3 Use Case Analysis

The functional scope of the Intelligent Monitoring module is visualized in Figure 10.

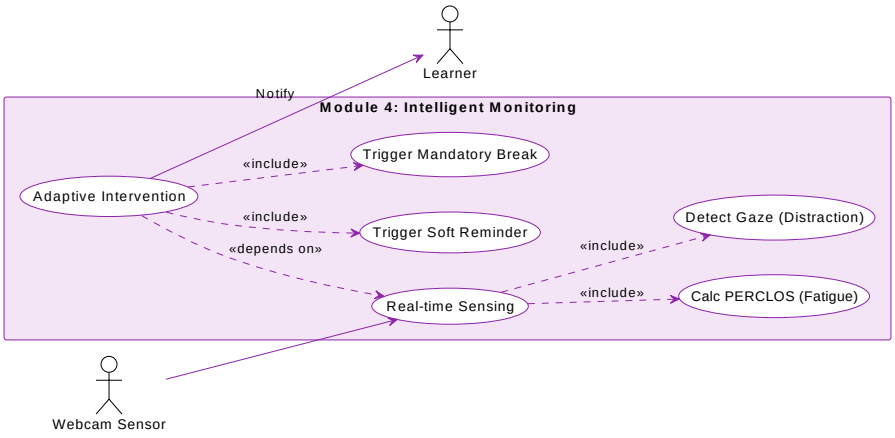

Figure 10: Use Case Diagram: Intelligent State Monitoring

The following tables detail the interaction flows for Sensing and Intervention.

Use Case Name	**Privacy-Preserving Real-time Sensing**
Brief Description	The system uses edge-side AI to quantify user fatigue (PERCLOS) and attention levels without transmitting video data.
Primary Actor	Webcam Sensor
Preconditions	User has authorized camera access.
Basic Flow	1. System initializes the local MediaPipe vision model. 2. System processes local video stream at 10 fps. 3. System extracts eyelid keypoints to calculate PERCLOS index. 4. System monitors gaze direction and tab switching frequency. 5. System generates a status vector (e.g., {`fatigue: 0.2, attention: 0.9`}) every 30 seconds. 6. System writes the desensitized vector to the local session log.
Alternative Flow	**1a.** (Permission Denied) If camera access is refused, the system degrades to mouse/keyboard activity tracking. **2a.** (Low Confidence) If lighting is poor, sensing is temporarily paused.
Postconditions	Real-time status vector is updated for the Intervention module.

Table 7: Use Case: Real-time Sensing

Use Case Name	**Adaptive Intervention Trigger**
Brief Description	Based on sensing data, the system triggers multi-level interventions when fatigue or distraction is detected.
Primary Actor	System
Preconditions	Real-time sensing module is active.
Basic Flow	1. System detects an anomaly in the status vector. 2. **Level 1 (Distraction):** If gaze deviates $> 10s$, trigger ”Ambient Breathing Light” on UI borders. 3. **Level 2 (Fatigue):** If PERCLOS > 0.4, trigger ”Mandatory Break” modal. 4. User chooses ”Take a Break” or ”I’m Fine.” 5. If ”Break” is selected, system suspends RAG services and starts a 5-minute timer. 6. After the break, system restores all functions.
Alternative Flow	**4a.** User selects ”Snooze,” silencing interventions for 15 minutes.
Postconditions	Intervention event is logged for habit analysis.

Table 8: Use Case: Adaptive Intervention

3.4.4 State Modeling

To address the requirement for dynamic behavioral analysis, a State Transition Diagram (Figure 11) is provided. This model defines the lifecycle of a user’s cognitive state within the system, illustrating how the system transitions users between ”Focused,” ”Distracted,” ”Fatigued,” and ”Resting” states based on sensor inputs and interaction events.

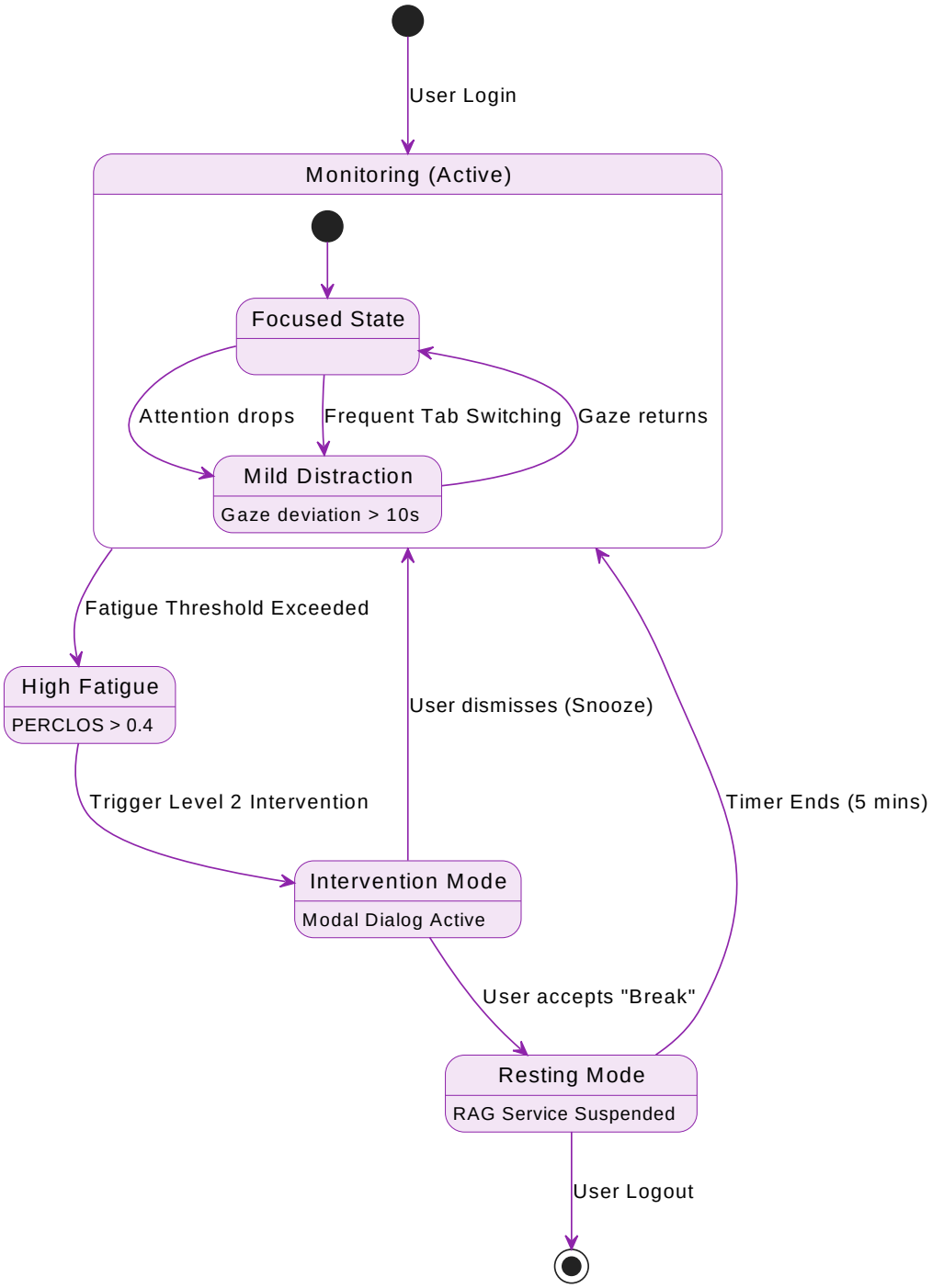

Figure 11: State Diagram: User Cognitive State Transitions

4 External Interface Requirements

4.1 User Interfaces

The user interface shall be implemented as a responsive Web application, strictly adhering to the following standards:

- **Technology Stack:** The frontend shall be built using **Vue 3** and the **Element Plus** component library to ensure consistency with the main LearnLoop platform.
- **Layout Design:** The editor interface shall feature a "Three-Column Layout": a collapsible "Logic Skeleton Tree" sidebar on the left, a central distraction-free writing area, and a floating "Recommendation Panel" on the right.
- **Visual Standards:** The system must support **Dark Mode** to reduce visual fatigue during high-intensity writing sessions.
- **Feedback Mechanism:** All AI-generated modifications (e.g., syntactic refactoring) must utilize **diff-highlighting** (e.g., Red strikethrough for deletions, Green background for insertions) to provide clear visual cues.

4.2 Hardware Interfaces

- **Client-side Webcam:** The system interfaces with the client device's camera via the browser's `MediaStream API`. It must handle hardware permission states (Granted/Denied) gracefully, automatically disabling state monitoring features if hardware access is unavailable.
- **Server-side GPU:** The backend inference engine interfaces with NVIDIA GPUs (e.g., A10 or T4) via **CUDA 11+** drivers to accelerate PyTorch-based LLM inference and vector embedding calculations.

4.3 Software Interfaces

- **LMS Platform Integration:** The system shall interface with the *LearnLoop* main platform using **OAuth 2.0** for Single Sign-On (SSO) authentication and to retrieve basic user profile data.
- **LLM Gateway:** The system shall not rely on a single model provider. Instead, it shall interface with an internal **Model Serving Layer** via RESTful APIs, allowing for the hot-swapping of underlying models (e.g., switching between GPT-4 and open-source LLaMA 3).
- **Database Interfaces:**
  - **PostgreSQL:** Accessed via SQLAlchemy (ORM) for persistent storage of user logs and profiles.
  - **Vector Database:** Interfaces with **pgvector** or **Pinecone** for high-dimensional semantic retrieval.

4.4 Communications Interfaces

- **Communication Protocols:** All client-server data transmission shall occur over HTTPS using TLS 1.2 or higher encryption standards to ensure data security.
- **Data Serialization:** All API payloads shall be formatted in strict JSON.
- **API Specification:** The interface definitions shall adhere to the OpenAPI 3.0 standard to facilitate automated documentation and client SDK generation.

5 Non-functional Requirements

5.1 Performance Requirements

The system must meet strict performance benchmarks to ensure a smooth user experience, particularly for real-time interactions.

- **Latency Targets:**
  - The `/suggest` inference endpoint (Instant Assistance) must maintain a P95 latency of ≤ 1.5 seconds and a P99 latency of ≤ 2.5 seconds under standard load.
  - The frontend rendering latency for the recommendation panel shall be ≤ 150 milliseconds.
- **Throughput & Concurrency:**
  - The system shall support at least 100 concurrent users per instance.
  - The system shall handle a sustained load of 50 Requests Per Second (RPS) for the inference API without degradation.
- **Resource Constraints:**
  - Under nominal load, the CPU utilization of single service instances shall not exceed 70%, and memory utilization shall not exceed 75%.

5.2 Safety Requirements

Safety requirements ensure the system handles failures gracefully without data loss or catastrophic errors.

- **Automated Failover (Degradation Strategy):**
  - If the primary LLM service experiences an error rate $> 1\%$ for over 1 minute, the system shall automatically downgrade to a BM25-based keyword retrieval mode within 5 seconds.

  - Once the LLM service recovers, the system shall automatically restore full functionality.

- **Data Safety:**
  - An automated Edit History Stack must be maintained locally. Users must be able to Undo/Redo any AI-applied changes to prevent accidental loss of original text.

5.3 Security Requirements

The system must strictly adhere to security standards to protect user data and intellectual property.

- **Encryption Standards:**
  - **Transmission:** All data in transit must be encrypted using TLS 1.2+.
  - **Storage:** Static data (e.g., user drafts, profiles) must be encrypted at rest using AES-256.
- **Access Control:**
  - Strict Role-Based Access Control (RBAC) shall be enforced. Critical operations (e.g., Corpus Management) are restricted to "Admin" roles only.
  - API Keys and Secrets must be injected via environment variables and never hard-coded in the source code.
- **Privacy Compliance:**
  - In compliance with the "Edge-Cloud" privacy architecture, raw video data from state monitoring must be processed locally and is strictly prohibited from being uploaded to the server.

5.4 Software Quality Attributes

- **Availability:** The system targets a Service Level Objective (SLO) of 99.0% monthly availability.
- **Maintainability:**
  - The codebase must adhere to standard linting rules.
  - Core service modules must maintain a unit test coverage of $\geq 70\%$.
- **Scalability:**
  - The stateless API layer shall support horizontal scaling via Kubernetes Horizontal Pod Autoscaler (HPA) to handle traffic bursts (e.g., 3x baseline load).

6 System Analysis Models

This section provides a set of visual models to detail the system's architecture, data structures, and dynamic behaviors, strictly following the Object-Oriented Analysis and Design (OOAD) methodology.

6.1 System Architecture (Package Diagram)

The **Package Diagram** (Figure 12) illustrates the high-level layered architecture of the WriteLoop system. It decouples the application into four distinct layers: Presentation, Service, Data Access, and External Interface, ensuring modularity and maintainability.

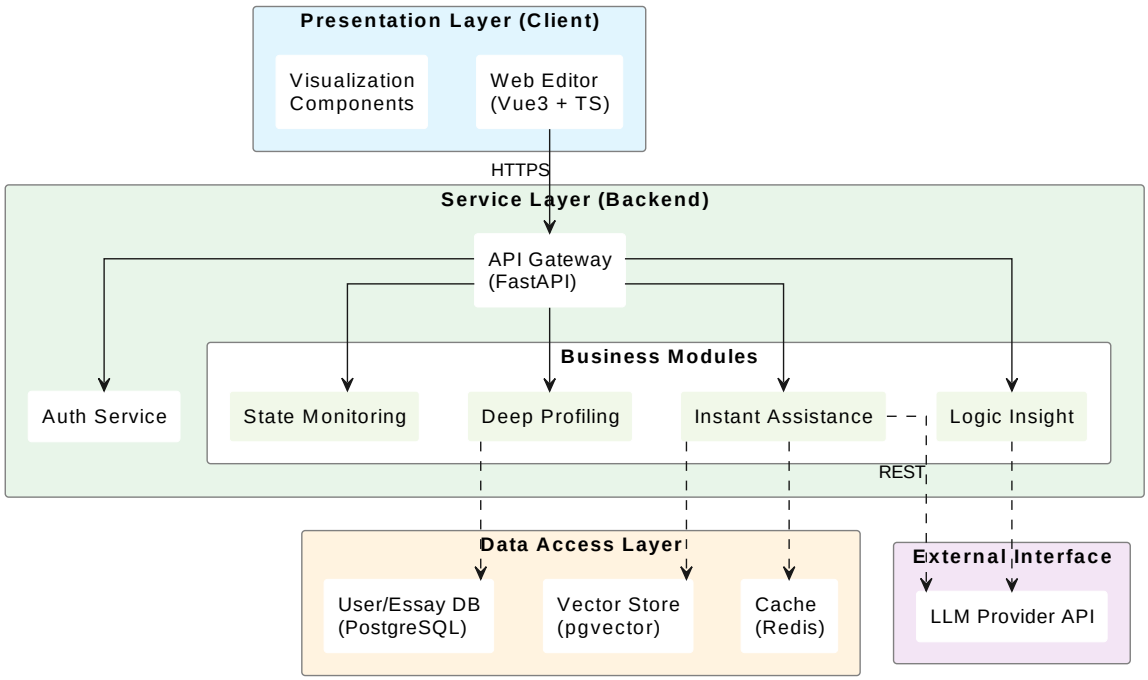

Figure 12: System Architecture Package Diagram

6.2 Domain Model (Class Diagram)

The **Class Diagram** (Figure 13) depicts the static structure of the core domain entities. It defines the relationships between the `User`, their generated `Essays`, the associated `UserProfile` metrics, and the AI-generated `Suggestions`.

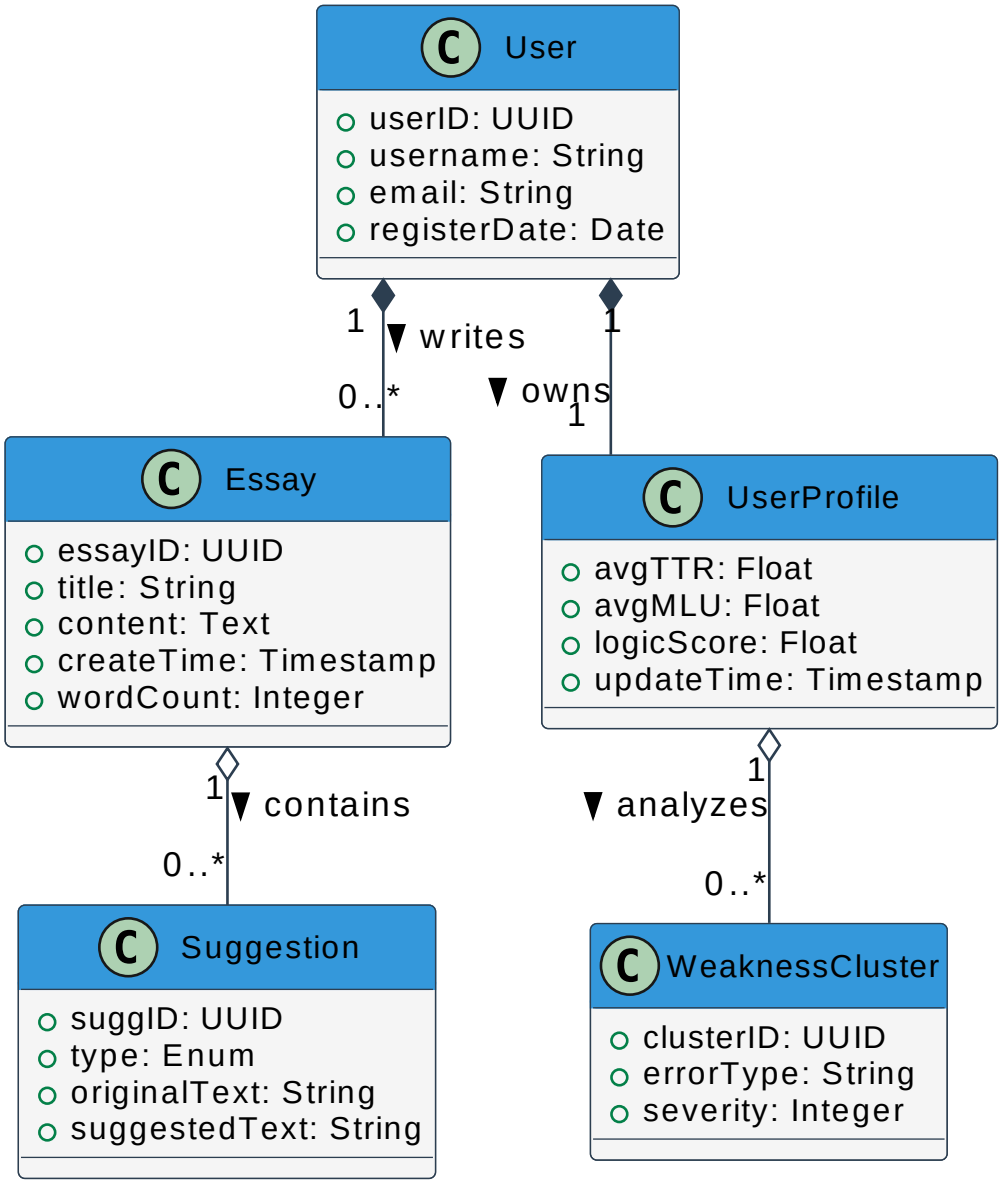

Figure 13: Domain Class Diagram

6.3 Dynamic Interaction (Sequence Diagram)

While Chapter 3 detailed specific functional flows, the Sequence Diagram below (Figure 14) illustrates the **macro-level interaction** for a full "Essay Submission and Analysis" cycle. It demonstrates how multiple subsystems coordinate asynchronously to process a user's submission.

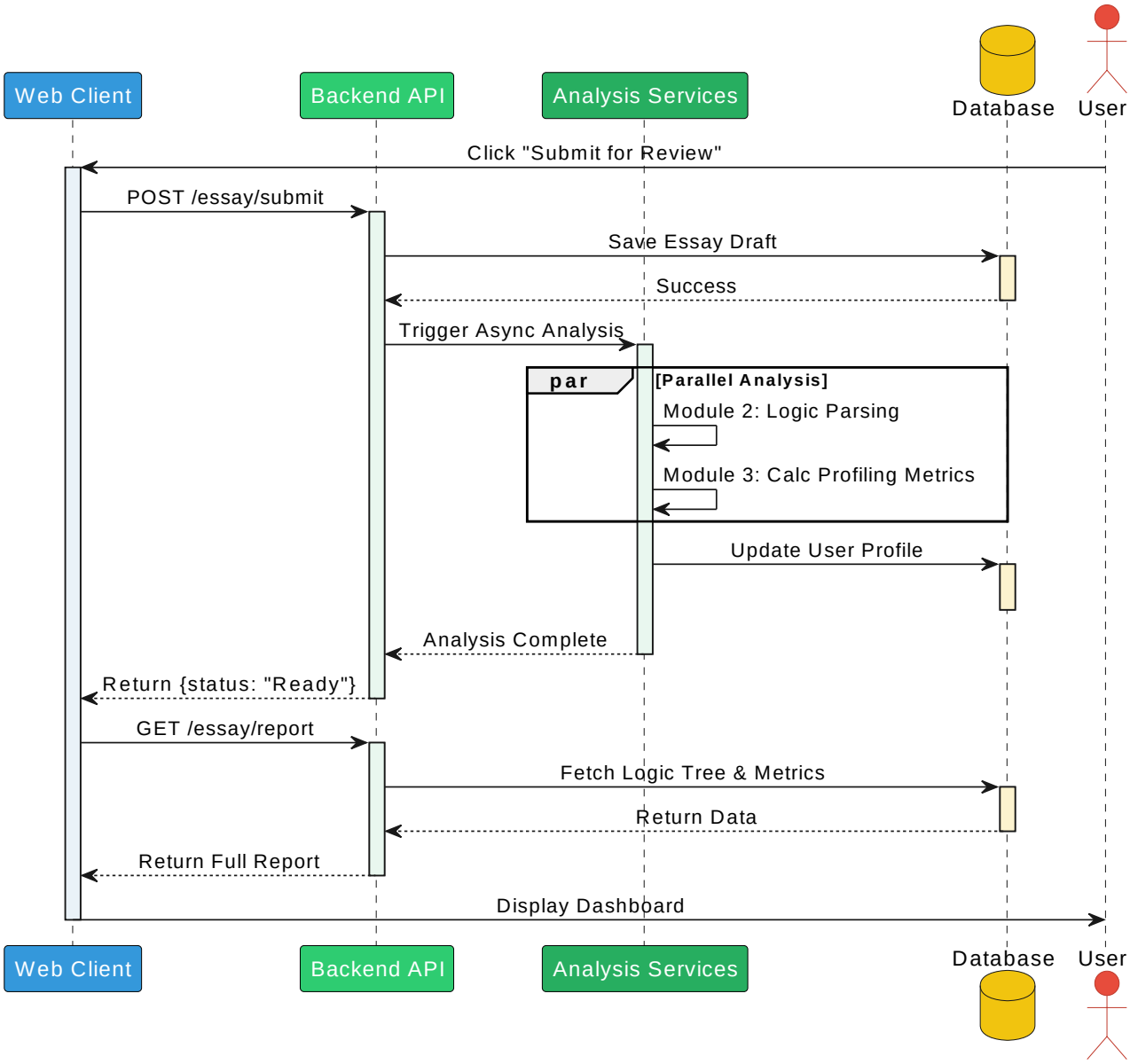

Figure 14: Sequence Diagram: Full Essay Submission Flow

6.4 State Modeling (State Diagram)

The State Diagram (Figure 15) models the lifecycle of an `Essay` object. It defines the valid transitions between "Drafting," "Analyzing," and "Reviewed" states, ensuring data consistency during the asynchronous AI inference process.

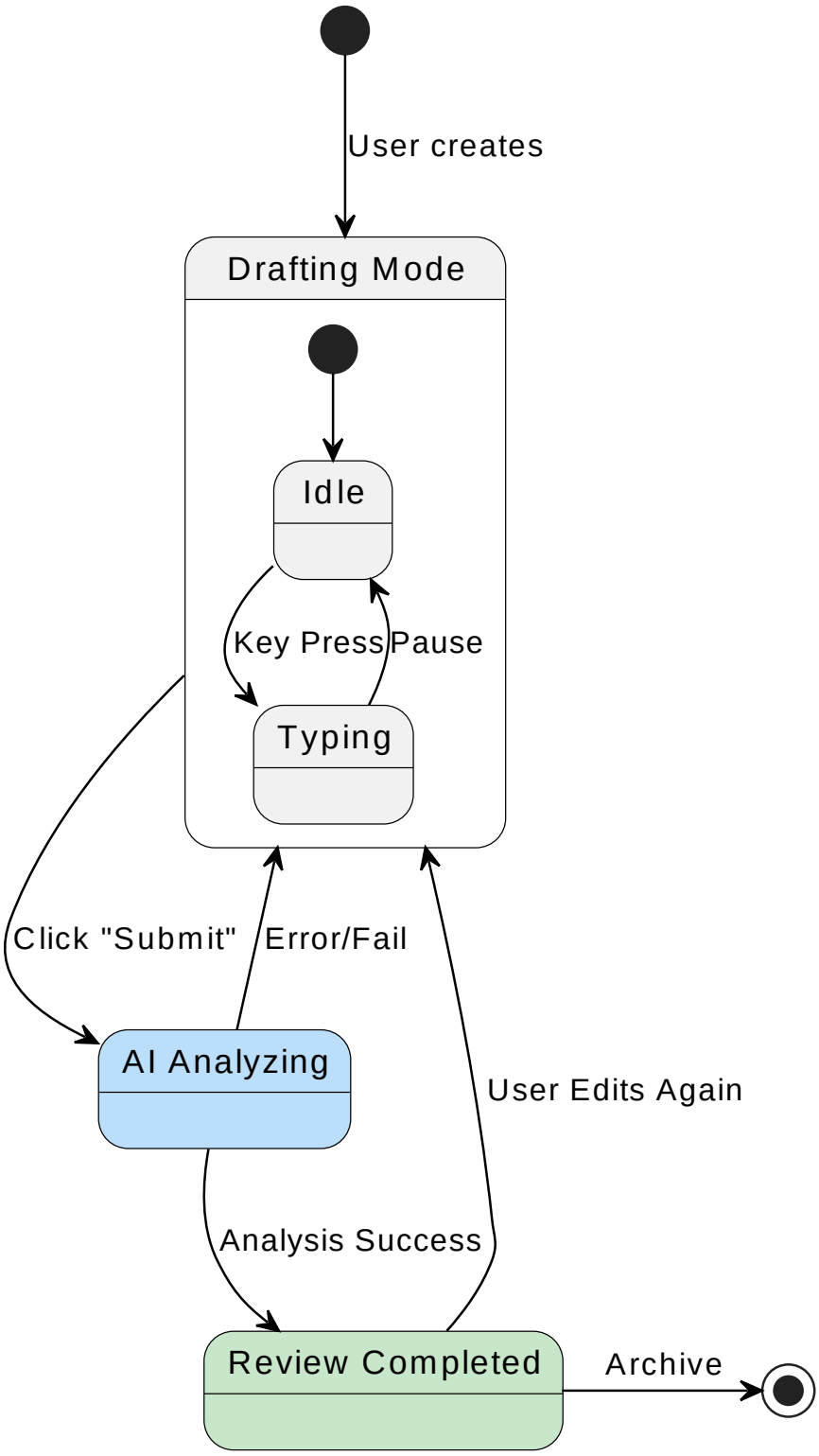

Figure 15: State Diagram: Essay Object Lifecycle

6.5 Deployment Model (Deployment Diagram)

To address the physical deployment requirements, the Deployment Diagram (Figure 16) illustrates the hardware/software mapping. It explicitly shows the ”Edge-Cloud” separation,

where the Webcam and MediaPipe models run on the Client Node to preserve privacy, while heavy LLM inference occurs on the Server Node equipped with GPU acceleration.

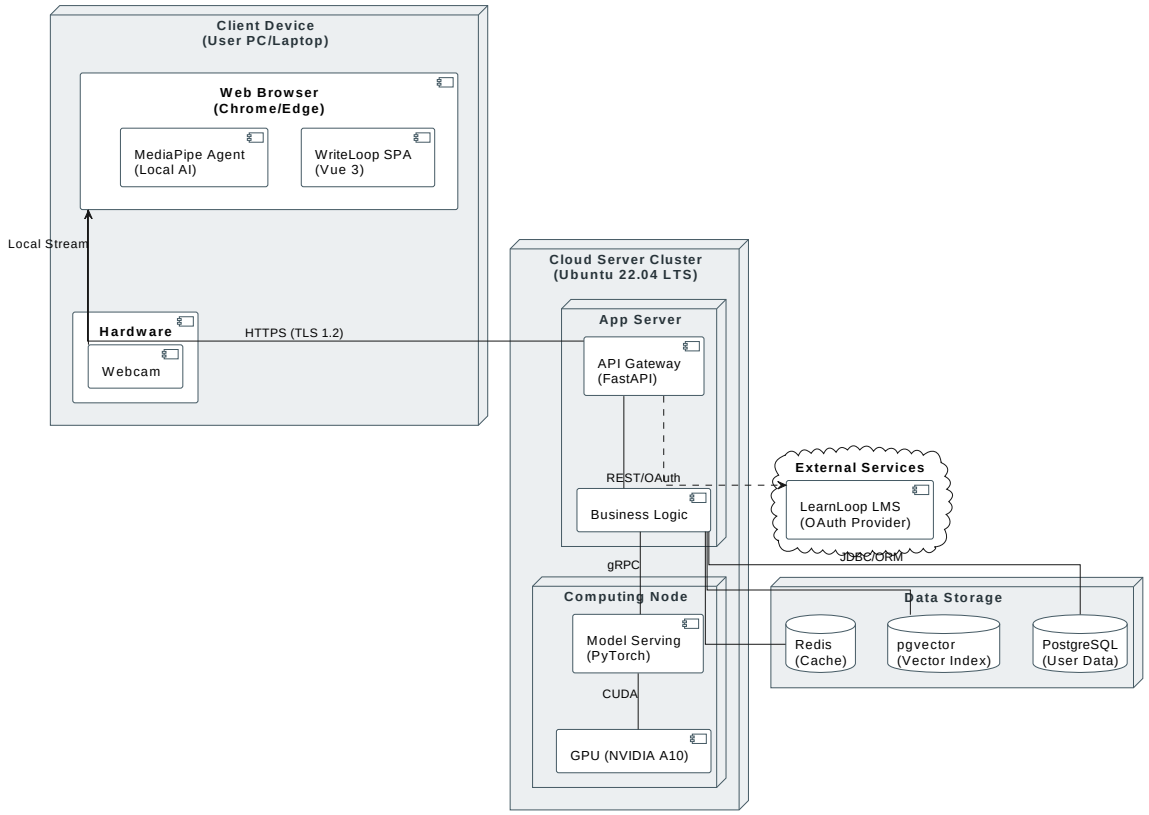

Figure 16: System Deployment Diagram

A Glossary

The following table provides definitions for the technical terms and acronyms used throughout this Software Requirements Specification.

Term/Acronym	Definition
RAG	**Retrieval-Augmented Generation**. A technique that enhances AI generation accuracy by retrieving relevant information from an external knowledge base (e.g., user reading history) before generating a response.
LLM	**Large Language Model**. A deep learning model (e.g., GPT-4) capable of understanding and generating human-like text based on vast training data.
TTR	**Type-Token Ratio**. A linguistic metric used to measure vocabulary diversity by calculating the ratio of unique words (types) to the total number of words (tokens).
MLU	**Mean Length of Utterance**. A metric indicating syntactic complexity, calculated by the average number of words per sentence.
PERCLOS	**Percentage of Eyelid Closure**. A physiological metric used to detect fatigue, defined as the proportion of time the eyes are more than 80% closed within a specific interval.
Edge AI	**Edge Artificial Intelligence**. The deployment of AI algorithms locally on the user's device (e.g., browser) rather than on a remote cloud server, ensuring data privacy and low latency.
Embeddings	Dense vector representations of text where semantically similar text segments are located close to each other in the vector space. Used for the Logic Insight module.

Table 9: Glossary of Terms

B System Constraints & Configuration Policies

This section defines the finalized configuration policies and operational constraints for the system. These definitions serve as the baseline for the MVP implementation, replacing previous TBD items with definitive engineering strategies.

B.1 Physiological Threshold Configuration (Fatigue Detection)

- **Policy Description:** To address individual differences in fatigue manifestation, the system adopts a **Configurable Threshold Strategy** rather than a hard-coded value.

- **Default Configuration:** The initial PERCLOS (Percentage of Eyelid Closure) threshold for triggering a "Level 2 Intervention" (Mandatory Break) is set to **0.4** by default. This value is derived from standard driver fatigue detection literature.
- **Admin Control:** The Backend Management System shall provide a configuration panel allowing authorized Administrators to adjust this global threshold within the safety range of **[0.2, 0.6]** without requiring code redeployment.

B.2 Data Retention & Privacy Compliance

- **Policy Description:** To strictly adhere to GDPR and local Data Security Laws, the system enforces a strict "Volatile vs. Persistent" data lifecycle policy.
- **Volatile Local Data (Edge-Side):** Raw video streams and intermediate feature vectors (e.g., facial landmarks) processed by the "Intelligent State Monitoring" module are classified as volatile. They must be stored **in-memory only** and are automatically purged immediately upon the termination of the browser session or after a rolling window of **24 hours**, whichever comes first.
- **Persistent User Data (Cloud-Side):** User writing drafts and generated metric logs (TTR, Logic Scores) are retained indefinitely to support longitudinal analysis (Feature FR-3.1). However, upon a user's explicit request for account deletion, all associated personal data must be physically removed from the database within **7 days**.

B.3 Network Degradation & Offline Strategy

- **Policy Description:** The system implements a **"Graceful Degradation"** strategy to handle network instability, prioritizing data safety over full AI functionality.
- **Offline Capability:** In the event of internet connection loss, the Web Editor shall remain functional for basic text input and local editing. All changes will be cached locally in the browser.
- **Feature Suspension:** Bandwidth-heavy or cloud-dependent features—specifically **Instant Assistance (RAG)** and **Logic Insight (LLM)**—will be automatically suspended. The UI shall display a "Reconnecting..." status indicator to inform the user that AI services are temporarily unavailable.
- **Auto-Recovery:** Once the network connection is restored, the system shall automatically synchronize the local cache with the cloud database and re-enable AI services.