SOFTWARE DESIGN DESCRIPTION (SDD)

WriteLoop Adaptive Writing Coach System

Prepared by:

Team 26

Name	Student ID
Xin Gao	2251356
Zhuobing Zhao	2252750
Jialu Wang	2351890
Yining Sun	2352736

January 2, 2026

Contents

1 Overview

1.1 Scope

The scope of this Software Design Description(SDD) is limited to the technical design of the WriteLoop Adaptive Writing Coach, a specialized intelligent support module integrated within the LearnLoop educational platform. This document establishes the information content and organization for the technical implementation of the system's four core modules:

- **Instant Writing Assistance:** Design for context-aware vocabulary and synatic recommendation engines using Retrieval-Augmented Generaiton.
- **Logic Insight:** Design for structural parsing, argumentative framework visualization, and coherence heatmap genneration.
- **Deep Profiling & Adaptive Tasks:** Design for time-series tracking of writing metrics and clustering algorithms for personalized task generation.
- **Intelligent State Monitoring:** Design for privacy-preserving, edge-side multimodal sensing for fatigue and attention detection.

This design is strictly focused on the upper-layer application of AI technologies for writing education. It explicitly excludes the development of underlying foundation models or general learning management system functionalities unrelated to writing. The SDD is intended for use in traditional software construction where design leads directly to implementation.

1.2 Purpose

The primary purpose of this Software Design Description is to specify the technical requirements, information content, and organization of the WriteLoop system to ensure it functions as an intelligent tutoring system capable of understanding both the "text" and the "learner". This document explicitly translates the gaps identified in traditional instruction—specifically in logical diagnosis and cognitive state management—into a robust technical architecture while specifying the requirements for design languages and viewpoints used to organize the system. Ultimately, it provides a standardized representation of the software design to effectively record and communicate vital technical details to both technical and managerial stakeholders throughout the development lifecycle.

1.3 Intended Audience

This document is intended for technical and managerial stakeholders involved in the development and maintenance of the WriteLoop system. The audience includes:

- **Software Designer and Architects:** To guide the necessary technical logic for coding the AI and presentation of design information.
- **Programmers and Developers:** To provide the necessary technical logic for coding the AI and edge-processing modules.
- **Project Managers:** To ensure the design is consistent, interchange-able, and well-organized for team coordination.
- **Quality Assurance Staff and Testers:** To evaluate the design against requirements and prepare test documentation.

1.4 Conformance

This SDD conforms to the IEEE Std 1016-2009 standard by satisfying all requirements defined in Clause 4 and Clause 5. All mandatory requirements with this document are denoted by the verb "shall".

2 Definitions

This section provides definitions for the terms used within this software Design Description.

2.1 Standard Design Terms

According to IEEE Std 1016-2009, the following terms describe the structural components of this designs:

- **Design Attribute:** An element of a design view that names a characteristic or property of a design entity, design relationship, or design constraint.
- **Design Concern:** An area of interest with respect to a software design, such as functionality, reliability, or performance.
- **Design Constraint:** An element of a design view that names and specifies a rule or restriction on a design entity, design attribute, or design relationship.
- **Design Element:** An item occurring in a design view, including design entities, relationships, attributes, or constraints.
- **Design Entity:** An element of a design view that is structurally, functionally, or otherwise distinct from other elements.
- **Design Rationale:** Information capturing the reasoning of the designer, including design options, trade-offs, and justifications for decisions made.

- **Design Relationship:** An element of a design view that names a connection or correspondence between design entities.
- **Design Stakeholder:** An individual or group having an interest in, or design concerns relative to, the software item.
- **Design View:** A representation comprised of one or more design elements to address a set of design concerns from a specified viewpoint.
- **Design Viewpoint:** The specification of the elements and conventions available for constructing and using a design view.

2.2 System-Specific Technical Terms

The following terms and acronyms define the core technologies and metrics employed by the WriteLoop system:

- **RAG:** Retrieval-Augmented Generation. A technique that enhances AI generation accuracy by retrieving relevant information from an external knowledge base, such as user reading history, before generating a response.
- **LLM:** Large Language Model. Large-scale pre-trained models capable of advanced natural language understanding and generation tasks.
- **TTR:** Type-Token Ratio. 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:** Mean Length of Utterance. 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. A standard physiological metric used to quantify fatigue levels based on the duration or proportion of eye closure over the pupil
- **Edge AI:** The deployment of AI algorithms locally on the user's device (e.g., browser) rather than a remote cloud server to ensure privacy and low latency.

3 System Architecture Design

This section defines the high-level architectural style, physical deployment topology, and the operational context of the WriteLoop system. The architecture is designed to address the critical quality attributes specified in the SRS, specifically the requirements for low-latency inference, strict user privacy regarding video data, and seamless integration with the LearnLoop ecosystem.

3.1 System Logical Architecture

The WriteLoop system is structured using a strict Layered Architecture pattern, which enforces a separation of concerns between the user interface, business logic, and data persistence mechanisms. This logical separation ensures modularity and facilitates independent maintenance of the frontend and backend subsystems.

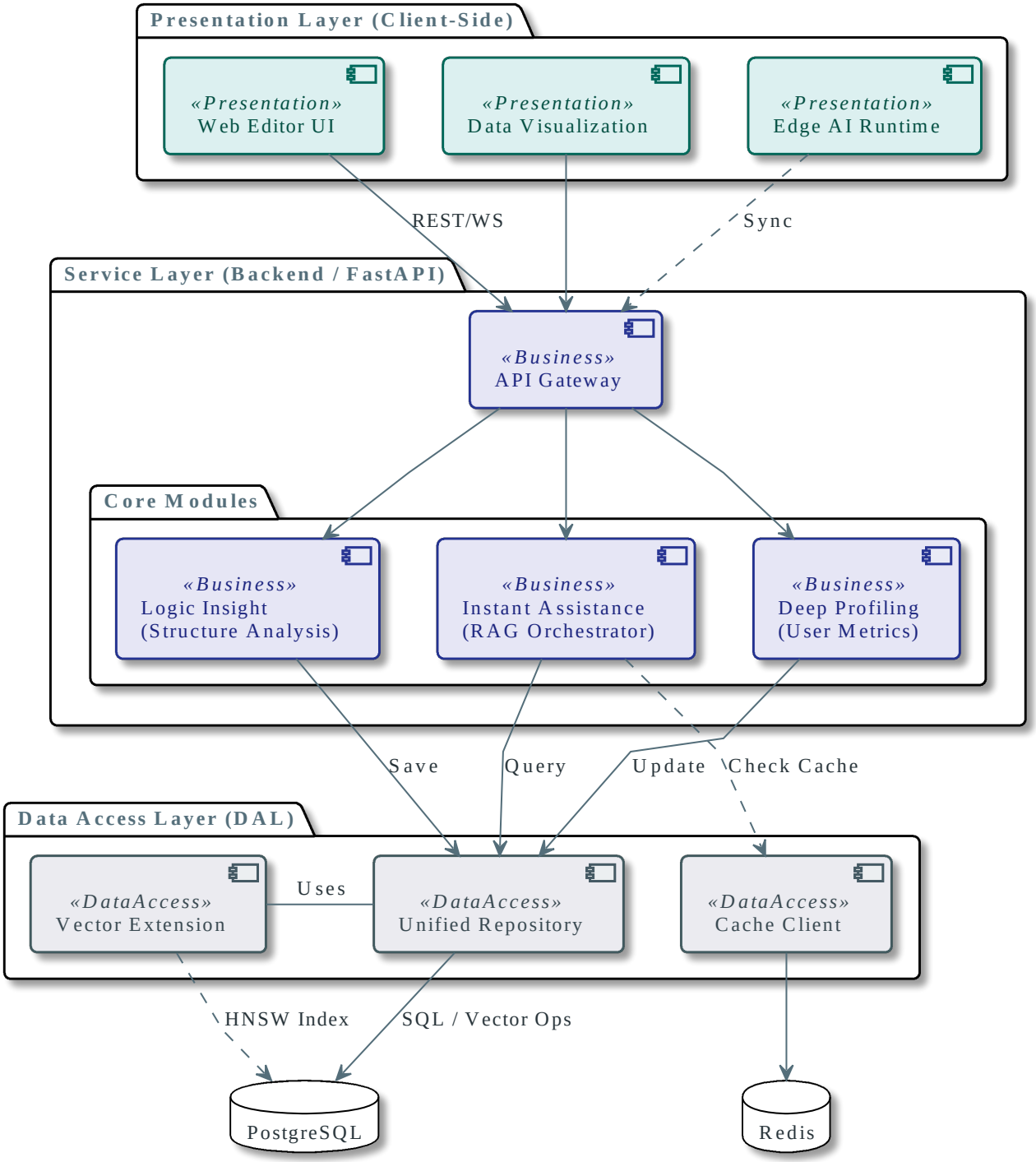

Figure 1: System Logical Architecture

Presentation Layer (Client-Side): Implemented as a Single Page Application using Vue 3 and TypeScript, this layer ensures interface responsiveness and type safety. It encapsulates the Web Editor for capturing real-time user interactions and integrates a MediaPipe-driven Edge AI Agent. By executing lightweight computer vision inference locally to derive PERCLOS and attention metrics, this design obviates the need for raw video transmission, strictly adhering to the privacy constraints mandated by the SRS.

Service Layer (Server-Side): Functioning as the backend core, this layer is exposed via a FastAPI Gateway that handles authentication, rate limiting, and request routing. It orches-

trates distinct business modules: the Instant Assistance Service manages the RAG pipeline for context retrieval; the Logic Insight Service leverages asynchronous task queues to parallelize structural parsing and heatmap generation; and the Deep Profiling Service executes batch processing of linguistic metrics and clustering algorithms for personalized user modeling.

Data Access Layer: This layer abstracts persistence through a hybrid storage strategy. It utilizes PostgreSQL accessed via an ORM for relational data while employing the pgvector extension to manage high-dimensional embeddings for efficient semantic search. To guarantee the system's sub-1.5s latency requirement, a Redis caching layer is integrated to manage transient session states and cache high-frequency retrieval results.

3.1.1 Functional Component Decomposition

Complementing the layered technical architecture, Figure 1.1 illustrates the system's structure from a functional decomposition perspective.

- Executive Control: The API Gateway functions as the central orchestrator, routing requests to specific functional domains.
- Core Subsystems: The business logic is encapsulated within four high-cohesion subsystems (Instant Assistance, Logic Insight, Deep Profiling, and State Monitoring). Each subsystem is further decomposed into atomic functional components.
- Separation of Concerns: This hierarchical structure explicitly separates the core algorithmic modules from external interface management, facilitating modular development and independent maintenance of specific features.

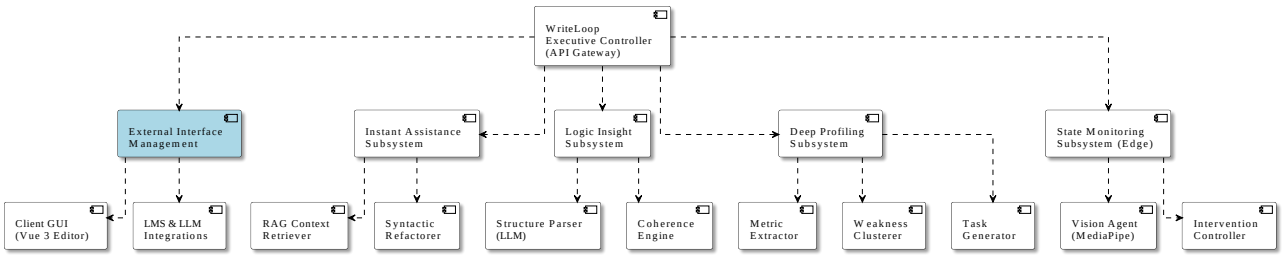

Figure 2: Functional Component Decomposition

3.2 Physical Deployment Architecture

The physical deployment of WriteLoop follows an Edge-Cloud Hybrid topology, explicitly mapped to optimize the trade-off between computational power and data privacy.

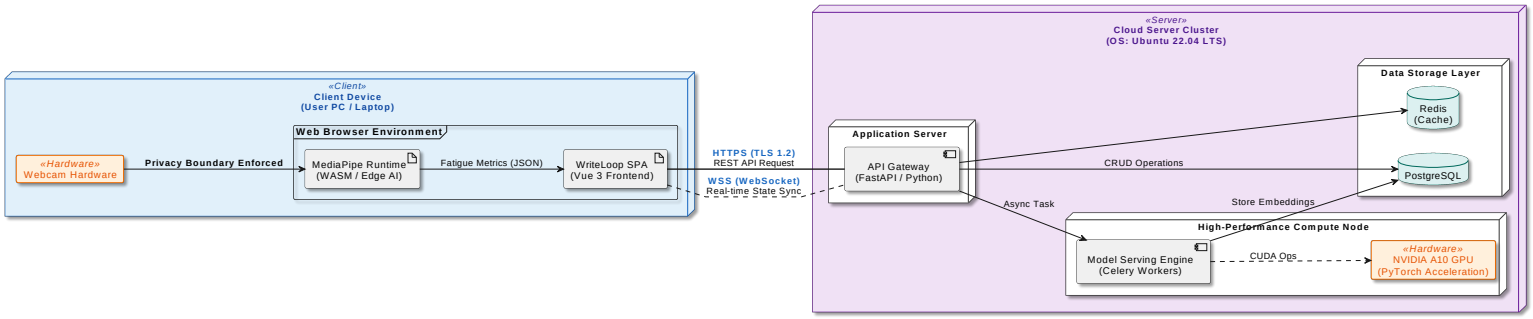

Figure 3: Physical Deployment Architecture

Client Computation Node (Edge): Operating within an edge computing paradigm, the Client Node executes the frontend and MediaPipe Agent within a browser environment. By interfacing directly with local hardware via the MediaStream API, it performs biometric analysis strictly in volatile memory. This architecture guarantees privacy by transmitting only derived scalar metrics, thereby precluding raw video egress.

Server Computation Node (Cloud): The Server Node comprises a high-performance Ubuntu cloud cluster leveraging NVIDIA A10 GPUs for accelerated PyTorch inference. It manages secure data transmission via TLS 1.2 and WebSocket protocols, while orchestrating data persistence through PostgreSQL and Redis to handle computationally intensive workloads.

External Integration Points: The system maintains secure interoperability with external services, synchronizing user data with LearnLoop LMS via OAuth 2.0. Additionally, it integrates with external LLM providers via RESTful APIs to offload generative tasks, employing a circuit-breaker pattern to mitigate dependency risks and ensure system resilience.

3.3 System Context and Boundaries

Figure 3 illustrates the operational boundaries of the WriteLoop system and its interactions with external entities. The system is depicted as a "Black Box" (Target System) to highlight its external interfaces and protocol dependencies.

Key architectural decisions regarding system boundaries include:

- External Dependencies: The system delegates identity management to the LearnLoop LMS via OAuth 2.0 (Identity Provider) and offloads generative inference tasks to an External LLM Service via RESTful APIs. This design ensures loose coupling with the school's existing infrastructure.
- Real-Time Interaction: Unlike traditional web apps, the interaction between the Learner and the system utilizes WebSocket (WSS) alongside standard HTTPS. This enables the server to push real-time feedback without client-side polling.
- Privacy Boundary: As shown in the diagram, the Webcam hardware interfaces directly with the browser via the MediaStream API. This strictly enforces the privacy requirement that raw video data remains within the client's execution environment and is never transmitted across the network boundary.

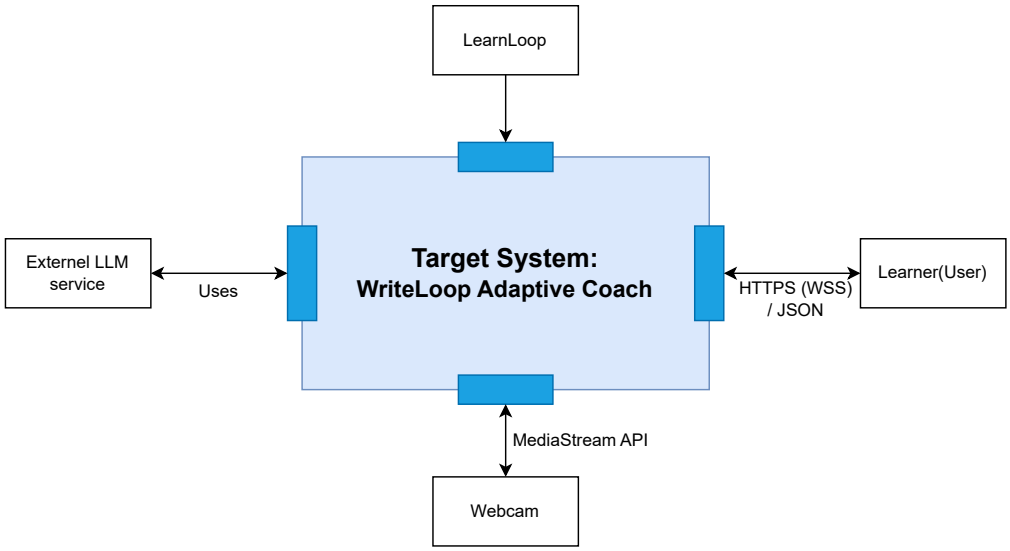

Figure 4: Architectural Context Diagram

The System Context Viewpoint defines the WriteLoop system as a specialized functional entity operating within the broader educational technology landscape. It treats the system as a "black box" to identify its external interfaces and dependencies.

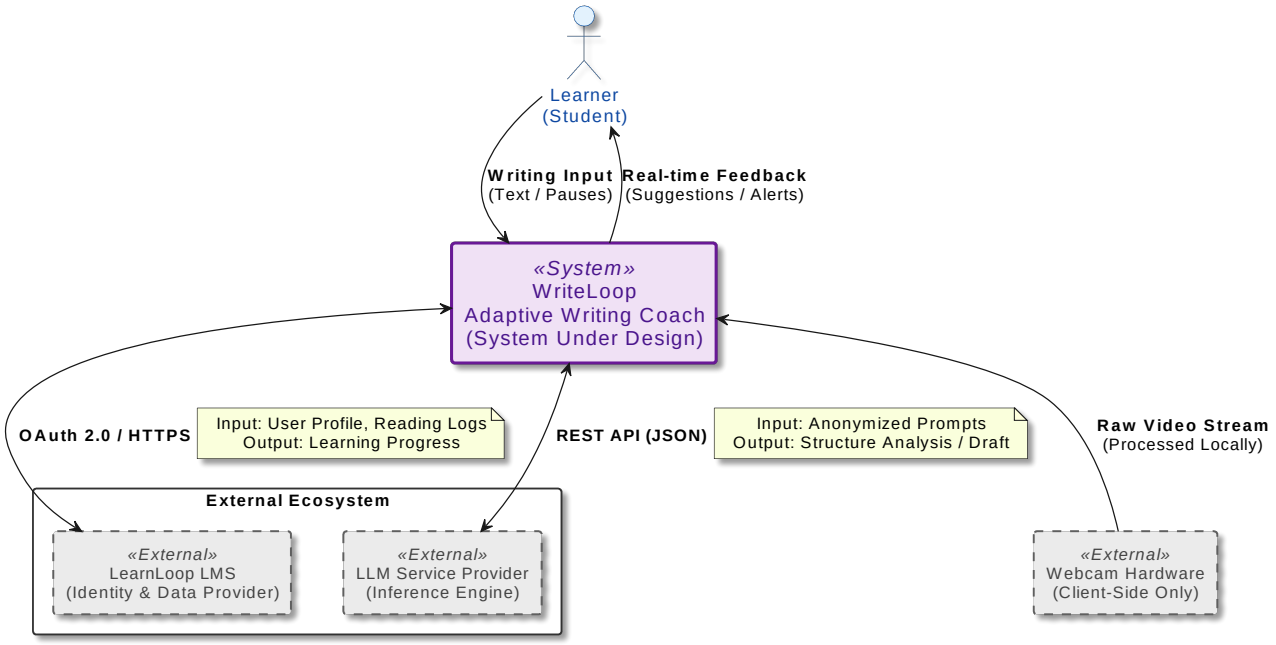

Figure 5: System Context Diagram

The system functions within a defined interaction boundary, engaging the Learner via the editor interface while integrating local webcam hardware within a strict browser sandbox to preserve privacy. Externally, it maintains interoperability with the LearnLoop LMS for identity federation and contextual data ingestion, and leverages external LLM services via a model-agnostic abstraction layer. This design supports model interchangeability and ensures fault tolerance, enabling the system to gracefully degrade to fundamental editing capabilities during external service interruptions.

4 Date Design

This chapter specifies the data architecture of the WriteLoop system. Given the adoption of PostgreSQL as the primary relational database, the system employs a Unified Storage Strategy: both structured transactional data and high-dimensional vector data are managed centrally by PostgreSQL. This architecture eliminates the need for an external vector search engine, ensuring strict transactional consistency and simplifying the infrastructure required for the RAG feature.

4.1 Conceptual Data Model

The core entity model centers on the User, who serves as the root aggregate for all personalized data. Each User is associated with a UserProfile, which aggregates longitudinal writing

metrics such as TTR and MLU. The Essay entity represents the core artifact, tracking the state transition from "Drafting" to "Analyzing" and finally to "Reviewed."

To support semantic retrieval, the system maintains ReadingMaterial entities. The textual content of these materials is stored in the relational database, while their corresponding VectorEmbeddings are mapped to an external vector index, linked via unique identifiers.

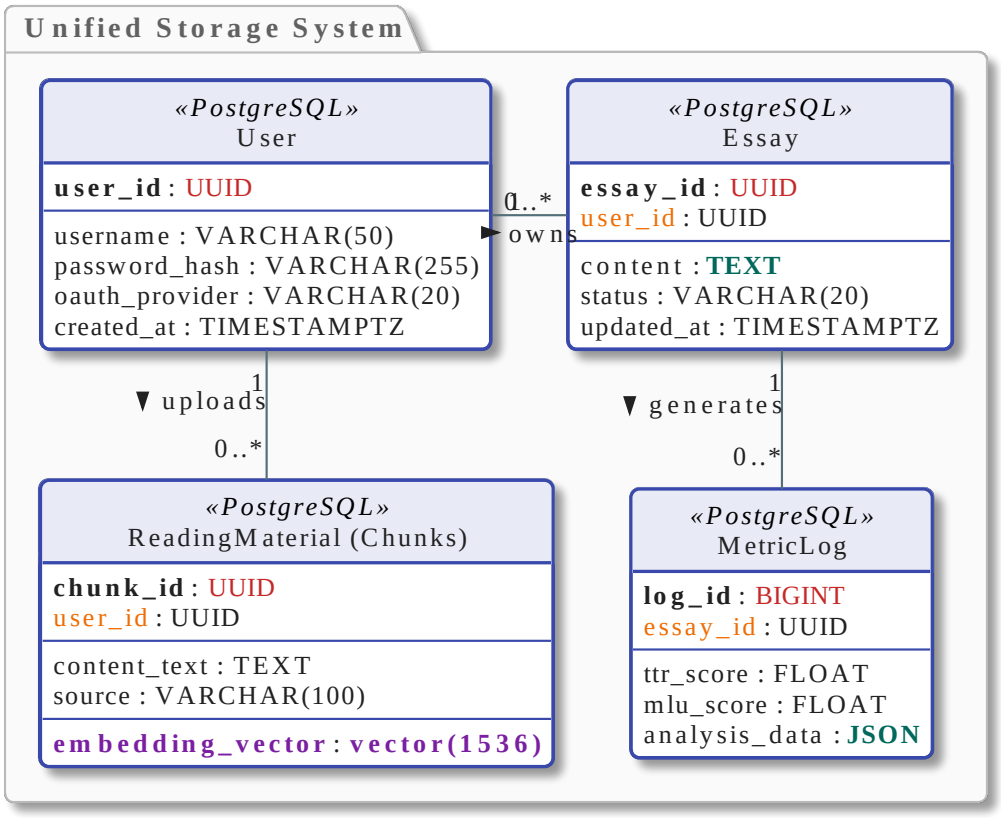

Figure 6: Data Architecture

4.2 Logical Datebase Schema

This section describes the physical schema design based on PostgreSQL. In the FastAPI backend, these schemas are defined using SQLAlchemy ORM models and connected via the asyncpg driver to ensure high-performance asynchronous execution. Furthermore, the pgvector extension is explicitly enabled to allow high-dimensional vector embeddings to be stored and queried natively alongside relational data, adhering to a unified storage architecture.

4.2.1 Relational Tables

Table 1: User Table Definition

Field Name	Data Type	Constraints	Description
user_id	UUID	PK, Not Null	Unique identifier (UUID).
username	VARCHAR(50)	Not Null	User's display name.
password_hash	VARCHAR(255)	Not Null	Bcrypt encrypted password string.
oauth_provider	VARCHAR(20)	Nullable	Source if not local auth.
created_at	DATETIME	Default NOW()	Account creation timestamp.

Table 2: Essay Table Definition

Field Name	Data Type	Constraints	Description
essay_id	UUID	PK, Not Null	Unique identifier (UUID).
user_id	UUID	FK, Not Null	References `users.user_id`.
content	TEXT	Nullable	Full text content of the draft.
status	VARCHAR(20)	Default `'draft'`	Enum: `'draft'`, `'analyzing'`, `'reviewed'`.
updated_at	DATETIME	On Update NOW()	Last modification timestamp.

Table 3: MetricLog Log Table Definition

Field Name	Data Type	Constraints	Description
log_id	BIGINT	PK, Auto Inc	Auto-incrementing primary key.
essay_id	CHAR(36)	FK, Not Null	References `essays.essay_id`.
ttr_score	FLOAT	Check (0–1)	Type-Token Ratio (Vocabulary diversity).
mlu_score	FLOAT	Not Null	Mean Length of Utterance (Complexity).
analysis_data	JSONB	Nullable	Stores Logic Tree structure & details.

4.2.2 Unified Vector Index Strategy

The system leverages PostgreSQL with the pgvector extension to store high-dimensional embeddings directly alongside relational data. This unified architecture simplifies the technology stack by eliminating external vector databases like FAISS, enabling complex hybrid queries within a single SQL transaction.

4.2.3 Data Flow Design

The system implements a comprehensive data flow strategy. Figure 5 details the component-level interactions during a write operation, ensuring transactional integrity.

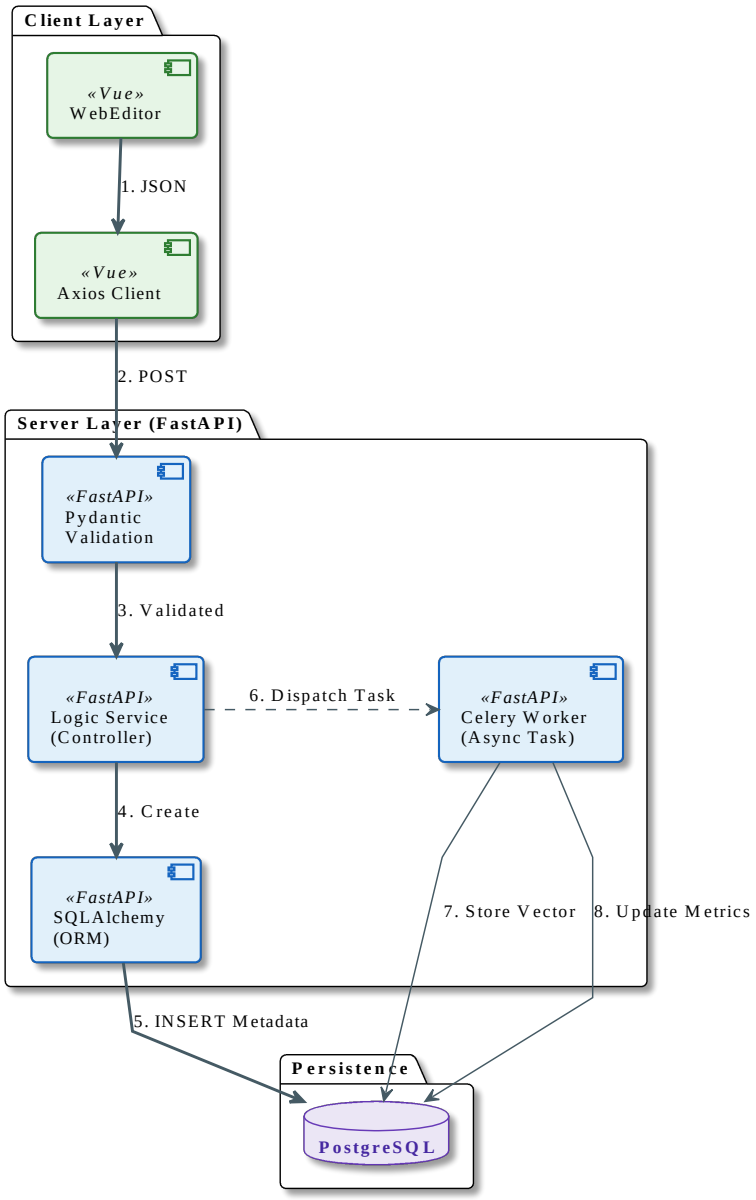

Figure 7: Data Flow Pipeline

Furthermore, Figure 6 illustrates the asynchronous RAG processing pipeline, highlighting the transformation from raw text to vectorized knowledge.

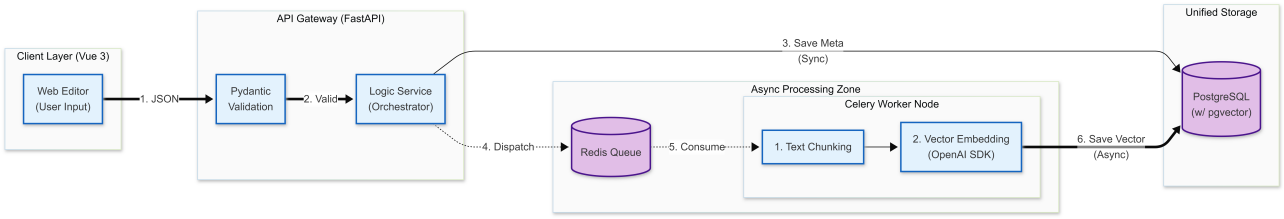

Figure 8: Data Flow Pipeline

5 Component-Level Design

This chapter details the internal component structure of the WriteLoop system. The system is decomposed into four high-cohesion subsystems, each adopting a Layered Pattern. To ensure scalability and loose coupling, interactions between subsystems are mediated through defined interface contracts and an event-driven mechanism.

5.1 Subsystem Collaboration

The system employs an Event-Driven Architecture to coordinate interactions between the Client-Side and Server-Side components. Subsystems do not communicate via tightly coupled direct calls; instead, they synchronize state through a Shared Database and a Message Broker. The Collaboration Mechanisms are as follows:

1. **Fatigue-Adaptive Feedback Loop:**
   The **State Monitoring Subsystem** (Client) acts as the system's sensory unit. Upon detecting user fatigue, it does not directly manipulate the editor. Instead, it broadcasts a system-level FatigueEvent.

   The **Instant Assistance Subsystem** (Server) subscribes to this event. Upon receipt, it automatically suppresses proactive RAG suggestions to reduce cognitive load, implementing the "Intelligent Silence" feature.

2. **Analysis-Driven Profiling Pipeline:**
   The **Logic Insight Subsystem** operates as an asynchronous service. When a user submits a draft, it performs structural analysis.

   Upon completion, it triggers an AnalysisComplete event. The Deep Profiling Subsystem consumes this event to recalculate the user's longitudinal metrics (TTR/MLU) and updates the User Profile in Database.

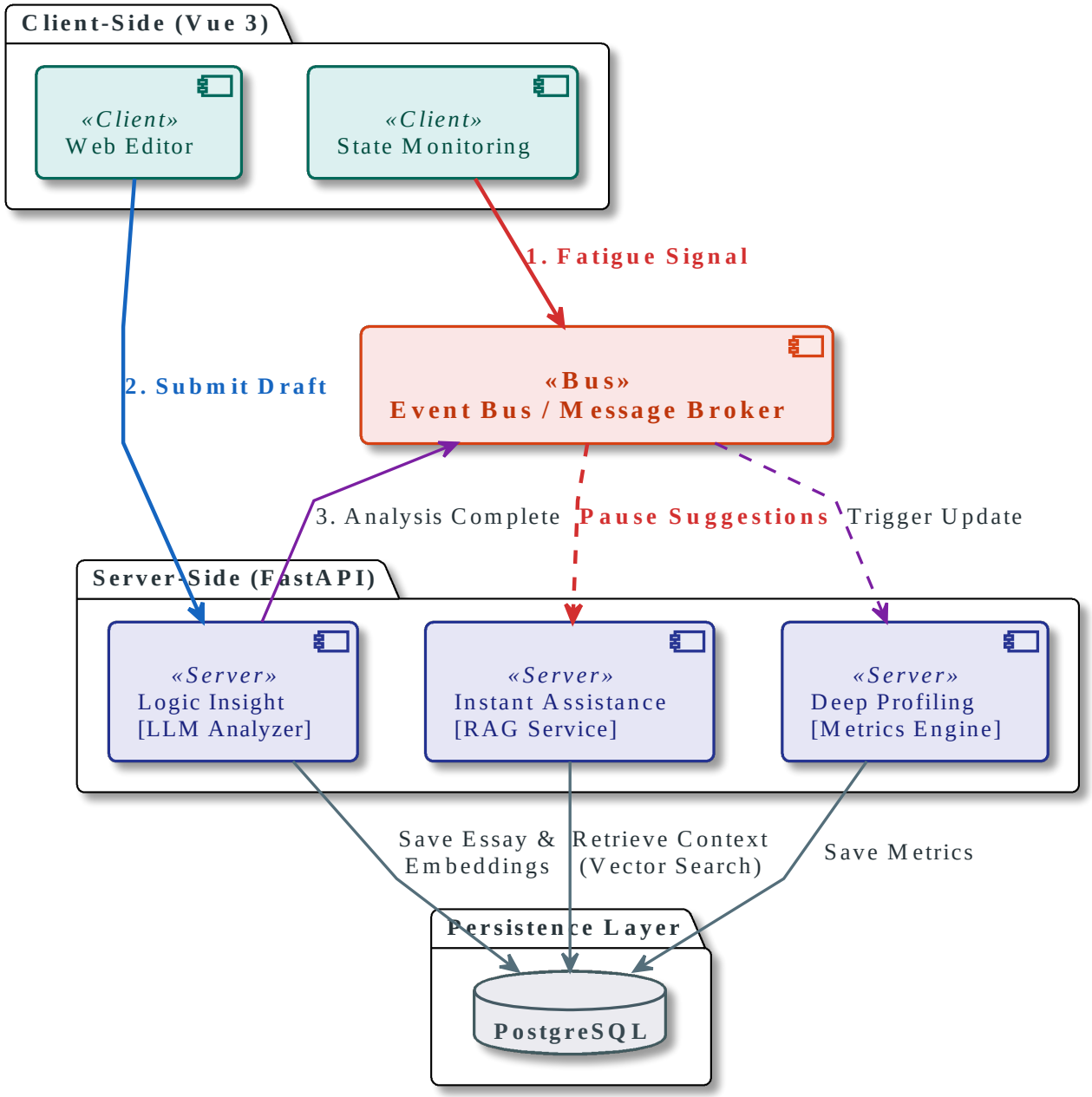

Figure 9: Subsystem Collaboration

5.2 Instant Assistance Subsystem

This subsystem is responsible for providing real-time writing support. A key architectural feature is its Dual-Source RAG Engine, which prioritizes the user's personal reading history to ensure pedagogical relevance, while maintaining a global academic corpus as a fallback to handle cold-start scenarios.

5.2.1 Functional Description

The Instant Assistance Subsystem implements the following core functions:

- **Personalized Context-Aware Completion:** Retrieves typical collocations and sentence

structures primarily from the user's Personal Reading Corpus to reinforce "Input-Output" transfer.

- **Cold-Start Fallback Strategy:** Automatically degrades to the Global Academic Corpus when the user's personal history is sparse or lacks relevant matches.
- **Stylistic Rewriting:** Rephrases selected text segments based on user-defined tones.

5.2.2 Class Design

The class design specifically addresses the need to manage two distinct vector indices.
SuggestionController(Interface Layer)

- **Description:** Handles HTTP requests(`POST/suggest`) and orchestrates the user context extraction.
- **Methods:** `+generate_suggestion(request:SuggestionRequest)->SuggestionResponse:` Entry point.

RAGService(Business Logic Layer)

- **Description:** The core orchestrator that implements the fallback logic. It decides where to search based on corpus sufficiency.
- **Attributes:**
  - `-personal_index:VectorStoreAdapter:` Interface to the user-specific vector partition.
  - `-global_index:VectorStoreAdapter:` Interface to the system-wide static vector index.
  - `-similarity_threshold:` Minimum score to accept a personal match.

Methods:

- `+get_suggestion(context:str,user_id:int)->str:` First calls search_personal(). If results are empty or score < threshold, calls search_global().
- `-_construct_prompt(user_text:str,retrieved_chunks:List[Chunk])->str:` Assembles the LLM prompt.

VectorRepository (Data Access Layer)

- **Description:**Manages connections to the underlying vector database
- **Methods:**
  - `+search_personal(user_id:int,query_vec:List[float])->List[Chunk]`

– +search_global(query_vec:List[float])->List[Chunk]

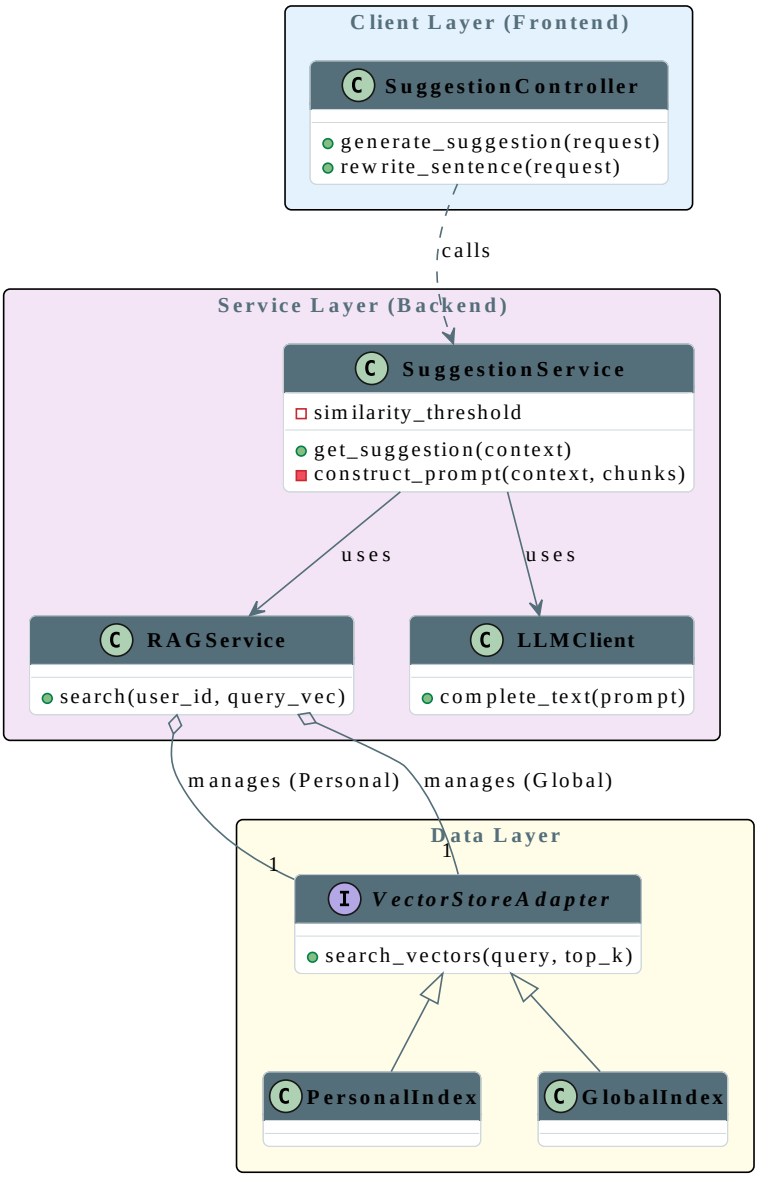

Figure 10: Instant Assistance Subsystem Class Diagram

5.2.3 Dynamic Behavior

To ensure pedagogical relevance while maintaining system robustness, this subsystem employs a Dual-Layer Retrieval Strategy. Instead of relying on a single data source, it prioritizes the user's Personal Reading History to reinforce learning transfer. Crucially, it implements a Cold-Start Fallback Mechanism that automatically queries the Global Academic Corpus when personal data is insufficient.

The Sequence Diagram below illustrates this dynamic decision-making process, highlighting the logical flow where the system evaluates retrieval confidence and switches data sources to guarantee high-quality suggestions.

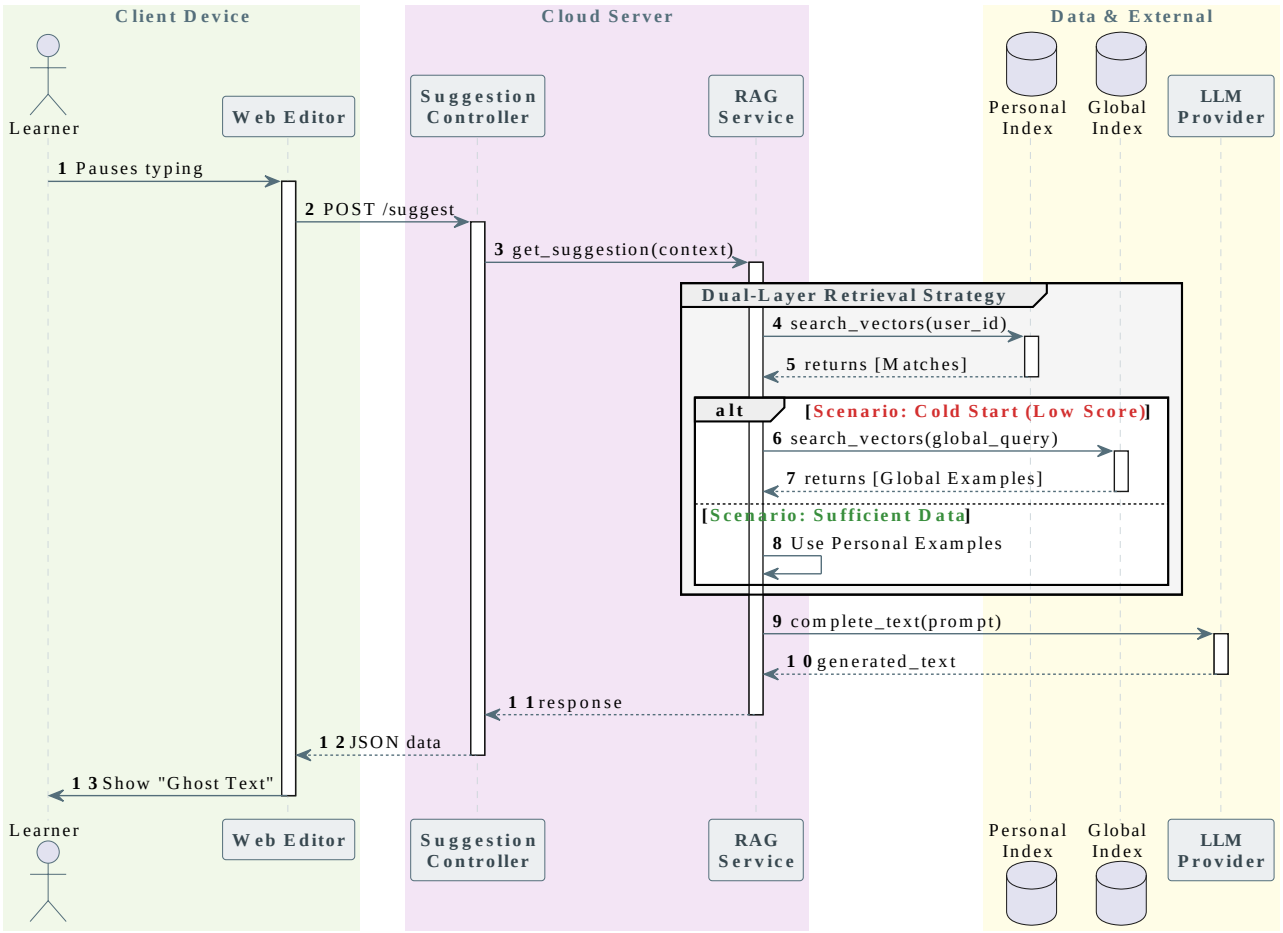

Figure 11: Instant Assistance Sequence Diagram

5.3 Logic Insight Subsystem

The Logic Insight Subsystem is responsible for the "Deep Diagnosis" of completed essays. Unlike the real-time nature of Instant Assistance, this subsystem operates in an asynchronous, high-throughput mode. It leverages a Parallel Pipeline Architecture to simultaneously execute Large Language Model inference for structural parsing and Vector Space computations for coherence analysis.

5.3.1 Functional Description

- **Argumentative Structure Decomposition:** Utilizes an LLM to perform a deep scan of the text, extracting the hierarchical "Skeleton Tree" (Thesis, Arguments, Evidence) and identifying logical loopholes (e.g., claims lacking evidence).
- **Coherence Heatmap Generation:** Splits the text into sentences, calculates high-dimensional embeddings, and computes the Cosine Similarity between adjacent vectors to visualize semantic flow.

5.3.2 Class Design

The subsystem follows a Task Orchestration Pattern. The LogicController accepts requests and offloads them to an AnalysisOrchestrator, which manages distinct worker engines for structure and coherence tasks.

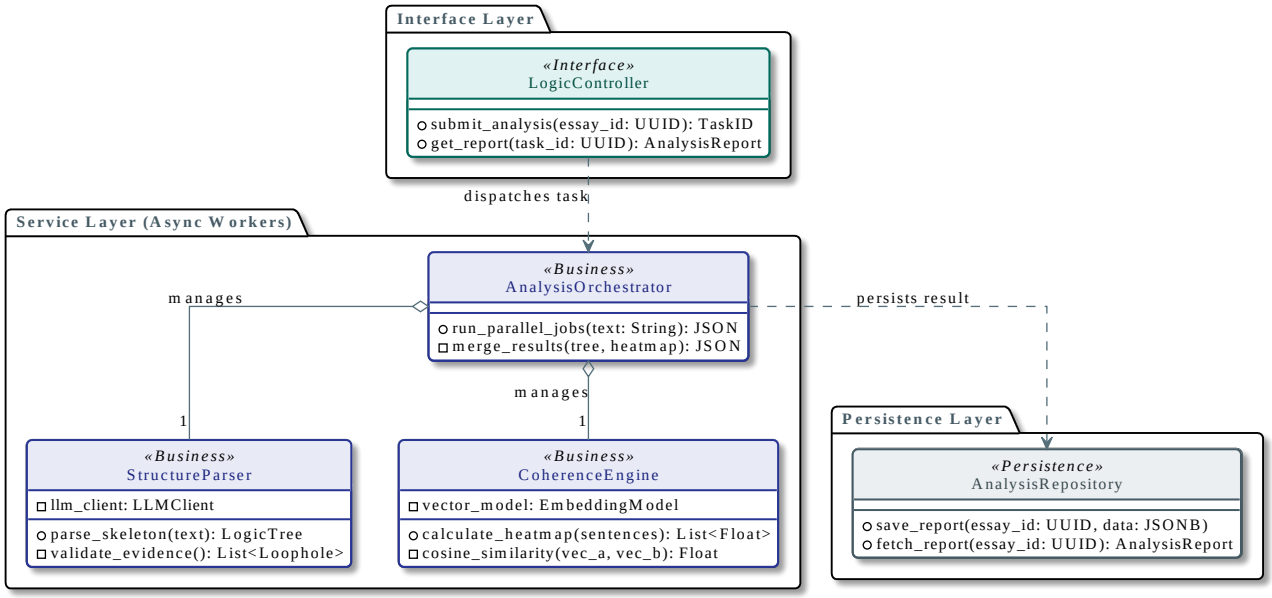

Figure 12: Logic Insight Class Diagram

5.3.3 Dynamic Behavior

To minimize user waiting time for long essays, this subsystem avoids sequential execution. Instead, it employs a Fork-Join Parallel execution model. The Activity Diagram below illustrates how the system "forks" the text processing into two concurrent streams—Structure Analysis (LLM-heavy) and Coherence Analysis (CPU/Vector-heavy)—and "joins" them to form the final report.

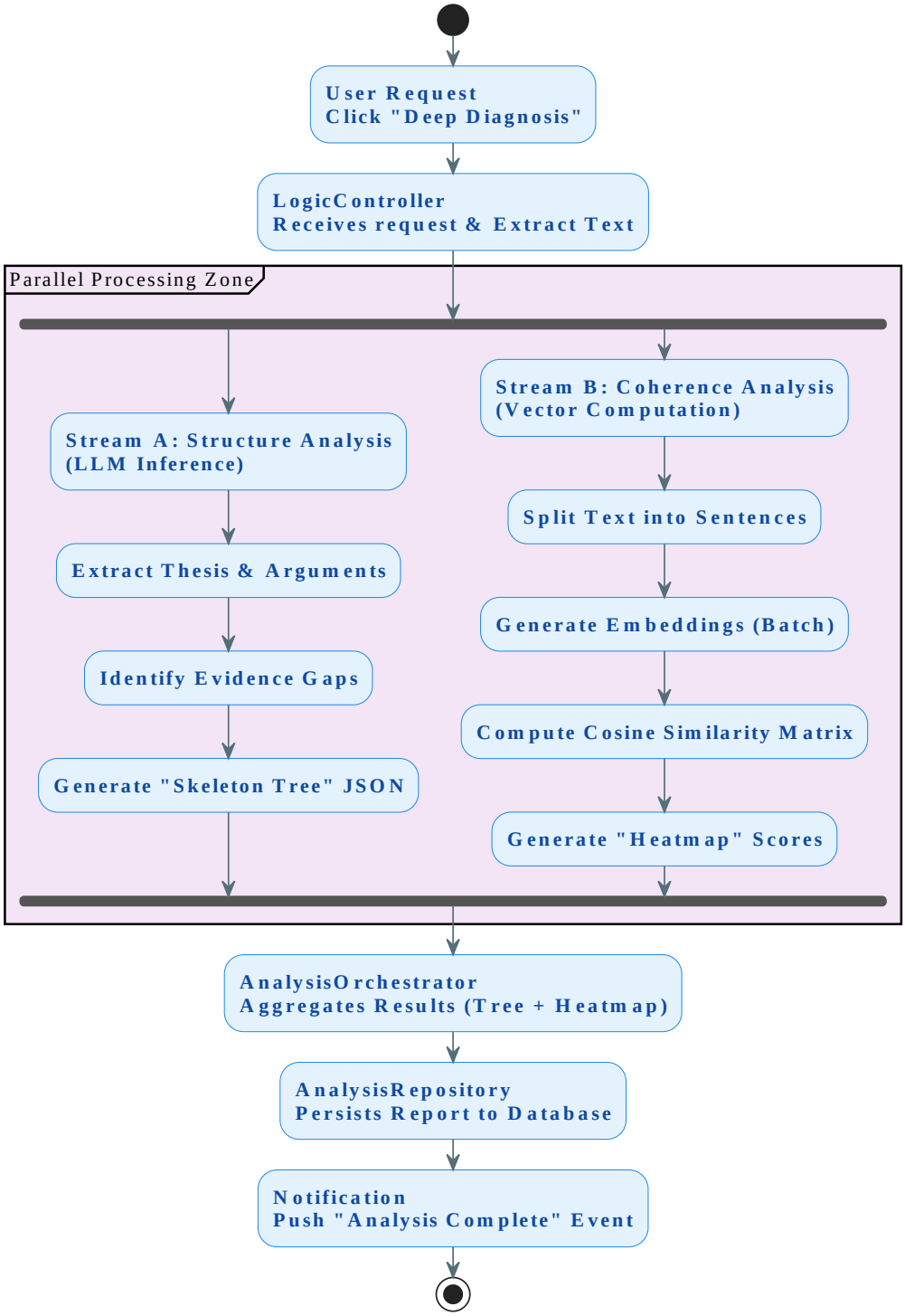

Figure 13: Logic Insight Parallel Processing Activity Diagram

5.4 Deep Profiling Subsystem

This subsystem is responsible for the longitudinal assessment of learner performance. It operates on a Hybrid Processing Model: instantaneous feature extraction is triggered via events upon essay submission, while computation-intensive pattern recognition (e.g., weakness clustering) is executed as periodic batch jobs to optimize resource utilization.

5.4.1 Functional Description

- **Multi-dimensional Feature Extraction:** Automatically computes linguistic metrics (TTR, MLU, Logic Score) for every submission to track progress over time.
- **Weakness Pattern Clustering:** periodically analyzes historical error logs using unsupervised learning algorithms to identify persistent skill gaps (e.g., "Tense Confusion Cluster").
- **Adaptive Task Generation:**Dynamically generates personalized training micro-tasks based on identified weakness clusters.

5.4.2 Class Design

The design follows a Strategy Pattern for the analysis engine, allowing different profiling strategies to be swapped without affecting the core service.

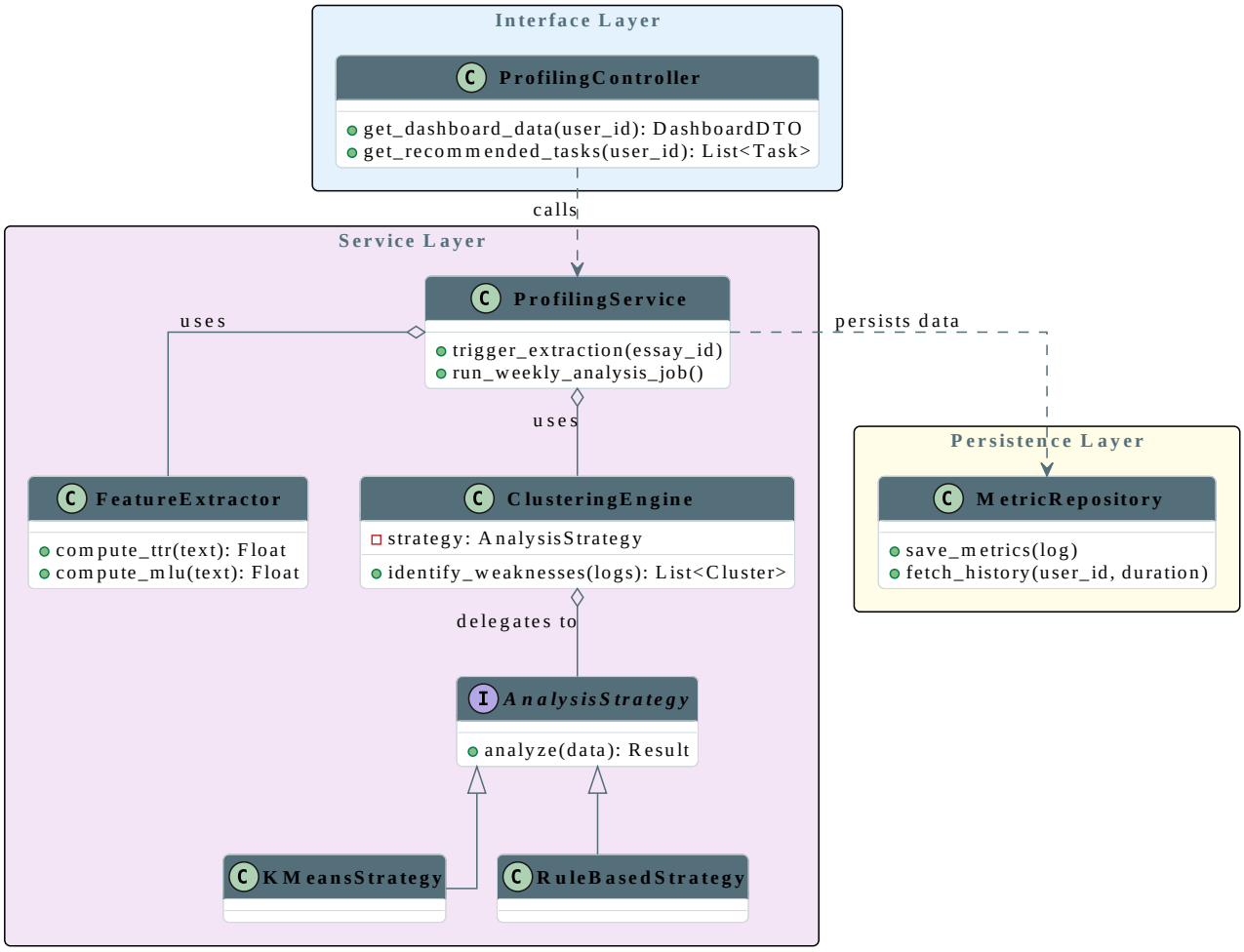

Figure 14: Deep Profiling Class Diagram

5.4.3 Dynamic Behavior

Unlike the user-triggered interactions in previous modules, this subsystem relies heavily on System-Triggered Automation.

The Sequence Diagram below illustrates the "Weekly Weakness Analysis" workflow. It demonstrates how the system autonomously wakes up, processes large datasets, and generates adaptive tasks without user intervention.

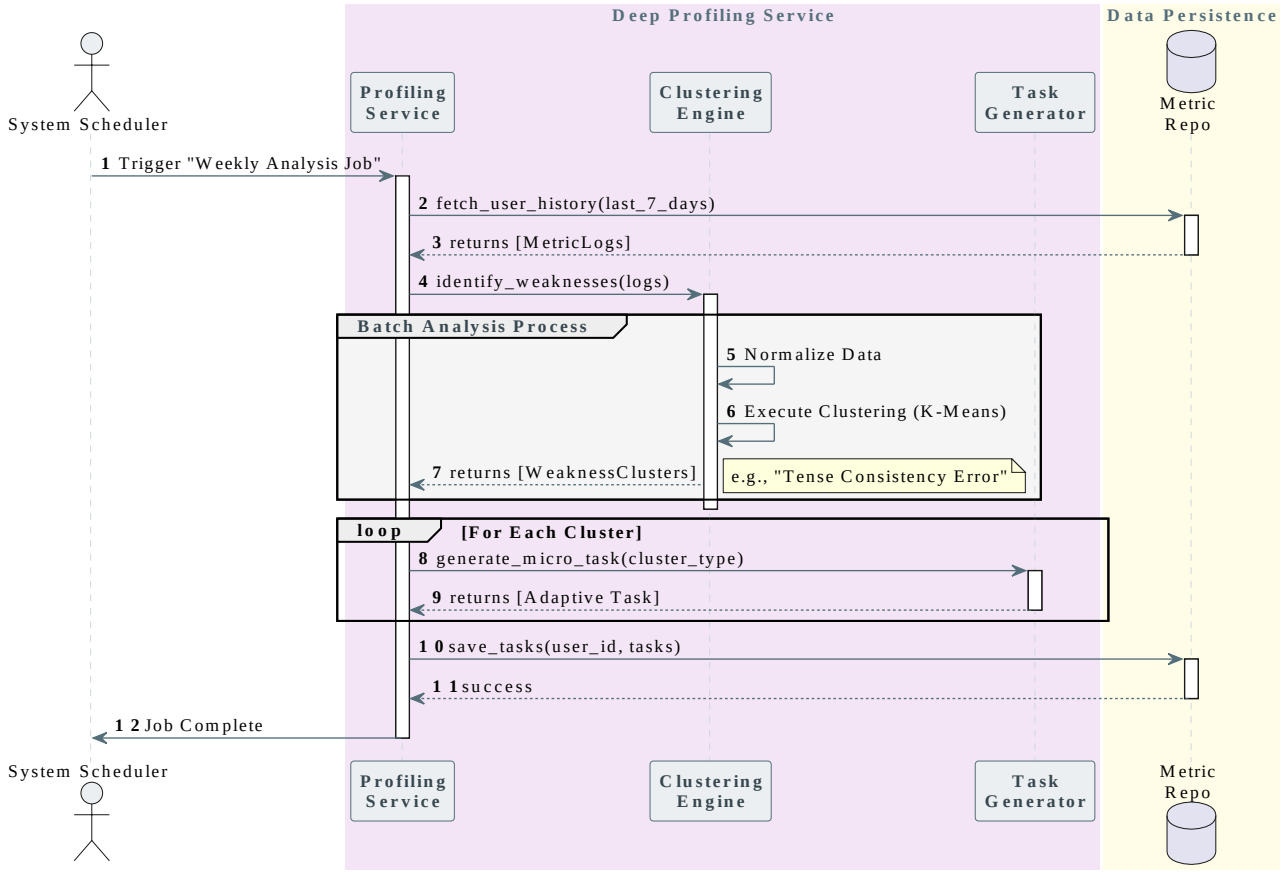

Figure 15: Deep Profiling Batch Processing Sequence Diagram

5.5 Intelligent State Monitoring Subsystem

This subsystem is architected as a Privacy-First Edge Module. Unlike other server-side components, it runs entirely within the user's browser context. It implements a Finite State Machine (FSM) to process high-frequency biometric signals (via MediaPipe) into stable cognitive states, ensuring zero leakage of raw video data.

5.5.1 Functional Description

- **Edge-Side Inference:** Wraps the MediaPipe Vision task in a Web Worker to perform non-blocking face landmark detection.
- **State Smoothing:** Converts noisy PERCLOS and gaze signals into stable states (e.g., "Focused" vs. "Fatigued") using a time-window smoothing algorithm.
- **Adaptive Intervention:** Triggers UI changes (e.g., dimming, modal alerts) based on the current FSM state.

5.5.2 Class Design

The design applies the Observer Pattern. The VisionAgent acts as the Subject producing raw signals, while the StateManager acts as the Observer that transitions states and notifies the InterventionController.

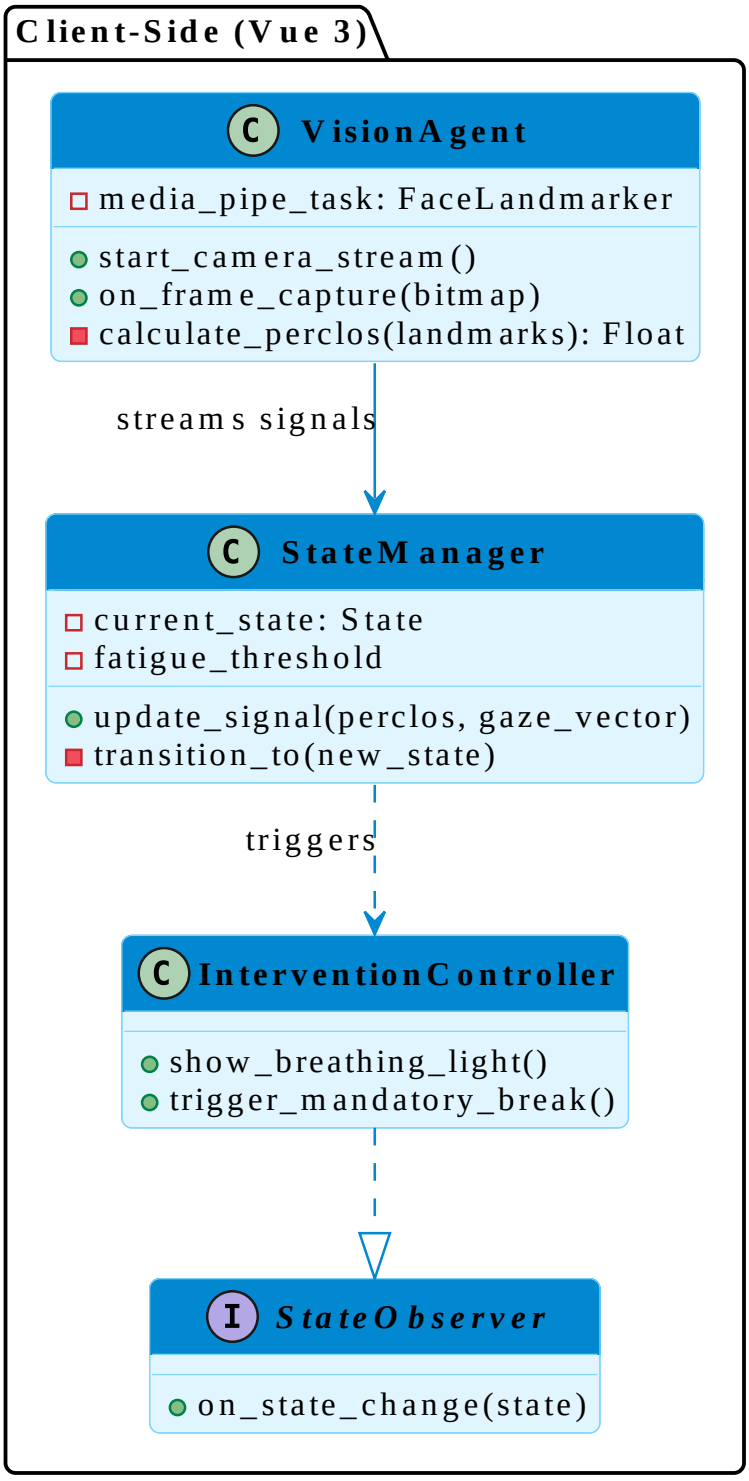

Figure 16: State Monitoring Class Diagram

5.5.3 Dynamic Behavior

To prevent "Alert Fatigue" (e.g., constantly popping up warnings if a user looks away for 1 second), the system employs a State Machine Diagram. This model defines strict transition rules: a user must exhibit distraction symptoms for a sustained period (Threshold) before the state changes.

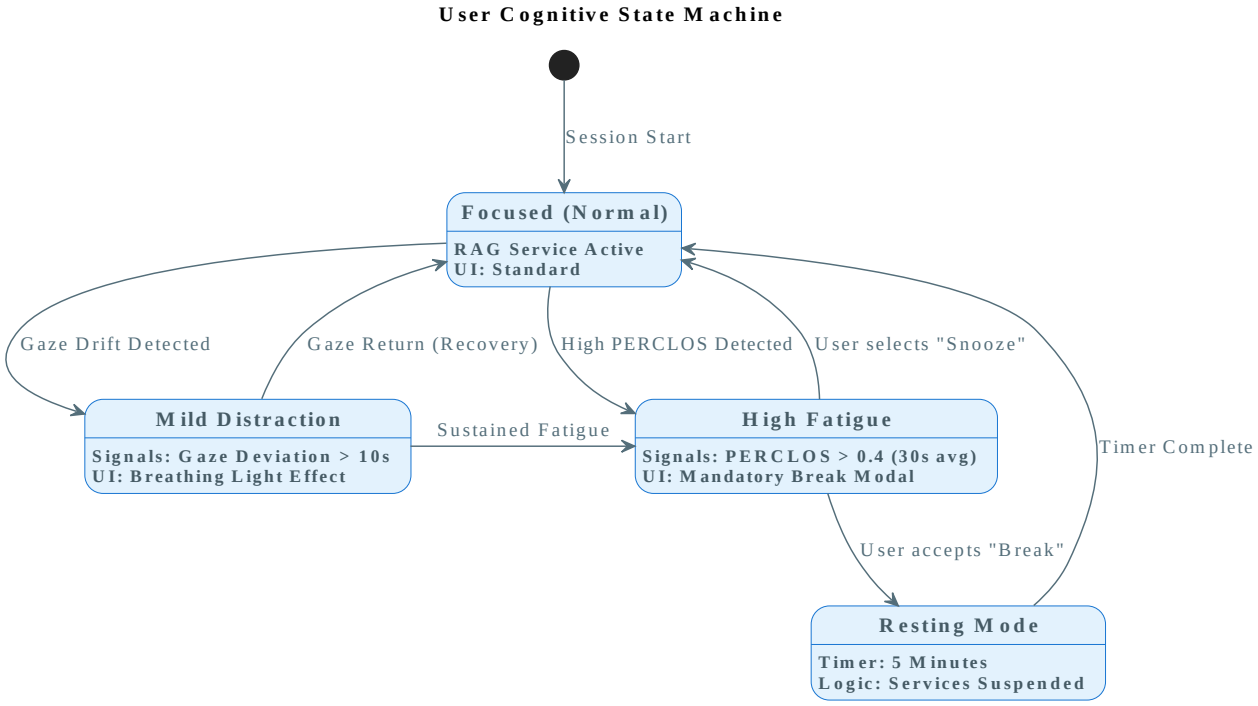

Figure 17: User Cognitive State Machine

6 Interface Design

This section documents the detailed interface specifications for the WriteLooop system, covering the user interface interactions, software application programming interfaces, and hardware integration points.

6.1 User Interface Design

The user interface adopts a "Cognitive-Scaffolding" design philosophy, dividing the workspace into functional zones to minimize distraction while providing real-time assistance.

6.1.1 Immersive Writing Editor

The core interaction surface is the Editor View, which strictly adheres to a distraction-free design philosophy by presenting a clean canvas where auxiliary assistance tools remain hidden by default to preserve user focus. Central to this layout is a Contextual Action Mechanism that prioritizes on-demand interaction over permanent interface clutter. The workflow is

fundamentally selection-based: when a user highlights a specific sentence or phrase, a Floating Action Bar automatically manifests adjacent to the selection; upon clicking the "Apply" trigger, a Bottom Action Panel animatedly slides up from the lower edge of the viewport. This panel serves as the decision hub, dynamically rendering AI-generated rewrite options or completion candidates. The interaction concludes when the user navigates through these options and selects a preferred variation, which instantly substitutes the original text in the canvas, thereby integrating complex AI assistance into a seamless, gesture-driven editing experience.

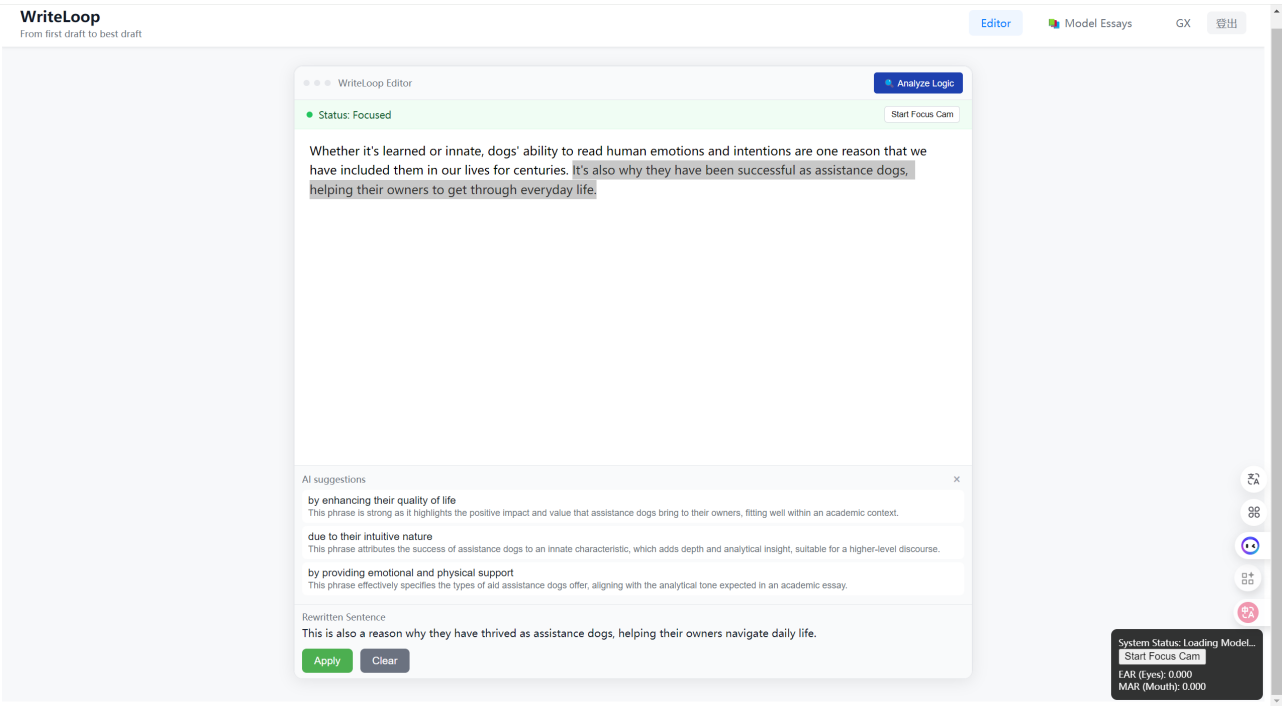

Figure 18: Immersive Writing Editor

6.1.2 Logic Insight Dashboard

Instead of redirecting users to a separate dashboard, the system employs an Overlay Analysis Mode that projects logical diagnostics directly onto the writing canvas. This design maintains the user's immersion while providing deep structural feedback.

- **Inline Coherence Heatmap:** The system visualizes semantic flow directly within the text. Rather than an abstract matrix, sentence backgrounds are color-coded based on their coherence scores.
- **Contextual Logic Indicators:** Specific logical flaws (such as "Claim without Evidence" or "Circular Reasoning") are flagged with margin indicators or floating icons adjacent to the problematic text. Clicking these indicators expands a side card explaining the logical gap, keeping the interface clean and focused.

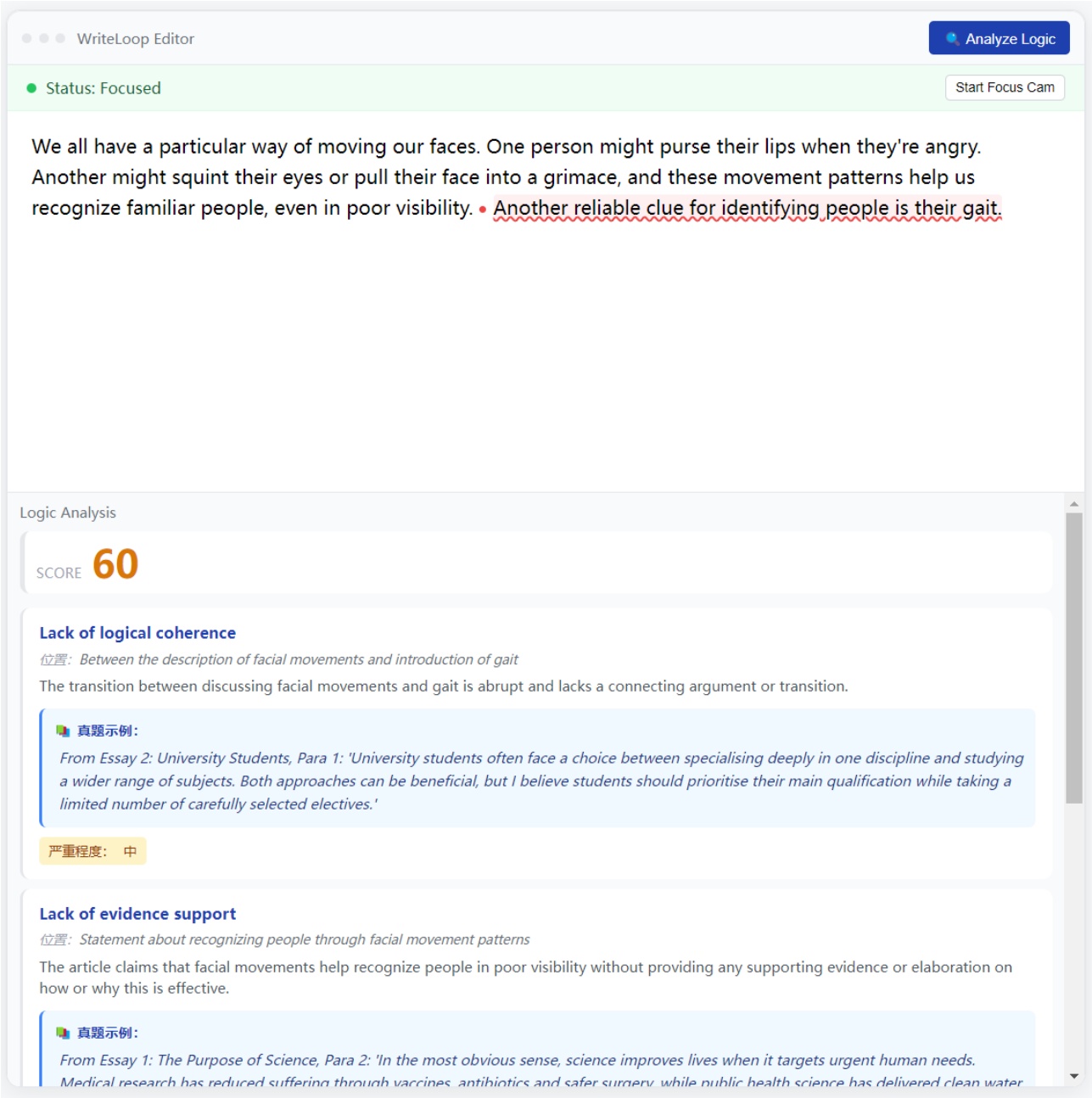

Figure 19: Editor in Analysis Mode showing Inline Coherence Highlights

6.1.3 Adaptive Task Workspace

The Adaptive Task Generation View is designed to provide immediate, context-aware training without navigating away from the writing context. The interface is vertically split into a reference area and an action area. **Original Text Context:** The upper portion of the viewport displays the Original Text. This provides necessary context, allowing the user to review their raw writing before engaging in optimization tasks, ensuring that all practice is grounded in their actual work. **Task Selection Cards:** Anchored at the bottom are three specific task options derived from the profiling engine. Each card represents a distinct improvement strategy tailored to the current draft's needs:

1. Craft a Clear Thesis Statement: Focuses on structural clarity. This module challenges

the user to refine their main argument, ensuring it is specific, arguable, and clearly positioned at the start of the text.

2. Connect General Statements with Examples: Focuses on logical depth. This module prompts the user to bridge the gap between abstract claims and concrete evidence, reinforcing the logical chain.

3. Vary Sentence Structures: Focuses on syntactic variety. This module targets repetitive sentence patterns, encouraging the user to mix simple, compound, and complex sentences to improve flow and readability.

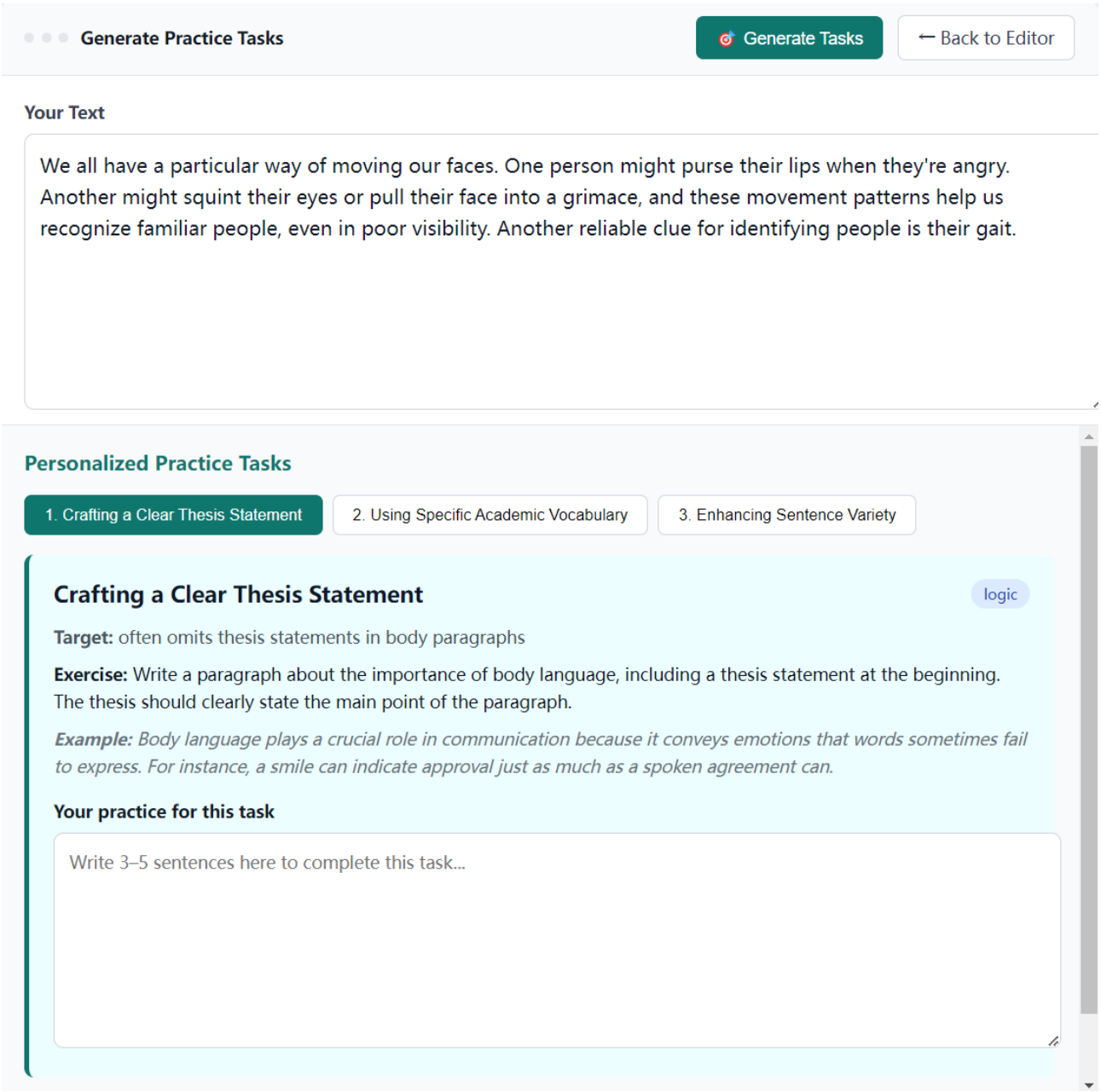

Figure 20: Adaptive Task Generation View

6.1.4 User Profiling Trends Dashboard

The User Profiling Trends Dashboard provides a holistic view of the learner's writing evolution, visualizing longitudinal data processed by the Deep Profiling Subsystem. Unlike the snapshot view of the Editor, this interface focuses on long-term growth patterns.

- **Longitudinal Trend Visualization:** The central area features interactive Multi-Metric Line Charts. These charts plot key linguistic indicators over time
- **Multidimensional Deep Profile:** Adjacent to the timeline, a Skill Radar Chart (or Capability Hexagon) maps the user's current proficiency across distinct dimensions: Grammar, Coherence, Vocabulary, Structure, and Conciseness. This "Deep Profile" instantly highlights balanced strengths and specific areas needing improvement.

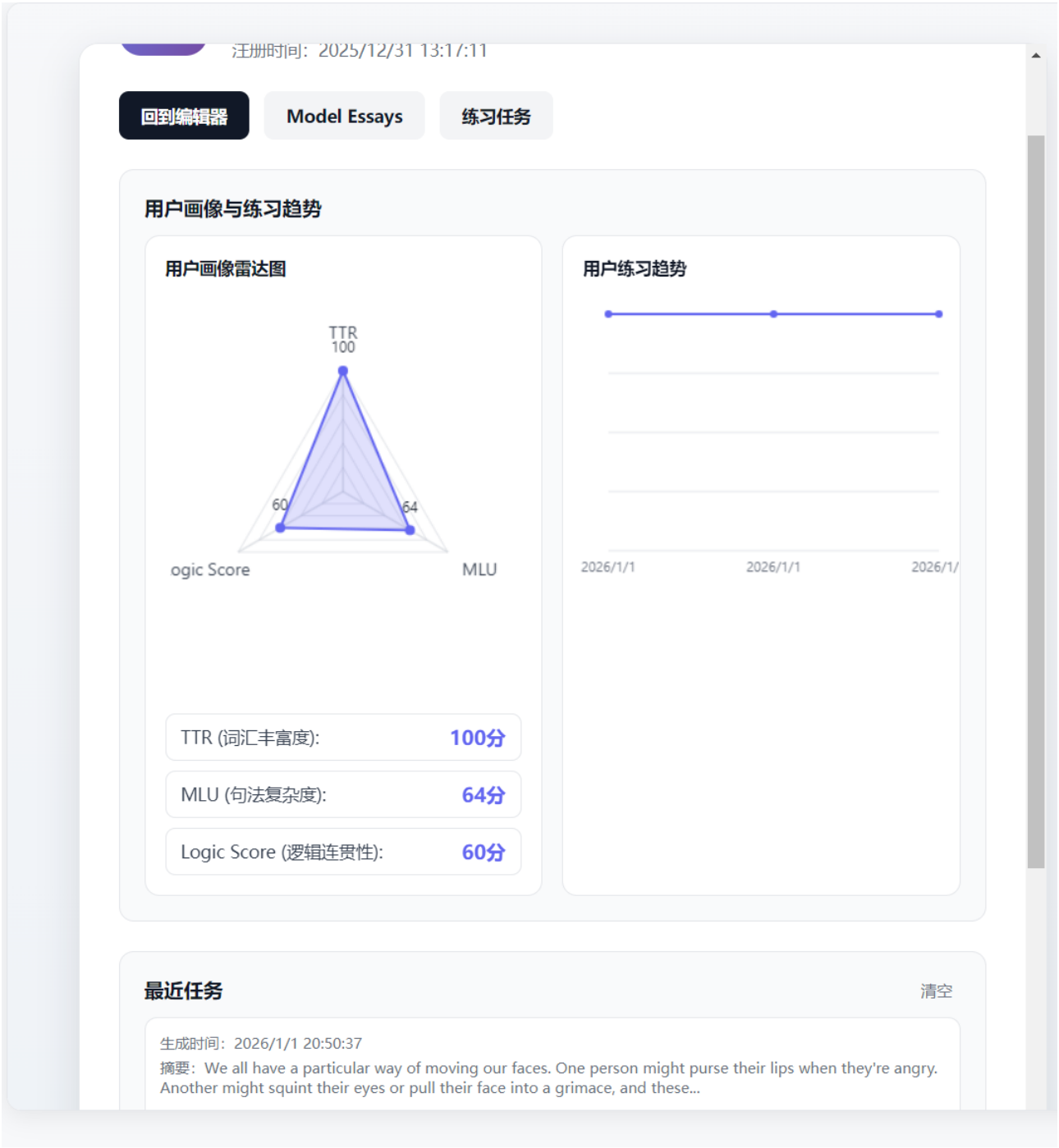

Figure 21: User Profiling Trends Dashboard

6.2 Software Interface Specifications

This section defines the precise interface contracts between the Client, the Server, and external subsystems. All application interfaces adhere to RESTFUL principles, utilizing JSON for data serialization and JWT fro stateless authentication.

6.2.1 Protocol Configuration

This table outlines the core configuration parameters for the application's API. It defines the protocol to be used based on the environment, the base URL that can be set through the

VITE_API_BASE_URL environment variable, the default JSON data format, UTF-8 character encoding, and token-based authentication using a Bearer token in the HTTP Authorization header. These settings ensure proper and secure integration between clients and the backend service in both development and production environments.

Configuration Item	Value / Description
Protocol	HTTPS (Production) / HTTP (Development)
Base URL	Configurable via `VITE_API_BASE_URL` • Production: `http://47.237.161.90/api/v1` • Development: `http://localhost:8000/api/v1`
Data Format	`application/json` (unless otherwise specified)
Character Encoding	UTF-8
Authentication	HTTP Header: `Authorization: Bearer <access_token>`

Table 4: API Configuration Settings

6.2.2 Authentication User Management

These interfaces manage identity verification and user profile synchronization, integrating with the WriteLoop LMS via OAuth 2.0.

Method	URI Path	Description	Request Body (JSON)	Response Body (JSON)
POST	/auth/login	Authenticates user credentials.	`{"username": "...", "password": "..." }`	`{"access_token": "...", "token_type": "bearer", "user": {...} }`
POST	/auth/register	Registers a new learner account.	`{"username": "...", "email": "...", "password": "..." }`	`{"user_id": "UUID", "status": "created" }`
GET	/users/me	Retrieves current user profile.	(None)	`{"id": "UUID", "role": "student", "preferences": {...} }`
PUT	/users/profile	Updates learner preferences (e.g., target exam).	`{"target_exam": "IELTS", "vocab_level": "C1" }`	`{"status": "success", "updated_at": "timestamp" }`

Table 5: Authentication Interfaces

6.2.3 Module 1: Instant Assistance Subsystem

his subsystem handles real-time writing support, including Contextual Recommendations (RAG) and Syntactic Refactoring (LLM).

Method	URI Path	Description	Request Body (JSON)	Response Body (JSON)
POST	/assist/suggest	Triggers RAG to retrieve collocations based on cursor context.	`{"text_context": "...", "cursor_pos": 150, "reading_history_ids": [...] }`	`{"suggestions": [{"content": "...", "citation": "Source...", "type": "completion"}] }`
POST	/assist/rewrite	Refactors selected text for academic tone or conciseness.	`{"selection": "...", "mode": "academic" }`	`{"rewritten_variants": [{"text": "...", "style": "academic", "diff_map": [...]}] }`

Table 6: Instant Assistance Interfaces

6.2.4 Module 2: Logic Insight Subsystem

This subsystem performs asynchronous deep analysis of essays, generating Argument Skeleton Trees and Coherence Heatmaps.

Method	URI Path	Description	Request Body (JSON)	Response Body (JSON)
POST	/analyze/logic	Submits full text for structural and coherence analysis.	`{"essay_id": "UUID", "full_text": "..." }`	`{"task_id": "UUID", "status": "processing", "est_time": 5 }`
GET	/analyze/report/{taskId}	Polling endpoint to retrieve analysis results.	(None)	`{"structure_tree": {"thesis": "...", "arguments": [...] }, "heatmap_scores": [{"sentence_id": 1, "score": 0.4 }] }`

Table 7: Logic Insight Interfaces

6.2.5 Module 3: Deep Profiling Adaptive Tasks

This subsystem manages longitudinal data tracking (TTR/MLU metrics) and generates personalized Micro-Tasks.

Method	URI Path	Description	Request Body (JSON)	Response Body (JSON)
GET	/dashboard/stats	Retrieves longitudinal trend data (TTR, MLU, Logic Score).	Query: `?range=month`	`{"ttr_trend": [...], "mlu_trend": [...], "weakness_tags": ["Grammar"] }`
POST	/tasks/generate	Generates adaptive micro-tasks based on weakness clusters.	`{"weakness_focus": ["coherence"], "count": 3 }`	`{"tasks": [{"id": 101, "prompt": "...", "reference": "..." }] }`
POST	/tasks/{id}/submit	Submits user response for a micro-task.	`{"user_input": "...", "time_taken": 45 }`	`{"score": 85, "feedback": "Good use of transitions." }`

Table 8: State Monitoring Interfaces

6.2.6 Module 4: Intelligent State Monitoring

While video processing is local (Edge AI), this subsystem logs intervention events (e.g., fatigue breaks) to the server for long-term habit analysis.

Method	URI Path	Description	Request Body (JSON)	Response Body (JSON)
GET	/corpus/essays	Retrieves list of reading materials/essays.	Query: `?page=1&limit=10`	`{"items": [{"id": "...", "title": "..." }], "total": 50 }`
GET	/corpus/essays/{id}	Retrieves specific essay content for context.	(None)	`{"id": "...", "content": "...", "metadata": {...} }`
POST	/corpus/upload	Uploads personal reading material for indexing.	FormData: file (PDF/TXT)	`{"doc_id": "...", "chunks_indexed": 15 }`

Table 9: Corpus Management API Endpoints

7 Quality Attribute Design

This section details the architectural tactics and design decisions employed to meet the non-functional requirements specified in the SRS, specifically focusing on Security, Performance, and Reliability.

7.1 Security Privacy Design

Given the educational nature of the system and the collection of biometric signals, security and privacy are paramount. The design implements a "Privacy-by-Design" strategy.

7.1.1 Communication Security

- Transport Layer Security: All data transmission between the Client (Vue 3) and Server (FastAPI), including WebSocket streams, is encrypted using TLS 1.2+.
- Authentication: Implements OAuth 2.0 Password Grant flow. Stateless JWT (JSON Web Tokens) are used for session management, with a short expiration time (60 minutes) and a refresh token mechanism.

  CORS Policy: Strict Cross-Origin Resource Sharing (CORS) configurations allow requests only from trusted domains.

7.1.2 Data At Rest

- Sensitive Data Encryption: User passwords are salted and hashed using bcrypt before storage in PostgreSQL.

- Anonymization: For research or model fine-tuning purposes, user writing data is decoupled from Personally Identifiable Information (PII) using a pseudo-ID mapping system.

7.2 Performance Scalability Design

The system is designed to handle high-concurrency usage typical of classroom settings, ensuring responsiveness.

7.2.1 Latency Optimization

- Asynchronous Processing: Computationally intensive tasks, specifically Logic Insight (LLM Analysis), are decoupled from the main request thread using Celery workers and Redis message queues. This prevents HTTP request timeouts and allows the frontend to display a "Analyzing..." status without freezing.
- Debouncing Strategy: To reduce API load, the Instant Assistance module implements a less debounce mechanism on the client side, ensuring requests are only sent when the user pauses typing.

7.2.2 Caching Strategy

- L1 Cache (Browser): User drafts are continuously cached in localStorage to enable instant load times and offline resilience.
- L2 Cache (Redis): High-frequency read data, such as user profiles and "Global Corpus" search results, are cached in Redis to reduce database hits.

7.2.3 Throughput Management

- Rate Limiting: To prevent API abuse and manage costs (OpenAI tokens), the API Gateway implements a Token Bucket algorithm, limiting users to a defined number of AI requests per minute.
- Connection Pooling: Database interactions utilize SQLAlchemy Connection Pooling to maintain a stable set of active connections, reducing the overhead of establishing new TCP connections for every request.

7.3 Maintainability Reliability Design

7.3.1 Modular Architecture

The strict separation of the Algorithm Layer (RAG/LLM logic) from the Application Layer (API routes) ensures that AI models can be upgraded (e.g., switching from GPT-4 to a local LLaMA model) without requiring code changes in the frontend or database schemas.

7.3.2 Graceful Degradation

The system implements a Circuit Breaker Pattern for external dependencies (OpenAI API).

- Failure Scenario: If the external LLM service times out or returns 5xx errors.
- Fallback Behavior: The system automatically switches to a "Basic Mode," disabling AI suggestions but keeping the Text Editor and Local Statistics (Word Count) fully functional. A user-friendly toast notification ("AI Services currently unavailable, switching to offline mode") is displayed.

7.3.3 Comprehensive Logging

Application Logs: Structured logs (JSON format) are collected for all API endpoints, recording request latency, status codes, and user IDs (excluding sensitive content) to facilitate rapid debugging.