AI Vector Knowledge Layer – Technical Specification

# 1. Overview

This specification defines the architecture and implementation details of the AI Vector Knowledge Layer for StudyHub. The system consists of four semantically searchable vector databases backed by ChromaDB, and is powered by a multi-agent RAG-based LLM system (Gemini). Each database supports search and content ingestion APIs, enabling a chat assistant (Perceptron) to retrieve accurate context for student queries while maintaining academic integrity.

# 2. Vector Databases

## FAQ

Stores static institutional info (e.g., grading policy, exam formats). Public access. Single-table.

## Course Guide

Stores semantic descriptions of what topics are covered or excluded in each course. Public access.

## Course Content

Stores lecture-level content for all courses, including PDFs and video transcripts. Metadata-rich and filterable.

## Personal Resources

User-uploaded notes per course, stored privately and accessed by the respective student only.

# 3. API Overview

Each vector database exposes two categories of endpoints:

1. Search APIs – semantic query → results (vector chunks or structured metadata)

2. Create APIs – used to ingest structured content into the database

# 4. API Specifications

## FAQ

POST /faq/search

Input: { 'query': 'Can I carry a calculator to the exam?' }  
Output: Top-K FAQ vector chunks (text)

POST /faq/add

Input: { 'text': 'Calculators are allowed in exams unless stated otherwise.', 'tags': ['exam', 'policy'] }

## Course Guide

POST /course-guide/search

Input: { 'query': 'Which courses cover SVD?' }  
Output: List of { course\_id, scope, score }

POST /course-guide/add

Input: { 'course\_id': 'MATH202', 'covered\_topics': [...], 'excluded\_topics': [...] }

## Course Content

POST /course-content/search

Input: { 'query': 'Explain recursion', 'filters': { 'course\_id': 'CS101', 'week\_id': 'W3' } }  
Output: Top-K lecture chunks

POST /course-content/add

Input: { 'course\_id': 'CS101', 'week\_id': 'W3', 'lecture\_id': 'L2', 'text': '...', 'topics': ['Recursion'] }

## Personal Resources

POST /personal-resources/search

Input: { 'query': 'Sorting in Java', 'filters': { 'course\_id': 'CS102', 'topic': 'Sorting' } }

POST /personal-resources/add

Input: { 'course\_id': 'CS102', 'text': '...', 'topics': ['Merge Sort'] }

## Integrity Check

POST /integrity-check/check

Input: { 'query': 'Explain pointers in C' }  
Output: List of { question\_id, similarity\_score }

POST /integrity-check/add

Input: { 'course\_id': 'CS101', 'assignment\_id': 'A1', 'question\_text': 'Explain pointers in C' }

# 5. Core Design Concepts

- All text is embedded using sentence-transformers (`all-MiniLM-L6-v2`) before storage in ChromaDB.

- Each vector DB uses metadata (e.g., course\_id, week\_id, scope) to support filtered semantic search.

- CourseGuide uniquely supports a 'scope' tag for covered/excluded topics.

- PersonalResources are namespaced per student and stored in private collections.

- IntegrityCheck DB only indexes questions from graded assignments, flagged via match score threshold.

- Reindexing is the preferred strategy for content updates (vs in-place edits).

# 6. Feasibility & Engineering Strategy

The architecture prioritizes clarity, feasibility, and modular development. Each vector database is independently updatable and accessible via REST APIs. Initial implementations focus on search and ingestion only—editing and deletion can be added post-MVP.