AI Agent Architecture Document

Study Buddy - Intelligent Academic Assistant

Data Science Internship Assignment

November 3, 2025

Abstract

This document details the architecture of Study Buddy, a multi-agent AI system designed to automate student workflow tasks. The system combines retrieval-augmented generation (RAG) for course material assistance with specialized LaTeX formatting capabilities, creating a comprehensive academic productivity tool. The architecture emphasizes modularity, scalability, and user-centric design while leveraging state-of-the-art language models and efficient inference strategies.

Contents

1 Executive Summary

Study Buddy is a multi-agent AI system designed to automate and enhance student workflow tasks. The system combines retrieval-augmented generation (RAG) for course material assistance with specialized LaTeX formatting capabilities, creating a comprehensive academic productivity tool.

2 System Architecture Overview

2.1 High-Level Architecture

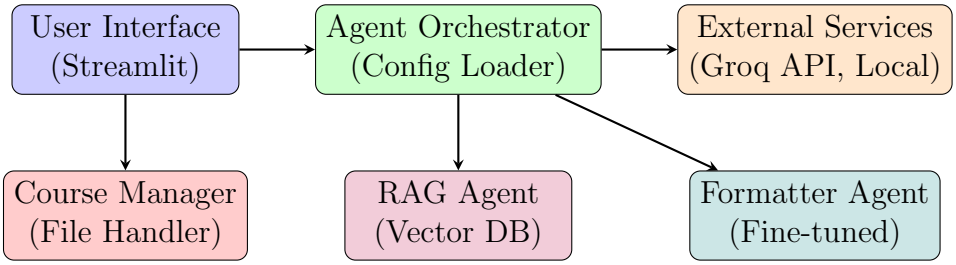

Figure 1: High-Level System Architecture

2.2 Core Design Principles

- **Modularity**: Independent agents with clear interfaces
- **Scalability**: Support for multiple courses and document types
- **Efficiency**: Optimized inference with mixed model strategy
- **User-Centric**: Intuitive interface with progressive disclosure

3 Component Architecture

3.1 User Interface Layer

3.1.1 Streamlit Web Application

Purpose: Provide accessible web interface for non-technical users

Components:

- **Course Management Interface**: File upload, course selection
- **RAG Query Interface**: Natural language question input
- **LaTeX Formatter Interface**: Text input and preview
- **Integrated Workflow**: Combined question-answer-formatting pipeline

Technology Choices:

- **Streamlit**: Rapid prototyping, Python-native, reactive updates

- **Reason**: Eliminates frontend development overhead, perfect for research prototypes

3.2 Agent Orchestration Layer

3.2.1 Configuration Manager

```
class ConfigLoader:
              Model configurations (Groq vs Local)
              Agent parameters (chunk size, temperature)
              Path management (courses, outputs, models)
              Environment variable handling
```

Listing 1: Configuration Manager Structure

Purpose: Centralized configuration management across agents
Technology: YAML configuration files with environment variable support

3.3 Data Management Layer

3.3.1 Course Manager

Components:

- **File Processor**: Handles PDF and TXT uploads
- **Document Storage**: Organized course directory structure
- **Format Handler**: Support for multiple document types

Supported Formats:

- **PDF**: Academic papers, slides, assignments
- **TXT**: Lecture notes, code, plain text content

Storage Structure:

```
courses/
          CS101-Algorithms/
              lecture_notes.pdf
              textbook_chapter.txt
              assignment_solutions.pdf
```

Listing 2: Course Directory Structure

4 AI Agent Components

4.1 RAG (Retrieval-Augmented Generation) Agent

4.1.1 Architecture

```
class RAGAgent:
               Document Loader (PDF, TXT)
               Text Splitter (Recursive character-based)
               Embedding Model (sentence-transformers/all-MiniLM-L6-v2)
               Vector Store (ChromaDB)
               Retrieval Engine (Similarity search)
               LLM Interface (Groq API)
```

Listing 3: RAG Agent Architecture

4.1.2 Processing Pipeline

1. **Document Ingestion** → Load and parse course materials
2. **Chunking** → Split into manageable text segments (1000 chars, 200 overlap)
3. **Vectorization** → Create embeddings using MiniLM model
4. **Storage** → Persist vectors in ChromaDB
5. **Retrieval** → Find relevant chunks for user queries
6. **Generation** → Synthesize answers using Groq's Llama 3.3 70B

4.1.3 Model Choices & Justification

Component	Technology	Rationale
Embedding	`all-MiniLM-L6-v2`	Lightweight (384d), fast, academic-optimized
Vector DB	ChromaDB	Simple, persistent, Python-native
LLM	Groq (Llama 3.3 70B)	High-quality reasoning, fast inference

Key Design Decisions:

- **Chunk Size 1000**: Balances context length with precision
- **Overlap 200**: Maintains context continuity between chunks
- **Top-3 Retrieval**: Provides sufficient context without overload

4.2 Formatter Agent

4.2.1 Architecture

```
class FormatterAgent:
               Model Manager (Base vs Fine-tuned detection)
               Prompt Engineering (LaTeX-specific templates)
               Inference Engine (Local model with optimizations)
               Validation System (LaTeX syntax checking)
```

Listing 4: Formatter Agent Architecture

4.2.2 Processing Pipeline

1. **Input Processing** → Receive text solution and document type
2. **Prompt Construction** → Apply LaTeX-specific templates
3. **Model Inference** → Generate LaTeX using Mistral 7B
4. **Output Validation** → Check for basic LaTeX structure
5. **Error Handling** → Provide warnings for common issues

4.2.3 Model Strategy

Base Model: `mistralai/Mistral-7B-Instruct-v0.2`

- **Reason**: Strong instruction-following capabilities
- **Advantage**: Open-source, commercially usable

Fine-tuning Approach: QLoRA (Quantized Low-Rank Adaptation)

- **Method**: 4-bit quantization with LoRA adapters
- **Target**: LaTeX formatting specialization
- **Efficiency**: ∼1% trainable parameters, runs on consumer hardware

Fine-tuning Data Strategy:

- **Source**: Mixed synthetic and educational content
- **Size**: 50-100 high-quality examples
- **Focus**: Mathematical expressions, academic structures

5 Interaction Flows

5.1 Course Setup Flow

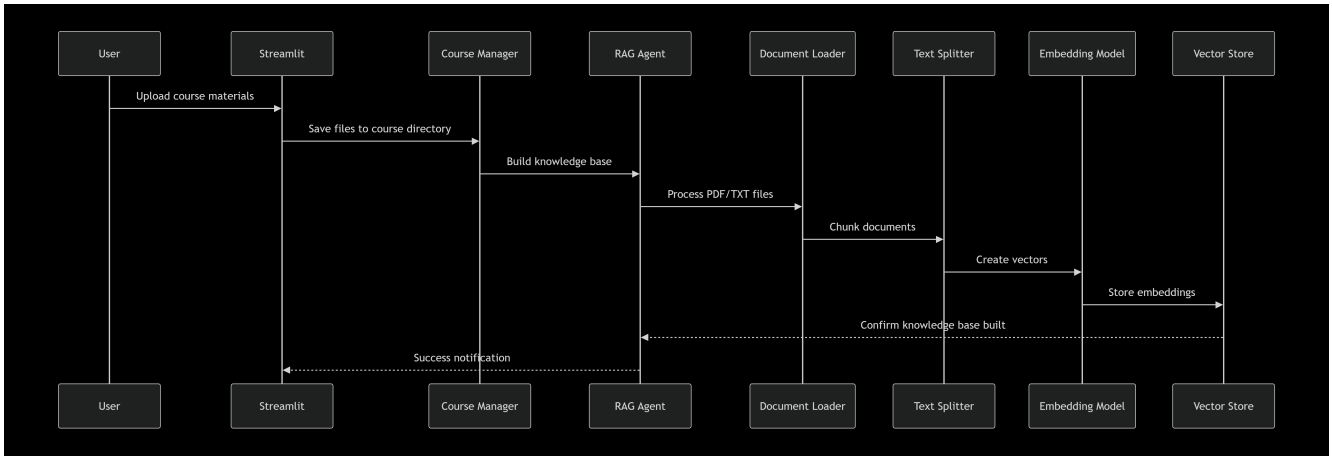

Figure 2: Course setup flow

5.2 RAG Query Flow

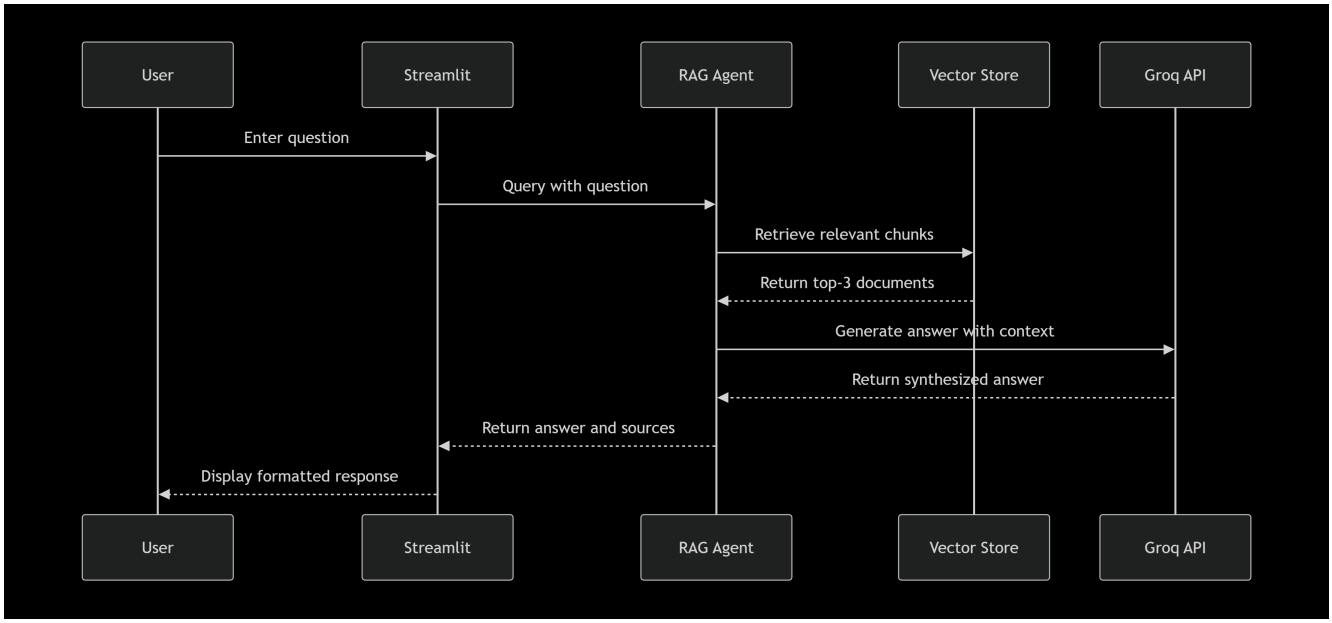

Figure 3: RAG Query Flow

5.3 LaTeX Formatting Flow

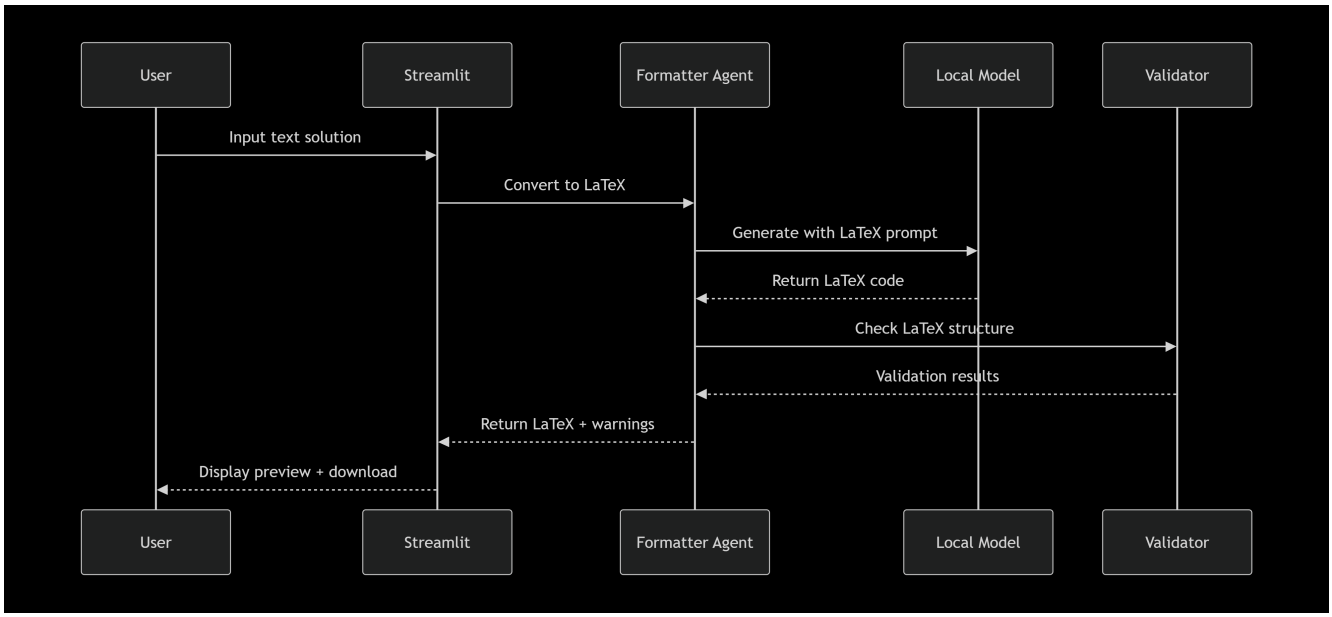

Figure 4: LaTeX Formatting Flow

5.4 Integrated Workflow

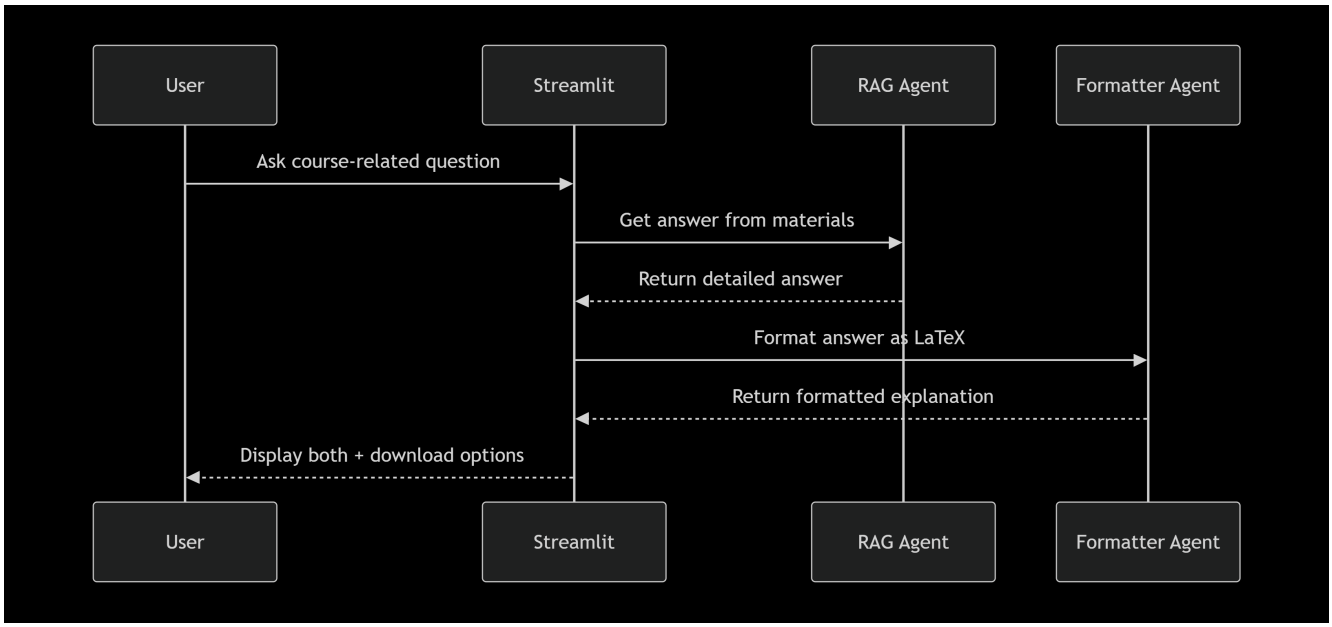

Figure 5: Integrated Workflow

6 Model Selection Justification

6.1 Groq API (RAG Agent)

Model: `llama-3.3-70b-versatile`

Reasoning:

- **Quality**: 70B parameter model provides excellent reasoning capabilities
- **Speed**: Groq's LPU architecture offers exceptional inference speed
- **Cost**: Efficient pricing for research and development
- **Context**: 128K token context window handles large documents

Alternative Considered: OpenAI GPT-4

- **Rejected**: Higher cost, slower inference for this use case

6.2 Local Model (Formatter Agent)

Base Model: `mistralai/Mistral-7B-Instruct-v0.2`

Reasoning:

- **Efficiency**: 7B parameters balance quality and resource requirements
- **Instruction-Tuning**: Optimized for following formatting instructions
- **Open Source**: No API dependencies, full control

- **Fine-tuning**: Well-supported for LoRA/QLoRA approaches

Fine-tuning Strategy:

- **QLoRA**: Enables fine-tuning on consumer hardware
- **Targeted Training**: Focus on LaTeX formatting patterns
- **Progressive Improvement**: Start with base, enhance with fine-tuning

6.3 Embedding Model

Model: `sentence-transformers/all-MiniLM-L6-v2`

Reasoning:

- **Performance**: Strong performance on academic text
- **Efficiency**: Small model size (80MB), fast inference
- **Dimension**: 384 dimensions provide good semantic capture
- **Proven**: Widely used in production RAG systems

7 Technical Implementation Details

7.1 Vector Database Design

```
ChromaDB Configuration:
          Collection: per_course
          Metadata: {source, chunk_index, course_name}
          Distance: cosine similarity
          Persistence: local disk storage
```

Listing 5: ChromaDB Configuration

7.2 Prompt Engineering

RAG System Prompt:

```
"You are a helpful study assistant. Answer based ONLY on provided
   context.
If answer cannot be found, say so clearly. Provide concise answers with
    citations."
```

Listing 6: RAG System Prompt

LaTeX Formatter System Prompt:

```
"You are an expert at converting academic solutions to properly
   formatted LaTeX.
Rules: Use $...$ for math, appropriate environments, ensure compilable
   code.
Return ONLY LaTeX without explanations."
```

Listing 7: LaTeX Formatter System Prompt

7.3 Error Handling Strategy

- **Graceful Degradation**: Fallback to base model if fine-tuned unavailable
- **Input Validation**: Check file types, text length, LaTeX structure
- **User Feedback**: Clear error messages with actionable steps
- **Recovery**: Automatic retry with simplified prompts when possible

8 Performance Considerations

8.1 Optimization Strategies

- **Lazy Loading**: Agents initialize only when needed
- **Caching**: Vector stores persist between sessions
- **Batch Processing**: Course setup happens asynchronously
- **Memory Management**: 4-bit quantization for local models

8.2 Scalability Measures

- **Modular Design**: Easy to add new document types or agents
- **Configuration-Driven**: Behavior changes without code modifications
- **Resource Awareness**: Different hardware configurations supported

9 Evaluation Methodology

9.1 RAG Agent Metrics

- **Retrieval Precision**: Relevance of retrieved chunks
- **Answer Quality**: Accuracy and completeness of responses
- **Source Attribution**: Correct citation of source materials
- **Response Time**: End-to-end query processing time

9.2 Formatter Agent Metrics

- **LaTeX Validity**: Percentage of compilable outputs
- **Formatting Quality**: Adherence to LaTeX best practices
- **Mathematical Accuracy**: Correct translation of math expressions
- **User Correction Time**: Time saved versus manual formatting

9.3 Integrated Workflow Metrics

- **End-to-End Accuracy**: Combined RAG + formatting quality
- **User Satisfaction**: Subjective ease of use and utility
- **Time Efficiency**: Total time savings versus manual process

10 Future Architecture Extensions

10.1 Planned Enhancements

1. **Multi-modal Support**: Image-to-LaTeX conversion
2. **Collaborative Features**: Multi-user course sharing
3. **Advanced Analytics**: Learning progress tracking
4. **Plugin System**: Third-party agent integrations

10.2 Scalability Roadmap

- **Cloud Deployment**: Docker containers, Kubernetes orchestration
- **Database Scaling**: Vector database clustering
- **Model Serving**: Dedicated inference servers
- **Caching Layers**: Redis for frequent queries

11 Conclusion

The Study Buddy architecture demonstrates a thoughtful balance between cutting-edge AI capabilities and practical usability. The multi-agent design allows specialized optimization for different tasks while maintaining a cohesive user experience. The choice of mixed inference strategy (cloud API for reasoning, local models for formatting) provides both quality and cost efficiency.

The system's modular architecture ensures maintainability and extensibility, while the Streamlit interface makes advanced AI capabilities accessible to non-technical users. This architecture serves as a solid foundation for both immediate academic use and future research development.

Appendices

Appendix A: Configuration Specifications

```yaml
inference:
  groq:
    model: "llama-3.3-70b-versatile"
    temperature: 0.1
    max_tokens: 1024

  local:
    formatter_model: "mistralai/Mistral-7B-Instruct-v0.2"
    device: "cuda"
    temperature: 0.1
    max_tokens: 2048

agents:
  rag:
    inference: "groq"
    embedding_model: "sentence-transformers/all-MiniLM-L6-v2"
    vector_store: "chroma"
    chunk_size: 1000
    chunk_overlap: 200

  formatter:
    inference: "local"
    base_model: "mistralai/Mistral-7B-Instruct-v0.2"
    fine_tuned_path: "./models/formatter_fine_tuned"
```

Listing 8: Default Configuration

Appendix B: API Documentation

Key endpoints and integration points for external services.

Appendix C: Deployment Guidelines

Step-by-step deployment instructions for various environments.

Appendix D: Troubleshooting Procedures

Common issues and their solutions for system maintenance.