Study Assistant Backend System

Technical Implementation Report

Naveen Prabakar

September 15, 2025

Abstract

This technical report presents a comprehensive analysis of a serverless Study Assistant backend system designed to provide intelligent tutoring across multiple domains including mathematics, SQL, and astronomy. The system leverages a modern cloud-native architecture utilizing Retrieval-Augmented Generation (RAG) with vector databases, containerized deployment, and AWS serverless infrastructure. Key components include PDF document ingestion with OCR capabilities, semantic search using Pinecone vector database, session management with Redis, and multi-domain query processing through OpenAI's GPT models.

Contents

1 Introduction

The Study Assistant represents a sophisticated educational technology solution designed to provide personalized tutoring assistance across multiple academic domains. This system addresses the growing need for accessible, intelligent educational support by combining state-of-the-art natural language processing, vector-based information retrieval, and cloud-native architecture principles.

The backend system serves as the core intelligence layer, processing user queries, retrieving relevant educational content, and generating contextually appropriate responses while maintaining session continuity and providing proper academic citations.

2 System Architecture

2.1 Overview

The Study Assistant employs a serverless, microservices-based architecture designed for scalability, cost-effectiveness, and maintainability. The system follows a three-tier architecture pattern:

- **Presentation Layer**: JavaScript/HTML/CSS frontend
- **Logic Layer**: Python-based Lambda function
- **Data Layer**: Pinecone vector database and Redis cache

2.2 Cloud Infrastructure

The deployment architecture leverages AWS services for optimal performance and scalability:

1. **Amazon ECR**: Container image registry
2. **AWS Lambda**: Serverless compute platform
3. **Amazon API Gateway**: RESTful API management
4. **External Services**: Pinecone vector database, Redis cache, OpenAI API

2.3 Data Flow Architecture

The system implements a sophisticated data flow supporting both document ingestion and query processing:

Document Ingestion Pipeline:

1. PDF document processing with OCR fallback
2. Text chunking and preprocessing
3. Embedding generation using OpenAI's text-embedding-3-small
4. Vector storage in Pinecone with metadata preservation

Query Processing Pipeline:

1. Query embedding generation
2. Semantic similarity search in Pinecone
3. Context retrieval and relevance filtering
4. Prompt engineering with retrieved context
5. Response generation using GPT-5-nano
6. Session state persistence in Redis

3 Technical Implementation

3.1 Core Dependencies

The system utilizes a carefully selected technology stack optimized for performance and functionality:

Component	Technology
PDF Processing	PyMuPDF (fitz)
OCR Engine	Tesseract (pytesseract)
Image Processing	PIL (Python Imaging Library)
Vector Database	Pinecone
Session Management	Redis
Language Model	OpenAI GPT-5-nano
Embedding Model	text-embedding-3-small
Numerical Computing	NumPy

Table 1: Technology Stack Components

3.2 Document Processing System

The document ingestion system implements a robust PDF processing pipeline with intelligent fallback mechanisms:

Text Extraction Algorithm:

1. Primary text extraction using PyMuPDF's native text extraction
2. OCR fallback for image-based or scanned documents
3. High-resolution rendering (300 DPI) for optimal OCR accuracy
4. Page-level granularity preservation for citation purposes

Embedding Strategy: The system employs OpenAI's text-embedding-3-small model (1536 dimensions) for semantic representation, providing:

- High-quality semantic understanding
- Efficient storage and retrieval
- Cross-domain applicability

3.3 Retrieval-Augmented Generation (RAG) Implementation

The RAG system implements sophisticated context retrieval and prompt engineering:

Similarity Search Configuration:

- Cosine similarity metric for semantic matching
- Configurable top-k retrieval (default: 5)
- Minimum similarity threshold (0.1) for relevance filtering
- Namespace-based content organization by subject domain

Prompt Engineering Strategy: The system implements domain-specific prompt templates with structured context integration:

- Role-specific system prompts for mathematics, SQL, and astronomy
- Context-aware prompt construction with retrieved content
- Citation formatting for academic integrity
- Educational disclaimers for responsible use

3.4 Session Management

Redis-based session management provides:

- Conversation history persistence
- Configurable TTL (3600 seconds)
- JSON serialization for complex data structures
- UUID-based session identification

4 Multi-Domain Support

4.1 Subject-Specific Configurations

The system supports three primary domains with tailored configurations:

Mathematics Domain:

- Step-by-step problem solving approach
- Mathematical notation preservation
- Theorem and proof support
- Formula explanation capabilities

SQL Domain:

- Query optimization focus

- Syntax correctness emphasis
- Performance consideration integration
- Database-specific dialect support

Astronomy Domain:

- Conceptual explanation priority
- Scientific citation requirements
- Visual concept description
- Current astronomical data integration

4.2 Namespace Architecture

Content organization utilizes Pinecone namespaces for efficient domain separation:

- Isolated vector spaces per subject
- Domain-specific retrieval optimization
- Flexible content management
- Scalable architecture for new domains

5 Deployment Strategy

5.1 Containerization

The system employs Docker containerization for consistent deployment:

Container Benefits:

- Environment consistency across development and production
- Dependency isolation and management
- Simplified deployment pipeline
- Scalability optimization

5.2 AWS Serverless Architecture

The deployment leverages AWS serverless services for optimal cost and performance:

Lambda Function Configuration:

- Event-driven execution model
- Automatic scaling capabilities
- Pay-per-request pricing model

- Cold start optimization through container reuse

API Gateway Integration:

- RESTful API endpoint management
- Request/response transformation
- Authentication and authorization support
- Rate limiting and throttling

5.3 Container Registry Management

Amazon ECR provides secure container image management:

- Secure image storage and versioning
- Integration with Lambda deployment
- Automated vulnerability scanning
- Access control and permissions

6 Performance Considerations

6.1 Optimization Strategies

The system implements several performance optimization techniques:

Vector Search Optimization:

- Configurable similarity thresholds
- Batch processing for document ingestion
- Efficient metadata storage and retrieval
- Namespace-based content partitioning

Memory Management:

- Numpy arrays for efficient embedding storage
- JSON serialization for Redis operations
- Batch upsert operations (100 vectors per batch)
- Preview text truncation (500 characters)

6.2 Scalability Architecture

The serverless architecture provides inherent scalability benefits:

- Automatic function scaling based on demand
- No infrastructure management overhead
- Cost optimization through pay-per-use model
- Global availability through AWS regions

7 Security and Compliance

7.1 API Security

The system implements security best practices:

- API key authentication for external services
- Environment variable management for sensitive data
- HTTPS encryption for all communications
- Input validation and sanitization

7.2 Data Privacy

Privacy considerations include:

- Session-based data isolation
- Configurable data retention policies
- No persistent storage of user queries
- Anonymized session identifiers

8 Error Handling and Reliability

8.1 Fault Tolerance

The system implements comprehensive error handling:

- OCR fallback for PDF processing failures
- Graceful degradation for missing context
- Redis connection error handling
- API rate limit management

8.2 Monitoring and Observability

Operational monitoring includes:

- Lambda function metrics and logging
- API Gateway request tracking
- External service health monitoring
- Performance metric collection

9 Future Enhancements

9.1 Planned Improvements

Potential system enhancements include:

- Multi-modal content support (images, diagrams)
- Advanced citation and reference management
- Personalized learning path recommendations
- Integration with additional educational domains
- Real-time collaborative features

9.2 Scalability Roadmap

Future scaling considerations:

- Microservices decomposition
- Multi-region deployment
- Advanced caching strategies
- Machine learning model optimization

10 Conclusion

The Study Assistant backend system represents a sophisticated implementation of modern cloud-native architecture principles combined with advanced natural language processing capabilities. The system successfully addresses key educational technology challenges through:

- Scalable serverless architecture
- Intelligent content retrieval and generation
- Multi-domain subject support

- Robust error handling and reliability
- Cost-effective deployment strategy

The implementation demonstrates effective utilization of cutting-edge technologies including vector databases, large language models, and containerized serverless deployment. The modular architecture provides excellent foundations for future enhancements and scaling requirements.

The system's design prioritizes educational effectiveness while maintaining technical excellence, providing a solid platform for intelligent tutoring applications across multiple academic domains.

11 Technical Appendix

11.1 Configuration Parameters

Parameter	Value	Purpose
EMBED_MODEL	text-embedding-3-small	Semantic embedding generation
GEN_MODEL	gpt-5-nano	Response generation
TOP_K	5	Retrieved context limit
MIN_SIM	0.1	Minimum similarity threshold
SESSION_TTL	3600	Session expiration (seconds)
PREVIEW_LEN	500	Text preview truncation

Table 2: System Configuration Parameters

11.2 API Endpoints

The system exposes a single POST endpoint through API Gateway:

- **Endpoint**: /study-assistant
- **Method**: POST
- **Content-Type**: application/json
- **Parameters**: subject, query, session_id, namespace