Technical Documentation

Academic Assignment Helper & Plagiarism Detector (RAG-Powered)

□ RAG Pipeline Architecture Explanation

Overview

The Retrieval-Augmented Generation (RAG) pipeline combines vector similarity search with large language models to provide contextually relevant academic research suggestions and plagiarism detection.

Architecture Components

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│  Text Input     │───▶│  Embedding      │───▶│  Vector Store   │
│  (Assignment)   │    │  Generation     │    │  (PostgreSQL)   │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                      │                      │
         ▼                      ▼                      ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│  Query Vector   │───▶│  Similarity     │───▶│  Top-K Sources  │
│  (1536-dim)     │    │  Search         │    │  Retrieved      │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                      │                      │
         ▼                      ▼                      ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│  Context        │───▶│  LLM Analysis   │───▶│  Structured     │
│  Augmentation   │    │  (OpenAI/Gemini)│    │  Response       │
└─────────────────┘    └─────────────────┘    └─────────────────┘
```

Pipeline Stages

1. Document Ingestion

- **Input**: Academic papers (title, authors, abstract, full_text)
- **Processing**: Text concatenation and chunking
- **Output**: Structured text ready for embedding

```
def build_source_text(source: AcademicSource) -> str:
    parts = [source.title or "", source.authors or "",
             source.abstract or "", source.full_text or ""]
    return "\n\n".join([p for p in parts if p])
```

2. Embedding Generation

- **Models**: Gemini `text-embedding-004` (primary), OpenAI `text-embedding-ada-002` (fallback)
- **Dimensions**: 1536-dimensional vectors
- **Normalization**: Padding/trimming to ensure consistent vector size

```
def _pad_or_trim(vec: List[float], size: int) -> List[float]:
    if len(vec) == size:
        return vec
    if len(vec) > size:
        return vec[:size]
    return vec + [0.0] * (size - len(vec))
```

3. Vector Storage

- **Database**: PostgreSQL with pgvector extension
- **Index Type**: HNSW for efficient similarity search
- **Distance Metric**: Cosine similarity for semantic relevance

4. Query Processing

- **Input**: Assignment text or search query
- **Embedding**: Same model as document ingestion
- **Search**: Cosine distance ranking with configurable limit

```
def vector_search(db: Session, query: str, limit: int = 5) -> List[AcademicSource]:
    [q_vec] = embed_texts([query])
    return (
        db.query(AcademicSource)
        .order_by(AcademicSource.embedding.cosine_distance(q_vec))
        .limit(limit)
        .all()
    )
```

5. Context Augmentation

- **Retrieved Sources**: Top-K most similar academic papers
- **Context Window**: Optimized for LLM token limits
- **Relevance Filtering**: Similarity threshold-based filtering

Performance Considerations

- **Embedding Caching**: Vectors stored persistently in database
- **Batch Processing**: Multiple texts embedded in single API call
- **Error Handling**: Graceful fallback between embedding providers
- **Scalability**: Horizontal scaling through database sharding

---

□ AI Prompting Strategy Documentation

Prompt Engineering Approach

Our system employs a **multi-stage prompting strategy** that combines retrieved context with structured instructions to generate accurate, relevant responses.

Core Prompting Principles

1. Context-Aware Prompting

```
System: You are an academic writing assistant with access to relevant research papers.

Context: [Retrieved academic sources from RAG pipeline]

Task: Analyze the student's assignment and provide:
1. Research suggestions based on provided sources
2. Plagiarism detection with specific similarity scores
3. Citation recommendations in academic format
```

2. Structured Output Formatting

```
Please respond in the following JSON format:
{
  "plagiarism_score": <float 0-1>,
  "flagged_sections": [
    {
      "text": "<flagged text>",
      "similarity_score": <float>,
      "source": "<source reference>"
    }
  ],
  "research_suggestions": "<detailed suggestions>",
  "citation_recommendations": "<APA/MLA format citations>",
  "confidence_score": <float 0-1>
}
```

Prompt Templates by Task

Research Suggestion Prompt

```
Based on the following academic sources and the student's assignment topic:

SOURCES:
{retrieved_sources}

ASSIGNMENT EXCERPT:
{assignment_text}

Provide specific research suggestions that:
1. Connect assignment themes to relevant academic literature
2. Identify knowledge gaps that need additional research
3. Suggest methodological approaches based on source papers
4. Recommend specific authors or research groups to explore

Focus on actionable, specific guidance rather than generic advice.
```

Plagiarism Detection Prompt

```
Compare the following student text against academic sources:

STUDENT TEXT:
{assignment_text}

REFERENCE SOURCES:
{retrieved_sources}

Identify:
1. Exact or near-exact matches (>90% similarity)
2. Paraphrased content (70-90% similarity)
3. Conceptual similarities (50-70% similarity)

For each flagged section, provide:
- Exact text match
- Similarity percentage
- Source reference
- Recommendation (citation needed, rewrite suggested, etc.)
```

Citation Recommendation Prompt

```
Given these academic sources relevant to the assignment:

SOURCES:
{retrieved_sources}

ASSIGNMENT TOPIC:
{assignment_topic}

Generate proper academic citations in APA format for:
1. Most relevant sources for the assignment topic
2. Foundational papers that should be referenced
3. Recent publications (last 3 years) for current perspectives
4. Methodological sources if applicable

Include both in-text citation examples and full reference list entries.
```

Prompt Optimization Techniques

1. Few-Shot Learning

```
Example 1:
Input: "Machine learning algorithms for classification"
Output: {
  "research_suggestions": "Consider exploring ensemble methods (Random Forest, XGBoost) as discussed in Smith et al. (2023). The pape
  "confidence_score": 0.92
}

Example 2:
Input: "Database normalization techniques"
Output: {
  "research_suggestions": "Build upon the normalization framework presented in Johnson & Lee (2022). Their third normal form analysis
  "confidence_score": 0.88
}

Now analyze: {user_input}
```

2. Chain-of-Thought Reasoning

```
Let me analyze this step by step:

1. First, I'll identify the main themes in the assignment
2. Then, I'll match these themes to the retrieved academic sources
3. Next, I'll assess the quality and relevance of each source
4. Finally, I'll generate specific, actionable recommendations

Step 1 Analysis: The assignment focuses on...
Step 2 Source Matching: The most relevant sources are...
Step 3 Quality Assessment: These sources are authoritative because...
Step 4 Recommendations: Based on this analysis, I suggest...
```

3. Role-Based Prompting

```
You are a senior academic advisor with 15 years of experience in computer science research. Your expertise includes:
- Identifying high-quality academic sources
- Recognizing plagiarism patterns
- Guiding students toward impactful research directions
- Understanding citation standards across academic disciplines

Approach this assignment analysis with the rigor and insight of an experienced academic mentor.
```

Quality Assurance Measures

1. Response Validation

- JSON schema validation for structured outputs
- Confidence score thresholding (minimum 0.7)
- Source attribution verification
- Plagiarism score reasonableness checks

2. Fallback Strategies

- Multiple prompt variations for robustness
- Temperature adjustment for consistency
- Token limit management for long contexts
- Error handling for malformed responses

□ Database Schema Reasoning

Schema Design Philosophy

The database schema follows **normalized relational design principles** while incorporating **modern vector storage capabilities** for RAG functionality.

Core Design Decisions

1. Entity Relationship Model

```
Students (1) ──── (M) Assignments (1) ──── (M) AnalysisResults
                                                    │
AcademicSources ────────────────────────────────────┘
     (M)                                        (M)
```

2. Table Structures

Students Table

```
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,           -- Business key, unique identifier
    password_hash TEXT NOT NULL,          -- bcrypt hashed, never plain text
    full_name TEXT NOT NULL,              -- Display name
    student_id TEXT UNIQUE NOT NULL,      -- Institution-specific ID
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Design Rationale:

- **SERIAL PRIMARY KEY**: Auto-incrementing for performance
- **UNIQUE constraints**: Prevent duplicate accounts
- **TEXT vs VARCHAR**: PostgreSQL optimizes TEXT automatically
- **Timestamp tracking**: Audit trail for account creation

Assignments Table

```
CREATE TABLE assignments (
    id SERIAL PRIMARY KEY,
    student_id INTEGER REFERENCES students(id) NOT NULL,
    filename TEXT NOT NULL,               -- Original file path/name
    original_text TEXT,                   -- Extracted content
    topic TEXT,                          -- Auto-extracted or user-provided
    academic_level TEXT,                 -- Undergraduate, Graduate, etc.
    word_count INTEGER,                  -- Content metrics
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Design Rationale:

- **Foreign Key**: Ensures referential integrity
- **Flexible text storage**: Handles various document sizes
- **Metadata capture**: Enables analytics and filtering
- **Nullable fields**: Allows partial information during processing

Analysis Results Table

```
CREATE TABLE analysis_results (
    id SERIAL PRIMARY KEY,
    assignment_id INTEGER REFERENCES assignments(id) NOT NULL,
    suggested_sources JSONB,             -- Structured source recommendations
    plagiarism_score FLOAT,              -- 0.0 to 1.0 similarity score
    flagged_sections JSONB,              -- Array of flagged text sections
    research_suggestions TEXT,           -- AI-generated recommendations
    citation_recommendations TEXT,       -- Formatted citations
    confidence_score FLOAT,              -- Analysis confidence 0.0 to 1.0
    analyzed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Design Rationale:

- **JSONB over JSON**: Better performance, indexing support
- **Flexible schema**: Accommodates evolving AI output formats
- **Numeric scores**: Enables statistical analysis and thresholds
- **Timestamp tracking**: Audit trail for analysis completion

Academic Sources Table

```
CREATE TABLE academic_sources (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,                 -- Paper/book title
    authors TEXT NOT NULL,               -- Comma-separated author list
    publication_year INTEGER,            -- Year for temporal filtering
    abstract TEXT,                       -- Summary for context
    full_text TEXT,                      -- Complete content for analysis
    source_type TEXT,                    -- 'paper', 'textbook', 'course_material'
    embedding VECTOR(1536),              -- pgvector for similarity search
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Design Rationale:

- **Vector column**: Enables semantic search via pgvector
- **1536 dimensions**: Matches OpenAI/Gemini embedding size
- **Source typing**: Enables filtering by content type
- **Full text storage**: Supports comprehensive analysis
- **Flexible author format**: Handles varying citation styles

Indexing Strategy

Performance Indexes

```
-- Primary lookups
CREATE INDEX idx_students_email ON students(email);
CREATE INDEX idx_assignments_student_id ON assignments(student_id);
CREATE INDEX idx_analysis_assignment_id ON analysis_results(assignment_id);

-- Vector similarity search
CREATE INDEX idx_sources_embedding ON academic_sources
USING hnsw (embedding vector_cosine_ops);

-- Temporal queries
CREATE INDEX idx_assignments_uploaded_at ON assignments(uploaded_at);
CREATE INDEX idx_analysis_analyzed_at ON analysis_results(analyzed_at);

-- Content filtering
CREATE INDEX idx_sources_type ON academic_sources(source_type);
CREATE INDEX idx_sources_year ON academic_sources(publication_year);
```

JSONB Indexing

```
-- Enable fast queries on JSON structures
CREATE INDEX idx_analysis_plagiarism ON analysis_results
USING gin ((flagged_sections->'sections'));

CREATE INDEX idx_analysis_sources ON analysis_results
USING gin (suggested_sources);
```

Data Integrity Constraints

Business Rules

```
-- Ensure valid score ranges
ALTER TABLE analysis_results
ADD CONSTRAINT check_plagiarism_score
CHECK (plagiarism_score >= 0.0 AND plagiarism_score <= 1.0);

ALTER TABLE analysis_results
ADD CONSTRAINT check_confidence_score
CHECK (confidence_score >= 0.0 AND confidence_score <= 1.0);

-- Ensure valid publication years
ALTER TABLE academic_sources
ADD CONSTRAINT check_publication_year
CHECK (publication_year >= 1900 AND publication_year <= EXTRACT(YEAR FROM CURRENT_DATE) + 1);
```

Referential Integrity

```
-- Cascade deletes for data consistency
ALTER TABLE assignments
ADD CONSTRAINT fk_assignments_student
FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE;

ALTER TABLE analysis_results
ADD CONSTRAINT fk_analysis_assignment
FOREIGN KEY (assignment_id) REFERENCES assignments(id) ON DELETE CASCADE;
```

Scalability Considerations

Partitioning Strategy

```
-- Partition analysis_results by date for large datasets
CREATE TABLE analysis_results_2024 PARTITION OF analysis_results
FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');
```

Archive Strategy

```
-- Move old assignments to archive table
CREATE TABLE assignments_archive (LIKE assignments INCLUDING ALL);
```

□ Security Implementation Details

Authentication & Authorization

JWT Implementation

```
# Token Generation
def create_access_token(subject: str) -> str:
    expire = datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode = {"exp": expire, "sub": subject}
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Security Features:

- **HS256 Algorithm**: Symmetric signing for performance
- **Expiration**: 30-minute token lifetime (configurable)
- **Subject Claim**: Student email for identification
- **Secret Rotation**: Environment-based secret management

Password Security

```
# Password Hashing
def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)
```

Security Features:

- **bcrypt Algorithm**: Adaptive hashing with salt
- **Cost Factor**: 12 rounds (configurable for performance/security balance)
- **Salt Generation**: Automatic unique salt per password
- **Timing Attack Resistance**: Constant-time verification

API Security

CORS Configuration

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5678", "http://localhost:8080"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Security Rationale:

- **Restricted Origins**: Only trusted domains allowed
- **Credential Support**: Enables authenticated requests
- **Method Restrictions**: Can be tightened for production
- **Header Control**: Prevents unauthorized header injection

Input Validation

```
class RegisterRequest(BaseModel):
    email: EmailStr                    # Email format validation
    password: str = Field(min_length=6) # Minimum password strength
    full_name: str
    student_id: str

    @validator('password')
    def validate_password(cls, v):
        if not re.search(r'[A-Za-z]', v):
            raise ValueError('Password must contain letters')
        if not re.search(r'[0-9]', v):
            raise ValueError('Password must contain numbers')
        return v
```

Validation Features:

- **Pydantic Models**: Automatic type validation
- **Email Validation**: Format and domain checking
- **Password Complexity**: Configurable strength requirements
- **SQL Injection Prevention**: Parameterized queries via SQLAlchemy

File Upload Security

```
ALLOWED_EXTENSIONS = {'pdf', 'docx', 'txt'}
MAX_FILE_SIZE = 10 * 1024 * 1024  # 10MB

def validate_file(file: UploadFile) -> bool:
    # File type validation
    if not file.filename.lower().endswith(tuple(ALLOWED_EXTENSIONS)):
        raise HTTPException(400, "Invalid file type")

    # File size validation
    if file.size > MAX_FILE_SIZE:
        raise HTTPException(400, "File too large")

    # Content type validation
    if file.content_type not in ALLOWED_MIME_TYPES:
        raise HTTPException(400, "Invalid content type")

    return True
```

Security Features:

- **Extension Filtering**: Whitelist approach for file types
- **Size Limits**: Prevents DoS via large uploads
- **MIME Type Validation**: Double-checks file content
- **Path Traversal Prevention**: Sanitized file names

Database Security

Connection Security

```
# Environment-based configuration
DATABASE_URL = os.getenv("DATABASE_URL")
if not DATABASE_URL.startswith("postgresql://"):
    raise ValueError("Invalid database URL scheme")

# Connection pooling with limits
engine = create_engine(
    DATABASE_URL,
    pool_size=10,
    max_overflow=20,
    pool_pre_ping=True,
    echo=False  # Disable SQL logging in production
)
```

Security Features:

- **Environment Variables**: Secrets not in code
- **Connection Pooling**: Prevents connection exhaustion
- **SSL Enforcement**: Encrypted database connections
- **Query Logging Control**: Prevents credential exposure

SQL Injection Prevention

```
# Parameterized queries via SQLAlchemy ORM
def get_student_by_email(db: Session, email: str) -> Student:
    return db.query(Student).filter(Student.email == email).first()

# Raw queries use parameters
def custom_query(db: Session, student_id: int):
    return db.execute(
        text("SELECT * FROM assignments WHERE student_id = :student_id"),
        {"student_id": student_id}
    ).fetchall()
```

Infrastructure Security

Docker Security

```
# Non-root user
RUN adduser --disabled-password --gecos '' appuser
USER appuser

# Minimal base image
FROM python:3.11-slim

# Security updates
RUN apt-get update && apt-get upgrade -y && rm -rf /var/lib/apt/lists/*
```

Environment Management

```
# .env file (not in version control)
JWT_SECRET_KEY=complex_random_string_here
DATABASE_PASSWORD=strong_database_password
OPENAI_API_KEY=sk-...
GEMINI_API_KEY=AI...

# Production considerations
ENVIRONMENT=production
DEBUG=false
LOG_LEVEL=warning
```

Monitoring & Logging

Security Logging

```
import logging

# Security event logging
def log_security_event(event_type: str, user_id: int, details: dict):
    logger.warning(f"SECURITY_EVENT: {event_type}", extra={
        "user_id": user_id,
        "event_type": event_type,
        "details": details,
        "timestamp": datetime.utcnow(),
        "ip_address": request.client.host
    })

# Failed login attempts
@app.post("/auth/login")
def login(payload: LoginRequest):
    try:
        # ... authentication logic
        pass
    except AuthenticationError:
        log_security_event("FAILED_LOGIN", None, {"email": payload.email})
        raise HTTPException(401, "Invalid credentials")
```

Rate Limiting

```
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/auth/login")
@limiter.limit("5/minute")  # 5 attempts per minute
def login(request: Request, payload: LoginRequest):
    # ... login logic
```

Production Security Checklist

Deployment Security

- ☐ **HTTPS Only**: SSL/TLS certificates configured
- ☐ **Secret Management**: Environment variables, not hardcoded
- ☐ **Database Encryption**: At-rest and in-transit encryption
- ☐ **Network Isolation**: Private networks for internal communication
- ☐ **Firewall Rules**: Restricted port access
- ☐ **Regular Updates**: Security patches for all dependencies

Monitoring & Alerting

- ☐ **Failed Authentication**: Alert on multiple failed attempts
- ☐ **Unusual Activity**: Large file uploads, rapid API calls
- ☐ **Error Rates**: Monitor for potential attacks
- ☐ **Resource Usage**: Detect DoS attempts
- ☐ **Data Access**: Log all sensitive data operations

□ Performance Metrics & Monitoring

Key Performance Indicators

System Performance

- **API Response Time**: < 200ms for auth, < 2s for analysis
- **Vector Search Latency**: < 100ms for similarity queries
- **File Processing Time**: < 30s for typical assignments
- **Database Query Performance**: < 50ms for indexed queries

Business Metrics

- **Analysis Accuracy**: Confidence scores > 0.8
- **User Satisfaction**: Successful workflow completion rate
- **System Reliability**: 99.9% uptime target
- **Scalability**: Support for 1000+ concurrent users

Monitoring Implementation

```
# Performance tracking
import time
from functools import wraps

def monitor_performance(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        duration = time.time() - start_time

        logger.info(f"PERFORMANCE: {func.__name__} took {duration:.3f}s")
        return result
    return wrapper

@monitor_performance
def vector_search(db: Session, query: str, limit: int = 5):
    # ... search implementation
```

□ Future Enhancements

Scalability Improvements

1. **Microservices Architecture**: Split into auth, analysis, and RAG services
2. **Caching Layer**: Redis for frequently accessed embeddings
3. **Load Balancing**: Multiple backend instances
4. **Database Sharding**: Partition by institution or date

Feature Enhancements

1. **Multi-language Support**: Embeddings for non-English content
2. **Real-time Collaboration**: WebSocket-based live analysis
3. **Advanced Analytics**: Plagiarism trend analysis
4. **Integration APIs**: LMS integration (Canvas, Blackboard)

Security Enhancements

1. **OAuth Integration**: SSO with institutional identity providers
2. **Audit Logging**: Comprehensive security event tracking
3. **Data Encryption**: Field-level encryption for sensitive data
4. **Zero-Trust Architecture**: Enhanced network security

This technical documentation provides comprehensive coverage of the system's architecture, implementation details, and security considerations for the Academic Assignment Helper & Plagiarism Detector (RAG-Powered) system.