Technical Specifications Document

AI-Powered Research Agent with RAG Capabilities

Document Information

- **Version**: 1.0
- **Date**: June 1, 2025
- **Author**: [Your Name]
- **Project Code**: RA-2025-001-TECH

---

1. System Architecture

1.1 High-Level Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────┐
│                      Research Agent System                      │
├─────────────────────────────────────────────────────────────────┤
│                                                                │
│  ┌─────────────┐    ┌─────────────┐    ┌─────────────┐         │
│  │    n8n      │    │   FastAPI   │    │  Streamlit  │         │
│  │ Orchestrator│◄──►│   Gateway   │◄──►│  Frontend   │         │
│  └─────────────┘    └─────────────┘    └─────────────┘         │
│                            │                                    │
│  ┌─────────────────────────────────────────────────────────────┐ │
│  │                 CrewAI Multi-Agent System                  │ │
│  │  ┌───────────┐  ┌───────────┐  ┌───────────┐  ┌─────────┐  │ │
│  │  │ Document  │  │ Research  │  │ RAG Query │  │ Analysis│  │ │
│  │  │ Agent     │  │ Agent     │  │ Agent     │  │ Agent   │  │ │
│  │  └───────────┘  └───────────┘  └───────────┘  └─────────┘  │ │
│  │  ┌───────────┐  ┌───────────┐                              │ │
│  │  │ Writer    │  │ QA        │                              │ │
│  │  │ Agent     │  │ Agent     │                              │ │
│  │  └───────────┘  └───────────┘                              │ │
│  └─────────────────────────────────────────────────────────────┘ │
│                            │                                    │
│  ┌─────────────┐    ┌─────────────┐    ┌─────────────┐         │
│  │   Milvus    │    │  OpenRouter │    │   Storage   │         │
│  │  Vector DB  │    │  LLM APIs   │    │   Layer     │         │
│  └─────────────┘    └─────────────┘    └─────────────┘         │
└─────────────────────────────────────────────────────────────────┘
```

1.2 Component Architecture

1.2.1 Presentation Layer

- **Streamlit Frontend**: Interactive web interface for Version 1.0
- **REST API Gateway**: FastAPI-based service layer
- **n8n Orchestrator**: Workflow automation and integration hub

1.2.2 Application Layer

- **CrewAI Framework**: Multi-agent coordination and task management
- **Agent Services**: Specialized microservices for specific research tasks
- **Task Queue**: Asynchronous job processing with Celery

1.2.3 Data Layer

- **Milvus Vector Database**: Embedding storage and similarity search
- **SQLite/PostgreSQL**: Metadata, job status, and user data
- **File Storage**: Document repository and report artifacts

1.2.4 Integration Layer

- **OpenRouter API**: Multi-LLM access and management
- **Web Scraping Services**: Content extraction and processing
- **Document Processing**: PDF, DOCX, TXT parsing and chunking

---

2. Agent Specifications

2.1 Agent Communication Protocol

python

```python
# Standard Agent Message Format
{
    "agent_id": "string",
    "task_id": "string",
    "message_type": "request|response|status",
    "payload": {
        "data": {},
        "metadata": {},
        "context": []
    },
    "timestamp": "ISO-8601",
    "priority": "low|medium|high"
}
```

2.2 Document Processing Agent

Technical Specifications:

- **Input**: File uploads (PDF, DOCX, TXT)
- **Processing**: PyMuPDF4LLM for clean text extraction
- **Chunking**: RecursiveCharacterTextSplitter (chunk_size=1000, overlap=200)
- **Embeddings**: sentence-transformers (all-MiniLM-L6-v2) - FREE local embeddings
- **LLM**: Ollama llama3.1:8b for content classification and metadata extraction
- **Output**: Vector embeddings stored in Milvus collection

Implementation Details:

python

```python
from sentence_transformers import SentenceTransformer
import ollama

class DocumentProcessingAgent:
    def __init__(self):
        # Free local embeddings
        self.embeddings = SentenceTransformer('all-MiniLM-L6-v2')
        self.llm_model = "llama3.1:8b"  # Local Ollama model
        self.vector_store = Milvus(
            embedding_function=self.embeddings,
            connection_args={"host": "localhost", "port": "19530"}
        )

    def process_document(self, file_path: str) -> Dict:
        # Extract text using PyMuPDF4LLM
        chunks = self.chunk_document(file_path)

        # Generate embeddings locally (FREE)
        embeddings = self.embeddings.encode(chunks)

        # Extract metadata using local LLM (FREE)
        metadata = ollama.chat(
            model=self.llm_model,
            messages=[{"role": "user", "content": f"Extract key metadata from: {chunks
        )

        return {"chunks": chunks, "embeddings": embeddings, "metadata": metadata}
```

2.3 Web Research Agent

Technical Specifications:

- **Search APIs**: Brave Search API (free tier), SerpAPI (fallback)
- **Content Extraction**: Jina Reader API (free tier) + local parsing
- **LLM Processing**: Ollama gemma2:9b for content analysis and source validation
- **Rate Limiting**: 10 requests/minute per API
- **Deduplication**: Content hashing and local similarity detection
- **Output**: Structured research data with source metadata

Web Scraping Configuration:

python

```python
import ollama

SCRAPING_CONFIG = {
    "user_agent": "ResearchAgent/1.0",
    "timeout": 30,
    "max_retries": 3,
    "rate_limit": 1.0,  # seconds between requests
    "allowed_domains": ["*"],  # Configurable whitelist
    "content_types": ["text/html", "application/pdf"]
}

class WebResearchAgent:
    def __init__(self):
        self.llm_model = "gemma2:9b"  # Local model for analysis

    def analyze_source_credibility(self, content: str, url: str) -> float:
        prompt = f"""Analyze the credibility of this source:
        URL: {url}
        Content preview: {content[:500]}
        Rate credibility from 0.0 to 1.0 and explain."""

        response = ollama.chat(
            model=self.llm_model,
            messages=[{"role": "user", "content": prompt}]
        )
        return self.extract_credibility_score(response)
```

2.4 RAG Query Agent

Technical Specifications:

- **Vector Search**: Milvus similarity search with configurable top_k
- **Embeddings**: Local sentence-transformers (all-MiniLM-L6-v2) - FREE
- **Hybrid Search**: Combines semantic and keyword matching
- **LLM Processing**: Ollama qwen2.5:14b for context synthesis and query understanding
- **Context Window**: Dynamic context assembly based on query complexity

Search Strategy:

python

```python
import ollama
from sentence_transformers import SentenceTransformer

class RAGQueryAgent:
    def __init__(self):
        self.embeddings = SentenceTransformer('all-MiniLM-L6-v2')
        self.llm_model = "qwen2.5:14b"  # Excellent for reasoning and synthesis

    def hybrid_search(self, query: str, top_k: int = 10):
        # Generate query embedding locally (FREE)
        query_embedding = self.embeddings.encode([query])

        # Semantic search using local embeddings
        semantic_results = self.vector_store.similarity_search(
            query_embedding, k=top_k
        )

        # Keyword search
        keyword_results = self.keyword_search(query, k=top_k)

        # Use local LLM to synthesize and rerank results (FREE)
        synthesis_prompt = f"""
        Analyze these search results for query: "{query}"
        Semantic results: {semantic_results[:3]}
        Keyword results: {keyword_results[:3]}
        Rank by relevance and synthesize key findings.
        """

        synthesis = ollama.chat(
            model=self.llm_model,
            messages=[{"role": "user", "content": synthesis_prompt}]
        )

        return self.parse_synthesis_results(synthesis)
```

2.5 Analysis Agent

Technical Specifications:

- **Primary LLM**: Ollama llama3.1:70b for complex analysis and pattern recognition
- **Fallback LLM**: Ollama gemma2:27b for moderate complexity tasks

- **Premium LLM**: Claude 3.5 Sonnet via OpenRouter (only for critical business reports)
- **Context Management**: Handles large contexts with local processing
- **Pattern Recognition**: Advanced reasoning with local models
- **Source Validation**: Cross-references multiple sources using local LLM inference

Implementation Strategy:

```python
import ollama

class AnalysisAgent:
    def __init__(self):
        self.primary_model = "llama3.1:70b"      # Best local reasoning
        self.fallback_model = "gemma2:27b"       # Faster alternative
        self.premium_model = "claude-3.5-sonnet" # Only for critical tasks

    def analyze_findings(self, research_data: Dict, complexity: str = "standard"):
        if complexity == "critical" and self.budget_allows_premium():
            # Use premium model only for critical business reports
            return self.premium_analysis(research_data)
        elif complexity == "complex":
            # Use best local model for complex analysis
            return self.local_analysis(research_data, self.primary_model)
        else:
            # Use faster local model for standard analysis
            return self.local_analysis(research_data, self.fallback_model)

    def local_analysis(self, data: Dict, model: str) -> Dict:
        prompt = f"""
        Analyze the following research findings and identify:
        1. Key patterns and trends
        2. Contradictions or gaps
        3. Strategic insights
        4. Recommendations

        Research Data: {data}
        """

        response = ollama.chat(
            model=model,
            messages=[{"role": "user", "content": prompt}]
        )

        return self.parse_analysis_response(response)
```

2.6 Report Writer Agent

Technical Specifications:

- **Primary LLM**: Ollama qwen2.5:14b (excellent writing capabilities)

- **Professional Reports**: Ollama llama3.1:70b for complex business reports
- **Premium Polish**: Claude 3.5 Sonnet via OpenRouter (only for final executive reports)
- **Templates**: Configurable report structures (Executive, Technical, Research)
- **Citation Format**: APA, MLA, Chicago style support
- **Output Formats**: Markdown, PDF (via WeasyPrint), DOCX
- **Quality Metrics**: Local readability analysis and citation validation

Implementation Strategy:

python

```python
import ollama

class ReportWriterAgent:
    def __init__(self):
        self.writing_model = "qwen2.5:14b"        # Excellent writing
        self.business_model = "llama3.1:70b"      # Complex business logic
        self.premium_model = "claude-3.5-sonnet"  # Final polish only

    def generate_report(self, analysis_data: Dict, report_type: str = "standard"):
        if report_type == "executive" and self.is_client_facing():
            # Use premium model for client-facing executive reports
            draft = self.local_report_generation(analysis_data, self.business_model)
            return self.premium_polish(draft)
        elif report_type == "business":
            # Use best local model for business reports
            return self.local_report_generation(analysis_data, self.business_model)
        else:
            # Use efficient writing model for standard reports
            return self.local_report_generation(analysis_data, self.writing_model)

    def local_report_generation(self, data: Dict, model: str) -> str:
        prompt = f"""
        Create a professional research report with:
        1. Executive Summary
        2. Key Findings
        3. Detailed Analysis
        4. Recommendations
        5. Source Citations

        Analysis Data: {data}
        Format: Professional business report
        """

        response = ollama.chat(
            model=model,
            messages=[{"role": "user", "content": prompt}]
        )

        return response['message']['content']
```

---

3. Database Design

3.1 Vector Database Schema (Milvus)

```python
# Document Collection Schema
DOCUMENT_COLLECTION_SCHEMA = {
    "collection_name": "research_documents",
    "fields": [
        {"name": "id", "type": "INT64", "is_primary": True, "auto_id": True},
        {"name": "document_id", "type": "VARCHAR", "max_length": 100},
        {"name": "chunk_id", "type": "VARCHAR", "max_length": 100},
        {"name": "embedding", "type": "FLOAT_VECTOR", "dim": 1536},
        {"name": "text_content", "type": "VARCHAR", "max_length": 2000},
        {"name": "metadata", "type": "JSON"}
    ],
    "index_params": {
        "index_type": "IVF_FLAT",
        "metric_type": "COSINE",
        "params": {"nlist": 1024}
    }
}
```

3.2 Relational Database Schema (SQLite/PostgreSQL)

sql

```sql
-- Research Jobs Table
CREATE TABLE research_jobs (
    id SERIAL PRIMARY KEY,
    job_id VARCHAR(100) UNIQUE NOT NULL,
    topic VARCHAR(500) NOT NULL,
    status VARCHAR(50) DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP,
    config JSON,
    results JSON
);

-- Documents Table
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    document_id VARCHAR(100) UNIQUE NOT NULL,
    filename VARCHAR(500),
    file_size INTEGER,
    content_type VARCHAR(100),
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processing_status VARCHAR(50),
    chunk_count INTEGER,
    metadata JSON
);

-- Research Sources Table
CREATE TABLE research_sources (
    id SERIAL PRIMARY KEY,
    job_id VARCHAR(100) REFERENCES research_jobs(job_id),
    source_url VARCHAR(1000),
    source_type VARCHAR(50),
    title VARCHAR(500),
    content_summary TEXT,
    credibility_score DECIMAL(3,2),
    retrieved_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

4. API Specifications

4.1 REST API Endpoints

4.1.1 Research Operations

yaml

```yaml
/api/v1/research:
  post:
    summary: Submit research request
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              topic:
                type: string
                description: Research topic or question
              config:
                type: object
                properties:
                  max_sources: {type: integer, default: 20}
                  report_format: {type: string, enum: [markdown, pdf, docx]}
                  use_rag: {type: boolean, default: true}
                  priority: {type: string, enum: [low, medium, high]}
    responses:
      '202':
        content:
          application/json:
            schema:
              type: object
              properties:
                job_id: {type: string}
                status: {type: string}
                estimated_completion: {type: string}

/api/v1/research/{job_id}:
  get:
    summary: Get research job status and results
    responses:
      '200':
        content:
          application/json:
            schema:
              type: object
              properties:
                job_id: {type: string}
                status: {type: string}
```

```
progress: {type: integer}
results: {type: object}
```

4.1.2 Document Management

```yaml
/api/v1/documents:
  post:
    summary: Upload document for RAG processing
    requestBody:
      content:
        multipart/form-data:
          schema:
            type: object
            properties:
              file: {type: string, format: binary}
              tags: {type: array, items: {type: string}}
    responses:
      '201':
        content:
          application/json:
            schema:
              type: object
              properties:
                document_id: {type: string}
                status: {type: string}
                processing_started: {type: boolean}

  get:
    summary: List uploaded documents
    responses:
      '200':
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  document_id: {type: string}
                  filename: {type: string}
                  status: {type: string}
                  chunk_count: {type: integer}
```

---

5. Configuration Management

5.1 Environment Configuration

python

```python
# config/settings.py
from pydantic import BaseSettings

class Settings(BaseSettings):
    # API Configuration (Minimal Usage)
    OPENROUTER_API_KEY: str = ""          # Only for critical tasks
    BRAVE_SEARCH_API_KEY: str = ""        # Free tier
    JINA_API_KEY: str = ""                # Free tier

    # Local Model Configuration
    OLLAMA_HOST: str = "localhost"
    OLLAMA_PORT: int = 11434

    # Model Selection Strategy
    PRIMARY_ANALYSIS_MODEL: str = "llama3.1:70b"
    FAST_PROCESSING_MODEL: str = "llama3.1:8b"
    WRITING_MODEL: str = "qwen2.5:14b"
    REASONING_MODEL: str = "gemma2:27b"

    # Premium Model Usage (Cost Control)
    ENABLE_PREMIUM_MODELS: bool = False
    PREMIUM_USAGE_LIMIT: int = 5          # Max premium calls per day
    MONTHLY_API_BUDGET: float = 15.0      # Maximum monthly spend

    # Database Configuration
    MILVUS_HOST: str = "localhost"
    MILVUS_PORT: int = 19530
    DATABASE_URL: str = "sqlite:///research_agent.db"

    # Agent Configuration
    MAX_CONCURRENT_JOBS: int = 3          # Reduced for local processing
    DEFAULT_CHUNK_SIZE: int = 1000
    DEFAULT_CHUNK_OVERLAP: int = 200
    MAX_SOURCES_PER_RESEARCH: int = 15    # Reduced to control costs

    # Local Embedding Configuration
    EMBEDDING_MODEL: str = "all-MiniLM-L6-v2"
    EMBEDDING_DEVICE: str = "cpu"         # "cuda" if GPU available

    class Config:
        env_file = ".env"
```

5.2 CrewAI Configuration

yaml

```yaml
# crew_config.yaml
agents:
  document_processor:
    role: "Document Processing Specialist"
    goal: "Process and embed documents for RAG capabilities using local models"
    backstory: "Expert in document parsing and local vector embeddings"
    tools: [document_loader, local_embedding_generator, vector_store]
    llm:
      provider: "ollama"
      model: "llama3.1:8b"
      temperature: 0.1

  researcher:
    role: "Web Research Specialist"
    goal: "Find and extract relevant information from web sources using local analysis"
    backstory: "Skilled researcher with expertise in local source validation"
    tools: [web_search, content_extractor, local_source_validator]
    llm:
      provider: "ollama"
      model: "gemma2:9b"
      temperature: 0.3

  rag_specialist:
    role: "Knowledge Retrieval Expert"
    goal: "Query internal documents and synthesize relevant context"
    backstory: "Expert in semantic search and context synthesis"
    tools: [vector_search, context_synthesizer]
    llm:
      provider: "ollama"
      model: "qwen2.5:14b"
      temperature: 0.2

  analyst:
    role: "Research Analyst"
    goal: "Synthesize information and identify key insights using advanced reasoning"
    backstory: "Analytical expert skilled in pattern recognition"
    tools: [content_synthesizer, pattern_analyzer]
    llm:
      provider: "ollama"
      model: "llama3.1:70b"     # Best local model for complex analysis
      temperature: 0.4
      fallback:
        provider: "ollama"
```

```yaml
        model: "gemma2:27b"     # Faster fallback

  writer:
    role: "Report Writer"
    goal: "Create well-structured, professional reports using local models"
    backstory: "Professional writer with expertise in business communication"
    tools: [report_generator, citation_manager]
    llm:
      provider: "ollama"
      model: "qwen2.5:14b"      # Excellent writing capabilities
      temperature: 0.6
      premium_fallback:         # Only for critical reports
        provider: "openrouter"
        model: "anthropic/claude-3.5-sonnet"
        condition: "executive_report"

  quality_assurance:
    role: "Quality Control Specialist"
    goal: "Validate report quality and accuracy using efficient local models"
    backstory: "Detail-oriented expert in quality assurance"
    tools: [quality_checker, citation_validator]
    llm:
      provider: "ollama"
      model: "llama3.1:8b"      # Fast validation
      temperature: 0.1

conductor:
  role: "Model Selection Coordinator"
  goal: "Optimize model selection based on task complexity and cost constraints"
  backstory: "Strategic coordinator focused on cost-effective AI operations"
  decision_matrix:
    simple_tasks: "llama3.1:8b"
    moderate_tasks: "gemma2:27b"
    complex_tasks: "llama3.1:70b"
    writing_tasks: "qwen2.5:14b"
    critical_business: "anthropic/claude-3.5-sonnet"  # Premium only when necessary
  cost_controls:
    daily_premium_limit: 5
    monthly_budget: 15.0
    fallback_strategy: "always_local"

tasks:
  document_processing:
    description: "Process uploaded documents using local embeddings and analysis"
```

```yaml
    expected_output: "Structured document embeddings with metadata"
    cost_tier: "free"

  research_task:
    description: "Conduct comprehensive web research using free APIs and local analysi
    expected_output: "Structured research findings with local source validation"
    cost_tier: "minimal"

  rag_synthesis:
    description: "Query internal documents and synthesize relevant context locally"
    expected_output: "Relevant document context with local reasoning"
    cost_tier: "free"

  analysis_task:
    description: "Analyze research findings using best available local model"
    expected_output: "Analytical summary with patterns and recommendations"
    cost_tier: "free"

  writing_task:
    description: "Generate professional research report using local writing model"
    expected_output: "Formatted report with executive summary and citations"
    cost_tier: "free"
    escalation:
      condition: "executive_report"
      premium_model: "anthropic/claude-3.5-sonnet"
      cost_tier: "premium"
```

---

6. Deployment Specifications

6.1 Version 1.0 - Local Deployment

System Requirements:

- **OS**: Windows 10/11, macOS 12+, Ubuntu 20.04+
- **RAM**: 32GB minimum, 64GB recommended (for large local models)
- **Storage**: 100GB available space (for model storage)
- **GPU**: NVIDIA RTX 4060+ (optional but recommended for faster inference)
- **Python**: 3.9+ with pip
- **Docker**: For Milvus database
- **Ollama**: For local model management

Required Local Models (Download via Ollama):

```bash
# Essential models (download these first)
ollama pull llama3.1:8b          # 4.7GB - Fast processing
ollama pull gemma2:9b            # 5.4GB - Efficient reasoning
ollama pull qwen2.5:14b          # 8.2GB - Excellent writing

# Advanced models (if you have sufficient RAM/storage)
ollama pull llama3.1:70b         # 40GB - Best reasoning (requires 64GB+ RAM)
ollama pull gemma2:27b           # 16GB - Advanced analysis
ollama pull deepseek-coder:6.7b  # 3.8GB - Code-specific tasks

# Total storage needed:
# Basic setup: ~18GB (8b + 9b + 14b models)
# Full setup: ~78GB (all models)
```

Installation Script:

```bash
#!/bin/bash
# setup_local_optimized.sh

echo "Setting up cost-optimized Research Agent..."

# Install Ollama
curl -fsSL https://ollama.ai/install.sh | sh

# Download essential models
ollama pull llama3.1:8b
ollama pull gemma2:9b
ollama pull qwen2.5:14b

# Create virtual environment
python -m venv research_agent_env
source research_agent_env/bin/activate  # Linux/Mac
# research_agent_env\Scripts\activate     # Windows

# Install dependencies
pip install -r requirements.txt

# Setup Milvus with Docker
docker-compose -f docker/milvus-docker-compose.yml up -d

# Initialize database
python scripts/init_db.py

# Configure environment (mostly free services)
cp .env.example .env
echo "Setup complete! Configure your free API keys in .env file"
echo "OpenRouter API key is optional - only needed for premium features"
```

#!/bin/bash

setup_local.sh

Create virtual environment

python -m venv research_agent_env

source research_agent_env/bin/activate # Linux/Mac

research_agent_env\Scripts\activate # Windows

Install dependencies

pip install -r requirements.txt

Setup Milvus with Docker

docker-compose -f docker/milvus-docker-compose.yml up -d

Initialize database

python scripts/init_db.py

Configure environment

cp .env.example .env

echo "Please configure your API keys in .env file"

```
### 6.2 Version 2.0 - Cloud Deployment

#### Infrastructure Requirements:
- **Kubernetes Cluster**: 3+ nodes, 8GB RAM per node
- **Load Balancer**: NGINX Ingress or cloud provider LB
- **Storage**: Persistent volumes for Milvus and PostgreSQL
- **Monitoring**: Prometheus + Grafana stack

#### Kubernetes Manifests:
```yaml
# k8s/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: research-agent

---
# k8s/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: research-agent-config
  namespace: research-agent
data:
  MILVUS_HOST: "milvus-service"
  MILVUS_PORT: "19530"
  MAX_CONCURRENT_JOBS: "10"
```

---

7. Security Specifications

7.1 Authentication & Authorization

API Security:

- **JWT Tokens**: For API authentication (Version 2.0)
- **API Rate Limiting**: 100 requests/hour per user
- **CORS Configuration**: Restricted to allowed origins
- **Input Validation**: Comprehensive request sanitization

Data Security:
```

- **Encryption at Rest**: AES-256 for sensitive data
- **Encryption in Transit**: TLS 1.3 for all communications
- **Secret Management**: Environment variables + HashiCorp Vault (Version 2.0)
- **Access Controls**: Role-based permissions for document access

7.2 Privacy & Compliance

Data Handling:

- **Document Retention**: Configurable purge policies
- **Logging**: No sensitive data in application logs
- **Audit Trail**: All research operations logged with timestamps
- **Data Minimization**: Only required data stored and processed

---

8. Monitoring & Observability

8.1 Application Monitoring

Metrics Collection:

- **Custom Metrics**: Research job completion rates, token usage, error rates
- **Performance Metrics**: Response times, throughput, resource utilization
- **Business Metrics**: User engagement, feature usage, cost per research

Logging Strategy:

python

```python
# logging_config.py
LOGGING_CONFIG = {
    'version': 1,
    'formatters': {
        'detailed': {
            'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
        }
    },
    'handlers': {
        'file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': 'logs/research_agent.log',
            'maxBytes': 10485760,  # 10MB
            'backupCount': 5
        }
    },
    'loggers': {
        'research_agent': {
            'level': 'INFO',
            'handlers': ['file']
        }
    }
}
```

8.2 Health Checks

Endpoint Specifications:

python

```python
@app.get("/health")
async def health_check():
    return {
        "status": "healthy",
        "timestamp": datetime.utcnow(),
        "services": {
            "milvus": check_milvus_connection(),
            "database": check_db_connection(),
            "openrouter": check_api_availability()
        }
    }
```

---

9. Testing Strategy

9.1 Unit Testing

- **Coverage Target**: 90%+ code coverage
- **Framework**: pytest with pytest-asyncio
- **Mocking**: unittest.mock for external API calls
- **Fixtures**: Shared test data and configurations

9.2 Integration Testing

- **Agent Communication**: Test CrewAI workflow execution
- **Database Operations**: Milvus and PostgreSQL integration
- **API Endpoints**: Full request/response cycle testing
- **Document Processing**: End-to-end file processing pipeline

9.3 Performance Testing

- **Load Testing**: Concurrent research job processing
- **Vector Search**: Query performance with large document sets
- **Memory Usage**: Embedding generation and storage efficiency
- **API Response Times**: Under various load conditions

---

10. Maintenance & Support

10.1 Backup & Recovery

- **Database Backups**: Daily automated backups of Milvus and PostgreSQL
- **Document Repository**: Synchronized backup of uploaded files
- **Configuration**: Version-controlled configuration management
- **Recovery Testing**: Monthly disaster recovery drills

10.2 Updates & Patches

- **Dependency Management**: Automated security updates
- **Model Updates**: Seamless LLM model version upgrades
- **Feature Releases**: Blue-green deployment strategy
- **Rollback Procedures**: Quick rollback capabilities for failed deployments