AI-Powered Research Assistant MCP Server

Software Requirements Specification (SRS) Document

Table of Contents

---

1. Introduction

1.1 Purpose

The AI-Powered Research Assistant MCP Server is designed to provide intelligent web research capabilities through the Model Context Protocol (MCP). It combines web scraping, document processing, and local LLM analysis to deliver comprehensive research insights without requiring external API costs.

1.2 Scope

This system will enable users to:

- Conduct intelligent web searches across multiple sources
- Extract and analyze content from web pages and documents
- Generate summaries and insights using local LLMs
- Create knowledge graphs from research findings

- Perform fact-checking across multiple sources
- Generate research reports and recommendations

1.3 Definitions and Acronyms

- **MCP**: Model Context Protocol - A standardized way for AI assistants to connect with external data sources
- **LLM**: Large Language Model
- **RAG**: Retrieval-Augmented Generation
- **NLP**: Natural Language Processing
- **PDF**: Portable Document Format
- **API**: Application Programming Interface

1.4 Target Audience

- Researchers and academics
- Students conducting literature reviews
- Content creators and journalists
- Business analysts and consultants
- Developers building AI-powered applications

---

2. System Overview

2.1 System Description

The AI Research Assistant MCP Server is a comprehensive research tool that leverages multiple free and open-source technologies to provide intelligent web research capabilities. The system integrates web search engines, document processors, and local AI models to deliver accurate, comprehensive research results.

2.2 Key Features

- **Multi-Source Web Search**: Aggregates results from DuckDuckGo, SearX, and academic databases
- **Intelligent Content Extraction**: Processes web pages, PDFs, and academic papers
- **AI-Powered Analysis**: Uses local LLMs for summarization and insight extraction
- **Knowledge Graph Generation**: Creates visual representations of research relationships
- **Fact-Checking**: Cross-references information across multiple sources
- **Research Report Generation**: Compiles findings into structured reports

2.3 System Benefits

- **Cost-Effective**: Uses only free and open-source tools
- **Privacy-Focused**: Processes data locally without external API calls

- **Comprehensive**: Covers web content, academic papers, and documents
- **Scalable**: Modular architecture supports easy extension
- **Reliable**: Multiple fallback search sources ensure robustness

---

3. Project Structure

```
ai_research_assistant_mcp/
├── README.md
├── LICENSE
├── requirements.txt
├── setup.py
├── pyproject.toml
├── docker-compose.yml
├── .env.example
├── .gitignore
│
├── src/
│   └── research_assistant/
│       ├── __init__.py
│       ├── main.py
│       ├── server.py              # Main MCP server entry point
│       ├── config.py              # Configuration management
│       │
│       ├── core/
│       │   ├── __init__.py
│       │   ├── mcp_server.py          # MCP protocol implementation
│       │   ├── tool_registry.py       # Tool registration and management
│       │   └── error_handler.py       # Global error handling
│       │
│       ├── search/
│       │   ├── __init__.py
│       │   ├── base_searcher.py       # Abstract search interface
│       │   ├── duckduckgo_search.py   # DuckDuckGo implementation
│       │   ├── searx_search.py        # SearX implementation
│       │   ├── selenium_search.py     # Selenium-based search
│       │   ├── academic_search.py     # Academic paper search
│       │   ├── search_aggregator.py   # Multi-source aggregation
│       │   └── search_utils.py        # Search utilities
│       │
│       ├── extraction/
│       │   ├── __init__.py
│       │   ├── base_extractor.py      # Abstract extractor interface
│       │   ├── web_extractor.py       # Web content extraction
│       │   ├── pdf_extractor.py       # PDF processing
│       │   ├── document_extractor.py  # Word/PowerPoint processing
│       │   ├── content_cleaner.py     # Text cleaning and preprocessing
│       │   └── extraction_utils.py     # Extraction utilities
│       │
│       ├── analysis/
│       │   ├── __init__.py
│       │   ├── base_analyzer.py        # Abstract analyzer interface
│       │   ├── summarizer.py           # Content summarization
│       │   ├── insight_extractor.py    # Key insight identification
│       │   ├── fact_checker.py         # Cross-reference verification
│       │   ├── sentiment_analyzer.py   # Sentiment analysis
│       │   ├── topic_modeler.py        # Topic modeling
│       │   └── analysis_utils.py       # Analysis utilities
│       │
│       ├── llm/
│       │   ├── __init__.py
│       │   ├── ollama_client.py        # Ollama integration
│       │   ├── langchain_chains.py     # LangChain workflows
│       │   ├── prompt_templates.py     # Prompt management
│       │   ├── model_manager.py        # Model loading and management
│       │   └── llm_utils.py            # LLM utilities
│       │
│       ├── storage/
│       │   ├── __init__.py
│       │   ├── vector_store.py         # ChromaDB integration
│       │   ├── cache_manager.py        # Result caching
│       │   ├── knowledge_graph.py      # Graph storage and retrieval
│       │   ├── session_manager.py      # Session data management
│       │   └── storage_utils.py        # Storage utilities
│       │
│       ├── tools/
│       │   ├── __init__.py
│       │   ├── research_tools.py       # Core research MCP tools
│       │   ├── analysis_tools.py       # Analysis MCP tools
│       │   ├── export_tools.py         # Export and reporting tools
│       │   ├── utility_tools.py        # Utility MCP tools
│       │   └── tool_schemas.py         # Tool input/output schemas
│       │
│       ├── reports/
│       │   ├── __init__.py
│       │   ├── report_generator.py     # Report compilation
│       │   ├── template_manager.py     # Report templates
│       │   ├── export_formats.py       # Multiple export formats
│       │   └── visualization.py        # Charts and graphs
│       │
│       ├── models/
│       │   ├── __init__.py
│       │   ├── research_models.py      # Research data models
│       │   ├── content_models.py       # Content data models
│       │   ├── analysis_models.py      # Analysis result models
│       │   └── response_models.py     # API response models
│       │
│       └── utils/
│           ├── __init__.py
│           ├── logger.py              # Logging configuration
│           ├── validators.py          # Input validation
│           ├── decorators.py          # Common decorators
│           ├── constants.py           # System constants
│           └── helpers.py             # Helper functions
│
├── config/
│   ├── models.yaml                    # Ollama model configurations
│   ├── search_sources.yaml            # Search engine configurations
│   ├── extraction_rules.yaml          # Content extraction rules
│   ├── analysis_templates.yaml        # Analysis prompt templates
│   └── logging.yaml                   # Logging configuration
│
├── data/
│   ├── models/                        # Downloaded models
│   ├── cache/                         # Cached search results
│   ├── vector_store/                  # ChromaDB data
│   └── knowledge_graphs/              # Generated knowledge graphs
│
├── tests/
│   ├── __init__.py
│   ├── conftest.py                    # Pytest configuration
│   ├── unit/
│   │   ├── test_search/
│   │   ├── test_extraction/
│   │   ├── test_analysis/
│   │   └── test_tools/
│   ├── integration/
│   │   ├── test_search_flow.py
│   │   ├── test_analysis_flow.py
│   │   └── test_mcp_integration.py
│   └── e2e/
│       ├── test_research_workflow.py
│       └── test_complete_pipeline.py
│
├── docs/
│   ├── README.md
│   ├── installation.md
│   ├── configuration.md
│   ├── api_reference.md
│   ├── user_guide.md
│   ├── developer_guide.md
│   ├── troubleshooting.md
│   └── examples/
│
```

```
│       ├── basic_research.md
│       ├── academic_research.md
│       └── business_analysis.md
│
├── scripts/
│   ├── setup.sh                  # Initial setup script
│   ├── download_models.sh             # Model download script
│   ├── start_server.sh              # Server startup script
│   └── run_tests.sh                # Test execution script
│
└── examples/
    ├── basic_usage.py
    ├── academic_research.py
    ├── market_research.py
    ├── fact_checking.py
    └── knowledge_graph_demo.py
```

---

4. Functional Requirements

4.1 Core Research Tools (FR-001 to FR-008)

FR-001: Multi-Source Web Search

- **Description**: Aggregate search results from multiple free search engines
- **Input**: Search query, result count, source preferences
- **Output**: Ranked search results with metadata
- **Priority**: High
- **Dependencies**: DuckDuckGo API, SearX instances

FR-002: Content Extraction and Processing

- **Description**: Extract and clean content from web pages and documents
- **Input**: URLs, file paths, extraction parameters
- **Output**: Cleaned text content with metadata
- **Priority**: High
- **Dependencies**: BeautifulSoup, PyMuPDF, Selenium

FR-003: AI-Powered Content Summarization

- **Description**: Generate intelligent summaries using local LLMs
- **Input**: Text content, summary length, focus areas
- **Output**: Structured summaries with key points
- **Priority**: High
- **Dependencies**: Ollama, LangChain

FR-004: Academic Paper Analysis

- **Description**: Search and analyze academic papers from multiple sources
- **Input**: Research query, publication filters
- **Output**: Paper summaries with citations and relevance scores
- **Priority**: Medium
- **Dependencies**: arXiv API, Semantic Scholar API

FR-005: Knowledge Graph Generation

- **Description**: Create visual knowledge graphs from research findings
- **Input**: Research results, relationship parameters
- **Output**: Interactive knowledge graph data
- **Priority**: Medium
- **Dependencies**: NetworkX, Graph visualization libraries

FR-006: Cross-Source Fact Checking

- **Description**: Verify claims across multiple reliable sources
- **Input**: Claims/statements, verification depth
- **Output**: Fact-check results with source citations
- **Priority**: Medium
- **Dependencies**: Multiple search sources, NLP analysis

FR-007: Research Report Generation

- **Description**: Compile findings into structured research reports
- **Input**: Research session data, report template
- **Output**: Formatted research report (multiple formats)
- **Priority**: Medium
- **Dependencies**: Report templates, Export libraries

FR-008: Research Session Management

- **Description**: Manage research sessions and maintain context
- **Input**: Session parameters, research history
- **Output**: Persistent research sessions
- **Priority**: Low
- **Dependencies**: Session storage, Context management

4.2 Analysis and Intelligence Tools (FR-009 to FR-014)

FR-009: Sentiment Analysis

- **Description**: Analyze sentiment of research content
- **Input**: Text content, analysis scope
- **Output**: Sentiment scores and emotional indicators
- **Priority**: Low
- **Dependencies**: NLP libraries, Sentiment models

FR-010: Topic Modeling

- **Description**: Identify and categorize topics in research content
- **Input**: Document collection, topic count
- **Output**: Topic clusters with representative keywords
- **Priority**: Low
- **Dependencies**: Topic modeling algorithms

FR-011: Research Question Generation

- **Description**: Generate relevant research questions from content
- **Input**: Research domain, content context
- **Output**: Prioritized research questions
- **Priority**: Medium
- **Dependencies**: LLM analysis, Question templates

FR-012: Citation and Reference Management

- **Description**: Extract and format citations from research sources
- **Input**: Academic content, citation style
- **Output**: Properly formatted citations
- **Priority**: Medium
- **Dependencies**: Citation parsing libraries

FR-013: Trend Analysis

- **Description**: Identify trends and patterns in research data
- **Input**: Time-series research data, analysis parameters
- **Output**: Trend reports with visualizations
- **Priority**: Low
- **Dependencies**: Statistical analysis libraries

FR-014: Research Recommendation Engine

- **Description**: Suggest related research areas and sources
- **Input**: Current research context, user preferences
- **Output**: Recommended research directions
- **Priority**: Low
- **Dependencies**: Machine learning models, Content similarity

---

5. Non-Functional Requirements

5.1 Performance Requirements (NFR-001 to NFR-005)

NFR-001: Response Time

- Search operations: < 10 seconds for standard queries
- Content extraction: < 30 seconds per document
- AI analysis: < 60 seconds for standard summaries

- Knowledge graph generation: < 120 seconds

NFR-002: Throughput

- Support minimum 10 concurrent research sessions
- Process up to 100 search queries per hour
- Handle 50 document extractions per hour

NFR-003: Scalability

- Horizontal scaling support through containerization
- Modular architecture for component scaling
- Resource usage optimization for local deployment

NFR-004: Memory Usage

- Maximum 4GB RAM usage under normal operation
- Efficient memory management for large documents
- Garbage collection optimization

NFR-005: Storage Efficiency

- Intelligent caching to reduce redundant operations
- Compressed storage for vector embeddings
- Configurable cache size limits

5.2 Reliability Requirements (NFR-006 to NFR-010)

NFR-006: Availability

- 99% uptime for local deployments
- Graceful degradation when search sources unavailable
- Automatic failover between search sources

NFR-007: Error Handling

- Comprehensive error logging and monitoring
- User-friendly error messages
- Automatic retry mechanisms for failed operations

NFR-008: Data Integrity

- Checksum verification for downloaded content
- Backup and recovery mechanisms
- Data validation at all input points

NFR-009: Fault Tolerance

- Resilient to individual component failures
- Circuit breaker patterns for external services

- Graceful handling of network interruptions

NFR-010: Recovery

- Automatic recovery from system crashes
- Session state persistence across restarts
- Database consistency checks on startup

5.3 Security Requirements (NFR-011 to NFR-015)

NFR-011: Data Privacy

- Local processing without external data transmission
- Secure handling of research content
- User data anonymization options

NFR-012: Input Validation

- Comprehensive input sanitization
- Protection against injection attacks
- URL validation and safety checks

NFR-013: Access Control

- Configurable user authentication
- Role-based access to features
- API key management for external services

NFR-014: Secure Communication

- HTTPS enforcement for web requests
- Certificate validation for external connections
- Secure local API endpoints

NFR-015: Audit Logging

- Comprehensive activity logging
- User action tracking
- Security event monitoring

5.4 Usability Requirements (NFR-016 to NFR-020)

NFR-016: User Interface

- Intuitive MCP tool interfaces
- Clear error messages and guidance
- Comprehensive documentation

NFR-017: Configuration

- Easy setup and configuration process
- Environment-specific configuration options
- Runtime configuration updates

NFR-018: Monitoring

- Real-time system status monitoring
- Performance metrics dashboard
- Resource usage tracking

NFR-019: Extensibility

- Plugin architecture for new search sources
- Custom analysis module support
- Configurable output formats

NFR-020: Compatibility

- Cross-platform support (Windows, macOS, Linux)
- Multiple Python version compatibility
- Container deployment support

---

6. Technology Stack

6.1 Core Technologies

- **Programming Language**: Python 3.9+
- **MCP Framework**: mcp>=1.0.0
- **AI/ML Framework**: LangChain, Ollama
- **Web Framework**: FastAPI (for API endpoints)
- **Database**: ChromaDB (vector storage), SQLite (metadata)

6.2 Search and Extraction

- **Search Engines**: DuckDuckGo, SearX, Academic APIs
- **Web Scraping**: Selenium, BeautifulSoup, Playwright
- **Document Processing**: PyMuPDF, python-docx, pdfplumber
- **Content Extraction**: trafilatura, newspaper3k

6.3 AI and Analysis

- **Local LLM**: Ollama (Llama2, CodeLlama)
- **Embeddings**: sentence-transformers, nomic-embed-text
- **Vector Storage**: ChromaDB, FAISS
- **NLP**: spaCy, NLTK, TextStat

6.4 Data Processing and Visualization

- **Data Processing**: pandas, numpy
- **Visualization**: matplotlib, plotly, networkx
- **Report Generation**: reportlab, jinja2
- **Graph Processing**: networkx, graphviz

6.5 Development and Deployment

- **Testing**: pytest, pytest-cov, pytest-mock
- **Containerization**: Docker, docker-compose
- **Logging**: structlog, rich
- **Configuration**: pydantic, python-dotenv

6.6 Free and Open Source Compliance

All technologies selected are:

- Open source with permissive licenses
- Free for commercial and non-commercial use
- Actively maintained with strong communities
- No API costs or usage limitations

---

7. System Architecture

7.1 High-Level Architecture

```
┌──────────────────────────────────────────────────────────
────┐
│              MCP Client Layer                    │
│           (Claude, VSCode, etc.)                  │
└─────────────────────────────┬────────────────────────────
────┘
              │ MCP Protocol
┌─────────────────────────────▼────────────────────────────
────┐
│            MCP Server Core                        │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐      │
│  │Tool Registry│ │Error Handler│ │Config Mgr  │     │
│  └──────────────┘ └──────────────┘ └──────────────┘      │
└─────────────────────────────┬────────────────────────────
────┘
              │
┌─────────────────────────────▼────────────────────────────
────┐
│            Research Tools Layer                    │
│  ┌────────────────┐ ┌────────────────┐ ┌────────────────┐      │
│  │Search Tools │ │Analysis Tools│ │Export Tools │     │
│  └────────────────┘ └────────────────┘ └────────────────┘      │
└───────┬────────────────────┬────────────────────┬─────────────────────
──────────┘
    │          │          │
┌───────▼───────┐ ┌──────────▼──────────┐ ┌────────▼────────┐
│Search    │ │Analysis     │ │Storage   │
│Engine    │ │Engine      │ │Engine    │
│Layer    │ │Layer       │ │Layer    │
└──────────────┘ └──────────────────┘ └──────────────┘
```

7.2 Component Architecture

7.2.1 MCP Server Core

- **Responsibility**: Handle MCP protocol communication
- **Components**: Tool registry, error handling, configuration
- **Interfaces**: MCP client communication, tool execution

7.2.2 Search Engine Layer

- **Responsibility**: Aggregate search results from multiple sources
- **Components**: DuckDuckGo, SearX, Selenium, Academic APIs
- **Interfaces**: Unified search API, result normalization

7.2.3 Analysis Engine Layer

- **Responsibility**: Process and analyze research content
- **Components**: LLM integration, NLP processing, insight extraction
- **Interfaces**: Content analysis API, result formatting

7.2.4 Storage Engine Layer

- **Responsibility**: Manage data persistence and caching
- **Components**: Vector store, cache management, session storage
- **Interfaces**: Data storage API, retrieval operations

7.3 Data Flow Architecture

```
Search Query → Search Aggregator → Multiple Search Sources
    ↓
Content URLs → Content Extractor → Clean Text Content
    ↓
Text Content → AI Analyzer → Insights & Summaries
    ↓
Analysis Results → Knowledge Graph → Visual Representation
    ↓
```

Final Results → Report Generator → Formatted Output

7.4 Integration Architecture

7.4.1 External Integrations

- **Free Search APIs**: DuckDuckGo, arXiv, Semantic Scholar
- **Local LLM**: Ollama with multiple model support
- **Web Scraping**: Multiple fallback mechanisms

7.4.2 Internal Integrations

- **Component Communication**: Event-driven architecture
- **Data Persistence**: Unified storage interface
- **Error Handling**: Centralized error management

---

8. API Specifications

8.1 MCP Tool Definitions

8.1.1 web_research_query

```
{
  "name": "web_research_query",
  "description": "Perform comprehensive web research on a given topic",
  "inputSchema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Research query or topic"
      },
      "max_results": {
        "type": "integer",
        "default": 20,
        "description": "Maximum number of results to return"
      },
      "sources": {
        "type": "array",
        "items": {"type": "string"},
        "default": ["web", "academic"],
        "description": "Search sources to include"
      },
      "depth": {
        "type": "string",
        "enum": ["shallow", "medium", "deep"],
        "default": "medium",
        "description": "Research depth level"
      }
    },
    "required": ["query"]
  }
}
```

8.1.2 summarize_content

```
{
  "name": "summarize_content",
  "description": "Summarize research content using AI analysis",
  "inputSchema": {
    "type": "object",
    "properties": {
      "content": {
        "type": "string",
        "description": "Content to summarize"
      },
      "summary_type": {
        "type": "string",
        "enum": ["brief", "detailed", "executive"],
        "default": "brief",
        "description": "Type of summary to generate"
      },
      "focus_areas": {
        "type": "array",
        "items": {"type": "string"},
        "description": "Specific areas to focus on in summary"
      },
      "max_length": {
        "type": "integer",
        "default": 500,
        "description": "Maximum summary length in words"
      }
    },
    "required": ["content"]
  }
}
```

8.1.3 extract_insights

```
{
  "name": "extract_insights",
  "description": "Extract key insights from research content",
  "inputSchema": {
    "type": "object",
    "properties": {
      "content": {
        "type": "string",
        "description": "Research content to analyze"
      },
      "insight_types": {
        "type": "array",
        "items": {
          "type": "string",
          "enum": ["trends", "patterns", "contradictions", "gaps", "opportunities"]
        },
        "default": ["trends", "patterns"],
        "description": "Types of insights to extract"
      },
      "confidence_threshold": {
        "type": "number",
        "minimum": 0.0,
        "maximum": 1.0,
        "default": 0.7,
        "description": "Minimum confidence level for insights"
      }
    },
    "required": ["content"]
  }
}
```

8.1.4 fact_check_claims

```
{
  "name": "fact_check_claims",
  "description": "Verify claims against multiple reliable sources",
  "inputSchema": {
    "type": "object",
    "properties": {
      "claims": {
        "type": "array",
        "items": {"type": "string"},
        "description": "Claims to fact-check"
      },
      "sources_required": {
        "type": "integer",
        "default": 3,
        "description": "Minimum number of sources for verification"
      },
      "include_sources": {
        "type": "boolean",
        "default": true,
        "description": "Include source citations in results"
      }
    },
    "required": ["claims"]
  }
}
```

8.1.5 generate_knowledge_graph

```
{
  "name": "generate_knowledge_graph",
  "description": "Create knowledge graph from research findings",
  "inputSchema": {
    "type": "object",
    "properties": {
      "research_data": {
        "type": "string",
        "description": "Research content for graph generation"
      },
      "entity_types": {
        "type": "array",
        "items": {"type": "string"},
        "default": ["person", "organization", "concept", "location"],
        "description": "Types of entities to extract"
      },
      "relationship_threshold": {
        "type": "number",
        "default": 0.5,
        "description": "Threshold for relationship strength"
      },
      "max_nodes": {
        "type": "integer",
        "default": 100,
        "description": "Maximum number of nodes in graph"
      }
    },
    "required": ["research_data"]
  }
}
```

8.1.6 generate_research_report

```
{
  "name": "generate_research_report",
  "description": "Compile research findings into structured report",
  "inputSchema": {
    "type": "object",
    "properties": {
      "research_session_id": {
        "type": "string",
        "description": "ID of research session to compile"
      },
      "report_type": {
        "type": "string",
        "enum": ["summary", "detailed", "academic", "business"],
        "default": "summary",
        "description": "Type of report to generate"
      },
      "format": {
        "type": "string",
        "enum": ["markdown", "html", "pdf", "json"],
        "default": "markdown",
        "description": "Output format for report"
      },
      "include_citations": {
        "type": "boolean",
        "default": true,
        "description": "Include source citations"
      }
    },
    "required": ["research_session_id"]
  }
}
```

8.2 Response Formats

8.2.1 Standard Success Response

```
{
  "success": true,
  "data": {
    "results": [],
    "metadata": {
      "query": "original query",
      "timestamp": "2024-01-01T00:00:00Z",
      "processing_time": 5.23,
      "sources_used": ["duckduckgo", "arxiv"],
      "total_results": 15
    }
  }
}
```

8.2.2 Error Response

```
{
  "success": false,
  "error": {
    "code": "SEARCH_FAILED",
    "message": "Unable to complete search request",
    "details": "Connection timeout to search service",
    "suggestions": [
      "Check internet connection",
      "Try a different search query",
      "Contact support if problem persists"
    ]
  }
}
```

---

9. Data Models

9.1 Core Data Models

9.1.1 SearchResult

```
class SearchResult:
    url: str
    title: str
    snippet: str
    source: str
    relevance_score: float
    timestamp: datetime
    metadata: Dict[str, Any]
```

9.1.2 ResearchContent

```
class ResearchContent:
    content_id: str
    text: str
    content_type: str  # web, pdf, academic
    source_url: str
    extraction_metadata: Dict[str, Any]
    processing_timestamp: datetime
```

9.1.3 AnalysisResult

```
class AnalysisResult:
    analysis_id: str
    content_id: str
    analysis_type: str
    results: Dict[str, Any]
    confidence_score: float
    model_used: str
    timestamp: datetime
```

9.1.4 KnowledgeGraph

```
class KnowledgeGraph:
    graph_id: str
    nodes: List[GraphNode]
    edges: List[GraphEdge]
    metadata: Dict[str, Any]
    creation_timestamp: datetime
```

9.1.5 ResearchSession

```
class ResearchSession:
    session_id: str
    user_id: str
    query: str
    search_results: List[SearchResult]
    analysis_results: List[AnalysisResult]
    knowledge_graphs: List[KnowledgeGraph]
    status: str
    created_at: datetime
    updated_at: datetime
```

9.2 Configuration Models

9.2.1 SearchConfig

```
class SearchConfig:
    enabled_sources: List[str]
    max_results_per_source: int
    timeout_seconds: int
    retry_attempts: int
    cache_duration: int
```

9.2.2 AnalysisConfig

```
class AnalysisConfig:
    default_model: str
    max_content_length: int
    confidence_threshold: float
    analysis_timeout: int
```

AI-Powered Research Assistant MCP Server

Software Requirements Specification (SRS) Document - Sections 10-14

---

10. User Stories

10.1 Researcher User Stories

US-001: Basic Web Research

As a researcher
I want to search multiple sources simultaneously
So that I can get comprehensive results without manually checking each source

Acceptance Criteria:

- Can search DuckDuckGo, SearX, and academic sources in one query
- Results are aggregated and ranked by relevance
- Search completes within 10 seconds for standard queries
- Duplicate results are automatically filtered

US-002: Academic Paper Analysis

As an academic researcher
I want to analyze academic papers from arXiv and Semantic Scholar
So that I can quickly understand key findings and methodologies

Acceptance Criteria:

- Can search academic databases with subject filters
- Papers are automatically summarized with key points
- Citations are extracted and formatted correctly
- Related papers are suggested based on content similarity

US-003: Content Summarization

As a content analyst
I want to get AI-generated summaries of web articles and documents
So that I can quickly understand large volumes of content

Acceptance Criteria:

- Can summarize web pages, PDFs, and documents
- Multiple summary types available (brief, detailed, executive)
- Key insights and trends are highlighted
- Summary accuracy is validated against source content

US-004: Fact Verification

As a journalist
I want to verify claims across multiple reliable sources
So that I can ensure information accuracy in my reporting

Acceptance Criteria:

- Can fact-check specific claims or statements
- Results show verification status with confidence scores
- Source citations are provided for all verifications
- Contradictory information is clearly highlighted

US-005: Knowledge Graph Creation

As a data analyst
I want to visualize relationships between research entities
So that I can understand complex topic interconnections

Acceptance Criteria:

- Can generate interactive knowledge graphs from research content
- Entities and relationships are automatically extracted
- Graph can be exported in multiple formats
- Interactive exploration features are available

10.2 Student User Stories

US-006: Literature Review Assistance

As a graduate student
I want to compile research from multiple sources for literature reviews
So that I can efficiently gather relevant academic sources

Acceptance Criteria:

- Can search across academic databases and web sources
- Results are categorized by research themes
- Citation formats are automatically generated
- Research gaps and opportunities are identified

US-007: Research Question Generation

As an undergraduate student
I want to generate research questions from topic exploration
So that I can develop focused research proposals

Acceptance Criteria:

- Can analyze topic content to suggest research questions
- Questions are ranked by feasibility and originality
- Background context is provided for each question
- Related research areas are suggested

10.3 Business Analyst User Stories

US-008: Market Research Analysis

As a business analyst
I want to research market trends and competitor information
So that I can provide data-driven business recommendations

Acceptance Criteria:

- Can search for industry reports and market data
- Trend analysis is performed on collected data
- Competitor information is automatically categorized
- Business insights are extracted and prioritized

US-009: Research Report Generation

As a consultant
I want to generate professional research reports from my findings
So that I can deliver comprehensive analysis to clients

Acceptance Criteria:

- Can compile research sessions into formatted reports
- Multiple report templates are available
- Charts and visualizations are automatically generated
- Reports can be exported in various formats (PDF, HTML, Word)

10.4 Developer User Stories

US-010: MCP Integration

As a developer
I want to integrate research capabilities into my AI assistant
So that my application can provide intelligent research features

Acceptance Criteria:

- MCP server can be easily configured and deployed

- All tools are accessible through MCP protocol
- Comprehensive API documentation is available
- Error handling provides clear feedback

---

11. Testing Strategy

11.1 Testing Approach

11.1.1 Test Philosophy

- **Test-Driven Development**: Core functionality developed with tests first
- **Continuous Testing**: Automated testing integrated into development workflow
- **Quality Gates**: Minimum test coverage and quality metrics enforced
- **Real-world Scenarios**: Tests based on actual user workflows

11.1.2 Testing Pyramid

- ┌─────────────────────┐
- │   E2E Tests   │ ← 10% (Full workflow validation)
- │   (Slow)   │
- ├─────────────────────┤
- │ Integration   │ ← 20% (Component interaction)
- │   Tests   │
- ├─────────────────────┤
- │   Unit Tests   │ ← 70% (Individual component logic)
- │   (Fast)   │
- └─────────────────────┘

11.2 Unit Testing Strategy

11.2.1 Search Components

- # Test Coverage Areas
- - DuckDuckGo search functionality
- - SearX integration and fallback
- - Academic paper search (arXiv, Semantic Scholar)
- - Search result aggregation and ranking
- - Content extraction from various sources
- - Error handling and retry mechanisms

11.2.2 Analysis Components

- # Test Coverage Areas
- - LLM integration (Ollama, LangChain)
- - Content summarization accuracy
- - Insight extraction validation
- - Fact-checking logic
- - Knowledge graph generation
- - Natural language processing pipelines

11.2.3 Storage Components

- # Test Coverage Areas
- - Vector store operations (ChromaDB)
- - Cache management and invalidation
- - Session persistence and recovery
- - Data serialization/deserialization
- - Query optimization

11.3 Integration Testing Strategy

11.3.1 Search-to-Analysis Flow

```
● def test_search_to_analysis_pipeline():
●     """Test complete search and analysis workflow"""
●     # Search for content
●     search_results = search_aggregator.search("AI research trends")
●     assert len(search_results) > 0
●
●     # Extract content
●     content = content_extractor.extract_from_urls(search_results.urls)
●     assert content.text is not None
●
●     # Analyze content
●     analysis = analyzer.summarize(content.text)
●     assert analysis.summary is not None
●     assert analysis.confidence_score > 0.7
```

11.3.2 MCP Protocol Integration

```
● def test_mcp_tool_execution():
●     """Test MCP tool registration and execution"""
●     # Register tools
●     mcp_server.register_tools(research_tools)
●
```

```
    # Execute web research tool
    result = mcp_server.execute_tool(
        "web_research_query",
        {"query": "machine learning", "max_results": 10}
    )

    assert result.success is True
    assert len(result.data.results) <= 10
```

11.4 End-to-End Testing Strategy

11.4.1 Complete Research Workflow

```
def test_complete_research_workflow():
    """Test full research session from query to report"""
    session_id = research_assistant.start_session()

    # Perform research
    research_results = research_assistant.research_query(
        session_id,
        "sustainable energy technologies"
    )

    # Generate insights
    insights = research_assistant.extract_insights(
        session_id,
        research_results
    )

    # Create knowledge graph
    graph = research_assistant.generate_knowledge_graph(
        session_id,
        research_results
    )

    # Generate report
    report = research_assistant.generate_report(
        session_id,
        format="markdown"
    )

    # Validate complete workflow
    assert research_results.total_sources >= 3
    assert len(insights.key_findings) > 0
    assert graph.node_count > 10
    assert len(report.content) > 1000
```

11.5 Performance Testing

11.5.1 Load Testing Scenarios

```
●  scenarios:
●    - name: "Concurrent Search Requests"
●      users: 10
●      duration: "5m"
●      requests_per_second: 2
●
●    - name: "Document Processing Load"
●      users: 5
●      duration: "10m"
●      files_per_minute: 20
●
●    - name: "AI Analysis Throughput"
●      users: 3
●      duration: "15m"
●      analysis_requests_per_minute: 10
```

11.5.2 Performance Benchmarks

```
●  performance_targets = {
●      "search_response_time": "< 10 seconds",
●      "content_extraction": "< 30 seconds",
●      "ai_analysis": "< 60 seconds",
●      "memory_usage": "< 4GB",
●      "concurrent_sessions": "> 10"
●  }
```

11.6 Testing Tools and Frameworks

11.6.1 Unit Testing Stack

- **pytest**: Primary testing framework
- **pytest-cov**: Code coverage reporting
- **pytest-mock**: Mocking and stubbing
- **pytest-asyncio**: Async testing support
- **factory_boy**: Test data generation

11.6.2 Integration Testing Tools

- **testcontainers**: Containerized test dependencies

- **responses**: HTTP request mocking
- **vcr.py**: HTTP interaction recording/playback
- **pytest-xdist**: Parallel test execution

11.6.3 Performance Testing Tools

- **locust**: Load testing framework
- **memory_profiler**: Memory usage monitoring
- **py-spy**: Python profiling tool
- **pytest-benchmark**: Performance benchmarking

11.7 Test Data Management

11.7.1 Test Data Strategy

```
# Sample test data structure
test_data = {
    "search_queries": [
        "artificial intelligence trends",
        "climate change research",
        "quantum computing applications"
    ],
    "sample_documents": [
        "research_paper_1.pdf",
        "web_article_1.html",
        "academic_abstract_1.txt"
    ],
    "expected_results": {
        "min_sources": 3,
        "confidence_threshold": 0.7,
        "processing_time_limit": 60
    }
}
```

11.7.2 Mock Data Generation

```
@pytest.fixture
def mock_search_results():
    """Generate realistic search result data"""
    return SearchResultFactory.create_batch(
        size=10,
        relevance_score__range=(0.5, 1.0),
        source__cycle=["duckduckgo", "searx", "arxiv"]
    )
```

12. Deployment Strategy

12.1 Deployment Architecture

12.1.1 Local Development Deployment

```
# docker-compose.dev.yml
version: '3.8'
services:
  research-assistant:
    build:
      context: .
      dockerfile: Dockerfile.dev
    ports:
      - "8000:8000"
    volumes:
      - ./src:/app/src
      - ./data:/app/data
    environment:
      - ENVIRONMENT=development
      - LOG_LEVEL=debug
    depends_on:
      - ollama
      - chromadb

  ollama:
    image: ollama/ollama:latest
    ports:
      - "11434:11434"
    volumes:
      - ollama_data:/root/.ollama
    environment:
      - OLLAMA_HOST=0.0.0.0

  chromadb:
    image: chromadb/chroma:latest
    ports:
      - "8001:8000"
    volumes:
      - chroma_data:/chroma/chroma
```

12.1.2 Production Deployment

```
# docker-compose.prod.yml
```

```
●  version: '3.8'
●  services:
●    research-assistant:
●      image: research-assistant:latest
●      restart: unless-stopped
●      ports:
●        - "8000:8000"
●      volumes:
●        - ./data:/app/data
●        - ./logs:/app/logs
●      environment:
●        - ENVIRONMENT=production
●        - LOG_LEVEL=info
●      healthcheck:
●        test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
●        interval: 30s
●        timeout: 10s
●        retries: 3
●
●    nginx:
●      image: nginx:alpine
●      restart: unless-stopped
●      ports:
●        - "80:80"
●        - "443:443"
●      volumes:
●        - ./nginx.conf:/etc/nginx/nginx.conf
●        - ./certs:/etc/nginx/certs
```

12.2 Container Configuration

12.2.1 Dockerfile

```
●  FROM python:3.11-slim
●
●  # System dependencies
●  RUN apt-get update && apt-get install -y \
●      curl \
●      git \
●      build-essential \
●      && rm -rf /var/lib/apt/lists/*
●
●  # Python dependencies
●  WORKDIR /app
●  COPY requirements.txt .
●  RUN pip install --no-cache-dir -r requirements.txt
```

```
# Application code
COPY src/ ./src/
COPY config/ ./config/
COPY scripts/ ./scripts/

# Create data directories
RUN mkdir -p data/models data/cache data/vector_store data/knowledge_graphs

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Run application
EXPOSE 8000
CMD ["python", "-m", "src.research_assistant.server"]
```

12.2.2 Multi-stage Build for Production

```
# Build stage
FROM python:3.11-slim as builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

# Production stage
FROM python:3.11-slim
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
COPY --from=builder /root/.local /root/.local
WORKDIR /app
COPY src/ ./src/
COPY config/ ./config/
ENV PATH=/root/.local/bin:$PATH
EXPOSE 8000
CMD ["python", "-m", "src.research_assistant.server"]
```

12.3 Environment Configuration

12.3.1 Development Environment

```
# .env.development
ENVIRONMENT=development
LOG_LEVEL=debug
DEBUG=true

```

- # Database URLs
- CHROMA_HOST=localhost
- CHROMA_PORT=8001
- OLLAMA_HOST=localhost
- OLLAMA_PORT=11434
-
- # Search Configuration
- DUCKDUCKGO_ENABLED=true
- SEARX_INSTANCES=searx.be,searx.org
- SELENIUM_HEADLESS=true
-
- # Model Configuration
- DEFAULT_LLM_MODEL=llama2:7b
- EMBEDDING_MODEL=nomic-embed-text
-
- # Cache Configuration
- CACHE_TTL=3600
- MAX_CACHE_SIZE=1000
-
- # Rate Limiting
- SEARCH_RATE_LIMIT=60
- ANALYSIS_RATE_LIMIT=30

12.3.2 Production Environment

- # .env.production
- ENVIRONMENT=production
- LOG_LEVEL=info
- DEBUG=false
-
- # Security
- SECRET_KEY=your-secret-key-here
- ALLOWED_HOSTS=your-domain.com,localhost
-
- # External Services
- CHROMA_HOST=chromadb
- CHROMA_PORT=8000
- OLLAMA_HOST=ollama
- OLLAMA_PORT=11434
-
- # Performance Tuning
- WORKER_PROCESSES=4
- MAX_CONCURRENT_REQUESTS=100
- TIMEOUT_SECONDS=300
-
- # Monitoring

- ENABLE_METRICS=true
- METRICS_PORT=9090

12.4 Deployment Scripts

12.4.1 Setup Script

- #!/bin/bash
- # scripts/setup.sh
-
- set -e
-
- echo "Setting up AI Research Assistant MCP Server..."
-
- # Check system requirements
- python3 --version
- docker --version
- docker-compose --version
-
- # Create directories
- mkdir -p data/{models,cache,vector_store,knowledge_graphs}
- mkdir -p logs
-
- # Copy configuration files
- cp .env.example .env
- echo "Please edit .env file with your configuration"
-
- # Download required models
- ./scripts/download_models.sh
-
- # Build and start services
- docker-compose up -d
-
- echo "Setup complete! Access the server at http://localhost:8000"

12.4.2 Model Download Script

- #!/bin/bash
- # scripts/download_models.sh
-
- set -e
-
- echo "Downloading required AI models..."
-
- # Check if Ollama is running

```
●  if ! curl -s http://localhost:11434/api/tags > /dev/null; then
●      echo "Starting Ollama service..."
●      docker-compose up -d ollama
●      sleep 10
●  fi
●
●  # Download LLM models
●  echo "Downloading Llama2 7B model..."
●  docker exec -it $(docker-compose ps -q ollama) ollama pull llama2:7b
●
●  echo "Downloading Code Llama model..."
●  docker exec -it $(docker-compose ps -q ollama) ollama pull codellama:7b
●
●  echo "Downloading embedding model..."
●  docker exec -it $(docker-compose ps -q ollama) ollama pull nomic-embed-text
●
●  echo "Model download complete!"
```

12.5 Monitoring and Logging

12.5.1 Logging Configuration

```
●  # config/logging.yaml
●  version: 1
●  disable_existing_loggers: false
●
●  formatters:
●    standard:
●      format: "%(asctime)s [%(levelname)s] %(name)s: %(message)s"
●    json:
●      format: '{"timestamp": "%(asctime)s", "level": "%(levelname)s", "logger":
   "%(name)s", "message": "%(message)s"}'
●
●  handlers:
●    console:
●      class: logging.StreamHandler
●      level: INFO
●      formatter: standard
●      stream: ext://sys.stdout
●
●    file:
●      class: logging.handlers.RotatingFileHandler
●      level: DEBUG
●      formatter: json
●      filename: logs/research_assistant.log
●      maxBytes: 10485760  # 10MB
●       backupCount: 5
●
●   loggers:
●     research_assistant:
●       level: DEBUG
●       handlers: [console, file]
●       propagate: false
●
●     uvicorn:
●       level: INFO
●       handlers: [console, file]
●       propagate: false
●
●   root:
●     level: INFO
●     handlers: [console, file]
```

12.5.2 Health Check Endpoints

```
●   # Health check implementation
●   @app.get("/health")
●   async def health_check():
●       """System health check endpoint"""
●       try:
●           # Check database connectivity
●           chroma_status = await check_chromadb_health()
●
●           # Check LLM availability
●           ollama_status = await check_ollama_health()
●
●           # Check system resources
●           memory_usage = psutil.virtual_memory().percent
●           disk_usage = psutil.disk_usage('/').percent
●
●           return {
●               "status": "healthy",
●               "timestamp": datetime.utcnow().isoformat(),
●               "services": {
●                   "chromadb": chroma_status,
●                   "ollama": ollama_status
●               },
●               "resources": {
●                   "memory_usage": memory_usage,
●                   "disk_usage": disk_usage
●               }
●           }
```

```
    except Exception as e:
        return {
            "status": "unhealthy",
            "error": str(e),
            "timestamp": datetime.utcnow().isoformat()
        }
```

12.6 Scaling Strategy

12.6.1 Horizontal Scaling

```
# docker-compose.scale.yml
version: '3.8'
services:
  research-assistant:
    image: research-assistant:latest
    deploy:
      replicas: 3
      resources:
        limits:
          memory: 2G
          cpus: '1.0'
    depends_on:
      - ollama
      - chromadb

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx-lb.conf:/etc/nginx/nginx.conf
    depends_on:
      - research-assistant
```

12.6.2 Load Balancer Configuration

```
# nginx-lb.conf
upstream research_assistant {
    server research-assistant:8000 max_fails=3 fail_timeout=30s;
    server research-assistant:8001 max_fails=3 fail_timeout=30s;
    server research-assistant:8002 max_fails=3 fail_timeout=30s;
}

server {
```

```
●       listen 80;
●
●       location / {
●           proxy_pass http://research_assistant;
●           proxy_set_header Host $host;
●           proxy_set_header X-Real-IP $remote_addr;
●           proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
●           proxy_connect_timeout 30s;
●           proxy_send_timeout 30s;
●           proxy_read_timeout 30s;
●       }
●
●       location /health {
●           access_log off;
●           proxy_pass http://research_assistant/health;
●       }
●   }
```

---

13. Risk Assessment

13.1 Technical Risks

13.1.1 High Priority Risks

RISK-001: Search Source Availability

- **Description**: External search services (DuckDuckGo, SearX) may become unavailable
- **Probability**: Medium
- **Impact**: High
- **Mitigation**:
  - Implement multiple fallback search sources
  - Cache recent search results for offline access
  - Circuit breaker pattern for failed services
  - Health monitoring with automatic failover

RISK-002: LLM Model Performance

- **Description**: Local LLM models may provide inconsistent or low-quality results
- **Probability**: Medium
- **Impact**: High
- **Mitigation**:
  - Support multiple LLM models with automatic fallback
  - Implement result quality scoring and validation

    - Regular model updates and performance benchmarking
    - User feedback integration for continuous improvement

RISK-003: Memory and Resource Constraints

- **Description**: Large documents and models may exceed available system resources
- **Probability**: High
- **Impact**: Medium
- **Mitigation**:
    - Implement chunking for large documents
    - Memory usage monitoring and optimization
    - Configurable resource limits and warnings
    - Efficient garbage collection and cleanup

RISK-004: Web Scraping Failures

- **Description**: Websites may block or limit scraping activities
- **Probability**: Medium
- **Impact**: Medium
- **Mitigation**:
    - Implement respectful scraping with delays
    - Use multiple scraping strategies (BeautifulSoup, Selenium)
    - Rotate user agents and implement proxy support
    - Graceful degradation when content unavailable

13.2 Operational Risks

13.2.1 Medium Priority Risks

RISK-005: Data Privacy and Compliance

- **Description**: Processing of sensitive research content may violate privacy regulations
- **Probability**: Low
- **Impact**: High
- **Mitigation**:
    - Local processing without external data transmission
    - Configurable data retention policies
    - User consent mechanisms for data processing
    - Audit logging for compliance tracking

RISK-006: System Configuration Complexity

- **Description**: Complex setup process may deter user adoption
- **Probability**: Medium
- **Impact**: Medium
- **Mitigation**:
    - Automated setup scripts and Docker deployment
    - Comprehensive documentation with examples

    - Default configurations for common use cases
    - Setup validation and troubleshooting guides

RISK-007: Dependency Management

- **Description**: Updates to dependencies may break functionality
- **Probability**: Medium
- **Impact**: Medium
- **Mitigation**:
    - Pin specific versions of critical dependencies
    - Automated testing for dependency updates
    - Staged rollout process for major updates
    - Rollback procedures for failed updates

13.3 Business Risks

13.3.1 Low Priority Risks

RISK-008: Open Source License Compliance

- **Description**: Improper use of open source components may create legal issues
- **Probability**: Low
- **Impact**: Medium
- **Mitigation**:
    - Regular license audits of all dependencies
    - Clear documentation of license requirements
    - Legal review of license compatibility
    - Contributor license agreements where needed

RISK-009: Community Support Dependency

- **Description**: Reliance on community-maintained tools may affect long-term support
- **Probability**: Medium
- **Impact**: Low
- **Mitigation**:
    - Monitor health of upstream projects
    - Maintain forks of critical dependencies
    - Contribute back to community projects
    - Build internal expertise on key components

RISK-010: Performance Degradation

- **Description**: System performance may degrade with increased usage
- **Probability**: Medium
- **Impact**: Medium
- **Mitigation**:
    - Performance monitoring and alerting
    - Scalable architecture design
    - Regular performance testing and optimization

    - Capacity planning and resource scaling

13.4 Risk Monitoring and Response

13.4.1 Risk Monitoring Framework

```
# Risk monitoring implementation
class RiskMonitor:
    def __init__(self):
        self.risk_metrics = {
            "search_success_rate": 0.95,  # 95% minimum success rate
            "llm_response_time": 60,      # 60 seconds maximum
            "memory_usage_threshold": 0.8, # 80% memory usage warning
            "error_rate_threshold": 0.05   # 5% error rate limit
        }

    async def monitor_risks(self):
        """Continuous risk monitoring"""
        while True:
            try:
                # Monitor search service health
                search_health = await self.check_search_services()

                # Monitor LLM performance
                llm_metrics = await self.check_llm_performance()

                # Monitor system resources
                system_metrics = await self.check_system_resources()

                # Trigger alerts if thresholds exceeded
                await self.process_risk_alerts(
                    search_health, llm_metrics, system_metrics
                )

            except Exception as e:
                logger.error(f"Risk monitoring error: {e}")

            await asyncio.sleep(60)  # Check every minute
```

13.4.2 Incident Response Plan

```
incident_response:
  severity_levels:
    critical:
      - Search services completely unavailable
      - LLM models not responding
```

- - System memory exhausted
- response_time: "< 15 minutes"
- escalation: "Immediate"
-
- high:
- - Degraded search performance
- - Partial LLM failures
- - High error rates
- response_time: "< 1 hour"
- escalation: "Within 2 hours"
-
- medium:
- - Individual source failures
- - Slow response times
- - Configuration issues
- response_time: "< 4 hours"
- escalation: "Next business day"
-
- response_procedures:
- - Assess impact and severity
- - Implement immediate mitigation
- - Notify stakeholders
- - Root cause analysis
- - Implement permanent fix
- - Post-incident review

---

14. Future Enhancements

14.1 Short-term Enhancements (3-6 months)

14.1.1 Advanced Search Capabilities

Enhancement: Multi-modal Search Support

- **Description**: Extend search capabilities to include images, videos, and audio content
- **Benefits**: More comprehensive research coverage
- **Implementation**:
  # Multi-modal search interfaceclass MultiModalSearcher: async def search_images(self, query: str) -> List[ImageResult]:   """Search for relevant images"""   pass   async def search_videos(self, query: str) -> List[VideoResult]:   """Search for educational videos"""   pass   async def extract_audio_content(self, urls: List[str]) -> List[AudioTranscript]:   """Extract content from audio/video sources"""   pass
-

- **Priority**: Medium
- **Effort**: 4-6 weeks

14.1.2 Enhanced Document Processing

Enhancement: Advanced Document Format Support

- **Description**: Support for presentations, spreadsheets, and specialized academic formats
- **Benefits**: Broader document compatibility
- **Implementation**:
  # Enhanced document processorsclass AdvancedDocumentProcessor: def process_presentation(self, file_path: str) -> DocumentContent: """Extract content from PowerPoint/Google Slides"""   pass   def process_spreadsheet(self, file_path: str) -> DocumentContent: """Extract structured data from Excel/CSV files"""   pass   def process_latex(self, file_path: str) -> DocumentContent: """Process LaTeX academic papers"""   pass
-
- **Priority**: High
- **Effort**: 3-4 weeks

14.1.3 Real-time Collaboration

Enhancement: Multi-user Research Sessions

- **Description**: Allow multiple users to collaborate on research projects
- **Benefits**: Enhanced team productivity
- **Implementation**:
  - WebSocket-based real-time updates
  - Shared research sessions with role-based access
  - Conflict resolution for concurrent edits
  - Activity tracking and notification system
- **Priority**: Low
- **Effort**: 6-8 weeks

14.2 Medium-term Enhancements (6-12 months)

14.2.1 Machine Learning Enhancements

Enhancement: Personalized Research Recommendations

- **Description**: Learn from user behavior to provide personalized research suggestions
- **Benefits**: Improved user experience and research efficiency
- **Implementation**:
  class PersonalizationEngine: def __init__(self):   self.user_profiles = {}   self.recommendation_model = CollaborativeFilteringModel()   async def get_personalized_recommendations(self, user_id: str, current_research: ResearchContext   ) -> List[Recommendation]:   """Generate personalized research recommendations"""   user_profile = self.user_profiles.get(user_id)   if

not user_profile: return self.get_default_recommendations(current_research)
return await self.recommendation_model.predict(user_profile,
current_research   )

-
- **Priority**: Medium
- **Effort**: 8-10 weeks

14.2.2 Advanced Analytics Dashboard

Enhancement: Research Analytics and Insights

- **Description**: Comprehensive dashboard for research patterns and productivity metrics
- **Benefits**: Data-driven research optimization
- **Features**:
    - Research productivity metrics
    - Topic trend analysis
    - Source reliability scoring
    - Collaboration analytics
    - Custom report generation
- **Priority**: Medium
- **Effort**: 6-8 weeks

14.2.3 API Gateway and Marketplace

Enhancement: Third-party Integration Platform

- **Description**: Allow third-party developers to create custom research tools
- **Benefits**: Ecosystem expansion and specialized tool development
- **Implementation**:
    - Plugin architecture with standardized APIs
    - Tool marketplace for sharing custom analyzers
    - SDK for third-party developers
    - Revenue sharing model for premium tools
- **Priority**: Low
- **Effort**: 12-16 weeks

14.3 Long-term Enhancements (12+ months)

14.3.1 Advanced AI Capabilities

Enhancement: Autonomous Research Agents

- **Description**: AI agents that can conduct independent research based on high-level objectives
- **Benefits**: Fully automated research workflows
- **Implementation**:
 class AutonomousResearchAgent: def __init__(self, objective: str):
self.objective = objective   self.

-
-