Research Agent System - Complete Flow Documentation

Version: 2.0.0 **Date:** 2025-01-09 **Status:** Production-Ready (ROI Model Architecture)

Table of Contents

---

1. Overview

System Purpose

The Research Agent System provides two specialized AI agents for **ROI Model Data Extraction** in the Triton platform. These agents extract quantitative financial metrics, variables, and formulas that enable mathematical ROI calculations rather than qualitative value propositions.

- **WebSearchAgent**: Researches healthcare companies to extract ROI Model data (cost savings, metrics, formulas)
- **DocumentAnalysisAgent**: Analyzes ROI Story documents (PDF, DOCX, TXT) from S3 to extract ROI Model components

Both agents use Claude Sonnet 4 via AWS Bedrock with a sophisticated 4-layer validation pipeline and automatic retry logic.

Architectural Shift: Value Propositions → ROI Models

```
graph LR
    subgraph "Old Architecture"
        A1[Value Proposition Text] --> B1[Template Generator]
        B1 --> C1[Static Dashboard Templates]
    end

    subgraph "New Architecture ROI Model"
        A2[ROI Story Document] --> B2[Research Agents]
        B2 --> C2[ROI Model Variables + Formulas]
        C2 --> D2[Dashboard Generator]
        D2 --> E2[Data-Driven Dashboards]
    end

    style C2 fill:#90EE90
    style E2 fill:#FFD700
```

Concept	Old System	New System (ROI Model)
Primary Entity	Value Proposition (qualitative)	**ROI Model** (quantitative)
Input	Client documents/ research	**ROI Story** (structured document)
Classification	None	**13 ROI Model Types**
Variables	Hardcoded	**Dynamic variables** extracted
Formulas	None	**Mathematical formulas** for calculations
Output	Static templates	**ROI Model → Dashboard Templates**

Key Features

□**Dual Research Modes**: Autonomous (15-25 searches) and Manual (5-15 searches) □**Multi-Tool Integration**: Google search, web scraper, S3 document reader □**ROI Model Extraction**: Extract variables, formulas, and assumptions for 13 ROI Model Types □**Robust Validation**: 4-layer validation with retry feedback □ **Async Processing**: Background task execution with job tracking □**Mock Mode**: Fallback to mock data when API keys unavailable

---

2. 13 ROI Model Types

Overview

Research agents extract data to populate one of **13 distinct ROI Model Types**. Each type represents a different healthcare cost reduction or value creation strategy with specific:

- **Variables**: Baseline costs, targets, intervention parameters
- **Formulas**: ROI calculations, payback periods, savings projections
- **Data Requirements**: Claims types, member data, clinical outcomes
- **Industry Benchmarks**: Typical ranges for metrics

Type Classification Matrix

#	ROI Model Type	Category	Key Metrics	Typical ROI Range
1	**Unit Price Reduction**	Cost Reduction	Unit cost PMPM, contract savings %	150-300%
2	**Site of Care Shift**	Utilization	HOPD vs freestanding cost delta	200-400%
3	**Provider Steering**	Cost Reduction	Tier differential, steering rate	180-350%
4		Administrative		500-1000%

#	ROI Model Type	Category	Key Metrics	Typical ROI Range
	Payment Integrity		Overpayment recovery $, edit hit rate	
5	**Utilization Reduction**	Utilization	Low-value service rate, reduction %	200-450%
6	**Medical Management**	Utilization	Chronic disease costs, outcomes	250-500%
7	**Episode Optimization**	Utilization	Episode cost, pathway adherence %	150-300%
8	**OON Mitigation**	Administrative	OON rate, cost differential	300-600%
9	**Leakage Recapture**	Utilization	Leakage %, in-system advantage	200-400%
10	**Pharmacy Optimization**	Cost Reduction	Pharmacy PMPM, specialty drug %	250-400%
11	**Supply Chain Validation**	Administrative	Device/implant cost variance	400-800%
12	**Admin Error Reduction**	Administrative	Duplicate payment $, errors	600-1200%
13	**Vendor Incentive ROI**	Utilization	Incentive $, steerage impact	100-250%

Research Agent Role

WebSearchAgent and **DocumentAnalysisAgent** extract:

1. **ROI Model Type Indicators**: Keywords, themes, and metrics suggesting which of 13 types applies
2. **Variable Values**: Baseline costs, population sizes, target metrics, intervention parameters

3. **Formula Components**: Cost calculations, savings formulas, ROI methodology
4. **Assumptions**: Industry benchmarks, default values, data source requirements
5. **Evidence**: Supporting sources, confidence scores, extracted page numbers

The extracted data feeds into the **ROI Model Builder Agent** (separate system) which constructs the complete ROI Model.

---

3. System Architecture

3.1 High-Level Architecture

```
graph TB
    subgraph "API Layer"
        API[FastAPI Application]
        WS_EP["/research/web-search"]
        DA_EP["/research/document-analysis"]
        STATUS_EP["/research/{job_id}"]
    end

    subgraph "Agent Layer"
        WSA[WebSearchAgent]
        DAA[DocumentAnalysisAgent]
        RETRY[Retry Wrapper]
        TEMPLATE[Agent Template]
    end

    subgraph "Tool Layer"
        GS[Google Search Tool]
        WS[Web Scraper Tool]
        S3[S3 Document Reader]
        MOCK[Mock Mode Fallback]
    end

    subgraph "LLM Layer"
        BEDROCK[AWS Bedrock]
        CLAUDE[Claude Sonnet 4]
    end
```

```
    subgraph "Validation Layer"
        JSON_EXT[JSON Extraction]
        JSON_PARSE[JSON Parsing]
        PYDANTIC[Pydantic Validation]
        BUSINESS[Business Logic Validation]
    end

    API --> WS_EP
    API --> DA_EP
    API --> STATUS_EP

    WS_EP --> WSA
    DA_EP --> DAA

    WSA --> RETRY
    DAA --> RETRY
    RETRY --> TEMPLATE

    TEMPLATE --> GS
    TEMPLATE --> WS
    TEMPLATE --> S3

    GS -.fallback.-> MOCK
    WS -.fallback.-> MOCK
    S3 -.fallback.-> MOCK

    TEMPLATE --> BEDROCK
    BEDROCK --> CLAUDE

    CLAUDE --> JSON_EXT
    JSON_EXT --> JSON_PARSE
    JSON_PARSE --> PYDANTIC
    PYDANTIC --> BUSINESS

    style API fill:#e1f5ff
    style WSA fill:#fff3e0
```

```
    style DAA fill:#fff3e0
    style BEDROCK fill:#f3e5f5
    style CLAUDE fill:#f3e5f5
```

3.2 Data Flow Overview (ROI Model Extraction)

```
sequenceDiagram
    participant User
    participant API
    participant BackgroundTask
    participant Agent
    participant Tools
    participant LLM
    participant Validation

    User->>API: POST /research/web-search<br/>(Extract ROI Model Data)
    API->>API: Create job (pending)
    API->>BackgroundTask: Schedule execution
    API-->>User: 202 Accepted (job_id)

    BackgroundTask->>BackgroundTask: Update status: in_progress
    BackgroundTask->>Agent: Create agent with retry
    Agent->>Tools: Load tools
    Tools-->>Agent: google_search, scrape_webpage

    Agent->>LLM: Execute ROI Model research
    loop For each search (15-25)
        LLM->>Tools: google_search(ROI metrics query)
        Tools-->>LLM: Search results
        LLM->>Tools: scrape_webpage(url)
        Tools-->>LLM: Page content with metrics
    end

    LLM-->>Agent: Raw response (ROI Model data)
    Agent->>Validation: Validate ROI Model extraction

    alt Validation Success
```

```
        Validation-->>Agent: Valid ROI Model data
        Agent-->>BackgroundTask: ROI Model extraction result
        BackgroundTask->>BackgroundTask: Update status: completed
    else Validation Failure
        Validation-->>Agent: Errors + Feedback
        Agent->>LLM: Retry with feedback
    end

    User->>API: GET /research/{job_id}
    API-->>User: Status + ROI Model Data
```

---

4. WebSearchAgent

Purpose: ROI Model Data Extraction via Web Search

The WebSearchAgent researches healthcare companies to extract **ROI Model data components**:

- **ROI Model Type Indicators**: Keywords and themes suggesting which of 13 ROI Model Types applies
- **Quantitative Metrics**: Baseline costs, savings percentages, population sizes, PMPM values
- **Formula Components**: ROI calculation methods, payback period formulas, savings calculations
- **Clinical Outcomes**: HbA1c reductions, admission rate decreases, adherence improvements
- **Financial Evidence**: Contract savings, cost avoidance, total cost of care reductions
- **Target Audiences**: Health plans, employers, providers, TPAs
- **Industry Benchmarks**: Typical ranges for metrics in each ROI Model Type

4.1 Agent Creation Flow

```
flowchart TD
    START([Start: create_web_search_agent_with_retry]) --> GET_MODEL{Mod
    GET_MODEL -->|No| DEFAULT[Get default model from config]
```

```
    GET_MODEL -->|Yes| USE_MODEL[Use provided model]

    DEFAULT --> CREATE_TEMPLATE[Create WebSearchAgentTemplate]
    USE_MODEL --> CREATE_TEMPLATE

    CREATE_TEMPLATE --> SET_MODE{Research Mode}
    SET_MODE -->|autonomous| AUTO["Set: 15-25 searches mode"]
    SET_MODE -->|manual| MANUAL["Set: 5-15 searches mode"]

    AUTO --> LOAD_INST[Load web_search_instructions.md]
    MANUAL --> LOAD_INST

    LOAD_INST --> ADD_CONTEXT[Append mode-specific context]
    ADD_CONTEXT --> LOAD_TOOLS[Load Tools]

    LOAD_TOOLS --> GS_TOOL[Google Search Tool]
    LOAD_TOOLS --> WS_TOOL[Web Scraper Tool]

    GS_TOOL --> CHECK_API{API Keys?}
    WS_TOOL --> CHECK_API

    CHECK_API -->|Present| REAL_MODE[Real API Mode]
    CHECK_API -->|Missing| MOCK_MODE[Mock Mode]

    REAL_MODE --> CREATE_AGENT[Create MareAgent]
    MOCK_MODE --> CREATE_AGENT

    CREATE_AGENT --> WRAP_RETRY[Wrap with retry logic]
    WRAP_RETRY --> SET_MAX[Set max_retries=3]
    SET_MAX --> RETURN([Return: Agent with retry])

    style START fill:#e8f5e9
    style RETURN fill:#e8f5e9
    style REAL_MODE fill:#c8e6c9
    style MOCK_MODE fill:#ffecb3
```

4.2 Execution Flow (Autonomous Mode) - ROI Model Data Extraction

```
flowchart TD
    START([Agent.run called]) --> BUILD_MSG[Build research message]
    BUILD_MSG --> COMPANY["Company: client_company_name"]
    BUILD_MSG --> INDUSTRY[Industry hint if provided]
    BUILD_MSG --> CONTEXT[Additional context if provided]

    COMPANY --> EXECUTE[Execute LLM with tools]
    INDUSTRY --> EXECUTE
    CONTEXT --> EXECUTE

    EXECUTE --> PHASE1["Phase 1: Company Information"]
    PHASE1 --> S1["google_search('company overview')"]
    S1 --> S2["google_search('company mission')"]
    S2 --> S3["scrape_webpage(company_website)"]

    S3 --> PHASE2["Phase 2: Product & Solutions"]
    PHASE2 --> S4["google_search('products solutions')"]
    S4 --> S5["google_search('platform features')"]

    S5 --> PHASE3["Phase 3: ROI Model Components"]
    PHASE3 --> S6["google_search('ROI cost savings')"]
    S6 --> S7["google_search('financial metrics PMPM')"]

    S7 --> PHASE4["Phase 4: Clinical Outcomes & Metrics"]
    PHASE4 --> S8["google_search('clinical outcomes data')"]
    S8 --> S9["google_search('quality metrics study')"]

    S9 --> PHASE5["Phase 5: Competitive Landscape"]
    PHASE5 --> S10["google_search('competitors')"]
    S10 --> S11["google_search('vs competitor ROI')"]

    S11 --> PHASE6["Phase 6: Market Presence"]
    PHASE6 --> S12["google_search('news press release')"]
    S12 --> S13["google_search('awards partnerships')"]
```

```
    S13 --> PHASE7["Phase 7: Customer ROI Evidence"]
    PHASE7 --> S14["google_search('case study ROI')"]
    S14 --> S15["google_search('customer results metrics')"]

    S15 --> CHECK_COUNT{Total searches >= 15?}
    CHECK_COUNT -->|No| MORE["google_search(additional ROI data)"]
    MORE --> CHECK_COUNT
    CHECK_COUNT -->|Yes| SYNTHESIZE[Synthesize ROI Model data into JSON]

    SYNTHESIZE --> RETURN_JSON[Return WebSearchResult with ROI Model dat
    RETURN_JSON --> END([End: Raw LLM response])

    style START fill:#e3f2fd
    style END fill:#e3f2fd
    style PHASE1 fill:#fff3e0
    style PHASE2 fill:#fff3e0
    style PHASE3 fill:#ffcccc
    style PHASE4 fill:#fff3e0
    style PHASE5 fill:#fff3e0
    style PHASE6 fill:#fff3e0
    style PHASE7 fill:#fff3e0
```

4.3 Execution Flow (Manual Mode) - ROI Model Data Extraction

```
flowchart TD
    START([Agent.run called with prompts]) --> PARSE_PROMPTS[Parse user

    PARSE_PROMPTS --> P1["Prompt 1: Research solution"]
    PARSE_PROMPTS --> P2["Prompt 2: Find clinical outcomes"]
    PARSE_PROMPTS --> P3["Prompt 3: Identify target segments"]

    P1 --> EXECUTE1[Execute searches for Prompt 1]
    EXECUTE1 --> S1["google_search(query 1)"]
    S1 --> S2["google_search(query 2)"]
    S2 --> SCRAPE1["scrape_webpage(url 1)"]
```

```
    P2 --> EXECUTE2[Execute searches for Prompt 2]
    EXECUTE2 --> S3["google_search(query 3)"]
    S3 --> S4["google_search(query 4)"]

    P3 --> EXECUTE3[Execute searches for Prompt 3]
    EXECUTE3 --> S5["google_search(query 5)"]

    SCRAPE1 --> CHECK_COUNT{Total searches >= 5?}
    S4 --> CHECK_COUNT
    S5 --> CHECK_COUNT

    CHECK_COUNT -->|No| MORE["Additional searches if needed"]
    MORE --> CHECK_COUNT

    CHECK_COUNT -->|Yes| SYNTHESIZE[Synthesize findings]
    SYNTHESIZE --> ADDRESS_P1[Address Prompt 1 findings]
    SYNTHESIZE --> ADDRESS_P2[Address Prompt 2 findings]
    SYNTHESIZE --> ADDRESS_P3[Address Prompt 3 findings]

    ADDRESS_P1 --> COMPILE[Compile WebSearchResult]
    ADDRESS_P2 --> COMPILE
    ADDRESS_P3 --> COMPILE

    COMPILE --> RETURN_JSON[Return JSON]
    RETURN_JSON --> END([End: Raw LLM response])

    style START fill:#e3f2fd
    style END fill:#e3f2fd
    style P1 fill:#e3f2fd
    style P2 fill:#e3f2fd
    style P3 fill:#e3f2fd
```

4.4 Tool Integration Flow

```
flowchart TD
    subgraph "Google Search Tool"
        GS_START[google_search called] --> GS_CHECK{TAVILY_API_KEY set?}
```

```
        GS_CHECK -->|Yes| GS_REAL[Call Tavily API]
        GS_CHECK -->|No| GS_MOCK[Return mock results]

        GS_REAL --> GS_PARSE[Parse results]
        GS_MOCK --> GS_PARSE

        GS_PARSE --> GS_RETURN["Return SearchResult[]"]
    end

    subgraph "Web Scraper Tool"
        WS_START[scrape_webpage called] --> WS_CHECK{FIRECRAWL_API_KEY s
        WS_CHECK -->|Yes| WS_FIRE[Call Firecrawl API]
        WS_CHECK -->|No| WS_BASIC[Basic scraping mode]

        WS_FIRE --> WS_CONVERT[Convert to markdown]
        WS_BASIC --> WS_CONVERT

        WS_CONVERT --> WS_RETURN[Return page content]
    end

    AGENT[LLM Agent Execution] --> GS_START
    AGENT --> WS_START

    GS_RETURN --> AGENT_CTX[Add to agent context]
    WS_RETURN --> AGENT_CTX

    AGENT_CTX --> NEXT_SEARCH{More searches needed?}
    NEXT_SEARCH -->|Yes| AGENT
    NEXT_SEARCH -->|No| SYNTHESIZE[Synthesize results]

    style GS_REAL fill:#c8e6c9
    style GS_MOCK fill:#ffecb3
    style WS_FIRE fill:#c8e6c9
    style WS_BASIC fill:#ffecb3
```

4.5 Retry Logic with Validation Feedback

```
flowchart TD
    START([run_with_retry called]) --> ATTEMPT_1["Attempt 1/3"]

    ATTEMPT_1 --> EXEC_1[Execute agent.run]
    EXEC_1 --> EXTRACT_1[Extract JSON from response]

    EXTRACT_1 --> JSON_OK_1{JSON extracted?}
    JSON_OK_1 -->|No| ERROR_1[Build error feedback]
    JSON_OK_1 -->|Yes| PARSE_1[Parse JSON]

    PARSE_1 --> PARSE_OK_1{Valid JSON syntax?}
    PARSE_OK_1 -->|No| ERROR_1
    PARSE_OK_1 -->|Yes| PYDANTIC_1[Pydantic validation]

    PYDANTIC_1 --> PYD_OK_1{Pydantic valid?}
    PYD_OK_1 -->|No| ERROR_1
    PYD_OK_1 -->|Yes| BUSINESS_1[Business logic validation]

    BUSINESS_1 --> BUS_OK_1{Business rules pass?}
    BUS_OK_1 -->|Yes| SUCCESS([Return validated result])
    BUS_OK_1 -->|No| ERROR_1

    ERROR_1 --> ATTEMPT_2["Attempt 2/3"]
    ATTEMPT_2 --> FEEDBACK_2["Append feedback to message:<br/>- Previous

    FEEDBACK_2 --> EXEC_2[Execute agent.run with feedback]
    EXEC_2 --> EXTRACT_2[Extract JSON]

    EXTRACT_2 --> JSON_OK_2{JSON extracted?}
    JSON_OK_2 -->|No| ERROR_2[Build error feedback]
    JSON_OK_2 -->|Yes| PARSE_2[Parse JSON]

    PARSE_2 --> PARSE_OK_2{Valid JSON syntax?}
    PARSE_OK_2 -->|No| ERROR_2
    PARSE_OK_2 -->|Yes| PYDANTIC_2[Pydantic validation]
```

```
    PYDANTIC_2 --> PYD_OK_2{Pydantic valid?}
    PYD_OK_2 -->|No| ERROR_2
    PYD_OK_2 -->|Yes| BUSINESS_2[Business logic validation]

    BUSINESS_2 --> BUS_OK_2{Business rules pass?}
    BUS_OK_2 -->|Yes| SUCCESS
    BUS_OK_2 -->|No| ERROR_2

    ERROR_2 --> ATTEMPT_3["Attempt 3/3"]
    ATTEMPT_3 --> FEEDBACK_3["Append cumulative feedback:<br/>- All prev

    FEEDBACK_3 --> EXEC_3[Execute agent.run with full feedback]
    EXEC_3 --> EXTRACT_3[Extract JSON]

    EXTRACT_3 --> JSON_OK_3{JSON extracted?}
    JSON_OK_3 -->|No| FAILURE[Raise RuntimeError]
    JSON_OK_3 -->|Yes| PARSE_3[Parse JSON]

    PARSE_3 --> PARSE_OK_3{Valid JSON syntax?}
    PARSE_OK_3 -->|No| FAILURE
    PARSE_OK_3 -->|Yes| PYDANTIC_3[Pydantic validation]

    PYDANTIC_3 --> PYD_OK_3{Pydantic valid?}
    PYD_OK_3 -->|No| FAILURE
    PYD_OK_3 -->|Yes| BUSINESS_3[Business logic validation]

    BUSINESS_3 --> BUS_OK_3{Business rules pass?}
    BUS_OK_3 -->|Yes| SUCCESS
    BUS_OK_3 -->|No| FAILURE

    style START fill:#e8f5e9
    style SUCCESS fill:#c8e6c9
    style FAILURE fill:#ffcdd2
    style FEEDBACK_2 fill:#fff9c4
    style FEEDBACK_3 fill:#ffecb3
```

---

5. DocumentAnalysisAgent

Purpose: ROI Model Data Extraction from ROI Story Documents

The DocumentAnalysisAgent analyzes **ROI Story documents** (client-uploaded PDFs, DOCX, TXT) to extract **ROI Model data components**:

- **ROI Model Type Indicators**: Document themes suggesting which of 13 ROI Model Types applies
- **Quantitative Metrics**: Baseline costs, target savings, population sizes, financial projections
- **Formula Components**: ROI calculations, cost methodologies referenced in documents
- **Clinical Metrics**: Patient outcomes, quality measures, utilization rates
- **Financial Data**: Intervention costs, implementation timelines, payback periods
- **Assumptions**: Stated assumptions, industry benchmarks mentioned
- **Data Requirements**: Identified data sources needed for calculations

5.1 Agent Creation Flow

```
flowchart TD
    START([Start: create_document_analysis_agent_with_retry]) --> GET_MO
    GET_MODEL -->|No| DEFAULT[Get default model]
    GET_MODEL -->|Yes| USE_MODEL[Use provided model]

    DEFAULT --> CREATE_TEMPLATE[Create DocumentAnalysisAgentTemplate]
    USE_MODEL --> CREATE_TEMPLATE

    CREATE_TEMPLATE --> LOAD_INST[Load document_analysis_instructions.md
    LOAD_INST --> LOAD_TOOL[Load S3 Document Reader Tool]

    LOAD_TOOL --> CHECK_AWS{AWS credentials?}
    CHECK_AWS -->|Present| REAL_S3[Real S3 mode]
    CHECK_AWS -->|Missing| MOCK_S3[Mock mode]

    REAL_S3 --> CREATE_AGENT[Create MareAgent]
    MOCK_S3 --> CREATE_AGENT
```

```
    CREATE_AGENT --> WRAP_RETRY[Wrap with retry logic]
    WRAP_RETRY --> SET_MAX[Set max_retries=3]
    SET_MAX --> RETURN([Return: Agent with retry])

    style START fill:#e8f5e9
    style RETURN fill:#e8f5e9
    style REAL_S3 fill:#c8e6c9
    style MOCK_S3 fill:#ffecb3
```

5.2 Document Analysis Execution Flow (ROI Story Data Extraction)

```
flowchart TD
    START([Agent.run called with ROI Story documents]) --> PARSE_DOCS[Pa

    PARSE_DOCS --> DOC1["Document 1: s3://path/roi_story.pdf"]
    PARSE_DOCS --> DOC2["Document 2: s3://path/case_study.pdf"]
    PARSE_DOCS --> DOC3["Document 3: s3://path/financial_projections.pdf

    DOC1 --> READ1["read_document('s3://path/roi_story.pdf')"]
    DOC2 --> READ2["read_document('s3://path/case_study.pdf')"]
    DOC3 --> READ3["read_document('s3://path/financial_projections.pdf')

    READ1 --> EXTRACT1[Extract text from PDF]
    READ2 --> EXTRACT2[Extract text from PDF]
    READ3 --> EXTRACT3[Extract text from PDF]

    EXTRACT1 --> ANALYZE1[Analyze Document 1]
    EXTRACT2 --> ANALYZE2[Analyze Document 2]
    EXTRACT3 --> ANALYZE3[Analyze Document 3]

    ANALYZE1 --> FIND_ROI1[Extract ROI Model components]
    ANALYZE1 --> FIND_VAR1[Extract variables PMPM, costs]
    ANALYZE1 --> FIND_FM1[Extract financial metrics]

    ANALYZE2 --> FIND_ROI2[Extract ROI Model components]
```

```
    ANALYZE2 --> FIND_VAR2[Extract variables]
    ANALYZE2 --> FIND_FM2[Extract financial metrics]

    ANALYZE3 --> FIND_ROI3[Extract ROI Model components]
    ANALYZE3 --> FIND_VAR3[Extract variables]
    ANALYZE3 --> FIND_FM3[Extract financial metrics]

    FIND_ROI1 --> SYNTHESIZE[Synthesize all findings]
    FIND_VAR1 --> SYNTHESIZE
    FIND_FM1 --> SYNTHESIZE
    FIND_ROI2 --> SYNTHESIZE
    FIND_VAR2 --> SYNTHESIZE
    FIND_FM2 --> SYNTHESIZE
    FIND_ROI3 --> SYNTHESIZE
    FIND_VAR3 --> SYNTHESIZE
    FIND_FM3 --> SYNTHESIZE

    SYNTHESIZE --> DEDUPE[Deduplicate across documents]
    DEDUPE --> CONF_SCORE[Calculate confidence scores]
    CONF_SCORE --> COMPILE[Compile DocumentAnalysisResult with ROI Model

    COMPILE --> RETURN_JSON[Return JSON]
    RETURN_JSON --> END([End: Raw LLM response])

    style START fill:#e3f2fd
    style END fill:#e3f2fd
    style ANALYZE1 fill:#fff3e0
    style ANALYZE2 fill:#fff3e0
    style ANALYZE3 fill:#fff3e0
```

4.3 Document Type Processing

```
flowchart TD
    START[read_document called] --> PARSE_PATH[Parse S3 path]
    PARSE_PATH --> EXTRACT_EXT[Extract file extension]

    EXTRACT_EXT --> CHECK_TYPE{File type?}
```

```
CHECK_TYPE -->|.pdf| PDF_FLOW[PDF Processing]
CHECK_TYPE -->|.docx| DOCX_FLOW[DOCX Processing]
CHECK_TYPE -->|.txt| TXT_FLOW[TXT Processing]
CHECK_TYPE -->|other| ERROR[Unsupported format]

PDF_FLOW --> DOWNLOAD_PDF[Download from S3]
DOWNLOAD_PDF --> PYPDF2[PyPDF2 extraction]
PYPDF2 --> EXTRACT_TEXT_PDF[Extract text from pages]
EXTRACT_TEXT_PDF --> COMBINE_PDF[Combine pages]

DOCX_FLOW --> DOWNLOAD_DOCX[Download from S3]
DOWNLOAD_DOCX --> PYTHON_DOCX[python-docx extraction]
PYTHON_DOCX --> EXTRACT_TEXT_DOCX[Extract paragraphs]
EXTRACT_TEXT_DOCX --> COMBINE_DOCX[Combine paragraphs]

TXT_FLOW --> DOWNLOAD_TXT[Download from S3]
DOWNLOAD_TXT --> READ_TXT[Read as UTF-8]

COMBINE_PDF --> RETURN[Return extracted text]
COMBINE_DOCX --> RETURN
READ_TXT --> RETURN
ERROR --> RETURN_EMPTY[Return empty string]

RETURN --> END([End: Document text])
RETURN_EMPTY --> END

style START fill:#e8f5e9
style END fill:#e8f5e9
style PDF_FLOW fill:#ffccbc
style DOCX_FLOW fill:#c5cae9
style TXT_FLOW fill:#dcedc8
style ERROR fill:#ffcdd2
```

4.4 Tool Integration (S3 Document Reader)

```
flowchart TD
    subgraph "S3 Document Reader Tool"
        TOOL_START[read_document called] --> VALIDATE_PATH{Valid S3 path
        VALIDATE_PATH -->|No| MOCK_PATH["Treat as mock/test path"]
        VALIDATE_PATH -->|Yes| CHECK_AWS{AWS configured?}

        CHECK_AWS -->|No| MOCK_MODE[Return mock document content]
        CHECK_AWS -->|Yes| CONNECT_S3[Connect to S3]

        CONNECT_S3 --> DOWNLOAD[Download document]
        DOWNLOAD --> DETECT_TYPE[Detect file type]

        DETECT_TYPE --> PDF_PROC["PDF: PyPDF2"]
        DETECT_TYPE --> DOCX_PROC["DOCX: python-docx"]
        DETECT_TYPE --> TXT_PROC["TXT: UTF-8 read"]

        PDF_PROC --> EXTRACT[Extract text]
        DOCX_PROC --> EXTRACT
        TXT_PROC --> EXTRACT

        MOCK_PATH --> MOCK_CONTENT[Generate mock content]
        MOCK_MODE --> MOCK_CONTENT

        MOCK_CONTENT --> RETURN[Return text content]
        EXTRACT --> RETURN
    end

    AGENT[LLM Agent Execution] --> TOOL_START
    RETURN --> AGENT_CTX[Add document content to context]

    AGENT_CTX --> MORE_DOCS{More documents?}
    MORE_DOCS -->|Yes| AGENT
    MORE_DOCS -->|No| ANALYZE[Analyze all documents]

    style CONNECT_S3 fill:#c8e6c9
```

```
    style MOCK_MODE fill:#ffecb3
    style MOCK_CONTENT fill:#ffecb3
```

4.5 Retry Logic (Same as WebSearchAgent)

```
flowchart LR
    START([Attempt 1]) --> VALIDATE_1[4-Layer Validation]
    VALIDATE_1 --> SUCCESS_1{Pass?}
    SUCCESS_1 -->|Yes| DONE([Return result])
    SUCCESS_1 -->|No| FEEDBACK_1[Build error feedback]

    FEEDBACK_1 --> ATTEMPT_2([Attempt 2])
    ATTEMPT_2 --> VALIDATE_2[4-Layer Validation]
    VALIDATE_2 --> SUCCESS_2{Pass?}
    SUCCESS_2 -->|Yes| DONE
    SUCCESS_2 -->|No| FEEDBACK_2[Add to error feedback]

    FEEDBACK_2 --> ATTEMPT_3([Attempt 3])
    ATTEMPT_3 --> VALIDATE_3[4-Layer Validation]
    VALIDATE_3 --> SUCCESS_3{Pass?}
    SUCCESS_3 -->|Yes| DONE
    SUCCESS_3 -->|No| FAIL([Raise RuntimeError])

    style START fill:#e3f2fd
    style DONE fill:#c8e6c9
    style FAIL fill:#ffcdd2
```

---

5. API Layer

5.1 API Request Flow (Sequence Diagram)

```
sequenceDiagram
    autonumber
    participant Client
    participant FastAPI
```

```
    participant BackgroundTask
    participant Agent
    participant Tools
    participant LLM
    participant Validation
    participant JobStore

    Client->>FastAPI: POST /research/web-search<br/>{company_name, mode,
    FastAPI->>FastAPI: Validate request<br/>(check manual mode has promp
    FastAPI->>FastAPI: Generate job_id<br/>"research_web_abc123"
    FastAPI->>JobStore: Create job record<br/>{status: "pending", progre
    FastAPI->>BackgroundTask: Schedule execute_web_search_research
    FastAPI-->>Client: 202 Accepted<br/>{job_id, estimated_time: 120s}

    Note over BackgroundTask: Async execution begins

    BackgroundTask->>JobStore: Update {status: "in_progress", progress:
    BackgroundTask->>Agent: create_web_search_agent_with_retry
    Agent->>Tools: Load tools (google_search, scrape_webpage)
    Tools-->>Agent: Tools ready

    BackgroundTask->>JobStore: Update {progress: 20%}
    BackgroundTask->>Agent: Build research message
    BackgroundTask->>JobStore: Update {progress: 30%}

    BackgroundTask->>Agent: agent.run(message)

    loop For each retry attempt (max 3)
        Agent->>LLM: Execute with tools
        loop For each search (15-25 searches)
            LLM->>Tools: google_search(query)
            Tools-->>LLM: Search results
            LLM->>Tools: scrape_webpage(url)
            Tools-->>LLM: Page content
        end
        LLM-->>Agent: Raw response text
```

```
        Agent->>Validation: Extract JSON
        Validation->>Validation: Parse JSON
        Validation->>Validation: Pydantic validation
        Validation->>Validation: Business logic validation

        alt Validation Success
            Validation-->>Agent: Valid WebSearchResult
            Note over Agent: Break retry loop
        else Validation Failure
            Validation-->>Agent: Error feedback
            Agent->>LLM: Retry with feedback
        end
    end

    Agent-->>BackgroundTask: WebSearchResult
    BackgroundTask->>JobStore: Update {status: "completed", progress: 10

    Note over Client: Client polls for status

    Client->>FastAPI: GET /research/{job_id}
    FastAPI->>JobStore: Query job
    JobStore-->>FastAPI: Job data
    FastAPI-->>Client: 200 OK<br/>{status: "completed", result}
```

5.2 Job State Machine

```
stateDiagram-v2
    [*] --> Pending: API creates job
    Pending --> InProgress: Background task starts

    InProgress --> Completed: Agent succeeds
    InProgress --> Failed: Agent fails<br/>(after 3 retries)
    InProgress --> Failed: Unexpected error

    Completed --> [*]
    Failed --> [*]
```

```
    note right of Pending
        progress_percent: 0%
        status: "pending"
    end note

    note right of InProgress
        progress_percent: 10% → 30% → 90%
        status: "in_progress"
    end note

    note right of Completed
        progress_percent: 100%
        status: "completed"
        result: WebSearchResult or DocumentAnalysisResult
    end note

    note right of Failed
        status: "failed"
        error: Error message
    end note
```

5.3 Background Task Progression

```
flowchart LR
    START([Background task starts]) --> P10["10%: Mark in_progress"]
    P10 --> P20["20%: Create agent"]
    P20 --> P30["30%: Build message"]
    P30 --> P90["90%: Agent execution<br/>This takes 60-120 seconds"]
    P90 --> P100["100%: Validation & save"]
    P100 --> DONE([Mark completed])

    style START fill:#e3f2fd
    style P90 fill:#fff9c4
    style DONE fill:#c8e6c9
```

5.4 API Endpoints

Method	Endpoint	Description	Response
POST	`/research/web-search`	Initiate web search research	202 Accepted + job_id
POST	`/research/document-analysis`	Initiate document analysis	202 Accepted + job_id
GET	`/research/{job_id}`	Get job status and results	200 OK + job data
GET	`/research/`	List jobs with filters	200 OK + paginated list
GET	`/research/stats/summary`	Get aggregate statistics	200 OK + stats
POST	`/research/validate`	Validate research result	200 OK + validation report

---

6. Validation Pipeline

6.1 Four-Layer Validation Process

```
flowchart TD
    START([Raw LLM response]) --> LAYER1["Layer 1: JSON Extraction"]

    LAYER1 --> EXTRACT[Extract JSON from text]
    EXTRACT --> MARKDOWN{Contains markdown?}
    MARKDOWN -->|Yes| REGEX["Regex: ```json(.*)```"]
    MARKDOWN -->|No| FIND["Find first { to last }"]

    REGEX --> JSON_STR[JSON string]
    FIND --> JSON_STR

    JSON_STR --> LAYER2["Layer 2: JSON Parsing"]
    LAYER2 --> PARSE[json.loads]
```

```
PARSE --> SYNTAX_OK{Valid JSON syntax?}
SYNTAX_OK -->|No| ERROR_PARSE[JSON syntax error]
SYNTAX_OK -->|Yes| LAYER3["Layer 3: Pydantic Validation"]

LAYER3 --> PYDANTIC[Pydantic model validation]
PYDANTIC --> TYPE_OK{All types valid?}
TYPE_OK -->|No| ERROR_PYDANTIC["Type/constraint error"]
TYPE_OK -->|Yes| FIELD_OK{All required fields?}
FIELD_OK -->|No| ERROR_PYDANTIC
FIELD_OK -->|Yes| LAYER4["Layer 4: Business Logic"]

LAYER4 --> BIZ_RULES[Business rule validation]

BIZ_RULES --> CHECK_SEARCHES{Minimum searches?}
CHECK_SEARCHES -->|No| WARN_SEARCHES["Warning: Too few searches"]
CHECK_SEARCHES -->|Yes| CHECK_VP{Min value props?}

CHECK_VP -->|No| ERROR_VP["Error: Missing value props"]
CHECK_VP -->|Yes| CHECK_DESC{Description length?}

CHECK_DESC -->|No| ERROR_DESC["Error: Description too short"]
CHECK_DESC -->|Yes| CHECK_CONF{Confidence reasonable?}

CHECK_CONF -->|No| WARN_CONF["Warning: Low confidence"]
CHECK_CONF -->|Yes| CHECK_SOURCES{Sufficient sources?}

CHECK_SOURCES -->|No| WARN_SOURCES["Warning: Few sources"]
CHECK_SOURCES -->|Yes| SUCCESS([Validation passed])

ERROR_PARSE --> RETRY{Attempt < max?}
ERROR_PYDANTIC --> RETRY
ERROR_VP --> RETRY
ERROR_DESC --> RETRY

RETRY -->|Yes| FEEDBACK[Build error feedback]
RETRY -->|No| FAIL([Validation failed])
```

```
    FEEDBACK --> APPEND[Append feedback to message]
    APPEND --> RERUN[Re-run agent with feedback]
    RERUN --> START

    WARN_SEARCHES --> SUCCESS
    WARN_CONF --> SUCCESS
    WARN_SOURCES --> SUCCESS

    style START fill:#e3f2fd
    style SUCCESS fill:#c8e6c9
    style FAIL fill:#ffcdd2
    style LAYER1 fill:#fff3e0
    style LAYER2 fill:#e1f5ff
    style LAYER3 fill:#f3e5f5
    style LAYER4 fill:#e8f5e9
```

6.2 Validation Rules

WebSearchResult Validation

```
flowchart TD
    START[WebSearchResult validation] --> CHECK_MODE{Research mode?}

    CHECK_MODE -->|autonomous| AUTO_CHECK["searches_performed >= 15"]
    CHECK_MODE -->|manual| MANUAL_CHECK["searches_performed >= 5"]

    AUTO_CHECK --> VP_CHECK["value_propositions.length >= 1"]
    MANUAL_CHECK --> VP_CHECK

    VP_CHECK --> DESC_CHECK["company_overview.description.length >= 50"]
    DESC_CHECK --> SOURCES_CHECK["sources.length >= 1"]
    SOURCES_CHECK --> CONF_CHECK["0.0 <= confidence_score <= 1.0"]

    CONF_CHECK --> HIGH_CONF["Any high-confidence value props?"]
    HIGH_CONF -->|No| WARN_HIGH["Warning: No high-confidence VPs"]
    HIGH_CONF -->|Yes| DONE
```

```
    WARN_HIGH --> DONE([Validation report])

    style START fill:#e3f2fd
    style DONE fill:#c8e6c9
```

DocumentAnalysisResult Validation

```
flowchart TD
    START[DocumentAnalysisResult validation] --> DOCS_CHECK["documents_a

    DOCS_CHECK --> VP_CHECK["extracted_value_propositions.length >= 1"]
    VP_CHECK --> VP_NAME["Each VP name.length >= 10"]
    VP_NAME --> VP_DESC["Each VP description.length >= 30"]

    VP_DESC --> CONF_CHECK["0.0 <= overall_confidence <= 1.0"]

    CONF_CHECK --> LOW_CONF{Confidence < 0.5?}
    LOW_CONF -->|Yes| WARN_CONF["Warning: Low confidence"]
    LOW_CONF -->|No| HIGH_VP

    HIGH_VP["Any high-confidence VPs?"]
    HIGH_VP -->|No| WARN_VP["Warning: No high-confidence extractions"]
    HIGH_VP -->|Yes| DONE

    WARN_CONF --> DONE
    WARN_VP --> DONE([Validation report])

    style START fill:#e3f2fd
    style DONE fill:#c8e6c9
```

---

7. Data Models

7.1 WebSearchResult Model Hierarchy

```
classDiagram
    class WebSearchResult {
        +int searches_performed
        +List~str~ queries
        +CompanyOverview company_overview
        +List~ValuePropositionEvidence~ value_propositions
        +List~ClinicalOutcomeEvidence~ clinical_outcomes
        +ROIFramework roi_framework
        +CompetitivePositioning competitive_positioning
        +List~str~ target_audiences
        +List~str~ sources
        +str research_mode
        +float confidence_score
        +List~str~ missing_information
        +List~str~ assumptions_made
        +datetime research_timestamp
    }

    class CompanyOverview {
        +str name
        +str description
        +str mission
        +List~str~ target_markets
        +str website
    }

    class ValuePropositionEvidence {
        +str name
        +str description
        +str evidence_type
        +List~str~ supporting_sources
        +str confidence
    }
```

```
    class ClinicalOutcomeEvidence {
        +str outcome
        +str metric_value
        +str evidence_type
        +str source
        +str confidence
    }

    class ROIFramework {
        +str typical_roi_range
        +str payback_period
        +List~str~ cost_savings_areas
        +str evidence_quality
        +List~str~ supporting_sources
    }

    class CompetitivePositioning {
        +List~str~ main_competitors
        +List~str~ unique_advantages
        +List~str~ market_differentiators
        +str market_position
    }

    WebSearchResult *-- CompanyOverview
    WebSearchResult *-- ValuePropositionEvidence
    WebSearchResult *-- ClinicalOutcomeEvidence
    WebSearchResult *-- ROIFramework
    WebSearchResult *-- CompetitivePositioning
```

7.2 DocumentAnalysisResult Model Hierarchy

```
classDiagram
    class DocumentAnalysisResult {
        +int documents_analyzed
        +List~str~ document_names
        +List~ExtractedValueProposition~ extracted_value_propositions
```

```
        +List~ExtractedClinicalOutcome~ clinical_outcomes
        +List~ExtractedFinancialMetric~ financial_metrics
        +List~str~ target_audiences
        +List~ExtractedCompetitiveAdvantage~ competitive_advantages
        +str additional_context
        +float overall_confidence
        +List~str~ missing_information
        +datetime analysis_timestamp
    }

    class ExtractedValueProposition {
        +str name
        +str description
        +Dict metrics
        +str source_document
        +List~int~ page_numbers
        +str confidence
    }

    class ExtractedClinicalOutcome {
        +str outcome
        +str metric_value
        +str source_document
        +List~int~ page_numbers
        +str confidence
    }

    class ExtractedFinancialMetric {
        +str metric_name
        +str value
        +str context
        +str source_document
        +List~int~ page_numbers
    }

    class ExtractedCompetitiveAdvantage {
        +str advantage
```

```
        +str supporting_evidence
        +str source_document
    }

    DocumentAnalysisResult *-- ExtractedValueProposition
    DocumentAnalysisResult *-- ExtractedClinicalOutcome
    DocumentAnalysisResult *-- ExtractedFinancialMetric
    DocumentAnalysisResult *-- ExtractedCompetitiveAdvantage
```

7.3 API Request/Response Models

```
classDiagram
    class WebSearchRequest {
        +str client_company_name
        +str research_mode
        +str industry_hint
        +List~str~ prompts
        +str additional_context
        +int max_searches
    }

    class DocumentAnalysisRequest {
        +List~str~ document_ids
        +str additional_context
    }

    class ResearchJobResponse {
        +str job_id
        +str status
        +str message
        +str research_type
        +datetime created_at
        +int estimated_completion_seconds
    }

    class ResearchJobStatusResponse {
        +str job_id
```

```
        +str status
        +str research_type
        +int progress_percent
        +datetime created_at
        +datetime started_at
        +datetime completed_at
        +dict result
        +str error
    }

    class ResearchJobListResponse {
        +int total
        +int page
        +int page_size
        +List~ResearchJobStatusResponse~ jobs
    }

    class ResearchStatsResponse {
        +int total_jobs
        +int web_search_jobs
        +int document_analysis_jobs
        +int completed_jobs
        +int failed_jobs
        +float average_duration_seconds
        +float success_rate
    }

    ResearchJobListResponse *-- ResearchJobStatusResponse
```

---

8. Component Specifications

8.1 WebSearchAgent

File: `agents/web_search_agent.py`

Key Functions:

- `create_web_search_agent()` - Creates base agent
- `create_web_search_agent_with_retry()` - Wraps with retry logic
- `extract_json_from_response()` - Extracts JSON from LLM output

Configuration:

```
 research_mode: "autonomous" | "manual"
max_retries: int = 3
model: AWS Bedrock Claude Sonnet 4
tools: [GoogleSearchTool, WebScraperTool]
```

Research Modes:

- **Autonomous**: 15-25 searches across 7 research areas
- **Manual**: 5-15 searches following user prompts

Instructions: `agents/templates/web_search_instructions.md` (376 lines)

8.2 DocumentAnalysisAgent

File: `agents/document_analysis_agent.py`

Key Functions:

- `create_document_analysis_agent()` - Creates base agent
- `create_document_analysis_agent_with_retry()` - Wraps with retry logic
- `extract_json_from_response()` - Extracts JSON from LLM output

Configuration:

```
 max_retries: int = 3
model: AWS Bedrock Claude Sonnet 4
tools: [S3DocumentReaderTool]
```

Supported Document Types:

- **PDF**: PyPDF2 extraction
- **DOCX**: python-docx extraction

- **TXT**: UTF-8 text reading

Instructions: `agents/templates/document_analysis_instructions.md` (345 lines)

8.3 Tool Specifications

Google Search Tool

File: `tools/google_search_tool.py`

API: Tavily Search API (AI-optimized search)

Mock Mode: Returns 3 mock search results when `TAVILY_API_KEY` not set

Function Signature:

```
def google_search(
    query: str,
    max_results: int = 5,
    include_raw_content: bool = False
) -> List[Dict[str, Any]]
```

Returns:

```
[{
    "title": str,
    "url": str,
    "snippet": str,
    "content": str,  # if include_raw_content=True
    "score": float
}]
```

Web Scraper Tool

File: `tools/web_scraper_tool.py`

API: Firecrawl API (structured web scraping)

Mock Mode: Returns placeholder content when `FIRECRAWL_API_KEY` not set

Function Signature:

```
def scrape_webpage(url: str) -> str
```

Returns: Markdown-formatted page content

S3 Document Reader Tool

File: `tools/s3_document_reader.py`

Dependencies: boto3, PyPDF2, python-docx

Mock Mode: Returns mock document content when AWS not configured

Function Signature:

```
def read_document(storage_path: str) -> str
```

Path Format: `s3://bucket/key` or `bucket/key`

Returns: Extracted text from document

---

9. Configuration & Setup

9.1 Environment Variables

```
 # Model Configuration
DEFAULT_MODEL_PROVIDER=aws_bedrock
DEFAULT_MODEL_NAME=us.anthropic.claude-sonnet-4-20250514-v1:0

# AWS Bedrock (Required)
AWS_PROFILE=your-profile
AWS_REGION=us-east-1

# Research Tool APIs (Optional - uses mock mode if not set)
TAVILY_API_KEY=tvly-...          # For Google search
FIRECRAWL_API_KEY=fc-...         # For web scraping
```

```
# API Server
API_HOST=0.0.0.0
API_PORT=8000

# Logging
LOG_LEVEL=INFO
DEBUG_MODE=false
```

9.2 Installation

```
 # Create virtual environment
python3 -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Configure environment
cp .env.example .env
# Edit .env with your credentials
```

9.3 Running the System

Start API Server:

```
uvicorn app:app --reload --port 8000
```

Access Documentation:

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Run Tests:

```
python tests/test_research_api.py
```

10. Example Usage

10.1 Web Search Research (Autonomous Mode)

Request:

```
curl -X POST "http://localhost:8000/research/web-search" \
  -H "Content-Type: application/json" \
  -d '{
    "client_company_name": "Livongo Health",
    "research_mode": "autonomous",
    "industry_hint": "diabetes management",
    "additional_context": "Focus on ROI and clinical outcomes"
  }'
```

Response (202 Accepted):

```
{
  "job_id": "research_web_abc123",
  "status": "pending",
  "message": "Web search research initiated for Livongo Health",
  "research_type": "web_search",
  "created_at": "2025-12-08T10:30:00Z",
  "estimated_completion_seconds": 120
}
```

Check Status:

```
curl "http://localhost:8000/research/research_web_abc123"
```

Response (Completed):

```
{
  "job_id": "research_web_abc123",
  "status": "completed",
```

```
  "research_type": "web_search",
  "progress_percent": 100,
  "result": {
    "searches_performed": 18,
    "queries": ["Livongo diabetes", "Livongo ROI", ...],
    "company_overview": {
      "name": "Livongo Health",
      "description": "Digital health company focused on...",
      "mission": "Empower people with chronic conditions",
      "target_markets": ["Health Plans", "Employers"],
      "website": "https://livongo.com"
    },
    "value_propositions": [{
      "name": "Cost Reduction through Prevention",
      "description": "Reduce diabetes-related costs...",
      "evidence_type": "explicit",
      "supporting_sources": ["https://livongo.com/roi"],
      "confidence": "high"
    }],
    "confidence_score": 0.85
  }
}
```

10.2 Document Analysis

Request:

```
curl -X POST "http://localhost:8000/research/document-analysis" \
  -H "Content-Type: application/json" \
  -d '{
    "document_ids": [
      "s3://triton-docs/client123/roi_sheet.pdf",
      "s3://triton-docs/client123/case_study.pdf"
    ],
    "additional_context": "Diabetes management for health plans"
  }'
```

Response (202 Accepted):


```
{
  "job_id": "research_doc_xyz789",
  "status": "pending",
  "message": "Document analysis initiated for 2 documents",
  "research_type": "document_analysis",
  "estimated_completion_seconds": 60
}
```


Result:


```
{
  "documents_analyzed": 2,
  "document_names": ["roi_sheet.pdf", "case_study.pdf"],
  "extracted_value_propositions": [{
    "name": "340% ROI in 24 Months",
    "description": "Achieve 340% return on investment...",
    "metrics": {"roi": "340%", "payback_months": 14},
    "source_document": "roi_sheet.pdf",
    "page_numbers": [1, 2],
    "confidence": "high"
  }],
  "financial_metrics": [{
    "metric_name": "24-Month ROI",
    "value": "340%",
    "context": "Based on 1000-member population",
    "source_document": "roi_sheet.pdf",
    "page_numbers": [1]
  }],
  "overall_confidence": 0.95
}
```

10.3 Python Client Example

```
 import requests
import time

BASE_URL = "http://localhost:8000"

# Initiate research
response = requests.post(f"{BASE_URL}/research/web-search", json={
    "client_company_name": "Livongo Health",
    "research_mode": "autonomous",
    "industry_hint": "diabetes management"
})

job_id = response.json()["job_id"]
print(f"Research job created: {job_id}")

# Poll for completion
while True:
    status_response = requests.get(f"{BASE_URL}/research/{job_id}")
    status = status_response.json()

    print(f"Status: {status['status']} ({status.get('progress_percent',

    if status["status"] == "completed":
        result = status["result"]
        print(f"\n✓ Research complete!")
        print(f"  - Searches: {result['searches_performed']}")
        print(f"  - Company: {result['company_overview']['name']}")
        print(f"  - Value props: {len(result['value_propositions'])}")
        print(f"  - Confidence: {result['confidence_score']}")
        break
    elif status["status"] == "failed":
        print(f"\n✗ Research failed: {status['error']}")
        break

    time.sleep(2)
```

Summary

This research agent system provides:

□**Comprehensive Web Research**: 15-25 searches across 7 key research areas □ **Document Intelligence**: Extract value propositions from client materials □**Robust Validation**: 4-layer validation with automatic retry □**Production-Ready API**: Async processing with job tracking □**Mock Mode Support**: Test without external API dependencies □**Rich Data Models**: Structured output with confidence scores

Total Implementation:

- 3,073 lines of production code
- 2 specialized agents
- 3 tool integrations
- 6 REST API endpoints
- 4-layer validation pipeline
- 15+ Mermaid diagrams (in this document)

Document Version: 1.0.0 **Last Updated:** 2025-12-08 **Status:** Complete □

⚠ Mermaid diagrams are shown as code blocks in this PDF.

For interactive diagrams, use VS Code Markdown Preview Enhanced or visit the HTML version.