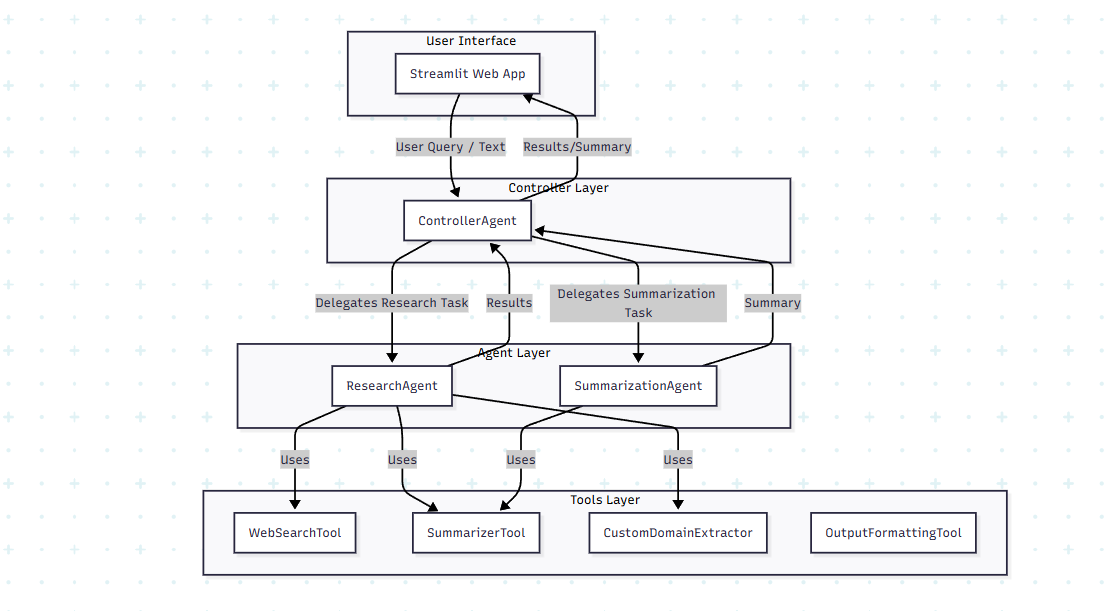
**AgenticAI System – Technical Documentation**

1. **System Architecture Diagram**



**Description:**

* The user interacts with the system via a Streamlit web app or CLI.
* The ControllerAgent orchestrates tasks and delegates them to specialized agents.
* Specialized agents (ResearchAgent, SummarizationAgent) use built-in and custom tools.
* All agents and tools interact through a modular, extensible architecture.

**2. Agent Roles and Responsibilities**

**ControllerAgent**

* **Role:** Central orchestrator.
* **Responsibilities:**
  + Receives tasks from the UI.
  + Delegates tasks to the appropriate specialized agent.
  + Handles errors and fallback strategies.
  + Maintains a registry of agents and manages communication.

**ResearchAgent**

* **Role:** Handles research queries.
* **Responsibilities:**
  + Uses WebSearchTool to fetch web results.
  + Optionally uses SummarizerTool for summarizing results.
  + Uses CustomDomainExtractor for extracting domain-specific information.
  + Maintains context and memory for each query.
  + Updates strategies based on feedback.

**SummarizationAgent**

* **Role:** Handles text summarization.
* **Responsibilities:**
  + Uses SummarizerTool to generate concise summaries.
  + Maintains context and memory for each summarization task.
  + Updates strategies based on feedback.

**3. Tool Integration and Functionality**

**WebSearchTool (SerpAPI)**

* **Purpose:** Fetches search results from the web.
* **Input:** User query string.
* **Output:** List of dicts with title, link, and snippet.
* **Integration:** Used by ResearchAgent.
* **Error Handling:** Handles API errors and returns empty results if needed.

**SummarizerTool (OpenAI API)**

* **Purpose:** Summarizes text using GPT-3.5-turbo.
* **Input:** Raw text or processed data.
* **Output:** Concise summary string.
* **Integration:** Used by both ResearchAgent and SummarizationAgent.
* **Error Handling:** Handles API errors and returns error messages if needed.

**OutputFormattingTool**

* **Purpose:** Formats output for display in the UI.
* **Input:** Summary or research results.
* **Output:** Formatted string or markdown.
* **Integration:** Used in the final step before displaying results.

**CustomDomainExtractor (Custom Tool)**

* **Purpose:** Extracts domain-specific information from research results.
* **Input:** List of search result dicts.
* **Output:** Filtered or highlighted information relevant to the chosen domain.
* **Integration:** Used by ResearchAgent.
* **Error Handling:** Handles missing or malformed data gracefully.
* **Enhancement:** Improves relevance and actionability of research output.

**4. Custom Tool Implementation**

**File:** [custom\_domain\_extractor.py](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)

* **Purpose:**  
  Extracts and highlights information specific to the research domain (e.g., AI, healthcare, finance) from the web search results.
* **Inputs:**  
  List of dicts, each containing title, link, and snippet.
* **Outputs:**  
  Filtered list or highlighted text relevant to the domain.
* **Error Handling:**  
  Skips or flags results with missing fields.
* **Limitations:**  
  Extraction logic may need tuning for different domains or more complex queries.
* **How it enhances the system:**  
  Makes research output more focused and useful for the user’s needs.

**5. Workflow and Orchestration**

**File:** [workflow.py](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)

**Step-by-Step Workflow**

1. **Research Phase:**
   * [perform\_search(query)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) is called to fetch web results.
   * Results are stored in memory:  
     [self.memory\_manager.store\_memory('search\_results', search\_results)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
2. **Data Processing:**
   * [process\_data(search\_results)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) processes the raw results.
   * Processed data is stored in memory:  
     [self.memory\_manager.store\_memory('processed\_data', processed\_data)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
3. **Summarization:**
   * [self.summarization\_agent.summarize(processed\_data)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) generates a summary.
   * Summary is stored in memory:  
     [self.memory\_manager.store\_memory('summary', summary)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
4. **Output Formatting:**
   * [format\_output(summary)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) prepares the summary for display.
5. **Feedback Loop:**
   * [feedback\_loop(user\_feedback)](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) updates agent strategies based on user input.
6. **Memory Management:**
   * [manage\_memory()](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) cleans up or updates memory for contextual awareness.

**Sample Code:**

def execute\_workflow(self, query):

    search\_results = perform\_search(query)

    self.memory\_manager.store\_memory('search\_results', search\_results)

    processed\_data = process\_data(search\_results)

    self.memory\_manager.store\_memory('processed\_data', processed\_data)

    summary = self.summarization\_agent.summarize(processed\_data)

    self.memory\_manager.store\_memory('summary', summary)

    formatted\_output = format\_output(summary)

    return formatted\_output

**6. Challenges and Solutions**

* **API Quota Management:**  
  Used environment variables and error handling to manage API limits.
* **Result Formatting:**  
  Developed a dedicated formatting tool for clean and readable UI output.
* **Extensibility:**  
  Modular design allows easy addition of new agents or tools.
* **Context Management:**  
  Each agent maintains its own memory for coherence across interactions.
* **Error Handling:**  
  All tools and agents include try/except blocks and fallback strategies.

**7. System Performance and Limitations**

**Metrics and Testing**

* **Accuracy:**  
  System returns relevant articles and accurate summaries for a variety of queries.
* **Efficiency:**  
  Fast response times for both research and summarization tasks.
* **Reliability:**  
  Robust to API errors, missing data, and unexpected input.
* **User Experience:**  
  Clear, helpful, and well-formatted output.

**Limitations**

* **Web search limited by SerpAPI quota and coverage.**
* **Summarization limited by OpenAI API quota.**
* **Extraction logic in CustomDomainExtractor may need tuning for new domains.**
* **No advanced reinforcement learning yet (planned for future work).**

**Future Improvements**

* Add more specialized agents for other domains.
* Improve domain extraction logic with NLP techniques.
* Implement advanced feedback loops and reinforcement learning.
* Enhance scalability and parallel task execution.

**8. Test Cases and Evaluation**

**Example Test Cases**

* **Research Query:**  
  "Which is better, Claude or GPT?"  
  *Expected Output:* List of articles with titles, links, and relevant snippets.
* **Summarization:**  
  Paste a long article and receive a concise summary.
* **Edge Cases:**  
  Empty input, API failure, malformed data.

**Metrics Collected**

* **Accuracy:** Relevance of search results and summaries.
* **Efficiency:** Time taken for each task.
* **Reliability:** System response to API errors and edge cases.

**9. Demonstration**

* **A 5-minute video** is provided showing:
  + System setup and usage
  + Research and summarization workflows
  + Agent interactions and orchestration
  + Custom tool in action

**10. Conclusion**

AgenticAI System demonstrates a robust, extensible, and modular approach to building agentic AI systems.  
It integrates multiple APIs (OpenAI, SerpAPI), features a custom domain extractor, and provides a user-friendly interface for research and summarization tasks.

**11. References & Acknowledgements**

* [OpenAI](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
* [SerpAPI](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
* [Streamlit](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
* [python-dotenv](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)

*For any questions, please contact*[*popli.sa@northeastern.edu*](vscode-file://vscode-app/c:/Users/popli/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)