SynthesisTalk - Collaborative Research Assistant

Final Project Report

Shahd Tarek 202202018

Salma Tawfik 202201254

Mohammed El-Sayed 202001872

Course: CSAI 422 – Applied Data Mining

Dr. Mohammed El-Beltagy

June, 2025

Table of Contents

1.System Architecture Description

SynthesisTalk is a smart, sophisticated AI research assistant platform that combines conversational AI with domain-specific tools for enhanced research workflows. It leverages advanced reasoning techniques like Chain-of-Thought (CoT) processing and ReAct (Reasoning + Acting) in addition to web search functionality, document analysis, and intelligent visualization capabilities. SynthesisTalk is built using modern FastAPI backend and a React frontend, demonstrating seamless tool integration and context-aware conversations.

The system is built using clean, two-tier architecture for better structure and maintainability:

- **Frontend:** A responsive React-based single-page app that gives users a smooth and engaging interface for their research tasks.
- **Backend:** A FastAPI service that handles all the logic—from managing AI reasoning and running tools to organize user data.

1.1 Backend Overview

The backend is implemented in the **`/backend`** directory and follows hierarchical architecture:

1. **Core Initialization:**
   - **main.py -** application initializer for FastAPI, initializing CORS policies, route registration, and global request-processing middleware.
2. **Routing Layer:**
   - **routers/ -** includes route handlers for core endpoints such as `/chat` (message handling) and `/tools` (summarize, visualize, search, etc.).
3. **Service Layer:**
   - **reasoning.py -** Implements advanced LLM reasoning techniques like Chain-of-Thought (CoT) and ReAct. It handles prompt creation, LLM communication, and structured output parsing.
   - **context_manager.py -** Keeps conversations context-aware by managing recent chat history.
   - **document_parser.py** Parses and extracts information from PDF and Word documents via pdfplumber and python-docx libraries.
   - **note_manager.py** - Lets users create and manage notes tied to their session.
   - **tool_manager.py** - Connects and manages all integrated tools.
   - **history_manager.py -** Handles storage and retrieval of user conversation history in conversations.json.
   - **search.py -** Provides web search capabilities using DuckDuckGo and summarizes web results in context.

- **export.py** - Turns plain text into well-formatted PDF files using ReportLab. It handles line wrapping, adds titles, and creates multi-page documents as needed—saving each file with a unique name in the exports/ folder.

4.Data Models:

- **models/schemas.py -** defines comprehensive Pydantic models ensuring type safety, request validation, and consistent API contracts across all endpoints.

5.Storage:

- The **data/** directory maintains JSON-based storage for conversation histories (conversations.json) and user notes (notes.json).

1.2 Frontend Overview

The react-based frontend is implemented in the `**/frontend**` directory and follows hierarchical architecture:

1. **Core Components:**
   - **App.jsx:** Root-level component overseeing global state, user authentication, and layout.
   - **ChatWindow.jsx:** Primary interface handling message rendering, tool selection, and user input.
   - **MessageBubble.jsx:** Intelligent message rendering with support for text, hyperlink, and data visualization.
2. **Specialized Panels:**
   - **ContextPanel.jsx:** Live rendering of conversation metadata, topic management, and source management.
   - **NotesPanel.jsx:** Sticky note-taking space with CRUD operations.
   - **DocumentUploader.jsx:** File upload space with progress display.

2. Implementation Details for Reasoning Techniques

2.1 Chain-Of-Thought (CoT) Reasoning

The Chain-of-Thought implementation guides the LLM through a series of ordered problem-solving steps. Instead of jumping directly to a solution, the system prompts the AI to think step by step, generating organized and coherent answers.

To make it more accurate, the platform includes a self-correction mechanism. After putting up an initial response, the system inspects whether the response is incomplete or not clear. If not of the mark, the AI attempts to edit its response twice—each time with more detailed, explicit reasoning.

Key Features of CoT:

- Promotes logical, step-by-step reasoning.
- Automatically checks for completeness and clarity.
- Refine responses through up to two iterative improvements.
- Considers recent chat history for enhancing context integration.

2.2 ReAct (Reason + Act)

The ReAct process combines reflective thought with real action through embedded tools. Each reason cycle has four steps:

1. Thought – The AI reflects on what it needs to take into account.

2. Action – It chooses and performs an appropriate tool (like Search, Clarify, or Summarize).

3. Observation – LLM interprets the result from the tool.

4. Final Answer – It generates a well-informed answer based on those results.

2.3 Context Management

To create adequately coherent and relevant conversation, SynthesisTalk uses a intelligent ContextManager that intelligently determines which parts of the chat history to use at any given time. This is especially important for instrument usage and long multi-turn conversation where the AI needs to "remember" what is going on.

Smart Context Selection

For tools like web search, the system identifies high-level user messages that triggered discussions such as long questions and correlates them with the recent messages. This enables the AI to understand the current context without being overloaded by unnecessary information.

Context Preparation Pipeline

1. The system goes through any input in a multi-step process prior to sending it to the AI:
2. System Message Generation: Adds tool-specific directives that are aware of the current activity.
3. History Filtering: Selects most critical recent messages on the basis of how much they are recent and important.
4. Input Enhancement: Re-formulates the query to what is needed by the tool or model.

5. Message Formatting: Adds correct roles (user, assistant, system) to enable the AI to correctly interpret each message.

Optimized for Other Tools

-Search: Targets identifying the main topic in order to make the query more specific.

-Summarization: Focuses on the whole recent conversation to offer greater, true summaries.

-Visualization: Repeats the last AI response if there is no fresh data. It utilizes this to produce visualizations.

-General Chat: Blends new and existing context in order to facilitate natural conversation.

3.**Tool Integration Approach**

A single system for handling all tool interactions, coordinated through the tool_manager.py module. Every tool acts in the same context-aware format.

Tool Categories and Features:

1. **Information Retrieval Tools:**
   -Search – Conducts web searches with broadened queries based on the current conversation.
   -Document Upload – Processes uploaded PDF files and automatically extracts summaries.

2. **Content Processing Tools:**
   -Summarize – Enables summarizing in different formats including plain text, bullet points, and JSON.

   -Clarify – Breaks down complex topics step by step using chain-of-thought descriptions.

3. **Analysis and Visualization Tools:**

   -Visualize – Extracts key data and generates charts or graphs in a smart way.

   -QA – Answers user questions using CoT, with self-correction mechanism.

4. **Utility Tools:**

   -Export – Transcribes chat content into a clean, nicely formatted PDF.

   -ReAct Agent – Combines reasoning and use of tools in a structured, intelligent response process.

4.Challenges & Solutions

4.1 Context Loss and History Injection

The system initially suffered from context fragmentation among tool switches, which led to incoherent replies and the absence of conversational continuity.

Solution Implemented:

- Context Limiting: Restricted context injection to the recent 10 messages for maintaining relevance within token limits
- Role-Aware Formatting: Forced appropriate message role assignment with ContextManager
- Tool-Specific Context: Context preparation specific to different tool types

4.2 Web Search Rate Limits

Search Query Failures due to DuckDuckGo rate limits triggered by full context being injected. Since several of the tools such as the ReAct agent are based on web search to complete their reasoning cycles, any failure in search capability affected overall performance right away. These crashes were most often the result of requesting too many pages of results in a short time or ambiguous queries that caused retrials repeatedly.

Solution Implemented:

- Reduced Prompt Size: We minimized the amount of context passed to the search API in order to maintain minimal request load and steer clear of rate limits.
- Query Enhancement: Implemented mini LLM prompts to rephrase and improve vague queries
- Fallback Mechanisms: error-handling code to trap search failures so that the system could react appropriately without interfering with the main flow.

4.3 Uploaded Documents Memory Issues

Initially, uploaded PDFs were summarized but not saved in conversation memory. As a result, follow-up questions like failed or resulted in LLM confusion.

Solution Implemented:

- modified upload endpoint to inject summaries into chat history

4.4 UI and React Compatibility

Initially developed using Create React App, the project caused build errors and

compatibility with modern libraries.

Solution Implemented:

- Vite Migration: Full project migration from CRA to Vite for faster performance and compatibility
- Library Updates: Updated all React dependencies to latest compatible versions

5.Evaluation of System Performance

5.1 Multi-Turn Memory and Tool Chaining

The system maintains coherent conversations using a 10-message context window. Smart context selection ensures smooth transitions between tools. Follow-up questions like “What does it mean?” or “Summarize that in bullets” correctly reference previous content.

5.2 Reasoning Quality and Accuracy

Chain-of-Thought (CoT) boosts the clarity and reasonableness of responses, while ReAct agents accurately select tools and execute reasoning cycles. Context integration enhances the accuracy and relevance of produced responses.

5.3 Visualization Fidelity

Charts generated from assistant messages were labeled, color-coded, and proportionate. Chart type was appropriately chosen based on dataset size.

5.4 Response Time & Latency

All tool calls returned results within 5-10 seconds on average, even during ReAct loops. Visual and PDF tools were slightly slower due to rendering.

6. Conclusion

SynthesisTalk brings together advanced AI reasoning and practical research tools in a seamless way, creating a smart and reliable assistant for research tasks. By combining step-by-step thinking (Chain-of-Thought) with action-driven responses (ReAct), and backing it all with smart context handling, the platform is able to hold meaningful, multi-turn conversations while smoothly using the right tools at the right time