# **Deep Research Report: Personalized AI Research and Coding Agent**

## **Overview**

This project aims to design and implement a personalized AI research and coding assistant capable of ingesting a user's evolving corpus of markdown notes, Word documents, and Python code files. The assistant uses advanced agentic workflows to:

1. **Ingest, semantically understand, and organize research content** using vector embeddings and knowledge graphs.
2. **Answer complex research queries** by planning, retrieving, reasoning, and synthesizing information.
3. **Write, test, and document Python code** based on prompts, retrieved examples, and user goals.
4. **Interface through a Streamlit UI** for conversational use.

Unlike traditional RAG systems, this assistant executes multi-step reasoning and code workflows, optionally verifying and debugging code before presentation.

## **System Goals**

1. **File Ingestion and Corpus Building** Load .md, .docx, and .py files into a structured vector and knowledge graph database.
2. **Hybrid Retrieval** Use both vector similarity search and logical graph querying to cover semantic and structured reasoning.
3. **Multi-LLM Support** Easily switch between GPT-4, Claude, Gemini, and DeepSeek models.
4. **Deep Research Workflow** Plan, retrieve, synthesize, verify, and produce structured research reports.
5. **Code Workflow** Generate code, test it, and revise until working. Document with comments or markdown sections.
6. **Streamlit UI** Conversational interface with mode selection (research or coding).

## **Architecture**

* **Vector Store**: FAISS or Chroma, populated with embedded text/code chunks.
* **Knowledge Graph**: In-memory graph using LangChain or Neo4j with entity/triple extraction.
* **LLM Router**: Determines research vs. coding mode.
* **LangGraph State**: Manages plan, retrievals, context, code iterations.
* **Tools**: search, graph\_query, run\_python, final\_answer, code\_debug.
* **UI**: Streamlit chat interface, mode toggle, and optional document upload.

## **Components**

### **File Ingestion**

* Parse .docx using python-docx, .md via standard I/O, .py using AST splitting.
* Chunk into ~500 tokens with metadata (source file, section header).
* Store embeddings with OpenAI/text-embedding-ada-002 or CodeBERT.

### **Vector + Graph Indexing**

* Create two retrieval pathways: similarity and logic.
* Use LLMs (e.g., GPT-4) to extract entities/triples for graph (e.g. function foo() in file bar.py).

### **LLM Layer**

* Configurable abstraction for:  
  + openai:gpt-4
  + anthropic:claude-3
  + google:gemini-pro
  + deepseek:chat

### **LangGraph Workflow**

**Research Path**:

* Plan → Retrieve → Summarize → Validate → Format Report

**Coding Path**:

* Retrieve → Generate Code → Run → Debug (loop max 3) → Document Code

### **Tools**

* run\_python: Exec environment with output capture
* final\_answer: Markdown report formatter
* debug\_code: Takes error and context, proposes fixes

## **Streamlit UI Design**

* Chatbox for user query
* Dropdown for LLM model
* Toggle for Research or Coding mode
* Output display with markdown rendering and copy-to-clipboard for code

## **Testing Protocol**

To test the agent:

1. Input this research report as a .md or .docx file to the corpus.
2. Ask the agent:  
   * "Produce a self-contained research report describing this project."
   * "Generate the code to build this agent including the UI."

Success = The agent produces:

* A new (non-identical) but logically equivalent and correct report.
* Functional, runnable code that matches or improves on the implementation.