AI Agent Architecture Document

Project: Academic Study-Guide Agent
Author: Rishabh Kumar

1. Manual Task & Agent Goal

Manual Task: The selected manual task is the process of reviewing long, unstructured university lecture notes (in PDF format) and manually converting them into two distinct study aids: (1) concise, hierarchical short notes, and (2) a visual mind map that outlines the key concepts and their relationships.

Agent Goal: The agent's goal is to fully automate this process. It takes a lecture PDF as input and produces two outputs: a clean notes.md file (short notes) and a mind_map.png image (visual mind map).

2. Agent Components

The agent is built using a "Planner" and "Executor" (tools) architecture. The "Planner" is the main agent script that orchestrates a fine-tuned model (the "Brain") and a set of custom tools (the "Hands").

A. Planner / Orchestrator (agent.py)

This is the main Python script that manages the entire workflow. It reasons about the overall goal, creates a multi-step plan, and then executes that plan by calling the other components in sequence.

B. Fine-Tuned Model: The "NoteExtractor" (The "Brain")

This is the core intelligence of the agent, fulfilling the **mandatory requirement** for a fine-tuned model.

- **Base Model:** google/gemma-2b-it
- **Fine-Tuning Method:** Parameter-Efficient Fine-Tuning (PEFT) using **LoRA** (Low-Rank Adaptation).
- **Purpose:** The model is fine-tuned for a single, specialized task: to read a block of unstructured lecture text and convert it into a **strict, hierarchical JSON object** that represents my personal note-taking style.

C. Custom Tools (The "Hands")

These three Python scripts act as the "Executors" or "Hands" of the agent, fulfilling the **optional requirement** for "custom Tools."

1. **pdf_extractor.py (Tool 1):**
    - **Input:** A string path to a PDF file (e.g., my_lecture.pdf).

    - **Action:** Uses the PyMuPDF library to extract all digital text from the PDF.
    - **Output:** Saves the raw text to a .txt file (e.g., temp_raw_text.txt).
2. **markdown_generator.py (Tool 2):**
    - **Input:** The JSON output file from the fine-tuned model (e.g., temp_model_output.json).
    - **Action:** Recursively parses the JSON's hierarchy.
    - **Output:** Writes a human-readable, formatted Markdown file (e.g., FINAL_NOTES.md) with nested bullet points.
3. **mindmap_generator.py (Tool 3):**
    - **Input:** The *same* JSON output file (e.g., temp_model_output.json).
    - **Action:** Recursively parses the JSON and generates a .dot file, a text-based representation of a graph. It then calls the Graphviz command-line utility (dot) to render this file.
    - **Output:** A visual mind map image (e.g., FINAL_MIND_MAP.png).

3. Interaction Flow (Reason, Plan, Execute)

The agent follows a clear "Reason, Plan, Execute" loop, managed by the main agent.py script.

1. **Reason:** The agent is given a target (my_lecture.pdf). The goal is to produce notes.md and mind_map.png. The agent reasons that this requires a multi-step process: PDF-to-Text, Text-to-JSON, JSON-to-Notes, and JSON-to-Map.
2. **Plan:** The agent's plan is as follows:
    - **Step 1:** Call pdf_extractor.py to convert the PDF into raw text.
    - **Step 2:** Call the fine-tuned NoteExtractor model with the raw text to get the structured JSON.
    - **Step 3:** Save the model's output to a temporary JSON file.
    - **Step 4:** Call markdown_generator.py on the JSON file to create the short notes.
    - **Step 5:** Call mindmap_generator.py on the JSON file to create the mind map image.
3. **Execute:** The agent.py script uses Python's subprocess module to execute each step in the plan sequentially. It handles the flow of data, saving the output of one step to be used as the input for the next, fully automating the manual task.

4. Models Used & Reasons for Choices

A. Base Model: google/gemma-2b-it

I chose gemma-2b-it as the base model because it is a powerful, instruction-following model that is small enough to be fine-tuned quickly on a free Google Colab GPU. Its non-gated (open) access also simplifies development.

B. Fine-Tuning Target: Adapted Style & Improved Reliability

This is the most important choice for the project and fulfills the **mandatory requirement** to

justify the fine-tuning.

A general-purpose, pre-trained model (like base Gemma or GPT-4) is **not suitable** for this task for two reasons:

1. **Lack of "Adapted Style":** A general model can summarize, but it cannot know *my personal, specific way* of structuring notes. It doesn't know what I consider a main_topic versus a detail or a child topic. The *only* way to teach the AI my personal mental model is to fine-tune it on examples of my style.
2. **Lack of "Improved Reliability":** My custom tools (markdown_generator.py and mindmap_generator.py) **require** a perfectly valid, strict JSON format as input. A general model may "hallucinate" and add extra text (like "Here is your JSON...") or make formatting errors (like a missing comma). This would break the entire agent pipeline.

By fine-tuning the model on 27 examples of *my specific style* (the dataset.jsonl file), I specialized it for this one task: **reliably converting unstructured lecture notes into my personal, machine-readable JSON format.**