**High-Level Overview of the Perplexia AI App Structure**

The Perplexia AI application is designed with a modular structure to support the staged development of an AI assistant. At its core, the design cleanly separates foundational logic (chat handling and tools) from week-specific implementations.

## **Project Structure**

**Directory Layout:**

perplexia\_ai/

├── core/

│ ├── \_\_init\_\_.py

│ └── chat\_interface.py → Core chat interface definition

├── tools/

│ ├── \_\_init\_\_.py

│ └── calculator.py → Calculator utility functions

├── week1/

│ ├── \_\_init\_\_.py

│ ├── factory.py → Factory that returns the appropriate part implementation

│ ├── part1.py → Query understanding logic

│ ├── part2.py → Basic tools logic

│ └── part3.py → Memory and context handling

├── app.py → Main application logic and Gradio setup

└── \_\_init\_\_.py → Package initializer

**Core Concept: Chat Interface Design**

### **ChatInterface: The AI Assistant Blueprint**

The ChatInterface is an abstract base class that all chat implementations inherit from.

**Two key methods:**

* initialize(): Called during setup. Used to load models, tools, and memory modules.
* process\_message(message, chat\_history): The core method that processes user input and returns a response based on current and past interactions.

**In Week 1**, the following implementations are built on top of this interface:

* **QueryUnderstandingChat (Part 1):** Focuses on classifying user queries.
* **BasicToolsChat (Part 2):** Introduces simple tool use (e.g., calculator).
* **MemoryChat (Part 3):** Adds support for basic memory and contextual tracking.

## **System Flow Overview**

1. The user starts the application by running run.py with a --mode argument (e.g., part1, part2, or part3).
2. This mode is mapped to a Week1Mode enum and passed to a factory method that instantiates the corresponding ChatInterface (one of three parts).
3. The selected chat implementation initializes its required components—LLMs, tools, or memory.
4. A Gradio web interface is launched, providing a chat interface for the user.
5. As the user sends messages, Gradio calls the respond() function, which delegates handling to the process\_message() method. This interaction loop continues until the Gradio server is stopped.

**A diagram is attached to appreciate the flow**

