**1. Problem Statement**  
You need to build an AI‐powered chatbot that:

* **Profiles investors** by classifying them into one of six risk categories (A–F).
* **Extracts** a fixed set of risk‐determinant parameters (e.g., volatility tolerance, investment horizon, loss mitigation strategy) through conversation.
* **Ensures full explainability**, so for every parameter it outputs both the chosen value and a natural‐language justification.
* **Does not itself decide** the final risk category, but hands off the collected parameters to a separate, transparent rule engine for classification.

**2. Solution Approach**

* **Hybrid Agent + Rule Engine**
  + **Agent’s Role:**
    - Ingest full structured context (customer master data, portfolio, asset allocation, trade history, behavior patterns).
    - Identify which parameters are still missing.
    - Ask targeted follow-up questions, and for each answer **infer** the parameter’s value **plus** a concise reasoning statement.
    - Return a complete parameter map of {parameter: {value, reasoning}}.
  + **Rule Engine’s Role:**
    - Take the fully-populated parameter map.
    - Apply a deterministic, auditable weighting or scoring table to compute the final risk category A–F.
    - Optionally produce a probability distribution across categories for downstream use or advisor review.
* **Explainability:**
  + All decision logic for risk classification lives in your rule engine (easy to audit/tweak).
  + All interpretive work (mapping a user’s words/behavior onto parameter values) is handled by the LLM agent, which also generates the “why” text accompanying each parameter.

**3. Implementation Plan**

| **Phase** | **Tasks** |
| --- | --- |
| **A. Data & Parameter Schema** | 1. Define the complete list of required parameters (e.g., volatility\_tolerance, investment\_horizon, loss\_mitigation, derivative\_exposure, etc.).  2. Design a JSON schema for the agent’s final output:```json |
| { |  |
| "volatility\_tolerance": {"value":"High","reasoning":"…"}, |  |
| "investment\_horizon": {"value":"Long-term","reasoning":"…"}, |  |
| … |  |
| } |  |

| \*\*B. Agent (LangChain + OpenAI)\*\* | 1. Create prompt templates that:<br> - Load the structured customer data.<br> - Ask only for missing parameters.<br> - Enforce JSON output of `{parameter, value, reasoning}`.<br>2. Implement a LangChain “tool” or OpenAI function for parameter extraction:<br> ```python

@tool

def extract\_parameter(context: Dict, user\_response: str) -> Dict:

# returns {"parameter": ..., "value": ..., "reasoning": ...}

1. Wire up a memory component to track which parameters are still pending. |  
   | **C. Rule Engine (Python)** | 1. Define weight mappings for each parameter option. 2. Write a scoring function that sums weights and maps total score to category A–F. 3. (Optional) Compute category probabilities. |  
   | **D. Orchestration & API** | 1. Build an MCP/Flask/FastAPI server: - Endpoint to start a session with full user JSON. - Endpoint to process each chat turn, calling the LLM agent and/or rule engine. 2. Store session state (parameters filled, chat history) in Redis or in-memory store. 3. Log all agent outputs (parameter + reasoning) and final classification for audit. |  
   | **E. UX & Delivery** | 1. Integrate with frontend: stepper UI that displays each question + the agent’s reasoning in real time. 2. On completion, hand off the final risk category to downstream systems or advisors. 3. Provide an “Explain My Profile” view showing each parameter’s value and justification. |

This architecture gives you:

* **Complete explainability** at the parameter level.
* **Regulatory compliance** via a transparent, auditable rule engine.
* **Adaptive, intelligent conversation** powered by OpenAI and LangChain.
* **Flexibility** to refine rules or parameters without retraining the LLM.