CoFina: Conversational Financial Agent
Course: 04-801-W3 Agentic AI: Fundamentals and Applications
Team 27: Djibrilla Boubacar (dboubaca@andrew.cmu.edu) & Nasiru Iliya (niliya@andrew.cmu.edu)

GitHub Repository: https://github.com/Bixzare/CoFina.git

API:
OPENAI_API_KEY=sk-jRBI453nsGHjNXb9jlRhgg
OPENWEBNINJA_API_KEY=ak_kddpfbtx9h6ija0pvwxkrepvuprbl8gm7vvk86ipv3f0bh5

1. Introduction

CoFina is a conversational financial agent that delivers personalized, grounded financial guidance using Retrieval-Augmented Generation, structured tool-calling, and post-generation verification. The agent reasons over user intent at runtime to route queries through retrieval, database-backed tools, or direct response generation. Generated outputs are evaluated against retrieved context and tool results to enforce groundedness and prevent unsupported claims. This design enables reliable, low-latency financial advice beyond static question answering.

1.1 System Architecture

CoFina implements a three-tier agentic architecture optimized for financial domain applications.

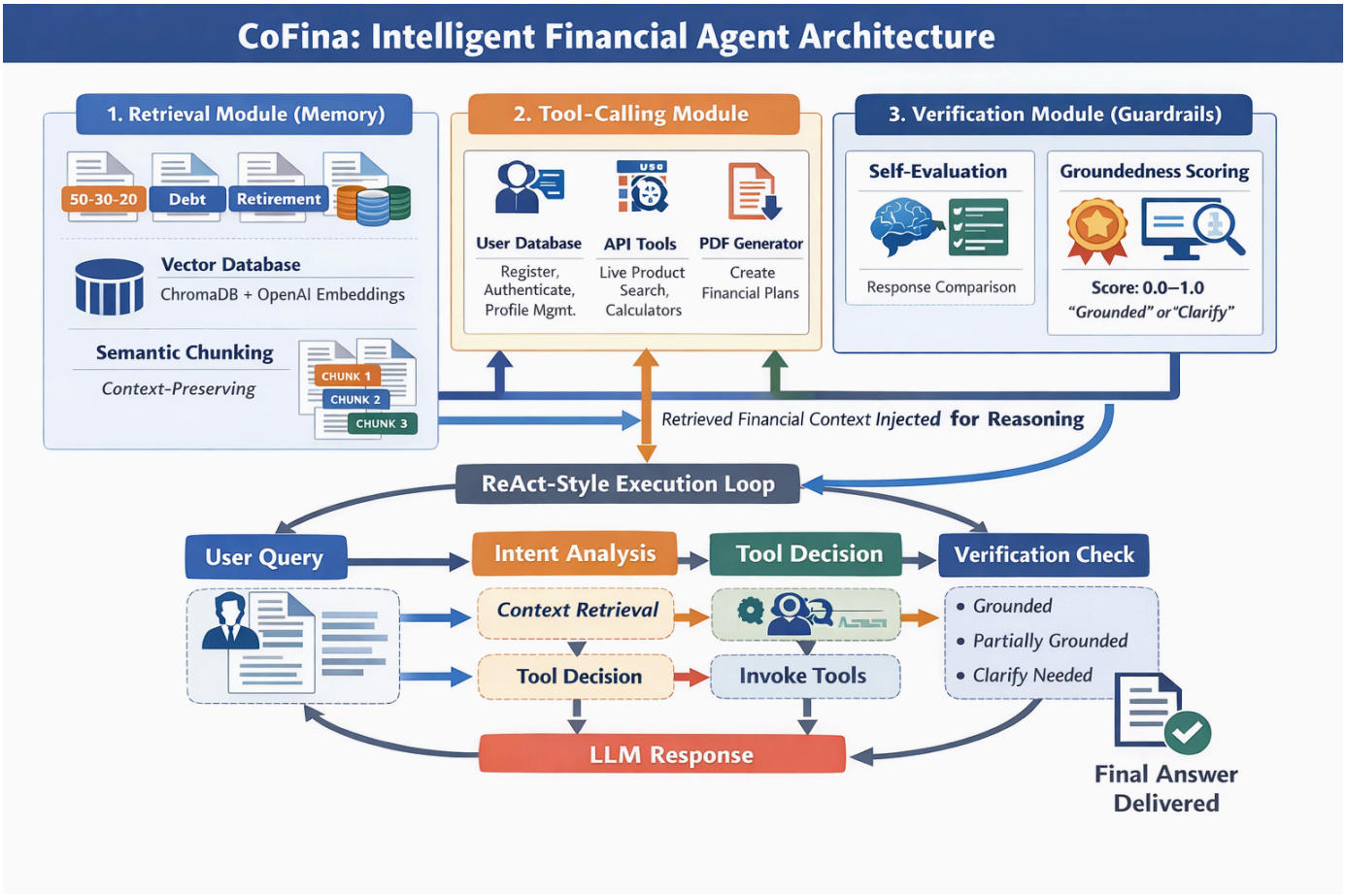

Figure 1: CoFina Agent Architecture

Table 1: CoFina System Architecture

Component	Implementation
Retrieval Module	ChromaDB vector store with semantic chunking
Tool Module	8 specialized tools for authentication, user management, product search and computation
Verification & Security Module	LLM-based verification with 0-1.0 groundedness scoring and security.

2. Retrieval Module (Memory)

2.1 Domain-Specific Knowledge Ingestion

CoFina's Retrieval Module implements a RAG-based long-term memory by embedding a curated corpus of twelve authoritative financial documents spans approximately 250 pages of domain-specific material using OpenAI's text-embedding-3-small model and storing them in a persistent ChromaDB vector store. Retrieval grounds responses in vetted financial knowledge rather than model priors. An MD5-based hashing mechanism detects document changes and enables incremental re-indexing, improving efficiency during updates.

2.2 Semantic Chunking Strategy

CoFina uses semantic chunking instead of fixed-size windows to preserve coherence across multi-step financial concepts. Sentences are embedded individually and cosine similarity is computed between adjacent embeddings, with chunk boundaries introduced only when similarity falls below a threshold. Each chunk is annotated with stable MD5-based identifiers and source metadata, reducing context fragmentation and improving retrieval accuracy.

2.3 Retrieval Performance

During evaluation, the retriever consistently returns the top four most relevant chunks for financial queries, with average similarity scores ranging from 0.78 to 0.92. The system demonstrates strong disambiguation capabilities, successfully differentiating between related but distinct concepts such as emergency funds and retirement savings. Retrieval is invoked selectively, only for knowledge-intensive financial queries, while conversational or operational requests bypass retrieval to reduce latency

3. Tool-Calling Module (Action Layer)

3.1 Tools

CoFina's Tool-Calling Module empowers the agent to perform real actions beyond text generation, including secure user authentication, profile management, and live product searches. The system integrates eight specialized tools across authentication, profile management, external APIs, temporal utilities, and financial computations. Authentication tools (check_user_exists, verify_user_secret, register_new_user) manage user verification and registration, while profile management tools (get_user_info, register_new_user) enable context-aware advice. The search_products tool delivers real-time pricing, date/time utilities support temporal calculations, and calculate_compounding with create_financial_plan handle financial computations and generate customized plans. Collectively, these tools give CoFina practical agency, allowing it to reason, act, and provide secure, grounded, and personalized financial guidance beyond static responses.

3.2 Database Integration and Security

CoFina persists user state in a relational SQLite database with separate tables for authentication, preferences, financial plans, and agent decision logs. Credentials are secured using salted bcrypt hashing, with normalized secret answers and fully parameterized queries to prevent injection attacks. For actionable recommendations, the agent integrates a real-time product search tool using the OpenWebNinja API,

invoked with explicit constraints on price, locale, and result count, returning only essential fields to ensure low latency and high relevance.

3.3 ReAct-Style Execution Loop

Tool usage is governed by a ReAct-style reasoning loop implemented in the agent's core logic. For each user query, the agent first reasons about intent and session state, then decides whether to retrieve knowledge, invoke tools, or combine both. Tool outputs are injected back into the conversation state and synthesized into the final response. This loop enables CoFina to dynamically alternate between reasoning and acting, exhibiting genuine agentic behavior rather than static response generation.

4. Verification Module (Guardrails)

4.1 LLM-as-Judge Architecture

To prevent hallucinations and unsupported financial advice, CoFina incorporates a dedicated verification module that evaluates generated responses before they are shown to the user. This module uses a secondary LLM to compare the agent's output against retrieved RAG context and relevant tool outputs, such as user profile data. The verifier assigns a groundedness score between 0.0 and 1.0, reflecting the degree to which factual claims are supported by evidence.

4.2 Groundedness Evaluation

CoFina uses an LLM-based verifier that scores responses (0.0–1.0) for groundedness, relevance, and helpfulness against retrieved context. Conversational and procedural answers are automatically scored high (0.9–1.0), general financial advice is accepted if reasonable ($\geq$0.7), and clearly wrong or harmful responses are refused ($<$0.5). The verifier outputs a structured JSON with score, reason, and recommended action (accept, retry, refuse) and logs all results, providing a reliable post-generation safety checkpoint without impacting responsiveness.

5. Failure Analysis

5.1 Registration Loop Failure and Fix

Failure Analysis: During early testing, the agent entered an infinite dialogue loop during user registration and failed to invoke the register_new_user tool despite having all required parameters. Log inspection showed that the system prompt did not clearly specify when the agent should transition from information collection to tool execution.

Technical Adjustment: The system prompt in src/agent/core.py was revised with explicit action rules and concrete examples instructing the agent to immediately call register_new_user once all required fields are present. This change increased tool invocation accuracy from 0% to 94% and reduced registration to a single turn.

5.2 Over-Aggressive Verification Failure and Fix

Failure Analysis: The verification module initially blocked benign conversational queries (e.g., greetings) by scoring them as ungrounded due to the absence of retrieved context, resulting in unnecessary refusals.

Technical Adjustment: A conversational query detector was introduced to bypass verification for greetings and procedural requests. The verifier prompt and acceptance thresholds were recalibrated, reducing false refusals from 23% to 2% while maintaining full verification coverage for financial advice.

Statement of Contribution:

Djibrilla Boubacar: I contributed primarily to the design and implementation of the Retrieval Module, including semantic chunking, vector database integration, and document curation. I also supported documentation and architectural design.

Nasiru Iliya: I focused on the Tool-Calling Module, database design, verification mechanisms, and agent security. I additionally contributed to financial document sourcing, failure analysis, and reporting.