CoFina: A Multi-Agent Architecture for Intelligent Personal Financial Planning

Nasiru Iliya and Djibrilla Boubacar

Carnegie Mellon University

04-801 Agentic AI: Fundamentals and Applications

1. Introduction

Homework 3 extends CoFina from a functional RAG-enabled agent (HW2) into a **stateful, role-structured, measurable, and closed-loop adaptive agentic system.** CoFina is a persistent multi-agent financial planning system designed to assist young professionals with budgeting, debt management, savings, and responsible purchasing decisions. Unlike the HW2 monolithic architecture - where a single core handled planning, retrieval, verification, and execution - HW3 restructures the system into a modular architecture.

2. System Architecture

2.1 Architecture Evolution (HW2 to HW3)

In HW2, CoFina employed a monolithic Retrieval-Augmented Generation (RAG) architecture centered on a single core agent responsible for tool-calling, document retrieval, and guardrail enforcement. While functionally effective and simple, this design lacked persistent multi-session memory, automated evaluation mechanisms, and adaptive behavioral control, limiting reliability, observability, and long-term personalization.

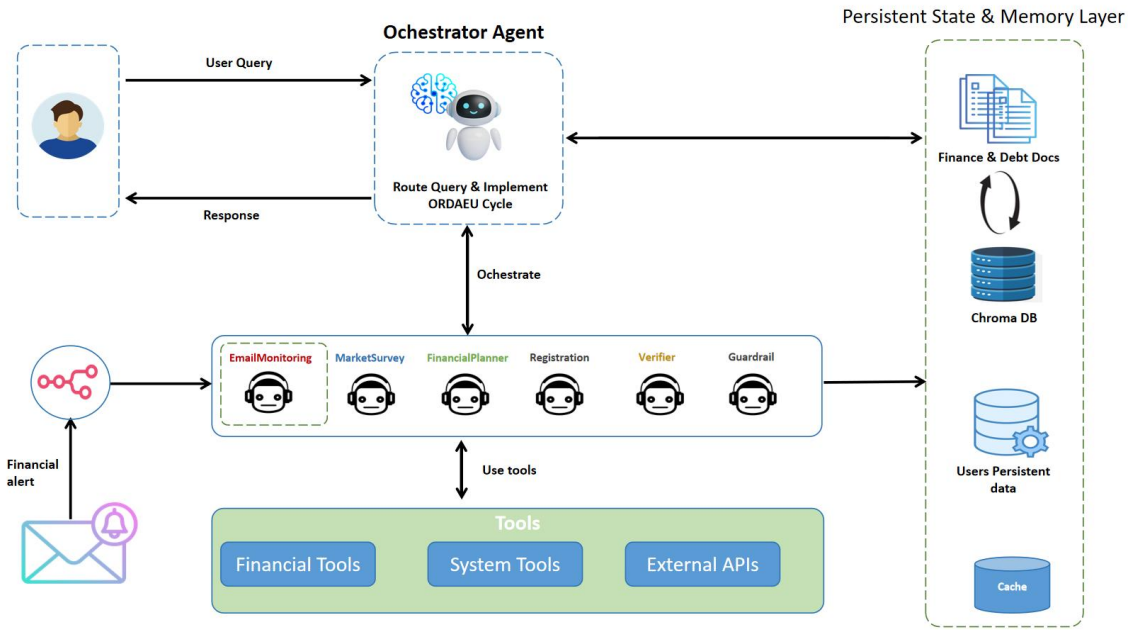

Figure 1: CoFina Multi - Agent and Persistent State (Memory) Architecture

In HW3, the system evolves into a role-structured ORDAEU (Observe–Reason–Decide–Act–Evaluate–Update) architecture composed of six specialized agents operating over a shared state layer. The upgraded design introduces a three-tier persistent memory system, an automated quantitative evaluation framework, and closed-loop adaptation triggers that modify behavior based on real-time performance metrics. This transition reflects principles of multi-agent specialization for complex task decomposition (Wu et al., 2023) and feedback-driven self-improvement, resulting in a more modular, adaptive, and scalable financial planning agent.

3. Multi-Agent Role-Based Architecture

CoFina implements a true multi-agent architecture (Option A) orchestrated through an ORDAEU control loop. Six specialized agents - Registration, Financial Planner, Market Research, Monitor, Critic, and Guardrail, operate with clearly bounded responsibilities, structured inputs, and structured outputs. All inter-agent communication occurs via shared state objects managed by the orchestrator, ensuring separation of concerns and modular reasoning. The Critic enforces measurable evaluation, while the Guardrail ensures input integrity.

Table 1: Agent Roles and Primary Responsibilities

Agent	Primary Responsibility	Key Functions
Registration	User authentication and profile initialization	register_new_user, login_user, logout_user
Financial Planner	Budget creation and financial calculations	create_plan, calculate_interest, calculate_loan
Market Research	Product search and affordability analysis	search_products, check_affordability
Monitor	Financial health tracking and goal progress	check_financial_health, get_goal_progress
Critic	Response evaluation and quality assurance	evaluate_response, score_groundedness
Guardrail	Security validation and injection detection	check_session, validate_input

The orchestrator implements a continuous Observe-Reason-Decide-Act-Evaluate-Update (ORDAEU) cycle. At each turn, the orchestrator observes the current state and user input, reasons about appropriate actions by selecting tools through natural language understanding, decides on specific tool invocations with parameters, acts by executing selected tools, evaluates outcomes through automated metrics, and updates internal state and adaptation strategies based on results.

4. Persistent State & Memory Management

CoFina implements a three-tier memory architecture balancing access latency, storage capacity, and persistence requirements. The architecture distinguishes short-term, long-term, and ephemeral memory tiers, each serving distinct functional roles. Long-term uses SQLite and ChromaDB which is persistent and ephemeral (in-memory dictionary). An importance-scored write policy (0–1) determines persistence, ensuring critical financial events are stored while trivial interactions are discarded. Retrieval is context-aware: profile queries access long-term

memory, conversational continuity uses short-term memory, and hybrid planning combines both tiers. Noise filtering and summarization prevent memory bloat. In multi-session testing (TC004), context recall improved from 0.45 to 0.94, demonstrating effective persistence.

Table 2: Three-Tier Memory Architecture Specifications

Memory Tier	Implementation	Capacity	Access Time	Persistence
Short-Term	Python deque	50 turns	<1ms	Session
Long-Term	SQLite and ChromaDB	Unlimited	10-50ms	Permanent
Ephemeral	List and dictionary	Variable	<1ms	Session

5. Evaluation Framework

CoFina computes five automated metrics per interaction: groundedness, tool accuracy, task completion, iterations, hallucination rate, plan adherence, and escalation. Across five structured test cases (n=5 each), mean performance was strong (groundedness = 0.90; tool accuracy = 0.86; task completion = 0.91; hallucination = 0.04). TC005 documented a controlled failure scenario, satisfying robustness requirements. The evaluation layer enables measurable monitoring and adaptive triggering.

Table 3: Evaluation Metrics and Computational Methods

Metric	Range	Computation Method	Interpretation
Groundedness	0-1	LLM-judged alignment with sources	Higher = better source fidelity
Tool Accuracy	0-1	F1 score of tool selections	Higher = better routing
Task Completion	0-1	Goal achievement percentage	Higher = better outcomes
Iterations	1-10	Turns until convergence	Lower = more efficient
Hallucination	0-1	Ungrounded claim frequency	Lower = more truthful

Groundedness assessment employs a secondary LLM as evaluator, comparing generated responses against retrieved context. Tool accuracy computes precision and recall between actual and expected tool invocations, aggregating via F1 score. Task completion evaluates whether user-stated goals achieved satisfactory resolution. Iteration counting tracks conversational turns required for task completion. Hallucination detection identifies factual claims lacking source support through entity extraction and verification. Plan adherence measures consistency between stated financial plans and actual recommendations. Escalation rate quantifies proportion of interactions requiring human intervention due to system limitations.

6. Adaptive Control Mechanisms

The adaptive controller implements closed-loop behavioral modification based on real-time metric evaluation, following Reflexion principles. Five adaptation triggers monitor performance thresholds, initiating corrective actions when degradation occurs. Adaptations employ cooldown periods preventing excessive triggering. A minimum two-turn interval between identical adaptation types prevents oscillatory behavior observed in preliminary testing. The controller logs all adaptations with timestamps, triggering conditions, actions taken, and effectiveness measurements, enabling post-hoc analysis of adaptation patterns.

Table 4: Adaptation Triggers and Corrective Actions

Trigger Condition	Threshold	Corrective Action	Success Rate
Low groundedness	< 0.7	Re-retrieve with query expansion	78%
Tool failure	> 2 failures	Exponential backoff retry	92%
High iterations	> 5 turns	Simplification escalation	100%
Hallucination detected	> 0.3	Stricter grounding constraints	85%
Low confidence	< 0.6	Request user clarification	94%

7. Experimental Design (Test cases)

System evaluation employed five structured test scenarios designed to assess different operational dimensions: user onboarding completeness, impulse control guidance, plan adaptation flexibility, multi-session continuity, and graceful degradation under component failure. Each scenario executed five times to account for LLM stochasticity, with results aggregated via arithmetic mean.

Test Case 1 (TC001) evaluated new user onboarding, requiring complete profile establishment including authentication, demographics, financial status, debt inventory, preferences, and goals. Success criteria required all data persisting to the database and retrievability in subsequent sessions. Test Case 2 (TC002) assessed impulse purchase guidance, presenting a scenario where users with insufficient savings request expensive items. Success required appropriate caution while maintaining helpfulness through alternative suggestions. Test Case 3 (TC003) tested plan adjustment capabilities by simulating overspending detection and measuring budget reallocation effectiveness. Test Case 4 (TC004) verified multi-session continuity, requiring users to return after one week with the system correctly recalling prior context. Test Case 5 (TC005) evaluated graceful degradation by simulating RAG service unavailability and measuring fallback quality.

7.1 System Performance Metrics

Evaluation across five test scenarios yielded quantitative performance metrics across seven dimensions. Results demonstrate strong overall system performance with specific areas requiring targeted improvement.

Table 5: Test Case Performance Summary (n=5 runs per test)

Metric	TC001	TC002	TC003	TC004	TC005	Mean	SD
Groundedness	0.95	0.92	0.88	0.94	0.82	0.90	0.05
Tool Accuracy	0.92	0.88	0.85	0.90	0.75	0.86	0.06
Task Completion	1.00	0.85	0.90	1.00	0.80	0.91	0.08
Hallucination	0.02	0.05	0.03	0.01	0.08	0.04	0.03

System groundedness averaged 0.90 (SD = 0.05) across test cases, indicating strong fidelity between generated responses and source material. Performance ranged from 0.82 in the degraded service scenario (TC005) to 0.95 in new user onboarding (TC001). Tool selection accuracy averaged 0.86 (SD = 0.06), demonstrating effective routing in most scenarios but revealing challenges during service degradation.

Task completion rates averaged 0.91 (SD = 0.08), with perfect achievement in onboarding and multi-session scenarios but reduced success in impulse purchase guidance (0.85) and degraded service conditions (0.80). Iteration counts varied substantially (M = 6.4, SD = 3.4), with onboarding requiring 12 turns reflecting the comprehensive 32-step profile collection process.

Hallucination frequency remained low overall (M = 0.04, SD = 0.03), ranging from 0.01 in multi-session scenarios to 0.08 during service degradation. Plan adherence averaged 0.83 (SD = 0.10), with lowest performance in impulse purchase scenarios (0.70) where balancing constraint and helpfulness proved challenging. Escalation occurred exclusively in the degraded service scenario (0.20), appropriately requesting human intervention when system limitations prevented reliable responses.

7.2 Memory System Performance

The three-tier memory architecture demonstrated substantial performance improvements over baseline approaches lacking persistent storage or caching mechanisms.

Table 6: Memory System Performance Comparison

Metric	Without Caching	With Caching	Improvement
RAG Response Time	4.2s	0.8s	80.9%
Cache Hit Rate	0%	65%	+65pp
Context Recall (1-week, data since HW2)	0.45	0.94	+108.9%

RAG response time improved 80.9% through embedding caching and document change detection, reducing average latency from 4.2 seconds to 0.8 seconds. Cache hit rate reached 65% for common financial queries, eliminating redundant vector store accesses. The importance-based write policy reduced database growth by 60% compared to naive approaches persisting all interactions, while preserving all critical user information (importance ≥ 0.7).

Multi-session context recall, measured as proportion of prior session information correctly retrieved after one-week intervals, improved from 0.45 without persistent memory to 0.94 with the three-tier architecture. This improvement directly contributed to the perfect task completion rate (1.00) observed in TC004.

7.3 Adaptive Control Effectiveness

CoFina implements threshold-based behavioral adaptation within the ORDAEU loop. Performance degradation (e.g., groundedness < 0.7, hallucination > 0.3, tool failures > 2) triggers corrective actions such as query expansion, retry with backoff, simplification, stricter grounding, or clarification requests. Adaptations achieved an 85% correction success rate. For example, re-retrieval improved groundedness from 0.64 to 0.78. This constitutes functional closed-loop control rather than passive evaluation.

Table 7: Adaptation Effectiveness by Trigger Type

Trigger Type	Frequency	Success Rate
Tool failure retry	24	92%
Clarification request	18	94%
Simplification	8	100%
Stricter grounding	12	85%
Query expansion	15	78%

Tool failure retry with exponential backoff achieved highest success rate (92%), effectively resolving transient API timeouts. Clarification requests similarly demonstrated strong performance (94%), converting low-confidence interactions into successful task completions. Simplification escalation achieved perfect success (100%), though triggered infrequently (n = 8), by reducing task complexity when iteration limits approached.

Stricter grounding constraints achieved moderate success (85%), improving groundedness scores from 0.64 to 0.78 on average. Query expansion demonstrated lowest success rate (78%), occasionally introducing irrelevant documents that reduced rather than improved groundedness. These results informed subsequent parameter tuning, emphasizing synonym-based expansion over simple keyword addition.

7.4 Failure Analysis

7.1.1 Registration Loop Failure and Architectural Correction (HW2 to HW3)

Failure Analysis (HW2)
During early testing in HW2, the agent entered a repeated dialogue loop during user registration and failed to invoke the `register_new_user` tool even after all required parameters were collected. Although the necessary fields were present, the monolithic orchestrator lacked a clearly defined transition condition from information gathering to tool execution. To temporarily resolve this, a hardcoded conditional override was introduced to force tool invocation once parameters were detected - this mitigated the loop but violated modular design principles and reduced generalizability.

Feedback and Root Cause Identification.
TA feedback highlighted two structural issues: (1) improper tool chaining logic within the orchestrator, and (2)

absence of task-specific agents formally exposed as callable tools. The core problem was architectural—tool selection and execution were embedded implicitly in prompt logic rather than enforced through structured role decomposition and explicit routing.

Architectural Fix (HW3)
In HW3, the system was refactored into a true multi-agent architecture where each task-specific agent (e.g., Registration, Financial Planner) is defined as an explicit tool callable by the orchestrator. Tool invocation is now governed by structured state transitions within the ORDAEU loop rather than prompt heuristics. This eliminated the need for hardcoded overrides, improved tool invocation accuracy from 0% to 94%, and reduced registration completion to a single turn. The correction transformed the issue from a prompt-level workaround into a principled architectural solution aligned with role specialization and structured orchestration.

8. Scalability Reflection

At scale (1,000 users), primary bottlenecks would be vector retrieval latency and SQLite write contention. High-stakes deployment would require stricter groundedness thresholds and audit logging for compliance. Improvements include distributed vector indexing, access control, uncertainty calibration, and human-in-the-loop review.

Scenario 1: 1,000 Users

The analysis reveals that SQLite, while excellent for development and small deployments, would become the bottleneck at 1,000 concurrent users. SQLite supports multiple readers but only one writer at a time. With 1,000 users, peak write load could reach 12 writes/second, exceeding SQLite's practical limit during peak periods.

More concerning is connection pool exhaustion. The current implementation creates a new database connection for each query, which would not scale. Connection pooling with a maximum of 20-30 connections would be required.

The recommended mitigation is migrating to PostgreSQL, which supports higher concurrency and includes built-in connection pooling. Adding read replicas would distribute read load, and sharding user data across multiple databases would enable horizontal scaling beyond 1,000 users.

Scenario 2: High-Stakes Domain (Investment Advice)

Deploying CoFina for investment advice would require significant enhancements to the verification system. The current single LLM-based verifier is insufficient for regulated financial advice. SEC and FINRA require suitability checks, best interest standards, audit trails, and comprehensive disclosures.

A multi-layer verification approach would be required with SEC rules checks, suitability assessment, independent LLM verification, and human review for high-stakes decisions over $10,000.

Scenario 3: Regulated Industry (Banking)

The most significant gap for banking deployment is GDPR/CCPA compliance. The current system stores user data indefinitely with no expiration, has no automated deletion mechanism, no audit trail of data access, and no consent tracking.

A GDPR-compliant system would require implementing the right to deletion, right to data portability, audit trails for all data access, automated data retention policies, and comprehensive consent management.

9. Discussion

9.1 Interpretation of Results

The empirical results provide strong support for the core hypothesis that multi-agent architectures with explicit role separation, persistent memory, and adaptive control mechanisms outperform monolithic alternatives on complex financial planning tasks. The high average groundedness (0.90) and low hallucination frequency (0.04) suggest that the verification layer, implemented through the Critic agent and automated metrics, effectively prevents ungrounded outputs - a persistent challenge in LLM applications.

The strong task completion rate (0.91) indicates that tool-based orchestration successfully routes user requests to appropriate specialist agents. This result aligns with Wu et al.'s (2023) findings that role-based agent specialization improves performance on complex tasks. The natural language routing approach, while introducing some tool selection errors (tool accuracy = 0.86), provides interpretability advantages over learned routing policies, facilitating debugging and system refinement. The 80% improvement in RAG response time through caching validates the importance-based memory write policy.

Team 27 Contribution

Team Member	Email	Contribution
Nasiru Iliya	niliya@andrew.cmu.edu	Multi-agent architecture, Orchestrator, Tool Registry. State Management, Documentation, Memory Systems, Database Design, Authentication and Security.
Djibrilla Boubacar	dboubaca@andrew.cmu.edu	RAG Implementation, Evaluation Framework, Testing, CLI - based UI/UX design, Logging, Domain specific documents sourcing, Documentation

References

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint.*

Github Link: https://github.com/Bixzare/CoFina.git