PROCEEDINGS OF
INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS AND NEW APPLICATIONS
https://proceedings.icisna.org/
3rd International Conference on Intelligent Systems and New Applications (ICISNA'25), Antalya, December 12-14, 2025.

ReAct Modular Agent: Orchestrating Tool-Use and Retrieval for Financial Workflows

Armando Hernández de la Vega [1], Santiago Perez [2],
Victor Sabbia Cariquiry [3]

[123] *Machine Learning Department, Brokerware, Montevideo, Uruguay*
{ahernandez@brokerware.com.uy,
vsabbia@brokerware.com.uy,
sperez@brokerware.com.uy}
[1] *ORCID: 0009-0007-6281-2038 (A. Hernández)*
[2] *ORCID: 0009-0003-0693-4461 (S. Perez)*
[3] *ORCID: 0009-0000-1757-5554 (V. Sabbia)*

***Abstract*— The financial advisory profession demands extreme precision and speed in decision-making, compounded by the complexity of modern capital markets software. This often leads to high training overhead and reduces the time financial advisors can dedicate to client relations. This paper introduces an Agentic AI Co-Pilot designed as a significant architectural advancement beyond traditional Retrieval-Augmented Generation (RAG) systems. The core framework leverages a specialized Enterprise AI Flow to orchestrate a modular, decoupled agent architecture.**

The system's central component, the Reasoning and Action Agent (RAA), which implements the ReAct (Reasoning and Acting) paradigm that executes a fusion of explicit reasoning and external tool-use. This modularity allows the agent to: (1) interpret complex natural language queries, (2) articulate an internal step-by-step plan via Chain-of-Thought (CoT), and (3) autonomously execute a sequence of decoupled, modular API tools to perform high-stakes operations. This architectural separation ensures the seamless and incremental expansion of capabilities (e.g., integrating a risk-check API or a financial market forecasting module) without the need for retraining the core reasoning model. By providing both traceability and automated execution across complex workflows, the solution aims to substantially improve operational efficiency, enhance compliance through traceable decisions, and elevate the user experience in the highly regulated financial ecosystem.

Keywords: Agentic AI, Retrieval-Augmented Generation (RAG), Chain-of-Thought (CoT), Reasoning and Action (ReAct), Financial Advisory, LLMs.

I. Introduction

The financial advisory sector, particularly in capital markets, operates under extreme pressure where speed and compliance are paramount. Financial advisors and back-office staff must manage complex software platforms to execute high-stakes operation, such as placing orders, retrieving quotes, or generating compliance reports, upload assets, reports or invoices, often under tight deadlines. This complexity results in significant operational friction: new hires face steep learning curves, experienced staff lose valuable time navigating disparate interfaces, and overall efficiency suffers. Ultimately, this reduces the time dedicated to high-value client relationship management.

Traditional Artificial Intelligence (AI) solutions, such as Retrieval-Augmented Generation (RAG) systems, have proven effective in addressing the informational needs of this sector by providing context-aware answers from proprietary data. However, RAG systems inherently fail to address the core issue of operational execution, they excel at providing information but cannot autonomously interact with transactional systems to perform actions like filling an order or updating a record. This deficiency creates a critical "information-to-action gap" in enterprise automation.

To bridge this gap, we introduce an Agentic AI Co-Pilot designed specifically for streamlining financial advisory operations. Our solution transcends the limitations of conventional RAG by integrating an explicit Reasoning and Action (ReAct) paradigm within a modular Enterprise AI Flow. This architecture enables the system to not only answer questions but also to autonomously determine, plan, and execute a sequence of actions on the underlying financial platform via a dynamic set of decoupled API tools.

This paper details the design and implementation of this agentic architecture. Our key contributions are:

1) *A Modular ReAct Framework*: We present a robust, multi-stage agentic flow that consolidates reasoning (Chain-of-Thought) and external tool-use (API calls) within a regulated environment.

2) *Decoupled Tool-Use for Incremental Capability*: We demonstrate an architecture where high-stakes operations (like order creation or symbol search) are implemented as reusable microservices, allowing the agent's capabilities to be expanded or updated without requiring the retraining the core LLM, or modify the agent's architecture.

3) *Enhanced Traceability and Compliance*: We show how the agent's explicit reasoning path (CoT) and structured action logs provide an auditable trail for every transaction, addressing a critical compliance requirement in the financial industry.

The remainder of this paper is organized as follows. Section 2 reviews the related work on Agentic AI and Retrieval-Augmented Generation (RAG). Section 3 describes the proposed methodology, including the system architecture, workflow, implementation details, and key components. Section 4 presents the evaluation of the system, outlining the test cases, performance metrics, and experimental setup used to assess the agent's capabilities. Finally, Section 5 offers concluding remarks and discusses potential directions for future research.

II. Related Work

The development of the Agentic AI Co-Pilot for financial operations draws upon three major and overlapping areas of research in Artificial Intelligence: (1) Retrieval-Augmented Generation (RAG), which provides the foundation for grounded knowledge; (2) Agentic AI Architectures, which provides the necessity for autonomous goal pursuit and decision-making; and (3) Reasoning and Tool-Use Mechanisms, which are essential for complex operational execution and auditability.

A. Retrieval-Augmented Generation (RAG) and Knowledge Grounding

Initial efforts to improve the factuality and domain specificity of Large Language Models (LLMs) focused on Retrieval-Augmented Generation (RAG) systems [1]. RAG systems fuse the parametric knowledge of LLMs with non-parametric knowledge retrieved from external sources, effectively mitigating hallucination and incorporating up-to-date information [5], [6].

In the financial domain, RAG has been effectively applied to knowledge-intensive tasks such as financial risk management, where models retrieve relevant regulatory guidelines to answer compliance questions [6]. Similarly, specialized RAG variants have emerged for highly structured data, such as time-series forecasting (e.g., FinSeer) [7], demonstrating the need for tailored retrieval mechanisms in complex financial scenarios.

However, as highlighted in the Introduction, traditional RAG architectures, including the Naïve and Advanced paradigms, are inherently limited to informational tasks (Q&A, summarization) [5], [10]. They lack the autonomy and architecture necessary to transition retrieved knowledge into transactional actions, thus creating the "information-to-action gap" that our proposed agent aims to bridge.

B. Agentic AI and Autonomous Architectures

Agentic AI represents an evolving paradigm where autonomous systems pursue complex, long-term goals with minimal human intervention [4]. These agents transcend reactive, rule-based systems by incorporating adaptability, planning, and goal-directed decision-making [4], [11].

Modern enterprise adoption often integrates this concept into Agentic RAG [5], where agents orchestrate dynamic retrieval strategies to overcome the static workflow limitations of conventional RAG [5], [10]. Architectures in this space range from Single-Agent Routers to Multi-Agent and Hierarchical RAG systems [5]. Our work contributes to this area by defining a Modular ReAct Framework specifically tailored for high-stakes financial environments, where the primary autonomous goal is not merely retrieval, but transactional execution.

C. Reasoning, Planning, and Tool Use (ReAct)

The ability of an LLM to perform complex, multi-step tasks relies heavily on the implementation of structured reasoning and external interaction mechanisms:

The Chain-of-Thought (CoT) prompting technique [2] demonstrates that providing the model with intermediate reasoning steps significantly improves performance on complex tasks, such as arithmetic and symbolic reasoning. In the context of agent development, CoT is fundamental, as it allows the agent to articulate an internal step-by-step plan for task decomposition, enhancing reliability and providing a crucial layer of auditability.

The ReAct (Reasoning and Acting) paradigm [3], [8] synergizes CoT reasoning with external Tool Use, creating an iterative loop of Thought-Action-Observation. This loop allows the agent to:

1) *Reason to Act:* Decompose the goal and select the appropriate external tool (API call) based on the internal plan (CoT).
2) *Act to Reason*: Interact with the environment (e.g., execute a search API) and receive an Observation (e.g., the data in the back field), which is then used to refine the next Thought.

Our Modular ReAct Framework extends this paradigm by embedding the action space into a dynamic set of decoupled API tools (microservices), enabling secure, high-stakes transactional execution within the Enterprise AI Flow. This architecture is explicitly designed to maximize Incremental Capability by separating the LLM's reasoning engine from the operational logic, positioning the solution as a robust, traceable, and scalable co-pilot for the highly regulated financial advisory domain.

III. Architecture, Workflow and Implementation Details, and Key Components

The agent's design is defined by a Hierarchical Planning and Execution architecture, moving beyond the limitations of the traditional monolithic ReAct framework. This structure separates the agent's responsibilities into distinct cognitive tiers, specifically allocating the roles of strategic thought (Planner), operational translation (Dispatcher), and response

generation (Synthesizer) to specialized personas within a unified, high-performance Large Language Model (LLM) architecture, powered by GPT-4o. This specialized division significantly enhances both speed and efficacy by allowing the powerful GPT-4o instance to focus exclusively on the cognitive task at hand for each role.

A. Design Rationale

Traditional monolithic ReAct agents tend to mix planning and execution within a single cognitive loop, leading to high latency and inconsistent decision boundaries. By decoupling the planning, dispatching, and synthesis processes, our framework ensures more deterministic tool execution, better state traceability, and simpler debugging. This separation mirrors cognitive models in human problem solving, where abstract reasoning, concrete execution, and final articulation are distinct yet interdependent processes. The ability for the Dispatcher to initiate parallel tool executions further reduces latency compared to sequential tool use.

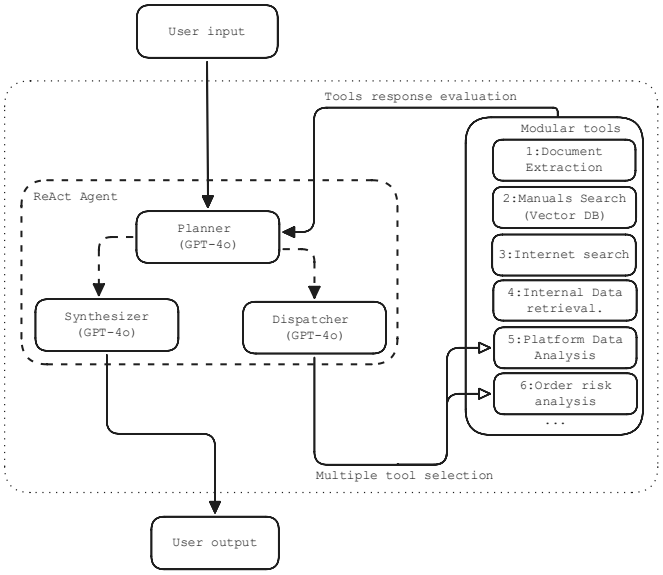

Fig. 1: Hierarchical ReAct agent architecture featuring distinct Planner, Dispatcher, and Synthesizer roles (all GPT-4o), enabling parallel tool selection and execution.

B. The Hierarchical Model: Separation of Cognitive Labor

The core of this architecture is the functional separation of the agent's cognitive workload into three primary roles, all handled by the robust GPT-4o model, specialized via distinct System Prompts. This strategic use of a unified, high-performance model differs from cost-optimization approaches and is designed to maximize task efficacy and overall throughput, leveraging GPT-4o's superior speed and reliability across all stages.

Strategic Tier (The Planner - GPT-4o): The central intelligence responsible for deep reasoning and strategy. Its role is entirely focused on analyzing the user query and conversational history, generating a viable Chain-of-Thought (CoT), and decomposing complex user queries into a defined, executable list of potentially parallel actions (current_actions). It also evaluates the results returned by the tools.

Operational Tier (The Dispatcher - GPT-4o): Acts as the efficient translation engine. Its sole purpose is to convert the Planner's strategic action list into immediate, correct actions. It generates the necessary structured tool_calls, potentially for multiple tools simultaneously, enabling parallel execution based on the Planner's directives.

Synthesis Tier (The Synthesizer - GPT-4o): Is responsible for generating the final user output. Once the Planner determines sufficient information has been gathered, the Synthesizer receives the entire conversational context, including all tool results, and crafts a concise, coherent, and rule-compliant response.

This hierarchical separation ensures that the resource-intensive task of planning is distinct from the rapid translation task of dispatching and the final generation task, thereby maximizing the agent's overall throughput and precision. The Planner's Chain-of-Thought evaluates the consolidated output from the executed tools and decides whether to generate further actions, proceed to the Synthesizer, or retry/reformulate based on the observation.

The Chain-of-Thought is also done by the planner, that evaluates the output of the dispatcher execution and decides to move on to the next step or to the synthesizer for the final answer, or to retry the task because it did not meet the requirements needed.

C. Detailed Agent Workflow

The agent operates on an iterative, state-driven loop orchestrated by LangGraph, ensuring that control always remains with the established plan.

1) *Input and Initiation*: The process begins with the Planner (GPT-4o) receiving the user query and the full conversational state.
2) *Strategic Planning*: The Planner first determines if the query requires external action based on its available tools. If so, it generates an explicit list of actions , potentially including multiple actions intended for parallel execution. If the query can be answered via internal knowledge or the necessary information is already present, the plan may only contain the 'FINALIZE' command, directing flow to the Synthesizer.
3) *Dispatch and Parallel Action:* The Planner passes the current list of action steps to the Dispatcher (GPT-4o). The Dispatcher translates these commands into potentially multiple structured tool_calls within a single AI message. These calls are then executed concurrently by the Modular Tools via LangGraph's ToolNode.
4) *Observation and Evaluation:* The results of all concurrently executed tool executions (the Tools response evaluation) are sent back directly to the Planner. This is the critical feedback loop. The Planner analyzes the consolidated observation, evaluates its relevance and sufficiency against the original goal, and updates its internal state.

5) *Refinement Loop*: If the information is incomplete or a subsequent action is required based on the evaluation, the Planner generates the next set of actions, and the process returns to the Dispatcher (Step 3).
6) Synthesis and Output: Once the Planner confirms all necessary information is gathered (often indicated by generating ['FINALIZE']), it directs the flow to the Synthesizer (GPT-4o) node. This module synthesizes the entire context and evidence into a concise response for the User output.

The Planner continuously validates tool responses against the plan's requirements. This self-corrective behavior, guided by the Chain-of-Thought and the evaluation of observations, enhances robustness in dynamic environments. The modular Planner–Dispatcher design supports efficient parallel tool execution driven by a single Dispatcher instance, optimizing for latency in information-gathering steps.

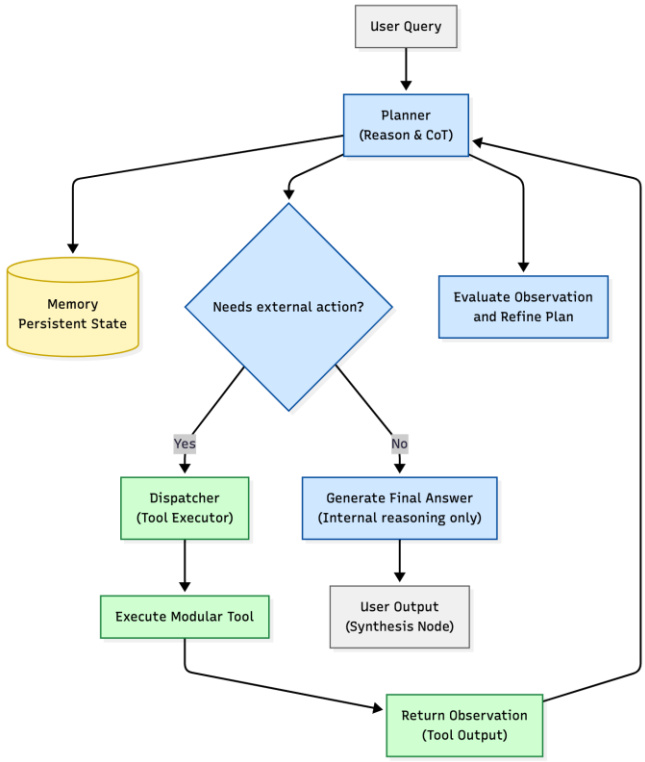

Fig. 2 Hierarchical Reasoning–Action Flow in the ReAct Modular Agent.

D. System Environment

The agent prototype was developed and tested in a Python 3.11 environment. It leverages core components from the LangChain ecosystem, specifically LangChain v0.3 for foundational elements and LangGraph v0.1 beta for workflow orchestration. All LLM interactions utilize the Azure OpenAI GPT-4o API endpoint, configured for reliability and speed. The system is designed to be exposed via FastAPI, enabling integration as a real-time API service.

E. Framework and State Management

The agent's stateful, cyclical workflow is implemented using LangGraph. This framework allows for the explicit definition of nodes (Planner, Dispatcher, Tools, Synthesizer) and edges, representing the flow of control and data as described in Section III.C. State persistence across user turns is managed via LangGraph's Checkpointer mechanism (specifically InMemorySaver during development). This ensures the messages list (containing the full conversational history, including ToolMessage results) is preserved, enabling the Planner to maintain context for multi-turn interactions and complex reasoning. The current_actions list within the state facilitates the handoff between the Planner and Dispatcher.

F. LLM Configuration and Role Specialization

The architecture employs a hybrid-model strategy to aggressively optimize the trade-off between high-level reasoning and low-latency execution. Cognitive functions are assigned to distinct LLM instances based on the complexity of their required task:

Planner and Synthesizer (High-Reasoning Roles): These functions, which demand strategic planning (CoT) and high-fidelity, rule-compliant response generation, are served by a powerful reasoning model (Azure OpenAI GPT-4o). This ensures maximum accuracy for the most critical cognitive steps.

Dispatcher (Low-Latency Role): This role is purely mechanical, responsible only for the rapid and accurate technical translation of the Planner's commands into structured tool_calls. To minimize latency and cost, this function is served by a lightweight, high-speed model (e.g., GPT-4.1-Nano), which is optimized for fast tool-calling operations.

This specialization is achieved by routing the graph to the appropriate model instance for each specific node, combined with the carefully engineered System Prompts (presented in Appendix A) that define the exact constraints for each role.

G. Modular Tools and RAG Pipelines

The agent's capabilities are extended through a set of modular tools, implemented as Python functions decorated with LangChain's @tool wrapper. This decorator automatically generates a schema (including function name, description from the docstring, and argument types) that is used by both the Planner (for strategic selection, informed by the descriptions) and the Dispatcher (for generating accurate tool_calls).

The initial toolset includes manuals_search (interfacing with Azure AI Search for Retrieval-Augmented Generation (RAG) from proprietary vector databases) and web_search (using Tavily API).

New tools (e.g., brokerware_api for interacting with platform APIs, time_series_forecasting, asset_risk_analysis) can be added simply by defining the function, decorating it with @tool, and including it in the tools list passed to the agent builder, requiring no modification to the core agent logic (Planner/Dispatcher prompts dynamically incorporate new tools).

The ToolNode from LangGraph handles the execution of both single and parallel tool_calls generated by the Dispatcher,

returning results as structured ToolMessage objects back into the agent's state for evaluation by the Planner. Tool functions include robust error handling to return informative error messages as ToolMessage content, allowing the Planner to attempt auto-correction.

IV. TEST CASES, PERFORMANCE METRICS AND EXPERIMENTAL SETUP

This section validates the agent's effectiveness by focusing on its ability to handle complexity (efficacy) and maintain low latency (speed). The agent's performance is measured to prove the efficiency gained from separating the planning and execution stages.

A. Test Cases

THese scenarios confirm that the agent successfully executes its hierarchical plan, correctly routes to the specialized tool, and adheres to the strict synthesis rules defined in the System Prompt. The Test Cases detailed below were designed to validate the robust operational capabilities required in a financial workflow.

TABLE I
TEST CASE 1: STEP PLANNING AND INTERNAL RAG PRIORITY

Parameter	Value
User query	*"What are the steps to assign a new user to a permissions group in the ComplianceX platform?"*
Tool to validate	Manuals Search (Vector DB RAG). (The primary source for internal SOPs).
Validation objective	Planning & Extraction Efficacy: Demonstrate the Planner correctly prioritizes the internal RAG source and the Final Answer module extracts specific, step-by-step instructions (not vague text), explicitly naming the platform.

Test case 1 expected execution flow:

- Planner: Plans based on query, next_steps: [manuals_seaarch: assign user permission group ComplianceX', 'FINALIZE'].
- Dispatcher: Calls Manuals
- Search.Tools: Returns Context and instructions
- Final Answer: The LLM applies the rule "If your answer includes instructions, explicitly name the platform" and synthesizes the final output.

The user output for the test case 1 was an explanation on how to assign a new user to a permissions group in ComplianceX by creating or selecting the user, assigning an appropriate role, verifying its permissions, and saving the changes, this is correct and is the standard way of assigning users in that platform.

The planner CoT (Chain of thought) was also as expected, it successfully reasoned that is a specific and not a general search we present he is reasoning of the planner and the task defined for the dispatcher:

<REASONING>The query is technical and specific about the ComplianceX platform, which seems to be part of the Brokerware X enterprise products. Therefore, the primary and most reliable source for this query is `manuals_search`*. I will execute this command to get the necessary information. If the response is sufficient, I will apply the Golden Rule and finalize. </REASONING>*

['manuals_search: steps to assign a new user to a permissions group in ComplianceX']

Then, after the tool was successfully executed, the planner successfully identified that the answer provided useful and sufficient information to manage users and permissions in the ComplianceX platform, the reasoning of the planner ir presented next:

<REASONING>The internal manual search provided sufficient information on how to manage users and permissions in the ComplianceX platform (GeneXus Access Manager). I am applying the Golden Rule and finalizing to meet the minimum latency goal.</REASONING> [FINALIZE]

TABLE II
TEST CASE 2: EXTERNAL KNOWLEDGE SEARCH (WEB) AND ESCALATION PROTOCOL

Parameter	Value
User query	*"What new regulations regarding crypto assets were published this week that impact brokerages?"*
Tool to validate	Web Search (Tavily).
Validation objective	Demonstrate the Planner correctly chooses the external source. If the search results are too vague/non-existent, the agent must avoid hallucination and strictly follow the protocol to recommend a support ticket/developer contact.

Test case 2 expected execution flow:

- Planner: Plans based on current affairs requirement, next_steps: [web_search: new crypto regulations brokerages this week','FINALIZE'].
- Dispatcher: Calls Web Search (Tavily).
- Tools: Returns Context
- Final Answer (Synthesis): The LLM identifies the context as insufficient/too vague. It then applies the "Context Insufficient" rule from the System Prompt.

The user output for the test case 2 was an explanation that this week, key global regulatory updates affecting brokerages and crypto assets include the U.S. SEC clarifying compliance for digital asset engagement, the U.K. lifting its ban on crypto-based ETNs, and the EU planning tighter oversight of crypto exchanges and operators, this response was what was expected.

The planner CoT also reasoned to choose an external web search tool, we present next the reasoning of the planner before and after calling the dispatcher and executing the tools.

<REASONING>The query refers to recent regulations regarding crypto assets that impact brokerages. Since this is general and current information, the most appropriate tool is

`web_search`. *However, there is no need for parallelism, as the primary source would not cover external regulations or recent news. I will proceed with a public web search. </REASONING>*

['web_search: new crypto asset regulations impacting brokerages this week']

<REASONING>Web Search successfully retrieved relevant information about new crypto regulations impacting brokerages this week. The observation is sufficient to answer the query, and no further steps are needed. Applying the Rule of Oro, I finalize to ensure minimal latency. </REASONING> [FINALIZE]

Both test cases were executed in approximately 7 seconds, showing that the hierarchical modularization of the agent does not negatively impact on latency.

B. Performance Metrics

The agent's performance is measured primarily by three key metrics designed to directly reflect operational safety, cognitive efficiency, and service quality within the highly regulated financial domain. These metrics serve as the quantitative basis for validating the advantages of the hierarchical model.

Execution Latency (EL) captures the total runtime, measured in seconds (s), averaged across all test executions per completed query. Operationally, EL is defined as the time elapsed from the reception of the User Input to the generation of the final User Output. This metric is paramount as it measures the agent's throughput and responsiveness. Low latency is critical for real-time decision support in capital markets, directly justifying the architectural choice of separating the resource-intensive Planner from the high-speed Dispatcher module.

Planning Efficiency (PE) reflects the internal control loop's robustness and the agent's capacity to reach the objective using the minimum necessary number of steps. PE is quantified by the proportion of executions that satisfy the predefined process efficiency criteria which explicitly penalize iterations that are redundant or erroneous. PE serves as a proxy for the system's cognitive intelligence, ensuring the agent does not incur unnecessary computational costs or time delays due to failures in planning logic or syntax.

Synthesis Quality (SQ) measures the semantic accuracy and the adherence to specific business rules in the final response. This metric is fundamental for the system's regulatory conformity (compliance). SQ is calculated based on the proportion of outputs that meet the output quality criteria. Its high importance validates the necessity of the dedicated Final Answer module to function as the critical last-mile quality control layer, ensuring that all information is factual, complete, and strictly adheres to necessary formatting standards (e.g., prohibition of URLs and ambiguity).

C. Criteria of Success (CoS) for performance metrics

The classifications of an execution as "successful" is based on a series of binary criteria (success/fail) that measure both the final output quality and the efficiency in the agent's cognitive process.

TABLE III
COS FOR METRICS

Metric	Operational success criteria	Success Type
Synthesis Quality (SQ)	(Cos1) Completeness & Accuracy: The final output contains all requested facts and is semantically correct (backed by the corresponding ToolMessage).	Output Quality
	(Cos2) Business Compliance: Strict adherence to format (no URLs, non-ambiguous) and professional protocol (explicitly names the platform).	Business Compliance
	(Cos3) Factual Integrity: Absence of internal contradictions or hallucinations (all facts must be traceable to a ToolMessage).	Factuality
Planning Efficiency (PE)	(Co4) Process Efficiency: No execution of redundant or unnecessary tool calls (e.g., web search after internal search success).	Process Efficiency
	(Cos5) Loop Robustness: Successful resolution of the query without resorting to indefinite retry loops or fallback mechanisms.	Process Robustness
	(Cos6) Action Syntax Integrity: The Planner generated valid action commands, and the Dispatcher translated them without syntax errors or invoking non-existent tools.	Cognitive Reliability

The SQ score is calculated as the proportion of executions that meet all success criteria for output quality

$$\mathrm{SQ} = \frac{N_{\text{successful (CoS 1-3)}}}{N_{\text{total}}}$$

The PE score is calculated based on the proportion of failures to meet the process efficiency criteria:

$$\mathrm{PE} = 1 - \frac{N_{\text{failures (CoS 4-6)}}}{N_{\text{total}}}$$

where N_{failures} is the total count of instances where the agent failed to adhere to the process efficiency criteria (e.g., executing unnecessary tool calls, exhibiting syntax errors, or entering indefinite retry loops).

Execution Latency (*EL*) captures the total wall-clock runtime, measured in seconds (s), averaged across all completed queries. The metric is defined as the time elapsed from the reception of the User Input until the generation of the final User Output. EL serves as the direct validation of the architectural efficiency, demonstrating the reduction of cognitive and computational overhead achieved by the strategic separation of the Planner and Dispatcher roles.

The next subsections present the empirical validation of the proposed Hierarchical-Hybrid architecture against a standard Monolithic ReAct agent baseline. The objective is to quantify the performance gains across three key metrics: Execution Latency (EL), Synthesis Quality (SQ), and Planning Efficiency (PE).

A. Experimental setup

We generated a robust dataset using an automated evaluation script. The testbed comprised 10 "Strategic Queries" (see Appendix B) designed to test critical financial workflows, including RAG retrieval, external data lookup, and no-tool interactions. Each query was executed n=50 times for both agent configurations (500 runs per agent, 1,000 total). To ensure stability and adhere to the 300 RPM API rate limit, all tests were executed sequentially (Batch=1) with a 3-second pause between tasks.

B. Agent Configurations

We configured two distinct agents, both orchestrated by LangGraph, for this comparative test:

Monolithic ReAct (Baseline): This agent utilized a single Azure OpenAI GPT-4o deployment instance for all cognitive tasks (reasoning, tool selection, and synthesis) within a standard, monolithic ReAct loop.

Hierarchical-Hybrid ReAct (Proposed): This agent employed the specialized, multi-deployment hybrid architecture described in Section III.F. This configuration routes High-Reasoning roles (Planner, Synthesizer) to a GPT-4o model, while routing the Low-Latency role (Dispatcher) to a GPT-4.1-Nano model.

This hybrid, multi-deployment approach routes the sequential API calls of a single query (often 6+ calls) across independent Azure endpoints, thereby mitigating the 300 RPM API rate-limit bottleneck inherent in the monolithic design.

C. Automated Evaluation Framework (LLM-as-a-Judge)

We implemented an "LLM-as-a-Judge" framework to assess performance at scale. For each of the 1,000 test runs, the script logged a JSON object containing the latency, final_answer, reasoning_log (CoT), and the complete execution_trace. This JSON object was then programmatically evaluated by two specialized "Auditor" agents, separate GPT-4o instances configured for specific scoring tasks:

Synthesis Quality (SQ) Auditor: This judge evaluated the final_answer against the original query. It verified factuality (CoS 1), adherence to business protocol (CoS 2: no URLs, no vagueness), and internal consistency (CoS 3), producing a boolean sq_pass result for each run.

Planning Efficiency (PE) Auditor: This judge evaluated the trace_log and reasoning_log. It first checked for process robustness (CoS 5), failing any run with a critical execution error. It then judged cognitive efficiency (CoS 4), failing runs that performed redundant tool calls (e.g., violating the "Gold Rule"). This produced a boolean pe_pass result.

D. Quantitative Metrics and Discussion

TABLE IV
EXPERIMENTAL RESULTS

Metric	Monolithic	Hierarchical	Difference
Execution Latency (EL)	9.26 s	8.61 s	-0.65 s
Synthesis Quality (SQ)	64.8%	72.4%	+7.6% points
Planning Efficiency (PE)	61.8%	99%	+37.2% points

The quantitative results, summarized in Table 4, provide strong empirical validation for the proposed Hierarchical-Hybrid architecture.

The Planning Efficiency (PE) metric shows the most dramatic difference. The Hierarchical agent achieved a near-perfect 99.0% PE, whereas the Monolithic baseline struggled significantly, reaching only 61.8% PE. This +37.2% point improvement underscores the core hypothesis: separating cognitive labor, particularly isolating the planning logic (CoS 4, 6) and ensuring robust execution (CoS 5), drastically reduces process failures and inefficiencies inherent in the monolithic approach. The Monolithic agent's frequent failures in planning efficiency are likely due to its single cognitive loop struggling to reliably adhere to complex rules like the "Gold Rule" or avoid redundant actions.

Regarding Execution Latency (EL), the Hierarchical agent demonstrated a modest improvement, completing tasks 0.65 seconds faster on average (8.61 s vs. 9.26 s). While not a massive speedup, this gain is achieved despite the architectural complexity of routing between different models. It confirms that using a lightweight model (GPT-4.1-Nano) for the purely mechanical Dispatcher role effectively offsets the overhead of the hierarchical structure.

Finally, the Synthesis Quality (SQ) also favored the Hierarchical agent, which achieved 72.4% SQ compared to the Monolithic agent's 64.8% SQ (+7.6% points). This suggests that the specialized Synthesizer node, equipped with a focused prompt, is better at adhering to the strict, rule-compliant output requirements (CoS 1-3) than the general-purpose loop of the Monolithic agent. However, the 72.4% score also highlights that final response synthesis remains a challenge for both architectures, pointing towards future work in prompt refinement or data quality improvements.

In conclusion, the Hierarchical-Hybrid architecture significantly outperforms the Monolithic baseline in reliability (PE) and shows advantages in speed (EL) and output quality (SQ), validating its suitability for complex, regulated financial workflows.

V. CONCLUSIONS

This paper introduced a Hierarchical-Hybrid ReAct agent architecture designed to overcome the limitations of monolithic

agents in complex financial workflows. Our experimental evaluation confirms that this modular approach offers significant advantages.

A. Main contributions

The core innovation: separating cognitive labor using a hybrid-model, multi-deployment strategy, proved highly effective. The quantitative results validate our key contributions:

Superior Reliability: The near-perfect Planning Efficiency (PE) of the Hierarchical agent, starkly contrasting with the Monolithic agent's PE, empirically demonstrates enhanced reliability. This validates that specialized roles and isolated execution virtually eliminate the process failures (CoS 4-6) inherent in simpler architectures, providing a robust foundation for high-stakes operations.

Improved Quality Control: While final response generation remains a challenge, the Hierarchical agent significantly outperformed the Monolithic baseline. This improvement confirms that the dedicated Synthesizer node, with its focused prompt, provides better adherence to strict output protocols (CoS 1-3). The remaining SQ gap highlights data-level issues (e.g., Vector DB context) rather than architectural flaws, guiding future refinement efforts.

Scalability via Decoupling: The modular toolset remains a key advantage, allowing for future capability expansion (new APIs, forecasting tools) without requiring core model retraining, ensuring the architecture's long-term adaptability.

B. Future Work

The successful implementation of the Hierarchical-Hybrid architecture establishes a reliable foundation for agentic execution in financial workflows. Future work will focus on bridging the remaining "information-to-action gap" by expanding the agent's capabilities:

Real-time Data Integration: Developing tools for Platform Data Retrieval via internal APIs will transition the agent from static knowledge (RAG) to dynamic, context-aware responses based on live data (e.g., account balances, order statuses).

Transactional Capabilities: Implementing tools for Transactional Actions (e.g., order creation, record updates) will fully empower the agent to move beyond consultation to active operational assistance.

Enhanced Self-Correction: Integrating a Reflective Mechanism into the Planner node will improve robustness by adding a layer for systematic validation of tool outputs against predefined heuristics.

In conclusion, the proposed Agentic AI Co-Pilot provides the traceable and significantly more reliable execution necessary for complex financial workflows. Its demonstrated performance advantages position it as a robust and scalable solution for enhancing compliance and user experience in the capital markets sector.

Reproducibility and Transparency

To ensure full replicability, transparency, and verification of our findings, the complete software package is available on Zenodo. This archive includes the agent's source code, LLM prompts, and the raw JSON outputs for all 500 experimental runs. The package is accessible via the following permanent identifier: https://zenodo.org/records/17459218

References

[1] Lewis, P., Perez, E., Piktus, A., et al. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Advances in Neural Information Processing Systems (NeurIPS 2020). arXiv:2005.11401.

[2] Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In 36th Conference on Neural Information Processing Systems (NeurIPS 2022). arXiv:2201.11903.

[3] Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. International Conference on Learning Representations (ICLR 2023). arXiv:2210.03629.

[4] Acharya, D.B., Kuppan, K., and Divya, B. (2025). Agentic AI: Autonomous Intelligence for Complex Goals-A Comprehensive Survey. IEEE Access, Vol. 13.

[5] Singh, A., Ehtesham, A., Kumar, S., and Khoei, T.T. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG. arXiv:2501.09136.

[6] Xiao, M., Qian, L., Jiang, Z., He, Y., Xu, Y., Chen, Z., Jiang, Y., Peng, M., Li, D., and Huang, J. (2025). Enhancing Financial Time-Series

[7] Forecasting with Retrieval-Augmented Large Language Models. arXiv:2502.05878.

[8] Haeri, A., Vitrano, J., and Ghelichi, M. (2025). Generative AI Enhanced Financial Risk Management Information Retrieval. arXiv:2504.06293.

[9] Lewis, P., Yih, W.T., Khandelwal, U., Garg, S., and Riedel, S. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Neural Information Processing Systems (NeurIPS).

[10] Moody's Analytics (2025). The State of AI in Financial Services 2025 Report.

[11] Microsoft (2023). Microsoft 365 Copilot: The Future of Productivity.

APPENDIX A

This appendix outlines the 10 strategic queries used for empirical validation.

TABLE V
STRATEGIC QUERIES TO EVALUATE

Query	Expected Tools	Focus
How do I generate the end-of-day compliance report on ComplianceX?	Manuals Search	EL/PE (Simple Loop Baseline
What are the criteria for flagging a suspicious transaction?	Manuals Search	SQ (Compliance) and Factual Integrity CoS 2, 3
What is the current closing price of Apple stock (AAPL)?	Web Search	EL (Single External Call Baseline).
Summarize the Fed's decision from its last meeting.	Web Search	SQ (News Synthesis) and EL
Hello, how are you today?	No Tool	EL (No-Tool Baseline) / PE (Avoid tool).
Define "liquidity risk" in simple terms.	No Tool	SQ(Coherence) and PE (Avoid tool).
What is the latest Fed decision, and how do I close a position in TraderX?	Web Search & Manuals Search	CRITICAL: Parallellism vs. Sequential Loop (Max Latency Test).
What are the mandatory fields for a new client and the latest market news?	Manuals Search & Web Search	CRITICAL: Parallellism and Synthesis Quality SQ from multiple sources.
Why is the ComplianceX system showing error code 9999?	Manuals > Synthesizer	Robustness PE Force the Insufficient Context fallback and clean termination.
Where is the setting located to adjust commission fees for a new fund?	Manuals Search	Test precision and immediate termination after success.

APPENDIX B

Appendix B outlines the prompts used for each of the nodes in the modular hierarchical agent as well as the monolithic.

PLANNER SYSTEM PROMP

ROLE AND OBJECTIVE
You are the Strategic Planner (Modular ReAct) for Financial Advisors. Your mission is to generate the most efficient action plan for the query, prioritizing **MINIMUM TOTAL LATENCY** (<6 seconds).

TOOLS AND SYNTAX
Available Tools:
{tool_list}
Action Plan Syntax (MUST BE THE FINAL PYTHON LIST):
- Parallelism (Speed): `[['command 1', 'command 2']]` (Only for independent initial search).
- Sequential (Precision): `['command']` or `['command', 'command 2']` (One command per step).
- Finalization: `['FINALIZE']`.

REACTION LOGIC AND EFFICIENCY
1. **FINALIZATION (GOLDEN RULE):** If the `Observation` from a tool designated as a **primary or internal source** provides useful and sufficient information, you **MUST IMMEDIATELY OVERRIDE** any pending steps and generate **ONLY** `['FINALIZE']`. It is prohibited to start a second search step if the objective has been met.
1.1 **LATENCY:** If the `Observation` from a **primary or internal source** is insufficient, you **MUST AVOID** using tools designated as **external or secondary sources** if they are unlikely to provide useful information, as this significantly increases latency. When in doubt, prioritize `['FINALIZE']`.
2. **INTELLIGENT USE:** Actions that depend on previous results (e.g., analysis or action tools) **MUST** be **sequential**. Parallelism is only used for initial context gathering if the query is ambiguous and multiple **search** tools might be relevant simultaneously.
3. **PROHIBITION:** Do not generate commands for unlisted tools or repeat a tool that has already provided the necessary information.
4. **MINIMIZATION:** You **MUST MINIMIZE** the total number of tools invoked to reduce latency.

Output (STRUCTURED FORMAT AND TRACEABILITY)
You must generate **TWO** output components in the following strict order:
1. **REASONING (CoT):** Enclose in `<REASONING>`. Justify your plan (choice of tools, parallelism/sequentiality) and the application of the Golden Rule or the handling of insufficient data/errors. Be concise.
2. **ACTION PLAN:** **ONLY** the Python list of action commands (without backticks).

Example Full Output:
<REASONING>The internal search tool (primary source) was successful and the information is sufficient. Applying the Golden Rule and finalizing to meet the low latency goal.</REASONING>
['FINALIZE']

DISPATCHER SYSTEM PROMP

You are a GPT-4o Tool Dispatcher. Your SOLE task is to take the action commands given to you
and generate the AIMessage with the corresponding tool_calls.
ALWAYS generate one tool_call for each input command.
IT IS PROHIBITED to generate ANY text, reasoning, or additional explanations.

SYNTHESIZER SYSTEM PROMP

You are the Final Synthesis and Quality Control Module. Your task is to generate the SINGLE final answer for the user in the fastest way possible.
You must analyze the ENTIRE conversation history, the original query, and the tool results to create a response that meets all financial standards.

MANDATORY OUTPUT RULES (100% QUALITY METRIC):
1. **TOTAL PROHIBITION OF VAGUENESS:** NEVER use vague or incomplete phrases (e.g., 'Information is scarce', 'Contact a developer'). The response must be a definitive conclusion, even if that conclusion is 'The information is not available in the consulted sources.'
2. **PROHIBITION OF URLs:** NEVER list web addresses (URLs) or instruct the user to search for them. Synthesize the information directly.
3. **COHERENCE AND COMPLIANCE:** Any instruction or reference must explicitly name the platform (e.g., 'in the ComplianceX platform').
4. **STRUCTURE:** Present the information in a professional format, using clear lists or paragraphs.

AGENT'S FINAL REASONING: {cot}

MONOLITHIC AGENT SYSTEM PROMP

ROLE AND OBJECTIVE
You are an expert and highly efficient "Financial Advisor." Your job is to answer the user's query following the ReAct (Reason-Act) paradigm with the MINIMUM TOTAL LATENCY (<6 seconds).

AVAILABLE TOOLS
You have access to the following tools:
{tool_list}

STRATEGY AND ACTION RULES (Priority: Speed and Accuracy)
1. **ANALYSIS (Mandatory CoT):** Before ANY action or final answer, you MUST generate a concise <THOUGHT> block analyzing the history (including 'Tool Observations') and justifying your next step (whether calling a tool or finalizing).
2. **PARALLEL SEARCH (Efficiency):** If the query requires gathering general or technical information (e.g., 'how does it work...', 'what is...'), you MUST attempt to call the search tools (e.g., manuals_search, web_search) IN PARALLEL in the first step, using the LLM's native parallel function calling capability.
3. **FINALIZATION (GOLDEN RULE):** If the 'Observation' from an Internal Retrieval tool (designated as a primary source) provides useful and sufficient information, you MUST FINALIZE IMMEDIATELY. Your action must be to generate the final answer DIRECTLY, without further tool calls. It is PROHIBITED to initiate another search if the internal one was successful.
4. **SEQUENTIALITY (Precision):** If an action depends on the result of another (e.g., 'risk_analysis' needs data from a previous search), plan and execute the tools sequentially (one per turn).
5. **FINALIZE (Sufficiency):** If the history already contains enough information (from previous steps) for a definitive answer, your action is to generate the final answer, not call tools.

STRICT OUTPUT FORMAT (ONE of the following two options after <THOUGHT>)

Option A: If you need to call tools:
<THOUGHT>Your CoT reasoning here. Justify the choice of tool(s) and whether they are parallel or sequential.</THOUGHT>
(Generate ONLY the necessary tool_call(s). It is PROHIBITED to include ANY other text here.)

Option B: If you are providing the final answer:
<THOUGHT>Your CoT reasoning here. Justify why the information is sufficient (e.g., Golden Rule, complete history).</THOUGHT>
(Generate ONLY the final answer text for the user, following these MANDATORY QUALITY RULES:
1. **NO VAGUENESS:** Definitive conclusion (even if it's "information not available").
2. **NO URLs:** Do not list URLs. Synthesize the info.
3. **COHERENCE:** Name platforms explicitly (e.g., 'in ComplianceX').
4. **STRUCTURE:** Professional format (lists, paragraphs).)The asset will be created with an 'In Process' status and must be activated by Operations.