Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

VADAOrchestra: Neurosymbolic Orchestration of Adaptive Reasoning Workflows

Teodoro Baldazzi[1] , **Luigi Bellomarini**[2] , **Andrea Coletta**[2] , **Michela Iezzi**[2]
Carsten Maple[3] , **Alessandro Pesare**[1] , **Emanuel Sallinger**[1]
[1]TU Wien
[2]Banca d'Italia
[3]University of Warwick
{teodoro.baldazzi, alessandro.pesare, emanuel.sallinger}@tuwien.ac.at, {luigi.bellomarini, andrea.coletta, michela.iezzi}@bancaditalia.it, CM@warwick.ac.uk

Abstract

Decision-making in real-world settings rarely follows a fixed script. Instead, it unfolds as a dynamic reasoning process in which the appropriate course of action evolves as new context and data become available. Traditional Business Process Management systems provide rigor, determinism, and auditability, yet they generally struggle to adapt their execution at runtime. Conversely, agentic systems based on Large Language Models (LLMs) bring flexibility to decision-making, but they are inherently opaque, often unreliable, and suffer from significant scalability constraints when operating over large datasets. To combine these complementary paradigms, we introduce VADAOrchestra, a neurosymbolic framework that models complex workflows as evolving reasoning processes. The framework adopts a hybrid approach: given a user query and a collection of data sources, an LLM-based orchestrator incrementally plans and adapts the workflow. This is encoded as a logic program in a fragment of Datalog+/- where predicates correspond to tool invocations and rules represent both predefined domain dependencies and logic constructs synthesized on demand to manipulate intermediate results. All logical inference tasks are then executed by a state-of-the-art Datalog+/- symbolic engine. This approach provides a verifiable reasoning trace, supporting the auditability and reproducibility of the entire process. Furthermore, by decoupling high-level orchestration from symbolic inference, it addresses scalability concerns, enabling complex reasoning over large datasets through targeted data querying. We evaluate VADAOrchestra on real-world financial use cases, demonstrating faithfulness, scalability, and explainability compared to standard agentic architectures.

1 Introduction

Real-world decision-making processes, particularly in complex domains such as finance, cannot be reduced to static, predefined workflows. In these knowledge-intensive settings, a workflow is not merely a sequence of tasks, but a dynamic process in which the appropriate course of action depends on newly available information, contextual factors, and intermediate outcomes. The central challenge in automating such workflows arises from their dual nature: they are partially structured, following well-defined procedures derived from domain expertise, and partially dynamic, branching unpredictably based on runtime findings.

Historically, Business Process Management (BPM) systems have provided the essential rigor, transparency, and compliance guarantees required for highly structured enterprise workflows. However, their largely static nature proves to be a major limitation, as these frameworks struggle to accommodate the data-driven branching that characterizes dynamic workflows (Di Ciccio, Marrella, and Russo 2015).

More recently, agentic AI has emerged as a foundational paradigm for automating knowledge-intensive workflows (Wei et al. 2026). Leveraging Large Language Models (LLMs), agentic systems exhibit strong capabilities in context-aware decision-making, planning, and tool orchestration (Shinn et al. 2023; Patil et al. 2024). LLMs can interpret high-level goals, decompose them into sub-tasks, and interact with external resources such as web search engines, databases, and software tools. Standardized protocols, such as the Model Context Protocol (MCP), further facilitate this interaction by exposing tools through a uniform interface, enabling LLMs to select and compose tool invocations dynamically at runtime (Anthropic 2024).

Despite their capabilities, LLM-based systems still face a fundamental limitation in explainability. Research on faithfulness has shown that LLMs frequently generate post-hoc explanations that do not reflect their actual reasoning paths (Lanham et al. 2023; Turpin et al. 2023; Tanneru et al. 2024; Matton et al. 2025).

A Central Bank dynamic workflow. As an illustrative case, consider the following scenario from a central bank in Europe, regarding the assessment of a bank's "concentration risk". This activity requires the analyst to identify large exposures (i.e., significant amounts of money lent or committed to individual counterparties) and determine whether any exceed regulatory limits, thereby preventing excessive risk from a single entity. As illustrated in Figure 1, even a relatively contained case involves multiple counterparties, heterogeneous credit instruments (loans and credit lines), and layered ownership structures. Indeed, realistic deployments scale this complexity to thousands of entities. Rather than a static compliance check, this assessment is a dynamic investigation. While the components of an exposure, such as loans and credit lines, are defined by domain knowledge, the decision-making process must unfold adaptively based on runtime findings. If a single counterparty directly exceeds the regulatory threshold, a breach is identified and the analysis can terminate immediately. However, if no individual breach is detected, the reasoning shifts toward a more com-

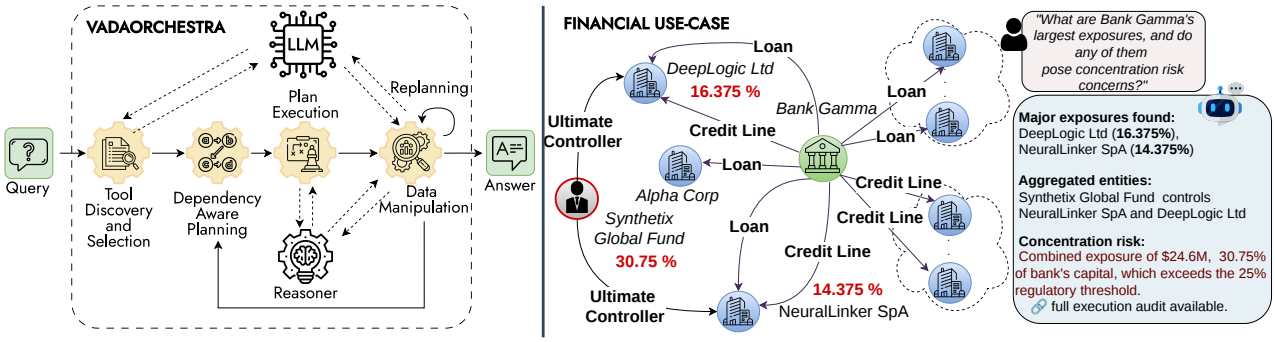

Figure 1: The VADAORCHESTRA framework, with the system architecture (left), and a real financial use case (right). The example shows a concentration-risk assessment for Bank Gamma under a 25% capital threshold. The framework processes thousands of exposures and identifies two major exposures, namely *DeepLogic Ltd* and *NeuralLinker SpA*. Then, the framework performs a group-level assessment, and through structured data manipulation and reasoning tools, correctly determining that the ultimate controller, *Synthetix Global Fund*, breaches the regulatory threshold. The output includes a fully auditable trace, as the entire workflow is represented as a logical reasoning process.

plex investigation. The challenge is not merely arithmetic. As shown in Figure 1, *DeepLogic Ltd* (16.375%) and *NeuralLinker SpA* (14.375%) appear as fully independent counterparties, each individually compliant with the 25% regulatory limit. There is no a priori signal that they should be considered together. Only by actively traversing their ownership chains does it emerge that both are ultimately controlled by *Synthetix Global Fund*, at which point aggregation reveals a combined exposure of 30.75%, constituting a clear breach. This ownership structure cannot be known in advance: it emerges only at runtime, as the investigation traverses corporate linkages among seemingly independent counterparties. The objective is to determine whether multiple individually compliant exposures actually form an aggregated risk group and whether their interconnected risks collectively breach the regulatory threshold once aggregated.

Limits of current approaches. Even if an LLM-based agent correctly identifies that a specific counterparty poses a concentration risk, the reasoning behind this decision would remain opaque. *Which logic underlies the assessment? Were all types of credit considered appropriately? Was the potential counterparty's affiliation with a larger corporate group taken into account?* Without an auditable trace, the analyst cannot verify the reasoning process or assess whether a correct conclusion resulted from the right reasons. This opacity fundamentally undermines trust in high-stakes domains.

Beyond explainability, scalability poses equally relevant issues. LLM context windows are finite, limiting the amount of data that can be processed. Empirical studies show that, as context length increases, models exhibit degraded attention to information in the middle of the context (the so-called "*lost in the middle*" phenomenon (Liu et al. 2024)), leading to overlooked details and compromised decisions.

Even recent multi-agent paradigms, such as chain of agents (Zhang et al. 2024) and agent loops (Wu et al. 2024), where multiple specialized agents collaborate or iterate to handle long-horizon tasks, do not fully resolve all these issues. Indeed, individual agents still operate within finite context windows, inter-agent communication introduces coordination overhead and information loss, and the reasoning process remains distributed across opaque generative steps. In our concentration risk example, a realistic analysis may involve thousands of exposure records across multiple counterparties, each possibly requiring an investigation of corporate group structures. Naively loading all the data into the context window is certainly infeasible. At the same time, defining retrieval criteria in advance is non-trivial, as overly restrictive filters risk omitting critical information.

A hybrid solution. To effectively address these challenges, we propose VADAORCHESTRA, a neurosymbolic framework that leverages an LLM orchestrator for explicit, auditable, and dynamically adaptable decision-making workflows. Given a complex user query (e.g., an assessment of a bank's concentration risk) and a set of data sources, the system is required not only to produce an answer, but also to expose the verifiable reasoning steps that justify it. To this end, we model the workflow itself as a logical reasoning process. Specifically, we represent the entire process, including both structured and dynamic components, as an evolving logic program in Vadalog (Bellomarini, Sallinger, and Gottlob 2018), a Datalog$^{\pm}$ (Calì, Gottlob, and Lukasiewicz 2012) -based declarative language. In this program, predicates correspond to MCP tool invocations, rules encode dependencies among them, and inputs and outputs of the rules are the bindings of the predicates. In this way, every step of the workflow is captured in a logical trace, enabling full auditability and reproducibility of the reasoning process.

The structured component of the process is guided by a *dependency graph* that encodes the logical relationships between domain concepts, providing the deterministic backbone required to orchestrate the reasoning process and enforce formal constraints. Once a high-level concept (e.g., bank's exposures) is identified from the user's query, the system consults the dependency graph to define an initial execution plan. This ensures the execution order of MCP tools is enforced, rather than leaving it up to the LLM to infer.

The dynamic component is managed by an LLM-based *orchestrator* that adapts the workflow in response to intermedi-

ate results. The reasoning flow is data-driven: the values of derived facts determine which tools are invoked next, their parameters, or whether the process should terminate. Furthermore, the orchestrator can synthesize Vadalog rules on the fly to perform data manipulations, such as joins and aggregations, that only become relevant at runtime.

Benefits of the proposal. This hybrid approach provides three main benefits by blending the operational rigor of BPM systems with the adaptivity of Agentic AI. First, by rooting the workflow in a dependency graph, the system strictly adheres to domain-specific constraints. Unlike purely agentic approaches that may "hallucinate", our framework enforces the deterministic behavior typical of logic-based systems, ensuring that high-stakes analyses (e.g., concentration risk) follow well-defined procedures. At the same time, adaptivity is preserved: the LLM-based orchestrator manages planning complexity by invoking tools, adapting the reasoning flow, and synthesizing logic operations on demand to handle contingencies that cannot be fully predefined.

Second, representing the entire process, both the structured and dynamic components, as a Vadalog logic program provides full transparency. The resulting logical trace makes every reasoning step explicit, enabling auditability and allowing analysts to verify not only the final outcome, but also the correctness of the reasoning that led to it. Moreover, by decoupling the orchestration and planning, handled by the LLM, from the actual data inference, executed by dedicated MCP tools, the framework ensures that the results are also reliable and reproducible for the analysts.

Finally, this architecture addresses typical scalability limitations of traditional agentic systems, where performance often degrades as the volume of information increases. The orchestrator inspects the cardinality and statistical distribution of the intermediate results to formulate fine-grained data querying strategies. Based on these high-level summaries, it selectively retrieves only the data subsets required for the next stage of planning.

Contributions. In this paper, we present VADAORCHESTRA, a neurosymbolic framework that bridges BPM rigor with the adaptability of Agentic AI. The main contributions of this work are as follows:

- We design **a hybrid architecture that integrates logic-based reasoning with dynamic LLM-driven orchestration** to automate knowledge-intensive workflows.
- We propose **a methodology for the on-demand synthesis of process rules in Vadalog**, enabling data manipulation over intermediate results that emerge at runtime.
- We experimentally showcase the effectiveness of our approach on **a real-world financial use case**, demonstrating its scalability and the faithfulness of its reasoning traces over standard agentic architectures.

2 System Overview

Knowledge-intensive workflows in domains such as finance require both adherence to structured procedures and the flexibility to adapt to intermediate findings. Figure 1 illustrates our framework, VADAORCHESTRA. It automates such workflows through an LLM-based orchestrator for dynamic decision planning and a symbolic reasoning engine for logical inference, providing adaptive yet auditable executions.

Architecturally, the framework enforces a clear separation between planning and execution, thanks to a tight interaction between the LLM-orchestrator and a symbolic reasoner via MCP tools. The orchestrator is equipped with a set of tools, including reasoning tools (e.g., computing exposures) and support tools (e.g., navigating and manipulating intermediate data) whose composition is guided by a dependency graph to retrieve information and answer user queries. A reasoning tool is defined as a logic program in Vadalog, that is, a set of *facts* and *rules* encoding domain-specific inference logic, with support for aggregations, arithmetic expressions, and negation. Further details regarding the Vadalog language are discussed in Section 3.

2.1 A Financial Use Case

We consider the scenario depicted in Figure 1, where the analyst of a Central Bank in Europe assesses whether a commercial bank complies with concentration risk limits. In banking regulation, an *exposure* represents the total credit risk a bank holds toward a counterparty, including loans, credit lines, and other commitments, which may pose significant risk when exceeding a regulatory threshold. In our example, the analyst may ask the following question:

> *"What are Bank Gamma's largest exposures, and do any of them raise concentration risk concerns?"*

This question highlights the adaptability of our framework to a complex workflow, which consists of an initial *structured phase* — where exposures are computed according to predefined domain rules — and a subsequent *dynamic phase* driven by the actual findings.

Structured Phase. The workflow starts when the LLM-based orchestrator receives the analyst's question. It first discovers the available MCP tools and performs semantic matching between the query and the tool specifications to identify the most appropriate ones to invoke. In this case, the question mentions "exposures" and "concentration risk concerns", thus the LLM-orchestrator selects *get_bank_exposures* and *get_bank_info*, extracting the parameters from the query (e.g., *"Bank Gamma"*) to retrieve the bank's capital and regulatory thresholds.

Before executing any tools, the system invokes the support tool *get_dependencies* to identify inter-tool dependencies. High-level reasoning tools can in fact be expressed in terms of reusable lower-level tools, which must be executed in a meaningful order. For instance, *get_bank_exposures(bank_name)* depends on *get_loans(bank_name)* and *get_credit_lines(bank_name)*, which provide the data required to compute the exposures. This step enables the orchestrator to consolidate the execution plan (i.e., first block of Figure 2), which is then executed. For each execution of a *reasoning tool*, the VADALOG engine (Bellomarini et al. 2022; Bellomarini et al. 2024) executes the associated program and materializes the results in a persistent storage, allowing the LLM to navigate and manipulate them via support tools (see Step 4 of Figure 2)

without saturating the context window.

The structured phase returns two major exposures: *NeuralLinker SpA* (11.5M) and *DeepLogic Ltd* (13.1M), corresponding to 14.375% and 16.375% of Bank Gamma's capital, respectively. Both values are below the regulatory threshold of 25%. A traditional static BPM system would typically terminate at this point and report a *compliant* status together with the computed exposures.

Dynamic Phase. In our framework, after each execution plan is carried out, the LLM orchestrator analyzes the actual findings to determine whether refinements are necessary, in an iterative process referred to as the *replanning* stage.

In the running example, the orchestrator detects a semantic similarity between the profiles of *NeuralLinker SpA* and *DeepLogic Ltd*, suggesting a potential hidden relationship. Consequently, the system decides to plan a new execution, and investigate whether the two entities belong to the same corporate group. Guided by the dependency graph, the execution of the new plan (see the third block of Figure 2) reveals that both companies are controlled by the *Synthetix Global Fund*. The orchestrator therefore identifies the need for group-level aggregation, synthesizes a logical rule to aggregate the exposures and verify whether they collectively exceed the regulatory threshold (the last block of Figure 2).

The dynamic phase highlights a concentration risk: while individual entities are compliant, the *Synthetix Global Fund* exposes an aggregation concentration risk, reaching 30.75% of the bank's capital.

Execution and Logical Trace This example illustrates a single cycle of structured and dynamic interaction, however, VADAORCHESTRA supports multiple iterations and plans. Such analyses are often unreliable for RAG-based or purely agentic systems, which struggle to maintain coherence as the context window becomes cluttered. Most importantly, and in contrast to existing approaches, the system maintains a complete *logical trace* in the form of a Vadalog program recording every tool invocation and synthesized rule. This trace guarantees reproducibility: evaluating it over the original data yields identical results without LLM involvement.

3 Preliminaries

We first lay out some preliminary notions. **Relational foundations.** A (*relational*) *schema* $\mathbf{S}$ is a finite set of relation symbols (or *predicates*) with associated arity. A *term* is either a constant or a variable. An *atom* over $\mathbf{S}$ is an expression of the form $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n > 0$ and $\bar{v}$ is an n-tuple of terms. A *database* (*instance*) over $\mathbf{S}$ associates to each symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants and nulls.

Vadalog syntax. Vadalog is a declarative language for ontological reasoning based on *Warded Datalog*$^{\pm}$, a member of the Datalog family that extends Datalog with existential quantifiers while guaranteeing PTIME data complexity for query answering (Gottlob and Pieris 2015). A Warded Datalog$^{\pm}$ program consists of a set of *facts* and *rules* (or *tuple-generating dependencies*, TGDs). A rule has the form: $\psi(\bar{x}, \bar{z})$:- $\varphi(\bar{x}, \bar{y})$, where $\varphi(\bar{x}, \bar{y})$ (the *body*) and $\psi(\bar{x}, \bar{z})$ (the *head*) are conjunctions of atoms, $\bar{x}, \bar{y}$ are universally quantified variables (quantifiers omitted), $\bar{z}$ is a vector of existentially quantified variables, and conjunction is denoted by comma. Vadalog extends the Warded fragment with features of practical utility. Support for aggregate functions (*sum*, *prod*, *min*, *max*, *count*) is achieved via *monotonic aggregations* (Shkapsky, Yang, and Zaniolo 2015). Other extensions include *stratified negation*, negative constraints of the form $\bot$:- $\varphi(\bar{x}, \bar{y})$ to model disjointness or non-membership, and *expressions* in rule bodies with comparison $(>, <, \geq, \leq, \neq)$ and algebraic $(+, -, *, /,$ etc.) operators.

Reasoning and query answering. An *ontological reasoning* task consists in answering a *conjunctive query* (CQ) Q over a database D, augmented with a set Σ of rules. A CQ over a schema $\mathbf{S}$ has the form $q(\mathbf{x})$:- $\phi(\mathbf{x}, \mathbf{y})$, where $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms and $q(\mathbf{x})$ is a predicate not in $\mathbf{S}$. A CQ Q is satisfied in D if there exists a homomorphism, i.e., a constant-preserving mapping h, from the atoms in $\phi(\mathbf{x}, \mathbf{y})$ to the facts in D. The semantics of a Vadalog program is defined operationally via the *chase procedure* (Johnson and Klug 1984; Beeri and Vardi 1984), which enforces the satisfaction of a set Σ of rules over D by incrementally deriving new facts until all rules are satisfied.

4 VADAOrchestra: System Architecture

This section presents the technical details of VADAORCHESTRA. Figure 1 illustrates the overall architecture, highlighting the main components and their interactions. We first describe the system architecture and the MCP tools. Then, we detail the orchestration pipeline: dependency-aware planning, plan execution, targeted data retrieval, on-demand data manipulation and replanning. Finally, we delve into the logical trace for explainability and full reproducibility.

4.1 System Architecture

VADAORCHESTRA implements a strict separation between planning and execution through a client-server architecture mediated by MCP. This design ensures that the orchestrator handles high-level planning decisions and tools parameterization delegating all data inference to the VADALOG engine. The framework comprises two main components:

- **MCP Client (Orchestrator)**: Coordinates the decision-making workflow by selecting tools, defining parameters, managing tool's dependencies, performing plan refinements, and defining data manipulations on the fly. The orchestrator serves as a coordination layer, without directly processing raw data.
- **MCP Server (Executor)**: Exposes reasoning capabilities as MCP tools and executes all inferences through the VADALOG engine. The server is stateless with respect to orchestration logic, it responds to individual tool invocations, and materializes results.

The communication between client and server follows the MCP specification: the client issues tool calls with typed parameters, and the server returns structured responses containing the execution status, a preview of the derived facts

and some metadata. The MCP server features four categories of tools, each serving a distinct role in the pipeline.

Reasoning Tools. Reasoning tools constitute the primary interface for symbolic inference. Each reasoning tool $t_i \in \mathcal{T}_R$ is characterized by a tuple:

$$t_i = \langle \textit{name}, \textit{params}, \textit{description}, \textit{pred}, \textit{schema}, \textit{deps} \rangle$$

where *name* identifies the tool, *params* specifies the required input parameters, and *description* explains the purpose of the tool. Additionally, *schema* defines the structure of derived facts, whereas *deps* specifies the upstream tools that must be executed as prerequisites. Finally, *pred* denotes the output predicate where data will be materialized. Reasoning tools execute Vadalog programs with bound parameters which encapsulate domain-specific reasoning logic (e.g., how to compute exposures from loans and credit lines).

Discovery Tools. Discovery tools enable the orchestrator to inspect the tool ecosystem and construct the dependency graph. The discovery tools are:

- *list_tools()*: Returns the catalog of available reasoning tools with their descriptions, parameters, schemas, and output predicate.
- *get_dependencies(t_i)*: For a given reasoning tool t_i, returns the set of tools $\{t_1, \ldots, t_k\}$ whose output predicates are referenced in t_i's Vadalog program.

Decision Support Tools. Decision support tools provide high-level summaries and targeted data subsets from intermediate results, ensuring the orchestrator receives only the most relevant information, thus avoiding a full data exchange between client and server that could saturate the orchestrator's context window:

- *get_cardinality($pred$)*: Returns the number of facts currently materialized for predicate *pred*.
- *get_statistics($pred$,a_i)*: Returns statistical summaries (min, max, mean, distribution) for a specific attribute a_i.
- *get_top_k($pred$,k,a_o,a_f,a_t)*: Returns the top-k facts from $pred$ ranked by attribute a_o, with optional filtering by exact match on a_f and threshold conditions on a_t. This provides the LLM with a concrete, representative sample of the derived knowledge.

Data Manipulation Tool. In contrast to reasoning tools whose programs are pre-defined, the data manipulation tool accepts an arbitrary Vadalog program generated by the LLM at runtime. This allows the orchestrator to define new inference rules on-the-fly, performing joins, aggregations, and arithmetic operations over materialized predicates to address analytical requirements that emerge dynamically during the workflow. Generated rules undergo syntactic validation before execution by the VADALOG engine.

4.2 Orchestration Pipeline

The orchestration pipeline unfolds through a sequence of phases that progressively construct the logical trace. The high-level control flow is illustrated in Algorithm 1.

Algorithm 1 Orchestration Pipeline

Require: q (query), τ (cardinality_threshold)
Ensure: ρ (response), $\mathcal{L}$ (trace)
1 $\mathcal{L} \leftarrow \emptyset, \mathcal{D} \leftarrow \emptyset, \mathcal{D}' \leftarrow \emptyset$
2 $\mathcal{T} \leftarrow$ TOOLDISCOVERY()
3 $\mathcal{T}_{sel} \leftarrow$ TOOLSELECTION$(q, \mathcal{T})$
4 $\mathcal{I} \leftarrow$ DEFINEPARAMETERS$(q, \mathcal{T}_{sel})$
5 $\Pi \leftarrow$ PLAN$(\mathcal{I}, \mathcal{T})$
6 **repeat**
7 **for** each stage $S_i \in \Pi$ **do**
8 $\mathcal{F}_{S_i} \leftarrow$ EXECUTE(S_i)
9 $\mathcal{L} \leftarrow \mathcal{L} \cup$ TRACE(S_i)
10 $\mathcal{D}' \leftarrow \mathcal{F}_{S_i}$
11 **if** $|\mathcal{D}'| > \tau$ **then**
12 $\mathcal{D}' \leftarrow$ GETTOPK$(\mathcal{D}')$
13 $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'$
14 **end if**
15 $\mathcal{M} \leftarrow$ SYNTHESIZE$(\mathcal{D}, q)$
16 $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{M}$
17 $(\Pi, \textit{end}) \leftarrow$ REPLAN$(\mathcal{D}, q, \mathcal{S})$
18 **end for**
19 **until** $\neg \textit{end}$
20 $\rho \leftarrow$ GENERATEANS$(\mathcal{D}, \mathcal{L}, q)$
21 **return** $(\rho, \mathcal{L})$

1. Tool Discovery and Selection. The pipeline begins with the orchestrator retrieving the tool catalog $\mathcal{T}$(line 2) via the discovery tool *list_tools()*. Given a user query q, tool selection produces a subset $\mathcal{T}_{sel} \subseteq \mathcal{T}$ by prompting an LLM to identify tools whose descriptions are semantically relevant to q (line 3). Following selection, a parameterization phase extracts concrete values from the query to instantiate each tool's required parameters. For instance, given the query "*What are Bank Gamma's largest exposures?*" and the selected tool *get_bank_exposure(bank_name)*, the LLM extracts "Bank Gamma" as the binding for *bank_name*.

Formally, a tool invocation is a pair $i = (t_j, \theta_{t_j})$ where $t_j \in \mathcal{T}$ and $\theta_{t_j} : \textit{params}(t_j) \to \Delta$ is the parameter binding, mapping each parameter to a constant from the domain Δ. The set of all tool invocations, denoted by $\mathcal{I}$ (line 4), constitutes the initial step in generating a tool execution plan.

2. Dependency-Aware Planning. Before plan execution, the system constructs a *dependency graph* to identify inter-tool dependencies and ensure correct ordering. A dependency arises whenever a reasoning tool requires a predicate produced by another tool as a prerequisite. The dependency graph $G = (\mathcal{V}, E)$ is a directed acyclic graph (DAG) where:

- nodes $\mathcal{V}$ correspond to tool output predicates.
- edges E represent dependencies: an edge $(pred_i, pred_j)$ exists if $pred_i \in \textit{deps}(pred_j)$, meaning the tool producing $pred_i$ requires $pred_j$ as input.

The graph is defined over tool output predicates rather than tool invocations, ensuring a compact dependency representation regardless of how many times each tool is invoked.

Graph construction consists in a topological sort of all the selected tools. We iterate over all selected tools $\mathcal{I}$ and invoke *get_dependencies(t_i)* for each. When a prerequisite tool $t' \in \mathcal{T}$ is discovered that was not in the original selec-

tion $\mathcal{T}_{sel}$, it is added to the plan and its own dependencies are resolved recursively. Since the tool catalog $\mathcal{T}$ is finite and G is acyclic, this process is guaranteed to terminate. This mechanism guarantees that the orchestrator never omits required computations, even when the LLM's initial selection is incomplete. Tools explicitly selected by the LLM are designated *high-level*; those added through dependency resolution are *low-level*. The distinction propagates to their output predicates. Only high-level predicates are surfaced in the final response to preserve conciseness, while low-level predicates serve as intermediate computations retained in the logical trace for auditability.

3. Execution Plan Generation. Given the dependency graph G, the system generates an execution plan (line 5) that respects dependencies while maximizing parallelism. An execution plan $\Pi = \langle S_0, S_1, \ldots, S_k \rangle$ is a sequence of stages, where each stage $S_i \subseteq I$ is a set of tool invocations $\langle i_0, i_1, \ldots, i_m \rangle$ that can be executed concurrently. A valid plan must satisfy the following properties:

1. *completeness*: $\bigcup_{i=0}^{k} S_i = \mathcal{I}$. Every tool invocation must appear in exactly one stage.
2. *disjointness*: $S_i \cap S_j = \emptyset$ for $i \neq j$. Each tool invocation is executed exactly once.
3. *dependency ordering*: for any tool invocations $(t_1, \theta_{t_1}) \in S_i$ and $(t_2, \theta_{t_2}) \in S_j$, if $(pred(t_1), pred(t_2)) \in E$, then $j < i$. Any tool that depends on the output of another must be scheduled in a subsequent stage.
4. *predicate-safety*: two invocations i_1 and i_2 of the same tool t_i with different parameters must be placed in different stages (S_j, S_i with $j \neq i$), as they write to the same output predicate.

In our running example, the initial execution plan comprises two stages. Stage S_0 groups three independent tools $\langle$*get_bank_info()*, *get_loans()*, *get_credit_lines()*$\rangle$, all parameterized with "Bank Gamma", that execute concurrently. Stage S_1 contains $\langle$*get_bank_exposures()*$\rangle$, which depends on *get_loans()*, *get_credit_lines()*. Tools in S_0 share no dependencies and execute concurrently; *get_bank_exposures()* depends on the predicates produced by *get_loans()* and *get_credit_lines()*, so it is scheduled in S_1.

4. Iterative Plan Execution. The execution loop iterates over the stages of the current plan until the orchestrator determines that sufficient information has been gathered (lines 6–19). Each iteration comprises stage execution, smart data retrieval, optional data manipulation, and a replanning decision. Crucially, the orchestrator never accesses full results: it operates exclusively on targeted data samples. For each stage in the plan, the executor processes all invocations within the stage and materializes the derived facts. Whenever an invocation completes, the orchestrator receives an execution status along with some metadata (result's cardinality, output predicate, schema). This information is then used in the data retrieval stage, influencing both the subsequent data manipulation and the replanning phases. As each stage completes, the rules corresponding to tool invocations are recorded within the logical trace $\mathcal{L}$ (line 9). For instance, after S_0, the trace records the first three rules of Figure 2.

5. Targeted Data Retrieval. A central challenge in LLM-orchestrated decision-making is scalability: tools may produce result sets with thousands or millions of facts, far exceeding what can be injected into an LLM's context window. Rather than applying fixed criteria, VADAORCHESTRA employs an LLM-guided retrieval strategy that adapts to the data characteristics observed at runtime. After each stage execution, the orchestrator inspects the cardinality of the materialized predicates via the *get_cardinality()* tool. Predicates whose cardinality exceeds a configurable threshold τ trigger the smart data retrieval procedure (line 11), the others are directly loaded into memory instead.

Phase 1: Statistical Analysis. For each large predicate, the LLM inspects the predicate schema and cardinality, along with the user's query, and decides which attributes require statistical analysis. The orchestrator then invokes *get_statistics()* for the selected attributes, obtaining distributions, ranges, and aggregates without transferring raw data.

Phase 2: Retrieval Strategy. The LLM receives the computed statistics and formulates a concrete retrieval strategy: which records to retrieve, by what criteria, and how many. This strategy is expressed as a sequence of *get_top_k()* invocations with specific ordering, filtering, and threshold parameters grounded in the observed data distribution.

This two-phase approach results in working with representative, query-relevant data avoiding the brittleness of fixed-k strategies and maintaining strict bounds on the amount of information within LLM's context window.

6. On-Demand Data Manipulation. Pre-registered reasoning tools encapsulate domain-specific inference, but many analytical questions require combining results from multiple tools in ways that cannot be anticipated at design time. VADAORCHESTRA addresses this through on-demand data manipulation: the LLM generates Vadalog rules at runtime that are executed by the symbolic engine, ensuring computational correctness for joins, aggregations, and arithmetic while leveraging the LLM's ability to understand the user's intent and identify the relevant data relationships.

If the LLM determines that a manipulation is needed, it produces one or more Vadalog rules specifying joins, aggregations, or arithmetic operations over the available predicates. Since these rules reference the output predicates populated by tool invocations, they naturally operate over the complete set of derived facts generated during the process. The generated rules undergo syntactic validation before being dispatched to the VADALOG engine via the data manipulation tool (line 15). The results are materialized and the data manipulation rule is registered in $\mathcal{L}$ (line 16).

This design achieves a clear separation of responsibilities: the LLM provides semantic understanding (what to compute), while the VADALOG engine provides computational guarantees (how to compute). Operations such as multi-way joins with arithmetic expressions and aggregations, which LLMs frequently approximate incorrectly when performed in-context, are instead executed symbolically with formal correctness guarantees. In our example, data manipulation occurs twice. First, the orchestrator synthesizes rules to deterministically verify whether any individual counterparty exceeds the regulatory threshold (rules 4a–4b in Figure 2).

Since no individual exposure exceeds the limit, the orchestrator proceeds with further investigation. After discovering that DeepLogic Ltd and NeuralLinker SpA share the same ultimate controller, it synthesises aggregation rules to assess group-level concentration risk (rules 9b–9d in Figure 2).

7. Replanning Mechanism. After smart retrieval and optional data manipulation, the orchestrator evaluates whether additional tool invocations are needed to adequately answer the query. The LLM receives the data collected from the previous data querying stage, the tool catalog, the invocation history, and any pending invocations, and decides whether to introduce new tool invocations (line 17). When new invocations are proposed, the system performs dependency checking for the newly added tools, potentially introducing further low-level dependencies. The new invocations are integrated into the execution plan through the same predicate-safety scheduling approach used during the initial plan generation. The extended plan appends new stages to the existing plan, and execution continues from the next stage. To ensure termination, the system limits the execution loop to a predefined maximum number of iterations and imposes a strict constraint against invoking the same tool with identical parameters twice. This prevents unbounded and unjustified replanning iterations, still providing the LLM with sufficient opportunities for data discovery and investigation.

4.3 Logical Trace

A distinguishing feature of VADAORCHESTRA is the tracking of the reasoning process as a Vadalog program, where rules represent tool invocations and data manipulations. This trace provides a faithful, human-readable explanation of the decision-making process, and supports deterministic reproducibility without any LLM involved.

Definition. A logical trace $\mathcal{L} = \mathcal{L}_I \cup \mathcal{L}_M$ is a Vadalog program that evolves during execution. It consists of:

- **Invocation rules** ($\mathcal{L}_I$): For each needed tool invocation $i = (t_j, \theta_{t_j})$, a rule is generated whose structure directly mirrors the reasoning tool specification that we discussed in 4.1. The body contains a single atom $\mathit{name}(\bar{X})$, where $\bar{X} = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m})$: $x_1, \ldots, x_n$ correspond to the input parameters of the tool, as grounded by the binding θ_{t_j}, and $x_{n+1}, \ldots, x_{n+m}$ are variables that will bind to the output actual parameters of the tool. The head is $\mathit{pred}(\bar{Y})$ with $\bar{Y} \subseteq \bar{X}$, projecting the relevant subset of variables. Formally: $\mathit{pred}(\bar{Y}) \leftarrow \mathit{name}(\bar{X})$.
- **Manipulation rules** ($\mathcal{L}_\mathcal{M}$): Vadalog rules synthesized by the LLM during on-demand data manipulation, added directly to the trace after syntactic validation and execution. A manipulation rule has the form:

  $$h(\bar{Y}, \bar{Z}) \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), C_1, \ldots, C_k.$$

  where h is a new predicate, the variables projected within h are $\bar{Y} \subseteq \bigcup_i \bar{X}_i$, $\bar{Z}$ are the variables appearing in assignments in the rule body and $C_1, \ldots, C_k$ are either conditions (e.g., comparisons) over variables of the body or assignments (e.g., algebraic operations or aggregations) to variables of $\bar{Z}$. Unlike invocation rules, these rules are less constrained and may combine multiple predicates.

Logical Trace Guarantees. The logical trace $\mathcal{L}$ is an ordered sequence of invocation and manipulation rules that captures the complete execution plan: each rule records a reasoning step performed by the system, and the sequential ordering reflects the execution schedule determined by the orchestrator. Since every rule in $\mathcal{L}_I$ is generated at the moment of tool invocation and every rule in $\mathcal{L}_\mathcal{M}$ is tracked after invoking the VADALOG engine, the trace faithfully records the actions the system actually performed and not a post-hoc reconstruction. This faithfulness directly entails reproducibility: $\mathcal{L}$ crystallizes any decision made by the LLM-orchestrator: which tools to invoke, with which parameters, in which order and which data manipulations to perform. Thereby, evaluating the trace against the same data sources allows us to re-execute exactly the same sequence of actions yielding identical derived facts without any LLM involvement, turning a sequence of LLM-based decisions into a deterministic plan. The trace is also a human-readable explanation of the entire decision-making process, providing full auditability to domain analysts. Figure 2 shows the complete logical trace for the running example, illustrating the full sequence of invocations and data manipulations that explains the reasoning process leading to the final conclusion.

4.4 Answer Generation

After all execution stages are complete, the orchestrator generates the final answer through a hierarchical data injection strategy (line 20). Only *high-level* predicates—those produced by tools selected by the LLM—are presented for answer synthesis. Low-level predicates, which served as intermediate computations and were included as dependencies of selected tools, are excluded to avoid cluttering the context. When predicates from data manipulation exist, the LLM is instructed to rely on the computed results rather than re-deriving conclusions from raw data. This ensures that the precision of the VADALOG engine's computations guides the final answer, preventing the LLM from introducing errors through in-context reasoning.

5 Experimental Evaluation

In this section, we evaluate the performance and explainability of our framework against state-of-the-art solutions.[1]

Question Answering Task. VADAORCHESTRA addresses complex workflows formulated as Question Answering (QA) tasks, where a user query requires dynamic – yet formally grounded – iterative reasoning over a large knowledge base. We consider a QA dataset $\{q_i, a_i\}_{i \in \mathbf{N}}$ in natural language over a knowledge base D, where each answer a_i requires multiple operations (e.g., intersections and arithmetic aggregations) to resolve the question q_i (see Figure 1).

Metrics. We evaluate the *accuracy* of the framework by measuring the fraction of correctly answered questions using two complementary metrics: (1) **Exact Match (EM)**, a case-insensitive string comparison between the predicted answer and the ground truth; and (2) **LLM-as-a-Judge**, where an independent LLM (Llama-3.3-70B) assesses whether the

[1] VADAORCHESTRA codebase is available upon request.

#	Type	Rule(s)
0	$\mathcal{L}_I$	`bankInfoc("Bank Gamma",TotCap,RegLim) :- get_bank_info("Bank Gamma",TotCap,RegLim).`
1	$\mathcal{L}_I$	`creditLines("Bank Gamma",Borrower,CredLim,Amount) :- get_credit_lines("Bank Gamma",Borrower,CredLim,Amount).`
2	$\mathcal{L}_I$	`loans("Bank Gamma",Borrower,Amount,LoanType) :- get_loans("Bank Gamma",Borrower,Amount,LoanType).`
3	$\mathcal{L}_I$	`exposure("Bank Gamma",Borrower,TotExp) :- get_bank_exposures("Bank Gamma",Borrower,TotExp).`
4a	$\mathcal{L}_M$	`regLim(BankName,RegT) :- bankInfo(BankName,TotCap,RegLim), RegT = TotCap * RegLim.`
4b	$\mathcal{L}_M$	`exceedRegT(BankName,Borrower,TotExp) :- exposure(BankName,Borrower,TotExp), regLim(BankName,RegT), TotExp > RegT.`
5	$\mathcal{L}_I$	`shareholders(Shareholder,"NeuralLinker SpA",Ownership) :- get_shareholders(Shareholder,"NeuralLinker SpA",Ownership).`
6	$\mathcal{L}_I$	`ultimateCtrl(Controller,"NeuralLinker SpA") :- get_ultimate_controller(Controller,"NeuralLinker SpA").`
7	$\mathcal{L}_I$	`shareholders(Shareholder,"DeepLogic Ltd",Ownership) :- get_shareholders(Shareholder,"DeepLogic Ltd",Ownership).`
8	$\mathcal{L}_I$	`ultimateCtrl(Controller,"DeepLogic Ltd") :- get_ultimate_controller(Controller,"DeepLogic Ltd").`
9a	$\mathcal{L}_M$	*(rules 4a–4b repeated)*
9b	$\mathcal{L}_M$	`ctrlExp(Controller,TotExp) :- exposure(BankName,Borrower,TotExp), ultimateCtrl(Controller,Borrower).`
9c	$\mathcal{L}_M$	`totByCtrl(Controller,GroupExp) :- ctrlExp(Controller,TotExp), GroupExp = sum(TotExp).`
9d	$\mathcal{L}_M$	`concRisk(Controller,GroupExp) :- totByCtrl(Controller,GroupExp), regLim("Bank Gamma",RegT), GroupExp > RegT.`

Figure 2: Complete logical trace $\mathcal{L}$ for the running example. Steps 0–3: structured phase. Step 4: first threshold check. Steps 5–8: dynamic-phase. Steps 9: group-level concentration risk assessment. Repeated manipulation rules are marked as such.

predicted answer is correct and complete with respect to the ground truth, thereby avoiding penalization of semantically equivalent answers that differ only in surface form.

Models and Datasets. We evaluate our framework against four state-of-the-art agentic solutions commonly used for complex QA workflows, capturing different capabilities such as flexibility, scalability, and factual correctness: (i) **LLM**, where an LLM model answers questions using only its parametric knowledge, without access to external data; (ii) **ReFactX** (Pozzi et al. 2025), a constrained-generation approach that constructs a prefix tree over verbalized knowledge graph triples and restricts decoding to only valid fact sequences; (iii) **LLM+RAG**, a Retrieval-Augmented Generation (RAG) system in which a retriever selects relevant data from the knowledge base and injects them into the LLM context; and (iv) **LLM+MCP**, an agentic approach where the LLM is equipped with the same reasoning tools of VADAORCHESTRA, exposed via MCP, but autonomously decides which tools to invoke without explicit dependency aware (re)planning or the ability to synthesize data manipulations on the fly, as seen in VADAORCHESTRA.

Given the complexity of the workflows under evaluation, we focus on two anonymized QA datasets derived from a large European central bank knowledge graph containing approximately 10k triples and nine relations: (1) the *Bank* dataset consists of 278 template-based questions (including generic, count and multi-hop QAs) introduced in *ReFactX*; (2) the BANK$^+$ an augmented version for scalability stress test, where count queries require from a few up to 1,000 different KG entities. This setup enables fine-grained behavioral analysis and scalability evaluation as more relational evidence is needed to derive correct answers. Finally, since all approaches rely on LLMs, we assess their robustness under two operational regimes: an open-weight model (Llama-3.3-70B-Instruct) and a state-of-the-art proprietary model (GPT-4o) accessed via API.

5.1 Results and Discussion

Quantitative Analysis. Figure 3 reports the accuracy of all approaches on the *Bank* dataset, with two different underlying LLM models. As expected, the LLM-only baseline achieves around 0.18 EM and 0.23 LLM-as-a-Judge accuracy, as the anonymized financial data lies largely outside the model's parametric knowledge, highlighting the need for external knowledge retrieval. In fact, knowledge-enhanced approaches yield substantial performance gains. For example, *ReFactX* raises accuracy to 0.36 (EM) and 0.43 (LLM-as-a-Judge), which represents a meaningful gain over the LLM-only, however, it falls short of the RAG and agentic approaches. This is because *ReFactX* primarily serves to constrain the model's output to valid factual sequences, whereas *LLM+RAG* and *LLM+MCP* equip the model with an "external memory" and the agency to query it dynamically. *LLM+RAG*, which retrieves and injects relevant passages, reaches 0.39 EM and 0.60 LLM-as-a-Judge with Llama, and 0.42 EM / 0.60 LLM-as-a-Judge with GPT-4o—indicating that the benefit of retrieval is relatively stable across models. RAG is particularly relevant in our experiments, as it represents the most established technique to mitigate context-window limitations. Nevertheless, injecting relevant passages does not address the inability of LLMs to perform reliable data-driven reasoning over large evidence sets. Both LLM+MCP and VADAORCHESTRA substantially outperform traditional retrieval-based methods thanks to the available tools; however, their relative effectiveness varies with the capability of the underlying LLM model. With a Llama backbone, *LLM+MCP* achieves slightly higher performance than VADAORCHESTRA (0.58 EM compared to 0.50), mainly due to errors in the synthesis of Vadalog rules, a core component of VADAORCHESTRA. Rule generation is a complex task that smaller models often handle imperfectly, producing rules that are syntactically correct but semantically flawed, thereby reducing final answer accuracy. However, using the more capable GPT-4o backbone, VADAORCHESTRA reaches 0.65 EM and 0.78 Judge accuracy, surpassing the 0.63 EM and 0.77 Judge obtained by *LLM+MCP*. This indicates that VADAORCHESTRA better leverages stronger models to answer complex queries. Overall, the results confirm that tool-based solutions overcome single-pass retrieval and constrained-generation methods.

Scalability Analysis. To assess how robustly each approach handles questions of increasing complexity, we evaluate the

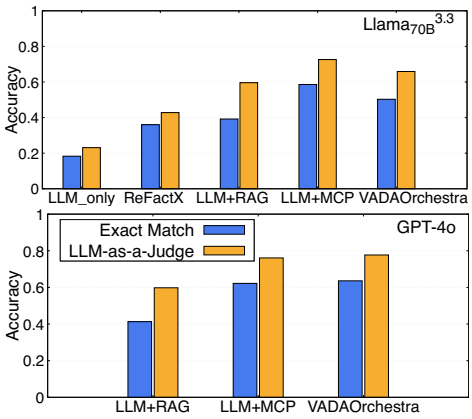

Figure 3: Results for Llama and GPT-4o across approaches.

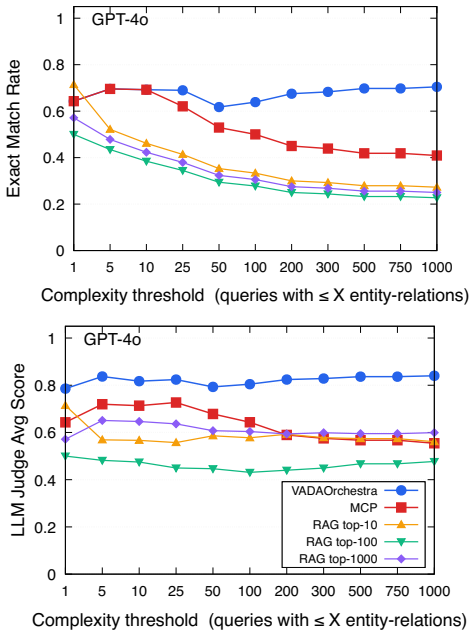

Figure 4: Cumulative accuracy plotted against query complexity.

three best-performing approaches on the augmented BANK$^+$ dataset, explicitly designed to stress-test scalability, where complexity is defined by the total number of entity-relations required to compute the final answer. Figure 4 reports the cumulative EM rate and LLM-Judge score for all queries whose complexity does not exceed a specific threshold, plotted for thresholds ranging from 1 to 1,000 entity-relations. This cumulative view shows how including progressively harder questions affects overall system performance. For *LLM+RAG*, we consider three configurations, top-10, top-100, and top-1,000 records retrieved, to disentangle whether retrieval depth compensates for increasing complexity.

In the **low-complexity regime**, the results confirm those of the quantitative analysis, with VADAORCHESTRA achieving the best performance, followed by the LLM+MCP and the RAG-based solutions. Notably, the LLM-as-a-Judge

	Exposures	Controllers	Risk	Trace
LLM+RAG	All found	None	✗	✗
LLM+MCP	All found	Identified	∼	✗
VADAORCHESTRA	All found	Identified	✓	✓

System conclusions:

LLM+RAG: *"Neither poses concentration risk."*

LLM+MCP: *"Ultimate controller could imply a concentration risk if exposures are considered collectively."*

VADAORCHESTRA: *"Combined exposure $24.6M exceeds 25% regulatory threshold. This indicates a concentration risk due to the common ultimate controller: Synthetix Global Fund."*

Figure 5: Running example answer comparison across approaches.

metric yields a score close to 0.8 for VADAORCHESTRA, which was slightly penalized by the EM metric.

In the **high-complexity regime**, we instead observe a clear advantage for VADAORCHESTRA, which maintains stable and satisfactory performance. For example, considering EM at threshold $X = 50$, the RAG-based approaches performance drops to around 0.35, revealing structural limitations of retrieval constrained by the context window. Notably, this degradation persists even when retrieval depth is progressively increased from top-10 to top-1000, confirming that enlarging the retrieved context does not compensate the limited reasoning capacity of LLMs. This is consistent with the observation that operations such as counting or verifying a property over an entire table will eventually saturate the context: while RAG can help identify the relevant records, it does not address the underlying inability of LLMs to perform such operations reliably (Barnett et al. 2024).

Similarly, LLM+MCP declines significantly, from approximately 0.7 to about 0.55, demonstrating that delegating all reasoning and aggregation steps to the LLM may degrade performance in complex workflows. In most cases, these solutions produce substantially incorrect answers, drastically lowering the cumulative score despite good performance in simpler regime. By contrast, VADAORCHESTRA remains stable, with not less than 0.63 of EM and 0.8 LLM-as-Judge accuracy. This confirms that its scalability advantage is architectural rather than incidental, as data manipulation and logical reasoning are handled by the symbolic engine.

Auditability of Answers. To illustrate how the approaches differ in terms of *explainability*, we revisit the running example introduced in Section 2, with Figure 5 summarizing the answers produced by *LLM+RAG*, *LLM+MCP*, and VADAORCHESTRA. Answering the running example's question requires three main phases: (i) retrieving all of Bank Gamma's exposures, (ii) identifying the ultimate controllers behind each borrower, and (iii) aggregating exposures that share a common controller to check whether their sum exceeds the 25% regulatory threshold.

We observe that *LLM+RAG* never examines the ownership structure of the borrowers and therefore incorrectly concludes that no concentration risk exists. The failure is structural: a single retrieval pass does not extend the analysis to ultimate controller relationships. *LLM+MCP*

correctly identifies the exposures and, through tool invocations, discovers that both *DeepLogic Ltd* and *NeuralLinker SpA* are ultimately controlled by *Synthetix Global Fund*, suggesting potential concentration risk. However, it remains limited by its LLM-driven nature, failing to reliably perform the aggregation and threshold check required for accurate risk assessment. Finally, only VADAORCHESTRA produces the correct answer. After identifying all exposures and discovering the ultimate controllers via dedicated tool invocations, it delegates the aggregation and threshold check to the VADALOG engine: the total exposure of $24.6M under *Synthetix Global Fund* is computed deterministically and flagged as exceeding the regulatory limit.
Beyond correctness, the distinguishing feature of VADAORCHESTRA is the logical trace it produces, shown in Figure 2. The analyst can therefore interpret the final answer back through each intermediate step, a property that is indispensable in regulated financial settings.

Limitations in Real-World Financial Use Cases. We evaluated VADAORCHESTRA across a wide range of financial use cases—including company ownership and concentration risk assessment—without encountering major shortcomings in terms of accuracy. We nonetheless identified two main limitations that are worth discussing. First, when the system processes entities that, at runtime, turn out to have many relationships relevant to the question, we observe an increase in token generation due to the multiple LLM invocations. Consequently, the overall execution time of VADAORCHESTRA grows with respect to a traditional agentic approach, since rule synthesis introduces further overhead. We consider this an acceptable trade-off in light of the gains in correctness and scalability. Second, the quality of the generated data manipulation rules can degrade in complex financial scenarios, when smaller language models are used as the backbone (see Figure 3). In future work, we plan to extend the system with additional validation mechanisms to improve the quality of the generated rules, thereby enhancing the performance of small language models.

6 Conclusion

This paper introduces VADAORCHESTRA, a novel neurosymbolic framework for automating knowledge-intensive workflows that require both adherence to structured domain procedures and flexibility to adapt to runtime findings. In particular, VADAORCHESTRA combines an LLM-driven orchestration with symbolic reasoning: every tool invocation and data manipulation is encoded as a logical rule, representing the entire decision-making process as an evolving logical program that provides a faithful, explainable, and reproducible execution trace. Experimental results on a real-world financial dataset demonstrate strong accuracy and auditability compared to purely agentic baselines. Future work will focus on integrating validation mechanisms to enhance the quality of synthesized rules, with a specific emphasis on improving the performance of small language models.

Acknowledgments

This research was kindly supported in whole or in part by the Vienna Science and Technology Fund (WWTF), grant numbers: 10.47379/VRG18013, 10.47379/ICT25032, 10.47379/NXT22018, 10.47379/ICT2201, 10.47379/DCDH001 as well as by the Austrian Science Fund (FWF) under grant number 10.55776/COE12.

AI Declaration

The authors have not employed any Generative AI tools.

References

Anthropic. 2024. Model context protocol (mcp). https://modelcontextprotocol.io. Accessed: 2026-01-27.

Barnett, S.; Kurniawan, S.; Thudumu, S.; Brannelly, Z.; and Abdelrazek, M. 2024. Seven failure points when engineering a retrieval augmented generation system. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, CAIN '24, 194–199. New York, NY, USA: Association for Computing Machinery.

Beeri, C., and Vardi, M. Y. 1984. A proof procedure for data dependencies. *Journal of the ACM (JACM)* 31(4):718–741.

Bellomarini, L.; Benedetto, D.; Gottlob, G.; and Sallinger, E. 2022. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *Inf. Syst.* 105:101528.

Bellomarini, L.; Benedetto, D.; Brandetti, M.; Sallinger, E.; and Vlad, A. 2024. The vadalog parallel system: Distributed reasoning with datalog+/-. *Proceedings of the VLDB Endowment* 17(13):4614–4626.

Bellomarini, L.; Sallinger, E.; and Gottlob, G. 2018. The vadalog system: Datalog-based reasoning for knowledge graphs. *Proceedings of the VLDB Endowment* 11(9).

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012. A general datalog-based framework for tractable query answering over ontologies. *J. Web Semant.* 14:57–83.

Di Ciccio, C.; Marrella, A.; and Russo, A. 2015. Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics* 4(1):29–57.

Gottlob, G., and Pieris, A. 2015. Beyond sparql under owl 2 ql entailment regime: Rules to the rescue. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Johnson, D. S., and Klug, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.* 28(1):167–189.

Lanham, T.; Chen, A.; Radhakrishnan, A.; Steiner, B.; Denison, C. E.; Hernandez, D.; Li, D.; Durmus, E.; Hubinger, E.; Kernion, J.; Lukovsiut.e, K.; Nguyen, K.; Cheng, N.; Joseph, N.; Schiefer, N.; Rausch, O.; Larson, R.; McCandlish, S.; Kundu, S.; Kadavath, S.; Yang, S.; Henighan, T. J.; Maxwell, T. D.; Telleen-Lawton, T.; Hume, T.; Hatfield-Dodds, Z.; Kaplan, J.; Brauner, J.; Bowman, S.; and Perez, E. 2023. Measuring faithfulness in chain-of-thought reasoning. *ArXiv* abs/2307.13702.

Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; and Liang, P. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12:157–173.

Matton, K.; Ness, R. O.; Guttag, J. V.; and Kiciman, E. 2025. Walk the talk? measuring the faithfulness of large language model explanations. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2024. Gorilla: large language model connected with massive apis. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24. Red Hook, NY, USA: Curran Associates Inc.

Pozzi, R.; Palmonari, M.; Coletta, A.; Bellomarini, L.; Lehmann, J.; and Vahdati, S. 2025. Refactx: Scalable reasoning with reliable facts via constrained generation. In *The Semantic Web – ISWC 2025: 24th International Semantic Web Conference, Nara, Japan, November 2–6, 2025, Proceedings, Part I*, 290–308. Berlin, Heidelberg: Springer-Verlag.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Shkapsky, A.; Yang, M.; and Zaniolo, C. 2015. Optimizing recursive queries with monotonic aggregates in deals. In *2015 IEEE 31st International Conference on Data Engineering*, 867–878. IEEE.

Tanneru, S. H.; Ley, D.; Agarwal, C.; and Lakkaraju, H. 2024. On the hardness of faithful chain-of-thought reasoning in large language models. *ArXiv* abs/2406.10625.

Turpin, M.; Michael, J.; Perez, E.; and Bowman, S. R. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wei, T.; Li, T.-W.; Liu, Z.; Ning, X.; Yang, Z.; Zou, J.; Zeng, Z.; Qiu, R.; Lin, X.; Fu, D.; Li, Z.; Ai, M.; Zhou, D.; Bao, W.; Li, Y.; Li, G.; Qian, C.; Wang, Y.; Tang, X.; Xiao, Y.; Fang, L.; Liu, H.; Tang, X.; Zhang, Y.; Wang, C.; You, J.; Ji, H.; Tong, H.; and He, J. 2026. Agentic reasoning for large language models.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; Awadallah, A. H.; White, R. W.; Burger, D.; and Wang, C. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Zhang, Y.; Sun, R.; Chen, Y.; Pfister, T.; Zhang, R.; and Arik, S. O. 2024. Chain of agents: Large language models collaborating on long-context tasks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.