

Rethinking Query Optimization for Multi-Agent Systems

Zoi Kaoudi
IT University of Copenhagen
Denmark
zoka@itu.dk

Ioana Giurgiu
IBM Research Europe
Switzerland
igi@zurich.ibm.com

ABSTRACT

The proliferation of large language models (LLMs) has accelerated the adoption of agent-based data pipelines. Yet current approaches remain ad hoc, relying on fixed structures, predefined LLMs, and single execution engines, without systematic optimization across heterogeneous data sources and engines. This paper presents NOMA, a query optimization framework for multi-agent data pipelines. We argue that optimizing agentic pipelines is a fundamentally different query optimization problem, with central challenges: (i) a multi-dimensional search and objective space, where topology, model, and engine choices must be optimized jointly across latency, cost, and accuracy; (ii) a variable pipeline topology; (iii) the co-existence of diverse data models, leaving no common operator algebra; and (iv) the significant cost of executing these pipelines. Our controlled experiment over a real-world 10-agent pipeline reveals extreme variance (153× cost, 5× latency, 25% quality) and that optimal plans are heterogeneous configurations no user would construct manually. Our analysis of real deployments confirms these inefficiencies are systematic. We present NOMA as an integrated optimization loop in which plan generation, cost estimation, runtime refinement, and semantic caching reinforce one another across executions, setting a community-wide research agenda on query optimization for multi-agent systems.

PVLDB Reference Format:

Zoi Kaoudi and Ioana Giurgiu. Rethinking Query Optimization for Multi-Agent Systems. PVLDB, 20(1): 46 - 52, 2026.
doi:10.14778/3845598.3845602

1 INTRODUCTION

The rapid advancement of large language models (LLMs), together with the emergence of tool-calling interfaces and agent orchestration frameworks, has led to the rise of multi-agent data pipelines: compositions of agents that reason, invoke tools, interact with heterogeneous data sources, and collaborate to execute complex data-driven tasks. These pipelines are increasingly used in real-world applications, ranging from customer support analytics to financial monitoring and supply-chain intelligence.

Data systems for agents. Despite the growing adoption of such pipelines, current approaches to constructing multi-agent pipelines remain fundamentally ad hoc. Developers rely on fixed pipeline structures, pre-selected models, and single execution environments,

resulting in pipelines that are often inefficient, brittle, and unable to adapt to evolving workload requirements. In these settings, the primary unit of computation is no longer the operator but the agent, and the primary artifact is not a fixed query plan but a dynamically structured pipeline of interacting agents. This shift calls for a new direction: *data systems for agents*. We must design systems where multi-agent pipelines themselves are the object of optimization. As a result, decisions about pipeline topology, model selection, and execution engines can no longer be made independently or ahead of time but must be jointly optimized under competing objectives such as latency, cost, and accuracy.

Challenges. This perspective is analogous to earlier “database for machine learning” efforts, which applied database principles to optimize ML pipelines [18, 25]. However, multi-agent pipelines introduce new challenges. First, the optimization space is inherently multi-dimensional, requiring joint reasoning over topology, models, and engines under competing objectives. Second, pipeline topology is itself a decision variable, meaning that the optimizer must determine not only how to execute a pipeline but also how to structure it. Third, the coexistence of structured data, unstructured text, embeddings, and streaming data eliminates any common operator algebra, preventing classical pruning techniques. Finally, execution is both expensive and stochastic, as LLM calls incur significant monetary cost and variability in output quality, making naive exploration of the search space infeasible.

NOMA. In this paper, we present NOMA, a query optimization framework designed for multi-agent data pipelines. NOMA treats agents as first-class units of computation and extends classical optimization principles, such as cost modeling, plan generation, and re-optimization, to a setting where plans are heterogeneous, stochastic, and structurally dynamic. The optimizer produces Pareto-optimal pipelines that balance latency, cost, and quality across heterogeneous execution environments. Our contributions include:

- (1) We quantify the cost of suboptimal choices on a real-world pipeline, showing 153× cost variance, 5× latency variance, and a 25% quality gap across the plan space under a fixed topology and show that varying topology reaches plans beyond it (Section 2).
- (2) We analyze multi-agent pipelines stemming from real deployments, showing that current practices leave substantial optimization potential unrealized (Section 3).
- (3) We formalize the novel problem of optimizing multi-agent data pipelines (Section 4).
- (4) We present NOMA, a vision for a query optimization framework for multi-agent pipelines, organized around four research challenges: variable multi-dimensional search, heterogeneous cost estimation, continuous optimization under stochastic execution, and semantic caching of pipeline artifacts (Section 5).

Related work. Existing research has largely focused on three lines of work. The first incorporates LLMs as user-defined functions

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 20, No. 1 ISSN 2150-8097.
doi:10.14778/3845598.3845602

Table 1: Positioning of NOMA against prior work.

Dimension	Model routing [6, 23]	Agent orchestration [5, 7, 30, 32]	LLM-in-DB [11, 16, 20, 24, 27]	NOMA
Topology a variable	×	✓	×	✓
Model assignment	✓	×	per-op	✓
Query engine selection	×	×	single engine	✓
Objective	cost-quality	single	partial	Pareto
Scope	per call	pipeline	operator	pipeline
Runtime adaptation	×	limited	×	continuous
Cross-pipeline reuse	×	partial*	×	semantic

*Helium [32] caches deterministic operator outputs across batches.

within traditional query processing pipelines [11, 16, 20, 24, 27], treating each LLM call as an operator within an otherwise fixed data system. The second targets model routing across LLMs, learning to dispatch each call to the most cost-effective model under quality constraints [6, 23]. The third focuses on serving efficiency and orchestration on top of agent frameworks, optimizing scheduling, resource use, and execution order across LLM calls [5, 7, 30, 32]. In all three, optimization targets individual operators or calls, making them cheaper, faster, or more accurate, while the overall pipeline structure remains fixed and grounded in a predefined data model and execution engine. A recent perspective [35] documents redundancy and inefficiency in agentic pipelines but leaves the joint optimization across our suggested dimensions unaddressed. Table 1 summarizes these distinctions.

Our vision is deliberately ambitious. We require a paradigm shift: from optimizing queries over fixed plans to continuously synthesizing and adapting pipelines. Optimization becomes a persistent system function rather than a pre-execution phase. We aim to redefine query optimization for multi-agent systems, establish the foundations for a new class of query optimizers that move beyond static pipelines toward adaptive and cost-aware execution, and identify key research questions that the community must address.

2 THE COST OF SUBOPTIMAL CHOICES

We begin by establishing that the optimization problem is both severe and unavoidable. Using a real-world multi-agent pipeline, we show that the space of possible executions exhibits extreme variance across cost, latency, and quality, and that optimal configurations cannot be constructed manually.

2.1 Real-world Multi-agent Pipeline

Consider the following real-world request from our customers:

Real-time Customer Support Reporting: Find all urgent cases reported by customers from live chats in the last 24 hours. Generate a detailed report and flag cases that need priority handling.

This pipeline decomposes into ten agents (left part of Figure 1), combining structured data (customer profiles from a data lake), unstructured data (sentiment analysis and urgency detection), and streaming data (real-time chat messages). We distinguish between two classes of agents: *stochastic*, whose outputs may vary across executions (light blue nodes), and *deterministic*, which return the same output for the same input (dark blue nodes).

The potential for optimization is substantial along three dimensions: how the pipeline should be structured (topology), which

models should be used (model selection), and which engines should execute each task (engine selection). We take a first step towards quantifying this potential with a controlled experiment.

2.2 Experimental Setup

Data. The pipeline processes approximately 2.8 GB across the four deterministic agents and between 8K and 80K input tokens per stochastic agent, depending on the task.

Engines. Each deterministic agent can be assigned to one of four engine configurations: Spark or Presto, each with 1 or 4 worker nodes. In our experimental setup, each agent has a *home engine* where its data resides (A2, A3, and A8 on Spark; A5 on Presto). Running on a different engine family incurs a data transfer penalty proportional to data volume.

LLMs. Each stochastic agent can be assigned to one of three LLMs spanning the cost-quality spectrum: Llama 3.1 8B (cheap, fast, accuracy 0.74–0.82), Qwen 2.5-72B (moderate cost and accuracy 0.89–0.93), and Claude Sonnet 3.5 (most accurate at 0.93–0.96 but orders of magnitude more expensive). To account for error propagation, we measure pipeline-level quality as the weighted geometric mean of per-agent accuracy, which we obtain by task-specific evaluation against reference labels: classification accuracy for sentiment and urgency detection, F1 for entity extraction, and reference-based scoring for report generation. The weights reflect each agent’s contribution to the final output and are set uniformly.

Plan space in two dimensions. To make the experiment feasible, we first assume a fixed topology. A plan is a complete assignment of an engine to every deterministic agent and an LLM to every stochastic agent, yielding $4^4 \times 3^6 = 186,624$ plans. We uniformly sample 10,000 and evaluate each on monetary cost, end-to-end latency, and output quality. To account for variance, we evaluate each sampled plan three times and report the average. Since topology is held fixed, this space covers only two of the three optimization dimensions; the variance reported below is therefore a lower bound on what joint optimization over topology, models, and engines would reveal. We show the topology’s additional benefit at the end of this section.

2.3 Results

Figure 2 shows the three-dimensional space of cost, latency, and accuracy. We make three main observations.

The plan space exhibits extreme variance. Across the 10,000 sampled plans (which explore only model and engine selection under a fixed topology), monetary cost ranges from \$0.01 to \$2.04 (a 153× spread), end-to-end latency from 42 s to 216 s (5.1×), and quality from 0.76 to 0.95 (25% gap). Average plans are far from optimal on any dimension, so default choices lead to poor outcomes.

The Pareto frontier is rich and non-trivial. We find 266 Pareto-optimal plans, representing 2.7% of the sampled space. The frontier spans from a cheapest plan at \$0.01 (42.5 s latency, 0.77 quality) to a highest-quality plan at \$1.84 (125.9 s, 0.95 quality)—a 138× cost increase for a 19% quality improvement, while tripling the latency. These configurations are highly unintuitive: they require fine-grained coordination between model selection and execution placement that developers do not perform in practice.

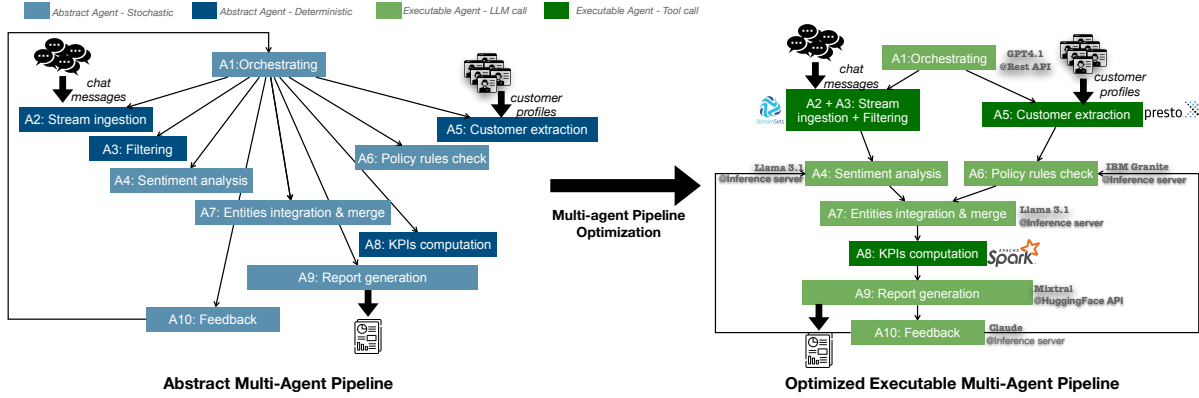


Figure 1: Initial multi-agent pipeline (left) and optimized multi-agent pipeline (right).

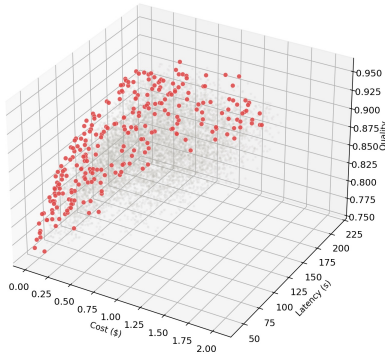


Figure 2: Pareto frontier for 10,000 randomly sampled plans.

Optimal plans are heterogeneous. The cheapest plan (\$0.01, 42.5 s, quality 0.77) assigns Llama 3.1 8B to all stochastic agents and places each deterministic agent on its home engine. The highest-quality plan (\$1.84, 125.9 s, quality 0.95) mixes Claude Sonnet 3.5 and Qwen 2.5-72B across stochastic agents, deliberately using the cheaper model where Claude’s advantage is marginal. A balanced plan (\$0.64, 58.9 s, quality 0.85) cuts cost by 65% and latency by 53% relative to the highest-quality plan by mixing all three models. Crucially, 96% of Pareto-optimal plans use mixed LLM assignments, a configuration that users rarely construct manually.

Topology as a third optimization axis. The variance above is measured under a fixed topology. To investigate the topology’s effect as well, we modify only the structure of a balanced plan found in the previous setting. In particular, we replace the single agent A4 with a conditional refinement loop that runs the cheapest model by default and escalates to Claude only when the confidence is low. This matches the accuracy of running Claude on every case while paying the expensive model price on a small minority, so the resulting pipeline lowers both cost and latency at equal quality (cost falls from \$0.64 to \$0.44, latency from 58.9s to 57s). This restructured pipeline dominates the balanced plan and lies outside the Pareto frontier of the two-dimensional space, reaching a point no model or engine assignment can reach without changing topology.

3 ANALYSIS OF MULTI-AGENT PIPELINES

We now ask whether this gap is systematic. We use 62 real agentic pipeline deployments spanning domains such as customer support, financial analytics, and supply-chain monitoring. We then construct and analyze 9,000 multi-agent pipelines by varying task compositions. Each pipeline comprises at least three tasks spanning data ingestion, schema mapping, feature engineering, analytical reasoning, and reporting. We report the following key findings.

Pipeline complexity is growing. Figure 3a shows the distribution of tasks per pipeline. We observe the highest density at 5–7 tasks, while pipelines with more than 10 tasks are not uncommon. As tools for automatically creating pipelines mature, we expect this distribution to shift toward even larger pipelines, much as application-generated SQL workloads have grown in complexity [26]. This trend matters because the optimization space grows combinatorially with the number of agents.

Model selection is driven by habit, not optimization. Across the corpus, 57% of tasks are stochastic (Figure 3b), and 21% of these use an LLM for diverse problems. Yet when we examine which models are actually used, we observe the same limited set recurring despite the existence of a much larger model landscape (Figure 3d). Users strongly prefer models they have already used and tend to favor smaller models over larger ones. In other words, model selection is not driven by task-specific cost-quality trade-offs, but by familiarity and convenience. This reinforces the conclusion of Section 2, that assigning one familiar model broadly across agents leaves substantial optimization potential unrealized.

Simple topologies dominate despite inefficiency. The most common structure is a simple chain (34%), followed by DAGs (25%), trees (24%), and branching chains (6%) as shown in Figure 3e. More complex structures are rare. This distribution reflects ease of development rather than efficiency: chains expose no parallelism, and none of the dominant structures naturally capture feedback loops.

Engine heterogeneity is underexploited. 43% of tasks involve a deterministic agent requiring a query engine (Figure 3c). Most of these run on structured databases (60%), while the remainder are split across streaming engines (8%), analytics engines (23%), and vector databases (9%). This confirms that engine assignment is rarely optimized.

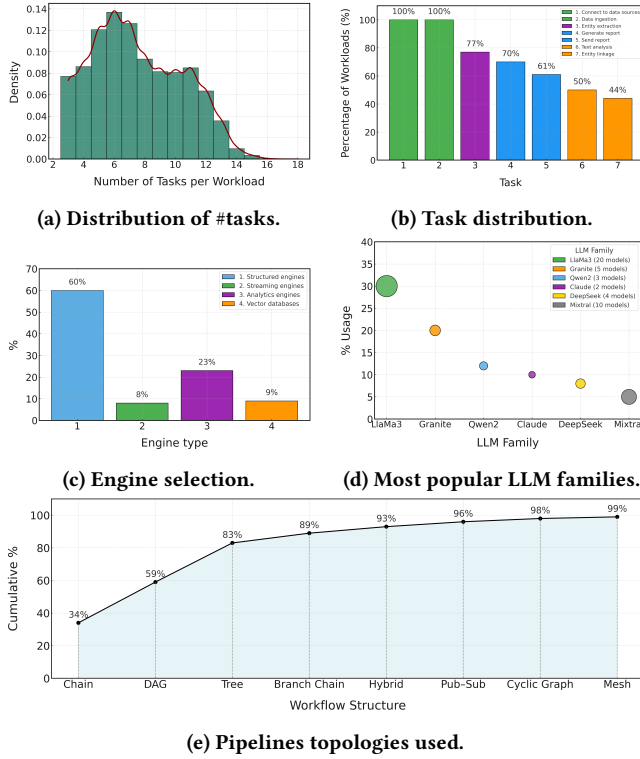


Figure 3: Analysis of multi-agent data pipelines.

Redundancy is pervasive. Core tasks such as data ingestion, entity extraction, and report generation recur across the majority of pipelines, yet repeated computations are rarely recognized or shared, leading to unnecessary cost and latency.

4 THE OPTIMIZATION PROBLEM

We now formalize the problem of optimizing multi-agent data pipelines. Given an abstract pipeline, the goal is to determine an executable pipeline by jointly selecting pipeline topology, model assignments, and execution engines under multiple objectives such as latency, cost, and accuracy. These decisions interact in complex and non-obvious ways: A single abstract pipeline can map to many structurally different executable pipelines, each with distinct performance characteristics.

Figure 1 illustrates this distinction. Starting from an abstract pipeline, agents can be merged, split, reordered, or parallelized, while model and engine assignments vary across configurations. This results in a large, heterogeneous search space. To formalize this setting, we define the following concepts.

Abstract multi-agent pipeline. At a conceptual level, a multi-agent pipeline is a logical composition of agents collaborating to accomplish a task. Each agent provides a specific capability, such as retrieval, sentiment analysis, or summarization, and interacts with other agents. Unlike relational systems, where the space of operations is fixed, the space of tasks that agents can perform is significantly broader and, in principle, unbounded. In practice, however, a system can only execute a finite set of agents. We therefore

assume the existence of a global agent registry $\mathcal{R} = \{a_1, a_2, \dots, a_n\}$ that defines the available agents together with their functional descriptions and expected inputs and outputs. Formally, we define an abstract multi-agent pipeline as a directed graph $G = (A, E)$, where $A \subseteq \mathcal{R}$ is the set of agents participating in the pipeline and $E \subseteq A \times A$ captures the communication between them. Each directed edge $(a_i, a_j) \in E$ denotes a dependency in which the output of agent a_i serves as input to agent a_j . The pipeline therefore specifies *what* tasks are to be performed and the information (i.e., control and/or data) flows between agents, while abstracting away from *how* each agent is implemented or executed and how they are connected. We distinguish between two classes of agents: *stochastic* agents, whose outputs may vary across executions and may be approximate (e.g., summarization), and *deterministic* agents, which return the same output for the same input (e.g., database retrieval).

Executable multi-agent pipeline. Given an abstract pipeline G , an executable pipeline specifies (i) the model assigned to each stochastic agent, (ii) the execution engine for each agent, and (iii) the structure that determines execution order, parallelism, and communication. Formally, an executable pipeline is a graph $G^* = (A^*, E^*)$ that may differ structurally from G , where A^* denotes the set of agents with concrete model and engine instantiations and E^* shows the communication among the instantiated agents.

Transformations. G^* may have a different structure from G because of a structure-changing transformation. We consider four transformations: (i) *fusion* merges two adjacent agents when the first feeds only the second; (ii) *split* decomposes an agent when its sub-agents reproduce its input-output contract; (iii) *reorder/parallelize* repositions or parallelizes agents at any point where their inputs remain available; and *conditional re-execution* adds a bounded guarded loop that re-invokes an agent while its quality estimate is low. In all cases the producer-consumer dependencies and each agent’s input-output contract are preserved, so that G^* stays semantically consistent with G .

Problem statement. Given an abstract pipeline $G = (A, E)$ and a set of optimization objectives $O = o_1, \dots, o_m$ (e.g., #tokens, monetary cost, quality, runtime, energy consumption, etc.), the goal is to find one or more executable pipelines $G^* = (A^*, E^*)$ that are Pareto-optimal with respect to O .

Note that among the objectives in O , quality is distinctive: unlike monetary cost or runtime, it has no universal metric and is task dependent. We therefore define the quality of an executable pipeline G^* as a task-specific function $q(G^*)$ which can be instantiated in different ways, e.g., via reference-based metrics (precision, recall, F1, exact match), model-based judges (LLM-as-judge, self-consistency), or domain-specific scores. The weighted geometric mean of per-agent accuracy used in Section 2 is one such instantiation.

This problem formulation shows why multi-agent pipeline optimization cannot be reduced to traditional query optimization. First, topology is a variable, rather than a plan with fixed operators. Second, optimization is inherently multi-objective, requiring trade-offs across latency, cost, and accuracy. Third, the absence of a common operator algebra prevents the use of classical pruning techniques. Together, these properties invalidate the core assumptions underlying traditional optimizers and require a fundamentally different approach, motivating the design of NOMA.

5 VISION AND RESEARCH CHALLENGES

Optimizing multi-agent pipelines requires rethinking each pillar of query optimization. We identify the main challenges that arise in this new setting and for each one we describe why existing approaches are insufficient, present our vision for how NOMA addresses it, and identify the open research questions that remain.

5.1 Variable Multi-Dimensional Search Space

The search space of a multi-agent pipeline spans three dimensions: pipeline topology, model selection, and execution engine assignment. These are highly inter-dependent: changing a model assignment changes the downstream workload that engine decisions must account for, merging two agents eliminates a model decision, while splitting an agent creates new ones. The set of decision variables is thus not fixed but changes with each structural alternative, yielding a vast, variable space where exhaustive enumeration is infeasible. Besides the complexity of the search space, the multi-objective nature of the problem compounds this difficulty. The system must explore a Pareto frontier over different user-specified objectives. The search process must create diverse candidates that reflect different trade-offs rather than converging to a single optimal plan. These challenges rule out classical enumeration-based approaches, where algebraic equivalences prune the search space and cost models estimate a single cost for the candidates.

Our vision. We propose using generative models trained on historical pipelines to generate executable pipelines directly in the joint optimization space, bypassing plan enumeration. In NOMA, a *multi-agent pipeline generator* produces candidates that vary across topology, model assignments, and engine configurations simultaneously, reflecting different positions on the Pareto frontier. Generative models have recently been proposed for relational query optimization [29], however this work operates over a fixed relational plan structure and optimizes for a single objective (execution time). Similarly, learned orchestrators [7] optimize execution order for a fixed agent set under a single objective. In contrast, NOMA’s generator must produce both pipeline structure and execution decisions jointly, without a fixed plan shape, without cardinality estimates, and across a multi-objective Pareto frontier, making it a fundamentally different generation problem.

Open research questions. Training generative models requires examples of good pipelines, yet obtaining them requires executing many non-optimal pipelines: How can we bootstrap training data without exhaustive, costly exploration? What representations capture both graph structure and agent semantics well enough to generalize to unseen pipelines? And how can generation be made cost-aware, so that candidates are likely near the Pareto frontier before the cost model evaluates them, avoiding wasted evaluations on poor candidates?

5.2 Unified Cost Models

Traditional and learned cost models assume a fixed plan structure and produce scalar estimates, assumptions that break in the multi-agent setting. First, topology is a variable: the cost model must compare pipelines differing in the number of agents, their connectivity, and their execution order. Learned cost models [14, 15, 28]

predict cost only within a given plan shape, while cross-engine optimizers [1, 4, 10, 17, 28] extend this to heterogeneous engines but still assume a fixed structure. Second, agent cost signals are not directly comparable: LLM agents incur token costs, stochastic latency, and variable quality, database agents depend on data volume and indexing, and streaming engines differ again. Quality propagation compounds this: a cheaper, lower-quality upstream model changes the volume and nature of downstream inputs, altering engine costs in ways a per-agent cost model cannot capture.

Our vision. We propose building *unified cost models* that map heterogeneous agent features into a shared representation and predict calibrated distributions over latency, monetary cost, and accuracy rather than scalar point estimates. Rather than a single monolithic model, we envision a family of cost models, one per objective, that share a common pipeline graph representation but specialize their output heads for the statistical properties of each objective. Distributional estimates are essential because agents are stochastic and quality propagates non-additively: a point estimate cannot capture the uncertainty introduced by upstream model choices, making it impossible to reason reliably about downstream costs. Conformal prediction [2, 21] can provide distribution-free calibration guarantees. Online recalibration [34] keeps the models accurate as pricing, hardware availability, and workloads shift. The three objectives differ significantly in how they can be learned. Monetary cost can be computed almost analytically: for stochastic agents it is a deterministic function of token volume and published per-token prices and for deterministic agents a function of data volume and resource-time at known rates; the only learned component is predicting volume before execution. Latency is largely profiled offline: per-agent latency distributions are measured per (model, engine, input-size) and composed along the pipeline, with a learned correction for contention when agents share resources. Quality is the only objective that is fully learned, label-scarce, and non-additive, and is therefore where the challenge concentrates. Context passing is accounted for when an execution plan is evaluated: the context an agent receives determines its input cost and latency and affects its output quality. Critically, the cost model need not be accurate in an absolute sense: it only needs to rank candidates well enough to prune generation and stay calibrated to flag when an execution deviates from prediction, triggering the refiner.

Open research questions. How can cost models compare configurations across structurally different topologies? How should heterogeneous cost signals (token usage, data volume, model accuracy) be reconciled into a unified representation? How can distributional estimates stay calibrated when quality propagates non-additively, so that uncertainty at one agent affects all downstream agents? And how should pipeline quality be defined and estimated efficiently, given that it is task-dependent and lacks a directly observable label?

5.3 Optimization Under Stochastic Execution

Traditional query optimization is a one-shot process: the optimizer produces a plan before execution begins and the executor carries it out without further intervention. This separation is possible because relational operators are deterministic and cost model errors manifest only as suboptimal performance, not as fundamentally changed downstream workloads. Multi-agent pipelines violate both

conditions. Stochastic agents introduce non-determinism: the same input can produce different outputs on different runs, and the quality of an upstream agent’s output directly affects the volume, distribution, and nature of downstream inputs. One-shot optimization cannot account for this because the actual workload is not known until execution begins. Existing approaches such as progressive optimization [22] and adaptive query processing [9] do not address this setting: they trigger re-optimization reactively when observed cardinalities deviate from estimates and re-planning is cheap relative to execution. In our setting, revision must be continuous and re-planning is expensive enough that the system must reason about whether the expected benefit of revision justifies its cost.

Our vision. We argue that multi-agent pipeline optimization should be continuous. In NOMA, a *continuous pipeline refiner* is a first-class component that participates in execution from start to finish. It uses a Bayesian optimization process where each agent completion yields an observation that updates cost model priors and revises downstream decisions accordingly. The refiner then applies an acquisition function to decide whether the expected gain from re-planning justifies its cost. BayesQO [31] has shown the effectiveness of Bayesian optimization for navigating a plan space in relational query optimization. In contrast, NOMA’s refiner must handle multi-objective optimization over a variable pipeline topology rather than a single objective over a fixed query structure. The refiner is tightly coupled with the cost models of Section 5.2.

Open research questions. How should the scope of revision be determined after each agent completion: revising only the next decision is cheap but myopic, while revising the full remaining pipeline is expensive but globally optimal? How can the system decide when revision pays off, given that planning cost is no longer negligible? How can stability be ensured under frequent revisions?

5.4 Semantic Caching Across Pipelines

Multi-agent workloads exhibit significant redundancy: data ingestion, entity extraction, summarization, and report generation recur across pipelines and often over overlapping inputs (see Section 3). In a setting where each LLM execution incurs non-trivial monetary cost, exploiting this redundancy through caching is not an optional enhancement but a core cost reduction mechanism. Traditional caching relies on syntactic equivalence — a cached result can be reused when queries match exactly or can be shown equivalent through algebraic rewriting. This assumption breaks down entirely in the multi-agent setting. Agent inputs are expressed in natural language, executions are non-deterministic, and there are no algebraic rewriting rules to establish equivalence. Two agent invocations that are semantically equivalent will not match syntactically, yet reusing a cached result may be entirely appropriate. Semantic caching has been studied since Dar et al [8] introduced predicate-based caching. Recent systems adapt it to LLM workloads via embedding similarity over queries [3, 13, 33]. These cache individual query–response pairs over flat inputs; they do not cache sub-plans or optimization decisions, reason about structural similarity across pipelines, or transfer configurations to structurally different pipelines—the challenges NOMA’s cache must address.

Our vision. We propose a *multi-purpose cache* that goes beyond caching intermediate results to also storing optimized sub-plans

and optimization decisions: model assignments and engine configurations that proved effective in prior pipelines. The cache indexes all these artifacts semantically rather than syntactically. It combines structural normalization with semantic embedding, with a lightweight coarse filter to eliminate obvious mismatches before invoking the more expensive embedding-based similarity check.

Open research questions. How should semantic equivalence be defined for agent inputs, balancing precision, recall, and similarity-computation cost? How can sub-plan reuse be supported across structurally different pipelines, where similarity is semantic rather than syntactic? And how should staleness be managed, and cached decisions transferred across pipelines?

5.5 NOMA as an Integrated Optimization Loop

The four challenges above are tightly coupled facets of a single optimization loop. NOMA’s contribution is that its components are mutually dependent and must be co-designed. The generator proposes candidates, the cost models score them, the refiner revises decisions at runtime, and the cache closes the loop by converting execution experience into reusable knowledge. Each of our envisioned solutions extends an existing line of work and does not necessarily constitute a standalone contribution. As a concrete agenda, we propose to address the challenges in the following order: (1) a unified cost model over a fixed topology; (2) pipeline generation built on top of it; (3) the continuous pipeline refiner; and (4) caching.

Evaluation. We will evaluate NOMA on a representative subset of our 9,000 pipelines, against baselines spanning a developer-default “habit” configuration, per-agent model routing, orchestration-only approaches, random sampling with Pareto filtering, and an oracle optimizer on small instances as an upper bound. We will report end-to-end cost, latency, and quality. In addition, we will measure optimization overhead, distance to a reference Pareto frontier, and component-level contributions, such as cache hit rate and refinement gains. We expect NOMA to deliver substantial reductions in monetary cost and latency at matched quality levels relative to developer-default baselines, while approaching the Pareto frontier obtained by exhaustive enumeration on small pipelines.

6 LOOKING AHEAD

Multi-agent data pipelines represent a fundamental shift in how data-driven tasks are executed. Yet, they remain largely unoptimized. The challenges of jointly optimizing topology, model selection, and execution engines require rethinking each pillar of query optimization. NOMA is our vision for a framework that addresses these challenges as an integrated optimization loop, where each pipeline execution makes the next one cheaper and better.

This direction aligns with emerging work on synthesized data systems [12, 19]. NOMA extends this trajectory to multi-agent workloads: instead of optimizing queries over a fixed system, it enables pipelines, execution strategies, and system configurations to co-evolve. In this view, optimization is no longer a stage in execution but a continuous function, and agentic data systems transition from fixed pipelines to adaptive, self-optimizing infrastructures. We hope this paper serves as a call to action for the database, systems, and ML communities to take on this problem together.

REFERENCES

- [1] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling Cross-Platform Data Processing - May The Big Data Be With You! -. *Proc. VLDB Endow.* 11, 11 (2018), 1414–1427.
- [2] Anastasios N. Angelopoulos and Stephen Bates. 2023. Conformal Prediction: A Gentle Introduction. *Found. Trends Mach. Learn.* 16, 4 (2023), 494–591. <https://doi.org/10.1561/2200000101>
- [3] Fu Bang. 2023. GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*. 212–218. <https://doi.org/10.18653/v1/2023.nlposs-1.24>
- [4] Kaustubh Beedkar, Bertty Contreras-Rojas, Haralampos Gavrilidis, Zoi Kaoudi, Volker Markl, Rodrigo Pardo-Meza, and Jorge-Arnulfo Quiané-Ruiz. 2023. Apache Wayang: A Unified Data Analytics Framework. *SIGMOD Rec.* 52, 3 (2023), 30–35.
- [5] Gohar Irfan Chaudhry, Esha Choukse, Haoran Qiu, Inigo Goiri, Rodrigo Fonseca, Adam Belay, and Ricardo Bianchini. 2026. Murakkab: Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. 567–587.
- [6] Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Trans. Mach. Learn. Res.* 2024 (2024).
- [7] Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. 2025. Multi-Agent Collaboration via Evolving Orchestration. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*.
- [8] Shaul Dar, Michael J. Franklin, Björn Þór Jónsson, Divesh Srivastava, and Michael Tan. 1996. Semantic Data Caching and Replacement. In *Proceedings of 22th International Conference on Very Large Data Bases, VLDB'96*. 330–341.
- [9] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. 2007. Adaptive Query Processing. *Foundations and Trends in Databases* 1, 1 (2007), 1–140.
- [10] Katerina Doka, Nikolaos Papaliou, Victor Giannakouris, Dimitrios Tsoumakos, and Nectarios Koziris. 2016. Mix 'n' match multi-engine analytics. In *2016 IEEE International Conference on Big Data (IEEE BigData 2016)*. 194–203. <https://doi.org/10.1109/BIGDATA.2016.7840605>
- [11] Anas Dorbani, Sunny Yasser, Jimmy Lin, and Amine Mhedhbi. 2025. Beyond Quacking: Deep Integration of Language Models and RAG into DuckDB. *Proc. VLDB Endow.* 18, 12 (2025), 5415–5418. <https://doi.org/10.14778/3750601.3750685>
- [12] Timo Eckmann, Matthias Jasny, Johannes Wehrstein, and Carsten Binnig. 2026. The Future Is Bespoke: Synthesizing One-Size-Fits-One DBMSs with LLM Coding Agents. *IEEE Data Eng. Bull.* 50, 1 (2026), 88–103.
- [13] Waris Gill, Mohamed Elidrisi, Pallavi Kalapatapu, Ammar Ahmed, Ali Anwar, and Muhammad Ali Gulzar. 2025. MeanCache: User-Centric Semantic Caching for Large Language Model Based Web Services. In *Proceedings of the 39th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1298–1310. <https://doi.org/10.1109/IPDPS64566.2025.00117>
- [14] Roman Heinrich, Xiao Li, Manisha Luthra, and Zoi Kaoudi. 2025. Learned Cost Models for Query Optimization: From Batch to Streaming Systems. *Proc. VLDB Endow.* 18, 12 (2025), 5482–5487. <https://doi.org/10.14778/3750601.3750699>
- [15] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How Good are Learned Cost Models, Really? Insights from Query Optimization Tasks. *Proc. ACM Manag. Data* 3, 3 (2025), 172:1–172:27. <https://doi.org/10.1145/3725309>
- [16] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proc. ACM Manag. Data* 2, 3 (2024), 186. <https://doi.org/10.1145/3654989>
- [17] Sebastian Kruse, Zoi Kaoudi, Bertty Contreras-Rojas, Sanjay Chawla, Felix Naumann, and Jorge-Arnulfo Quiané-Ruiz. 2020. RHEEMix in the data jungle: a cost-based optimizer for cross-platform systems. *VLDB J.* 29, 6 (2020), 1287–1310.
- [18] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017*. ACM, 1717–1722. <https://doi.org/10.1145/3035918.3054775>
- [19] Jiale Lao and Immanuel Trummer. 2026. GenDB: The Next Generation of Query Processing - Synthesized, Not Engineered. *CoRR abs/2603.02081* (2026). <https://doi.org/10.48550/ARXIV.2603.02081> arXiv:2603.02081
- [20] Chunwei Liu, Matthew Russo, Michael J. Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael J. Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpsest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *15th Conference on Innovative Data Systems Research, CIDR 2025*. www.cidrdb.org.
- [21] Hanwen Liu, Shashank Giridhara, and Ibrahim Sabek. 2025. Conformal Prediction for Verifiable Learned Query Optimization. *Proc. VLDB Endow.* 18, 8 (2025), 2653–2666.
- [22] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. 2004. Robust Query Processing through Progressive Optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 659–670. <https://doi.org/10.1145/1007568.1007642>
- [23] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*.
- [24] Liana Patel, Siddharth Jha, Melissa Z. Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Towards AI-Based Data Analytics with Accuracy Guarantees. *Proc. VLDB Endow.* 18, 11 (2025), 4171–4184. <https://doi.org/10.14778/3749646.3749685>
- [25] Sergey Redyuk, Zoi Kaoudi, Sebastian Schelter, and Volker Markl. 2024. Assisted design of data science pipelines. *VLDB J.* 33, 4 (2024), 1129–1153. <https://doi.org/10.1007/S00778-024-00835-2>
- [26] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. *Proc. VLDB Endow.* 18, 11 (2025), 4144–4157. <https://doi.org/10.14778/3749646.3749683>
- [27] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *Proc. VLDB Endow.* 18, 9 (2025), 3035–3048. <https://doi.org/10.14778/3746405.3746426>
- [28] András Strausz, Niels Pardon, and Ioana Giurgiu. 2025. A Learned Cost Model-based Cross-engine Optimizer for SQL Workloads. In *Proceedings of Workshops at the 51st International Conference on Very Large Data Bases*.
- [29] Pavel Sulimov, Claude Lehmann, and Kurt Stockinger. 2025. GenJoin: Conditional Generative Plan-to-Plan Query Optimizer that Learns from Subplan Hints. *Proc. ACM Manag. Data* 3, 4 (2025), 247:1–247:25. <https://doi.org/10.1145/3749165>
- [30] Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. 2025. Towards End-to-End Optimization of LLM-based Applications with Ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2025*. 1302–1316. <https://doi.org/10.1145/3676641.3716278>
- [31] Jeffrey Tao, Natalie Maus, Haydn Thomas Jones, Yimeng Zeng, Jacob R. Gardner, and Ryan Marcus. 2025. Learned Offline Query Planning via Bayesian Optimization. *Proc. ACM Manag. Data* 3, 3 (2025), 179:1–179:29. <https://doi.org/10.1145/3725316>
- [32] Noppanat Wadlom, Junyi Shen, and Yao Lu. 2026. Efficient LLM Serving for Agentic Workflows: A Data Systems Perspective. *Proc. ACM Manag. Data* 4, 3 (2026), 169:1–169:29. <https://doi.org/10.1145/3802046>
- [33] Jianxin Yan, Wangze Ni, Lei Chen, Xuemin Lin, Peng Cheng, Zhan Qin, and Kui Ren. 2025. ContextCache: Context-Aware Semantic Cache for Multi-Turn Queries in Large Language Models. *Proc. VLDB Endow.* 18, 12 (2025), 5391–5394. <https://doi.org/10.14778/3750601.3750679>
- [34] Qihan Zhang, Shaolin Xie, and Ibrahim Sabek. 2025. LIMAO: A Framework for Lifelong Modular Learned Query Optimization. *Proc. VLDB Endow.* 18, 11 (2025), 4546–4559. <https://doi.org/10.14778/3749646.3749712>
- [35] Yingli Zhou and Shu Wang. 2025. Towards the Next Generation of Agent Systems: From RAG to Agentic AI [Vision]. In *Proceedings of Workshops at the 51st International Conference on Very Large Data Bases*.