Query Optimization in the Wild: From Cross-platform Data Systems to Multi-Agent Systems

Zoi Kaoudi[0000−0003−4520−5360]

IT University of Copenhagen, Copenhagen, Denmark
zoi@itu.dk

Abstract. For decades the query optimizer has been the intellectual center of data management: the component that decides, on our behalf, how a declarative request should be executed. This talk traces a personal research path in which that same idea—separating what we want computed from how it is computed, and searching a large space of alternatives for a good plan—keeps resurfacing in new and larger guises. My journey passes through cross-platform data processing, where Apache Wayang made the execution engine the new unit of choice, moves on through the shift from hand-crafted to learned cost models and optimizers, and reaches today's frontier: the optimization of multi-agent data pipelines, where the optimizer must jointly choose pipeline topology, language models, and execution engines under genuinely multi-objective, Pareto trade-offs. Along the way a new objective has become impossible to ignore: energy. Treating energy as a first-class optimization goal, alongside latency, monetary cost, and accuracy, is both a scientific opportunity for our community and a societal necessity.

Keywords: Query optimization · Cross-platform data processing · Learned optimizers · Multi-agent data pipelines · Sustainable computing

1 Query Optimization as a Way of Thinking

The founding promise of data management is a declarative one: say *what* you want computed, not *how* to compute it, and let the system find a good way to do it. The query optimizer is what makes that promise real. It takes as input an abstract (non-specific) query[1] representation (a.k.a. logical plan) and outputs a concrete specification of how such a query should be executed (a.k.a. execution or physical plan). It is easy to treat the query optimizer as an implementation detail (i.e., a box between the parser and the executor) but we have come to see it as one of the most durable ideas in data systems: the separation of intent from execution, coupled with a principled search over a space of alternative plans.

Ioannidis abstracted the relational query optimizer's components in 1996 [9]. As shown in Fig. 1, a relational query optimizer requires three main components:

[1] The query here can be an SQL query, a dataflow job, an ML/AI pipeline, or any other user input.

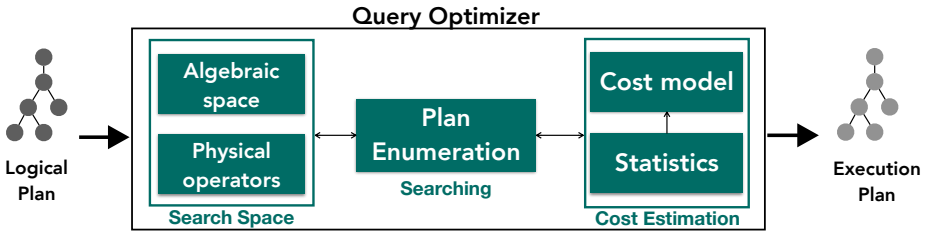


Fig. 1. Main components of a traditional relational query optimizer (adapted from [9])

a *search space* of candidate plans that are equivalent in result but not in cost; *cost estimation* that predicts how expensive each candidate will be; and *plan enumeration*, the strategy that searches this space for a good plan. Interestingly, whatever the era or setting, the same components remain valid well beyond the relational setting. What to choose among (search space), how to score the choices (cost estimation), and how to search among them (plan enumeration) can vary.

The thesis of this talk is that the idea of query optimization is remarkably portable. Every time our computational substrate or data has grown more heterogeneous, the optimizer question has returned, only bigger. When data took shapes the relational model was not built for (e.g., XML, RDF), the question was how to optimize over a different data model. When data outgrew a single machine, the question became *how to distribute*. When it outgrew a single system, the question became *which engine*. And now, as we build data pipelines out of stochastic AI agents, the question is returning in its most open-ended form yet. My aim here is to trace that arc through my own work, and to argue that the optimizer, properly reimagined, is the right lens for building AI systems that are not only efficient but also sustainable.

2 One Query, Many Engines

It is well-known that no single system is best at everything. Relational engines, dataflow systems, graph processors, streaming platforms, and specialized stores each dominate some region of the workload space. Recent data analytics tasks have become increasingly complex, often spanning several of these at once. *Cross-platform data processing* takes this heterogeneity as a first class design assumption rather than an inconvenience: a single analytical task should be free to span multiple engines, with each sub-computation running wherever it is cheapest or fastest.

This has been the central concern of Apache Wayang[2] [2], which started as a research prototype called Rheem [1].[3] In Wayang, a task is expressed as a platform-agnostic plan and its optimizer outputs a platform-specific plan. Fig. 2

[2] https://wayang.apache.org/
[3] Rheem was donated to the Apache Software Foundation and renamed Wayang.

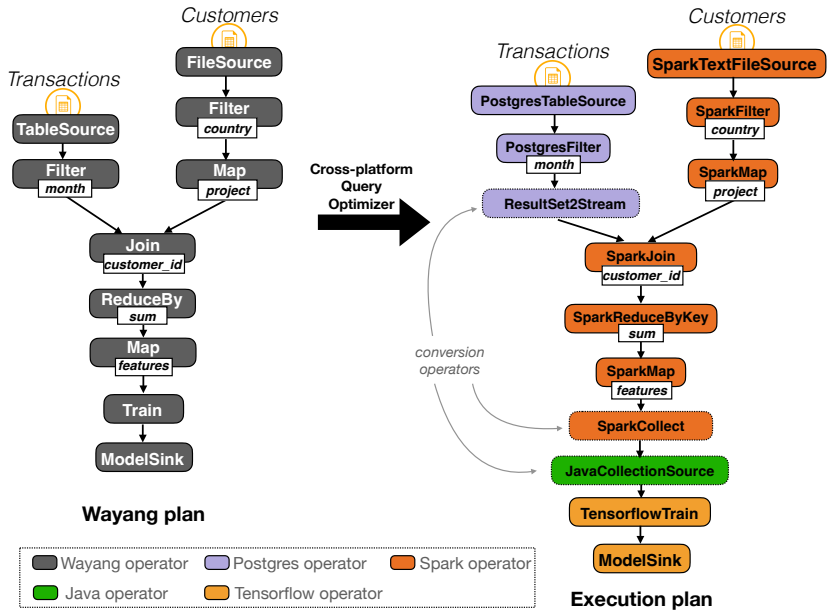


Fig. 2. An example cross-platform optimization task in Apache Wayang. *Left:* the platform-agnostic Wayang plan, which joins transactions with customers, aggregates per customer, extracts features, and trains a model. *Right:* one execution plan chosen by the cross-platform optimizer, which pushes the transaction filter down to Postgres, runs the join and aggregation on Spark, and trains on TensorFlow, inserting conversion operators (e.g., `ResultSet2Stream`, `SparkCollect`) to move data across engines.

shows an example analytical task that trains a machine learning model over two datasets (a *Transactions* table and a *Customers* file) to predict customer churn. The platform-agnostic Wayang plan (left) filters transactions by month, filters and projects customers by country, joins the two on `customer_id`, sums the amount spent per customer, extracts features, and trains a model. None of these operators names a concrete engine; deciding where each one runs is the cross-platform query optimizer's job. The execution plan on the right is one such choice: the transaction filter is pushed down to Postgres, where that table already resides; the customer processing, join, and aggregation run on Spark; and training runs on TensorFlow. Because consecutive operators may land on different engines, the optimizer additionally inserts *conversion operators* (e.g., `ResultSet2Stream`, `SparkCollect`) that move data across platform boundaries. This single task thus spans four systems.

We have built a cost-based optimizer that uses cost formulas, with profiling to tune the coefficients of these formulas, a novel plan enumeration algorithm, and a lossless pruning approach to navigate the search space [12]. Crucially, data movement is not modeled as file-based intermediate data exchange, which can be very costly, nor as a fixed one-to-one link between each pair of platforms,

which scales poorly, but as a channel conversion graph: a graph whose paths capture the many possible ways and intermediate data representations by which data can be moved from one platform to another. A dedicated data movement optimization process then selects the least costly path through this graph [13].

3 From Hand-Crafted to Learned Optimizers

An optimizer is only as good as its model of the environment it operates on. The cost model is that model, and hand-tuned cost models do not scale gracefully to a landscape of constantly changing, heterogeneous engines. This is what motivated us to replace the mathematical cost models of Wayang with a machine learning model. In [11], we show how a cross-platform query optimizer can make more informed decisions with less effort by utilizing a regression model and a vector-based plan enumeration algorithm.

Since then, learned query optimization has become a very active area, at least in academia, spanning the whole optimizer pipeline [5, 18, 6]. One line replaces individual estimators, learning cardinalities [19, 21] or predicting operator and query costs directly, increasingly with models designed to transfer across databases and workloads in a zero-shot fashion [23, 7]. A second line goes end-to-end, learning the plan-selection policy itself, as in Neo [17], Bao [16], and Balsa [22].[4] Even in this learned setting, the structure of Fig. 1 still holds. What changes is only what fills each box: cost estimation is now performed by a learned model rather than analytical formulas, its statistics are replaced by training data, and plan enumeration may be driven by reinforcement learning rather than a symbolic search.

The deeper shift, though, is not just replacing each component with a learned counterpart but rethinking the problem and the search. Starting from the relational setting, note that the optimizer never truly needs the cost of a plan; it needs only to know which plan is better. This means that instead of estimating cost, it suffices to estimate *rank*. We can thus reframe query optimization as a learning-to-rank problem [3], using a listwise learning-to-rank model together with an adaptive dynamic-programming enumeration algorithm that, at each level of the plan lattice, prunes the candidate set down to an adaptively chosen number of plans.

The second rethinking concerns enumeration itself. If enumeration is the bottleneck, why enumerate at all? A promising direction is to treat cross-platform query optimization not as a search problem but as a prediction to be learned, mapping a platform-agnostic plan directly to a good execution plan and reserving explicit search for offline refinement. Whether this can outperform classical enumeration across the full cross-platform space is one of the open questions that remain to be answered.

[4] The above list of references is by no means exhaustive.

4 The New Frontier: Multi-Agent Data Pipelines

Large language models have quietly introduced a new execution substrate. We now build *multi-agent data pipelines*: systems in which several agents, backed by different models and tools, collaborate to solve complex data tasks. Yet today's pipelines are largely ad hoc: hand-wired topologies, a single pre-chosen model, and a single execution engine behind the tools. The result is fragmented, redundant, and often startlingly expensive.

Seen through the optimizer's lens, this is the same problem again, generalized along three axes and against three objectives. The optimizer must choose the pipeline topology (which agents exist and how they connect), the model for each step (which LLM at which accuracy–cost point), and the execution engine for each operation. This calls for a new direction, which we term *data systems for agents*: systems in which the multi-agent pipeline itself is the object of optimization, analogous to earlier efforts that brought database principles to bear on machine learning pipelines [14]. This is the vision behind NOMA [10], a next-generation class of query optimizers for multi-agent systems that co-optimizes topology, model, and engine. The optimizer's task, moreover, is now inevitably multi-objective: a classical optimizer largely minimizes latency, while an optimizer for agentic workflows must balance three objectives, i.e., latency, accuracy, and monetary cost, navigating a Pareto frontier rather than optimizing a scalar.

Three things make this genuinely new. First, the space of operations is effectively unbounded: agents are not confined to a fixed algebra like relational operators, so the optimizer needs a far more open and extensible design. Second, the operators are stochastic: the same agent, on the same input, varies in latency, token usage, and output accuracy, so a point cost is not enough—the optimizer must reason about calibrated distributions over each objective. Third, the environment drifts: model prices, capabilities, and availability change week to week, so optimization must be continuous rather than one-shot. The prize for getting this right is large. In real multi-agent pipelines we have observed a cost gap of up to 153× between naive and well-optimized configurations.

There is, however, a deeper break. Through every stage of this talk, the structure of Fig. 1 held: optimization ran to completion and handed a finished plan to execution, cleanly separated. In the agentic setting, for the first time, that separation may have to go. Because an agent's output determines what the pipeline does next, the topology is data-dependent and cannot be fully fixed in advance; and because a step's cost and accuracy are revealed only once it has run, the estimates that drive optimization become available only during execution. Optimization can therefore no longer cleanly precede execution: the two must be interleaved, with the optimizer acting less as a compiler that emits a plan and more as a runtime controller.

5 Energy as a First-Class Optimization Objective

There is another objective that our field has either not paid attention to or has treated as an afterthought: energy. It can no longer be one. AI workloads are

projected to push data centers' electricity demand to around 945 TWh by 2030, comparable to the entire current electricity consumption of Japan [8]. When a single pipeline can differ by two orders of magnitude in energy depending on how it is configured, the choice of plan is no longer only a performance or economic question but also an environmental one.

Energy is not entirely new to our field: the database community raised energy-aware query optimization over a decade ago [4, 20, 15]. Even then, the picture was mixed. Some work showed that response time could be traded for energy, batching or slowing execution to reach more efficient points [15]; other work found that, within a single server, the fastest plan is typically also the most energy-efficient, leaving little room for a separate energy-aware optimizer [20]. Perhaps because of this ambiguity or because of the limited benefits, these ideas were never adopted in practice. The line of work largely went quiet. We can no longer afford that. In today's heterogeneous landscape, the question our community set aside is open and pressing again.

The encouraging news is that our community is unusually well-equipped to take this on. We already think natively in terms of cost models, search over alternatives, and multi-objective trade-offs; extending the objective vector to include Joules, and treating latency, monetary cost, accuracy, and energy as axes of a single *Pareto frontier*, is a natural continuation of what optimizers have always done. What it requires is deliberate: energy-aware cost primitives that generalize across engines, models, and hardware; optimization procedures that navigate a multi-objective frontier under user-defined budgets and quality constraints; and, not least, benchmarks that measure energy as seriously as we measure runtime. From this perspective, Green AI need not be a separate agenda from query optimization, but query optimization that accounts for all of its costs, energy included.

6 Outlook

The idea of query optimization—keep the query's intent separate from its execution and let an optimizer search the space of possible plans for a good one—has now stretched from a single machine to heterogeneous engines to fleets of stochastic agents. At each step the community was tempted to declare the optimizer a solved or secondary problem, and at each step the substrate changed underneath us and the problem returned, larger and more consequential. I expect this to continue. The open challenges I find most pressing are a scalable multi-objective optimizer for agentic pipelines; optimization that is robust to stochasticity and drift; and theory and benchmarks that treat energy as a first-class citizen. None of these is niche: together they describe how data analytics can remain not only powerful, but affordable and sustainable, in the age of AI. I look forward to working on them with this community.

Acknowledgments. The research described in this talk is the product of many collaborations. I am deeply grateful to the students, postdocs, and colleagues who shaped

it. In particular, I would like to thank Juri Petersen and Mads Sejer Pedersen for their ongoing work on cross-platform optimization, Xiao Li for ongoing work on learning-to-rank query optimization, and my collaborator Ioana Giurgiu for the work and fruitful discussions that gave life to this new line of research on multi-agent systems. This work was partially supported by the Carlsberg Foundation (CF-23-0836).

References

1. Agrawal, D., Chawla, S., Contreras-Rojas, B., Elmagarmid, A.K., Idris, Y., Kaoudi, Z., Kruse, S., Lucas, J., Mansour, E., Ouzzani, M., Papotti, P., Quiané-Ruiz, J., Tang, N., Thirumuruganathan, S., Troudi, A.: RHEEM: enabling cross-platform data processing - may the big data be with you! -. Proc. VLDB Endow. **11**(11), 1414–1427 (2018)
2. Beedkar, K., Contreras-Rojas, B., Gavriilidis, H., Kaoudi, Z., Markl, V., Pardo-Meza, R., Quiané-Ruiz, J.: Apache wayang: A unified data analytics framework. SIGMOD Rec. **52**(3), 30–35 (2023). https://doi.org/10.1145/3631504.3631510, https://doi.org/10.1145/3631504.3631510
3. Behr, H., Markl, V., Kaoudi, Z.: Learn what really matters: A learning-to-rank approach for ml-based query optimization. In: Datenbanksysteme für Business, Technologie und Web (BTW). LNI, vol. P-331, pp. 535–554 (2023)
4. Harizopoulos, S., Shah, M.A., Meza, J., Ranganathan, P.: Energy efficiency: The new holy grail of data management systems research. In: CIDR (2009)
5. Heinrich, R., Li, X., Luthra, M., Kaoudi, Z.: Learned Cost Models for Query Optimization: From Batch to Streaming Systems. Proc. VLDB Endow. **18**(12), 5482–5487 (2025)
6. Heinrich, R., Luthra, M., Wehrstein, J., Kornmayer, H., Binnig, C.: How good are learned cost models, really? insights from query optimization tasks. Proc. ACM Manag. Data **3**(3), 172:1–172:27 (2025)
7. Hilprecht, B., Binnig, C.: Zero-shot cost models for out-of-the-box learned cost prediction. Proc. VLDB Endow. **15**(11), 2361–2374 (2022)
8. International Energy Agency: Energy and AI. Tech. rep., IEA, Paris (2025), https://www.iea.org/reports/energy-and-ai
9. Ioannidis, Y.E.: Query optimization. ACM Comput. Surv. **28**(1), 121–123 (1996)
10. Kaoudi, Z., Giurgiu, I.: Rethinking query optimization for multi-agent systems [vision]. CoRR **abs/2512.11001** (2025), https://doi.org/10.48550/arXiv.2512.11001
11. Kaoudi, Z., Quiané-Ruiz, J., Contreras-Rojas, B., Pardo-Meza, R., Troudi, A., Chawla, S.: Ml-based cross-platform query optimization. In: ICDE. pp. 1489–1500 (2020)
12. Kruse, S., Kaoudi, Z., Contreras-Rojas, B., Chawla, S., Naumann, F., Quiané-Ruiz, J.: Rheemix in the data jungle: a cost-based optimizer for cross-platform systems. VLDB J. **29**(6), 1287–1310 (2020). https://doi.org/10.1007/s00778-020-00612-x, https://doi.org/10.1007/s00778-020-00612-x
13. Kruse, S., Kaoudi, Z., Quiané-Ruiz, J., Chawla, S., Naumann, F., Contreras-Rojas, B.: Optimizing cross-platform data movement. In: 35th IEEE International Conference on Data Engineering, ICDE. pp. 1642–1645 (2019). https://doi.org/10.1109/ICDE.2019.00162, https://doi.org/10.1109/ICDE.2019.00162
14. Kumar, A., Boehm, M., Yang, J.: Data management in machine learning: Challenges, techniques, and systems. In: SIGMOD. pp. 1717–1722 (2017)

15. Lang, W., Patel, J.M.: Towards eco-friendly database management systems. In: CIDR (2009)
16. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: Making learned query optimization practical. In: SIGMOD. pp. 1275–1288 (2021)
17. Marcus, R., Negi, P., Mao, H., Zhang, C., Alizadeh, M., Kraska, T., Papaemmanouil, O., Tatbul, N.: Neo: A learned query optimizer. Proc. VLDB Endow. **12**(11), 1705–1718 (2019)
18. Mparmparousis, K., Tsapelas, C., Koutrika, G.: What drives learned optimizer performance? A systematic evaluation. In: EDBT. pp. 605–618 (2026)
19. Sun, J., Zhang, J., Sun, Z., Li, G., Tang, N.: Learned cardinality estimation: A design space exploration and a comparative evaluation. Proc. VLDB Endow. **15**(1), 85–97 (2021)
20. Tsirogiannis, D., Harizopoulos, S., Shah, M.A.: Analyzing the energy efficiency of a database server. In: SIGMOD. pp. 231–242. ACM, New York, NY, USA (2010)
21. Wang, X., Qu, C., Wu, W., Wang, J., Zhou, Q.: Are we ready for learned cardinality estimation? Proc. VLDB Endow. **14**(9), 1640–1654 (2021)
22. Yang, Z., Chiang, W., Luan, S., Mittal, G., Luo, M., Stoica, I.: Balsa: Learning a query optimizer without expert demonstrations. In: SIGMOD. pp. 931–944 (2022)
23. Zhang, Q., Xie, S., Sabek, I.: LIMAO: A framework for lifelong modular learned query optimization. Proc. VLDB Endow. **18**(11), 4546–4559 (2025)