Carnot: Interpretable, Interactive, and Optimized Execution of Deep Research Queries

Matthew Russo
MIT
mdrusso@csail.mit.edu

Yash Agarwal
MIT
yashaga@mit.edu

Tianyu Li
MIT
litianyu@mit.edu

Zhuohan Gu
MIT
zgu15@mit.edu

Michael Cafarella
MIT
michjc@csail.mit.edu

Omar Khattab
MIT
okhattab@mit.edu

Tim Kraska
MIT
kraska@mit.edu

Samuel Madden
MIT
madden@csail.mit.edu

ABSTRACT

Enterprises increasingly seek to query data lakes using natural language via AI-driven tools like semantic operators or deep research agents. However, the latter operates as an opaque black box, hiding its intermediate reasoning and data retrieval steps, and failing to expose controls for managing API costs and execution latency. Meanwhile, the former can be prohibitively expensive for enterprise-scale data lakes. Consequently, analysts using these systems lack the agency to intercept hallucinated premises, verify intermediate results, or correct the system's trajectory. We present Carnot, an interactive execution engine for AI-driven analytics. Carnot compiles natural language requests into physical execution graphs and surfaces them through an interactive notebook interface. Rather than waiting blindly for a final output, users can critique the plan, incrementally execute operators, inspect intermediate data, or directly edit the underlying code or semantic operator instructions. Carnot's query optimizer will optimize the query with respect to cost or latency constraints provided by the user. Our demo will showcase how Carnot helps users achieve efficient and verifiable insights on workloads motivated by real enterprise use cases.

PVLDB Reference Format:
Matthew Russo, Yash Agarwal, Tianyu Li, Zhuohan Gu, Michael Cafarella, Omar Khattab, Tim Kraska, and Samuel Madden. Carnot: Interpretable, Interactive, and Optimized Execution of Deep Research Queries. PVLDB, 19(12): 4642 - 4645, 2026.
doi:10.14778/3827998.3828086

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mitdbg/carnot.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828086

1 INTRODUCTION

Organizations increasingly deploy AI-driven query systems to extract insights from unstructured and multi-modal data lakes. For example, a business analyst might issue a natural language query to a Deep Research system which can autonomously explore data, write code, and interpret execution results in a loop to produce a final answer to their question [1, 2, 5, 7, 11]. The human's job is primarily to verify that the output is correct. However, these systems are often *black boxes*: they accept a query and emit a result, exposing no control over the intermediate chain of retrievals, transformations, and LLM invocations that produced it.

Consider a consumer product analyst who asks an AI system to *"summarize the top complaints about our products and identify which attributes drive negative reviews the most."* Suppose the system returns a polished paragraph claiming there are only minor complaints and that three attributes drive most of the negative reviews. The analyst has no way of checking which documents were retrieved, whether the LLM's filter for "complaints" silently excluded important data sources, or whether one of the attributes was hallucinated without support from the source data. Any serious analyst, before accepting the result, must carefully read through the reasoning trace (if one is available), and potentially verify parts of the analysis by hand, negating the benefits of automation.

The problem is not merely one of trust; it is one of *control*. In contrast to Deep Research systems, semantic query processing engines [4, 6, 8–10, 12] execute SQL-like semantic query plans. However, once a query is submitted, the user can do little to detect faulty operators mid-execution, apply insights unearthed from partial execution, or selectively patch and re-execute parts of the pipeline. Returning to our example, the analyst would only realize their query is ill-specified *after* paying the full cost of query execution. They must then manually edit their semantic operator code, and discover whether the fix works only after paying a second time.

In short, today's AI-driven query systems fail to provide both *visibility*—the ability to inspect every prompt, code block, and intermediate result before, during, and after execution—and *steerability*—the ability to intervene at any step without discarding prior computation. To address this challenge, we present Carnot

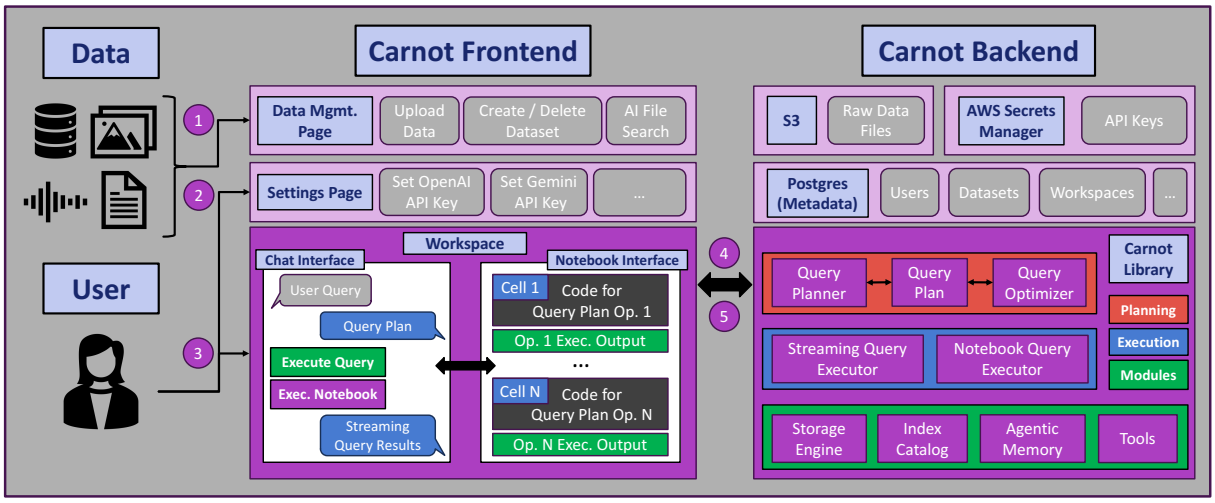

Figure 1: An overview of the Carnot system. Users create datasets from uploaded raw data files (Step 1) and provide API keys which may be used by Carnot to perform LLM-based data processing (Step 2). (Local storage and models are also supported for private deployments.) Users execute queries against their dataset(s) in natural language via a chat interface or via an equivalent representation of the physical plan in a notebook (Step 3). Any feedback provided in the chat and / or edits made to the notebook will be sent to Carnot's query planner which will re-optimize the query plan in response (Steps 4 and 5).

(Figure 1), an interactive execution engine for NL queries over unstructured data lakes. When a user issues a natural language query, Carnot's automated planner compiles it into a Directed Acyclic Graph (DAG) of physical operators, combining semantic operators (e.g., LLM-based filters and extractors) with agentic operators (e.g., data exploration, Python code generation and execution). Unlike prior semantic operator systems, Carnot materializes the DAG as a notebook interface where each cell corresponds to a single operator. Users can inspect any cell at multiple levels of abstraction—a summary of the operator's intent, the generated code or prompt, or the raw input/output data pairs—and execute the graph one operator at a time. When a user edits a cell to correct a mistake, Carnot's optimizer will automatically re-optimize and re-execute the cell while reusing the results of unaffected upstream cells.

This feedback loop is the core technical enabler of Carnot's interactive model. In our previous example, the analyst first inspects the proposed plan of the agent and corrects any faulty assumptions in plan-space. They then execute the first few operators and discover that the LLM's filter only looks at the "ratings" column and excludes the free-form "review text" field. They edit the filter prompt directly in its notebook cell. Carnot resumes execution from the corrected point—leaving the upstream data retrieval and preprocessing steps untouched. The result is a system that combines the optimizable structure of a database query plan with the iterative flexibility of an agent, while keeping the human in the loop at every step.

Our demo showcases Carnot's interactive planning and execution over datasets from academic benchmarks [3] and enterprise workloads motivated by industry partners. To see transparent execution in action, attendees will issue queries and inspect the resulting DAGs before any computation begins; they may drill into any operator to view its generated code, prompt, or a natural language summary of its intent. To experience steerability, attendees will execute a query step-by-step, intercept a faulty intermediate result, and edit the corresponding notebook cell; Carnot will then resume execution from the corrected point without re-running upstream operators. Finally, to highlight Carnot's cost-aware optimization, attendees will configure explicit cost and latency constraints in the interface; Carnot will show how the planner dynamically swaps operator implementations to meet their budget.

2 SYSTEM OVERVIEW

2.1 Carnot Frontend

Carnot's data management page enables users to upload raw data files (e.g., a `.zip` archive) to Carnot's data management page and organize them into one or more named *datasets*—for example, a collection of research papers, a set of supplier invoices, or a dataset of product reviews (Step 1 in fig. 1). Each dataset may be annotated with a description of its contents, which Carnot's query planner consults when generating execution plans. The user also registers API keys for one or more supported LLM providers (Step 2).

To issue a query, the user selects the relevant datasets and types a natural language question into the chat interface (Step 3). Carnot's backend responds with a physical query plan, displayed in complementary views: a linearized bullet-list summary in the chat window, and a visual DAG beside it. Crucially, execution has not yet begun—Carnot pauses here to let the user critique the proposed plan before any additional tokens are spent. The user may issue follow-up instructions to revise the plan, execute it end-to-end with results streamed back to the chat, or open it as an interactive *notebook*.

The notebook is the primary vehicle for Carnot's visibility and steerability guarantees. Each cell corresponds to a single physical operator in the DAG, obtained by topologically sorting the plan so that data-source operators appear first. Users can inspect any cell at three levels of abstraction: a natural language summary

of the operator's intent, the generated code or LLM prompt, and the raw input/output data pairs. Operators can be executed one cell at a time, and any cell may be edited in-place—for instance, to revise a semantic filter's predicate or to swap an LLM call for a deterministic Python expression. When the user commits an edit, Carnot re-executes the updated operator while reusing the cached results of unaffected upstream cells (Steps 4 and 5). Users may also set explicit cost and latency budgets; Carnot's optimizer respects these constraints by dynamically selecting cheaper or faster operator implementations.

2.2 Carnot Backend

The backend is responsible for managing the metadata and state that support the frontend and dispatching query planning, optimization, and execution requests to the Carnot library. For the former, Carnot stores user identifiers, dataset definitions, and the state of each conversation–notebook pair (together called a *workspace*) in a Postgres database. User-uploaded data files are placed in a private S3 bucket. (Carnot's storage and LLM abstractions can be swapped for local filesystem(s) and local models in deployments where privacy is a concern.) Using stored state, the backend translates user actions into calls against the Carnot library, described next.

2.3 Carnot Library

The library provides modular components for planning, optimizing, and executing queries. Its core abstraction is a `Dataset` class—a wrapper around a list of files that exposes a set of chainable operators. These include *semantic operators* (LLM-based maps, filters, joins, etc.) and *agentic operators* (data exploration, Python code generation and execution, general-purpose reasoning). A user or agent constructs a query plan by chaining operators into a DAG over one or more input datasets; data is only materialized when `Dataset.run()` is called. The `Dataset` class is JSON serializable, which enables it to be directly visualized and edited in the notebook.

The `QueryPlanner` is an LLM agent with few-shot demonstrations for composing plans in Carnot's operator vocabulary. To avoid premature use of expensive LLM operators, the planner first invokes a *data discovery* subagent that samples the dataset, performs schema analysis, and summarizes dataset structure. The `QueryOptimizer` follows, inspired by the Cascades-style optimizer of Abacus [8]. It shares a similar set of implementation and transformation rules, but differs in its use of an LLM agent to select the final physical plan from the estimated Pareto frontier of cost, latency, and quality. The library also contains a streaming and notebook execution layer, a storage engine with caching support, and an index catalog for managing semantic indices.

2.4 Putting It Together

Figure 2 illustrates a complete Carnot session for the query *"Which of these papers is about AI?"* In the center panel, the chat interface displays the query alongside the linearized plan and visual DAG; the user can inspect the plan's logic before choosing to execute it in the chat or open it as a notebook. On the right, the notebook's code cells reveal the Carnot library calls—including the semantic filter's condition and the semantic map's field description—giving the user full transparency into each operator's behavior. On the left, the chat interface shows per-operator results with cost annotations (e.g., $0.0087 for the semantic map step), a final answer, and a downloadable CSV of the output (similar outputs are available in the notebook).

3 DEMONSTRATION SCENARIOS

Our demonstration is organized around three scenarios that let attendees experience Carnot's capabilities first-hand on the provided datasets, using the interface shown in Figure 2.

3.1 Datasets

We will provide attendees with access to two datasets. The first dataset is an anonymized set of 1,000 consumer product reviews from an industry partner. We anonymized the reviews by stripping the real product names and product identifiers from the dataset, while also having an LLM paraphrase the review text and titles. We inspected the anonymized reviews to ensure that their sentiment (and content) remained faithful to their original form. The second dataset will be the `legal` workload from KramaBench [3] an academic benchmark for data analytics.

3.2 Transparent Plan Inspection

This scenario highlights Carnot's ability to inspect every operator before any computation begins. The attendee will issue a free-form natural language query—for example, *"summarize the top complaints about our products and identify which attributes drive negative reviews the most."* against our enterprise dataset. Carnot will return a logical plan displayed both as a nested summary in the chat and as a DAG. The attendee will examine the DAG of operators, including per-operator arguments such as filter conditions and group by fields, in order to understand the logic of the query plan. Before executing a single operator, the attendee can verify that the plan retrieves data from the intended sources, that the filter criteria match their expectations, and that the aggregation logic is sound.

3.3 Steering via the Chat Interface

This scenario demonstrates high-level *steerability* through conversational feedback. Continuing from the plan above, the attendee notices the filter operator only inspects the "ratings" field. Rather than editing the operator directly, they type a follow-up instruction in the chat: *"Be sure to also consider any negative comments left in the 'review text' field."* Carnot's planner treats this instruction as a constraint, re-plans the affected operators, and presents an updated DAG in the chat. The attendee can then execute the revised plan in the chat, streaming per-operator results back to the chat interface. If the results look satisfactory, they accept the output. If further refinement is needed, they may continue the conversation—or drill down into different views for fine-grained control.

3.4 Editing via the Notebook Interface

This scenario showcases Carnot's most distinctive capability: cell-level editing with query re-execution and re-optimization. The attendee opens the current query plan as a notebook, executes the first few cells, and inspects the intermediate outputs. Suppose the semantic filter's LLM prompt produces false negatives—it classifies legitimate complaints as neutral feedback. The attendee edits the

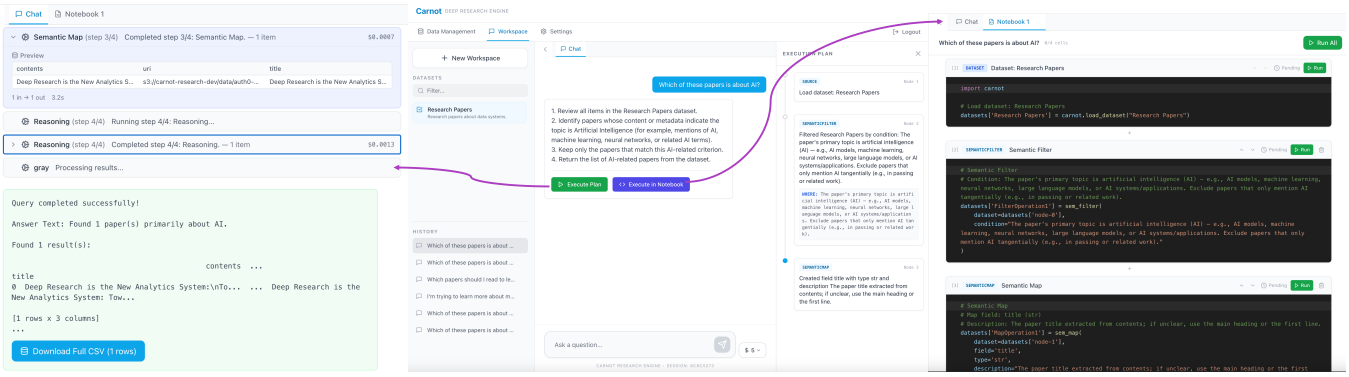

Figure 2: A CARNOT workspace answering the query *"Which of these papers is about AI?"* Center: the chat interface with the linearized plan and visual DAG. Right: the notebook's code cells, exposing the CARNOT library calls that implement each operator. Left: the finished query execution showing per-operator results, cost annotations, and the final output.

cell directly to make sure the prompt instruction captures forms of feedback it previously misclassified. Upon saving the edit, the updated cell appears in the notebook and CARNOT's optimizer re-optimizes the implementation of the corresponding operator. Upstream cells whose results remain valid are *not* re-executed. The attendee then resumes execution from the corrected cell, iterating on individual operators until the final output meets their standards.

To highlight cost-aware optimization, the attendee will also configure explicit cost and latency constraints via the interface. CARNOT will show how the planner dynamically swaps operator implementations—e.g., replacing a large frontier model with a smaller, cheaper one for a low-precision filter—to meet the specified budget while preserving result quality where it matters most.

4 LIMITATIONS AND FUTURE WORK

CARNOT does not provide guarantees on plan accuracy or correctness. We could modify the optimizer to provide statistical guarantees with respect to an oracle [6], but this is a soft guarantee since the oracle can also be wrong. Alternatively, better interfaces may improve users' ability to verify plan correctness. In the future, we also plan to extend CARNOT to support search over the web (akin to traditional Deep Research) and general tool calling. We also aim to explore improvements to the notebook interface which can better highlight the non-linear nature of most plan DAGs.

ACKNOWLEDGMENTS

We are grateful for the support from the DARPA ASKEM Award HR00112220042, the ARPA-H Biomedical Data Fabric project, NSF DBI 2327954, a grant from Liberty Mutual, a Google Research Award, and the Amazon Research Award. Additionally, our work has been supported by contributions from Amazon, Google, and Intel as part of the MIT Data Systems and AI Lab (DSAIL) at MIT, along with NSF IIS 1900933. This research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Anthropic. 2025. *Claude takes research to new places.* Retrieved July 31, 2025 from https://www.anthropic.com/news/research

[2] Google. 2025. *Gemini Deep Research.* Retrieved July 31, 2025 from https://gemini.google/overview/deep-research/

[3] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, SIVAPRASAD SUDHIR, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Mike Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. 2026. KRAMABENCH: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. In *The Fourteenth International Conference on Learning Representations.* https://openreview.net/forum?id=fZfUdeCC5X

[4] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. CIDR.

[5] OpenAI. 2025. *Deep Research System Card.* Retrieved July 31, 2025 from https://cdn.openai.com/deep-research-system-card.pdf

[6] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proc. VLDB Endow.* 18, 11 (July 2025), 4171–4184. https://doi.org/10.14778/3749646.3749685

[7] Perplexity. 2025. *Introducing Perplexity Deep Research.* Retrieved July 31, 2025 from https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research

[8] Matthew Russo, Sivaprasad Sudhir, Gerardo Vitagliano, Chunwei Liu, Tim Kraska, Samuel Madden, and Michael Cafarella. 2025. Abacus: A Cost-Based Optimizer for Semantic Operator Systems. arXiv:2505.14661 [cs.DB] https://arxiv.org/abs/2505.14661

[9] 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 (May 2025), 3035–3048. https://doi.org/10.14778/3746405.3746426

[10] Shreya Shankar, Bhavya Chopra, Mawil Hasan, Stephen Lee, Bjoern Hartmann, Joseph Hellerstein, Aditya Parameswaran, and Eugene Wu. 2025. Steering Semantic Data Processing With DocWrangler. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25).* Association for Computing Machinery, New York, NY, USA, Article 84, 18 pages. https://doi.org/10.1145/3746059.3747625

[11] SmolAgents. 2025. *Open-source DeepResearch – Freeing our search agents.* Retrieved July 31, 2025 from https://huggingface.co/blog/open-deep-research

[12] Lindsey Linxi Wei, Shreya Shankar, Sepanta Zeighami, Yeounoh Chung, Fatma Ozcan, and Aditya G. Parameswaran. 2026. Multi-Objective Agentic Rewrites for Unstructured Data Processing. arXiv:2512.02289 [cs.DB] https://arxiv.org/abs/2512.02289