Deep Research is the New Analytics System: Towards Building the Runtime for AI-Driven Analytics

Matthew Russo
MIT
Cambridge, MA, USA
mdrusso@csail.mit.edu

Tim Kraska
MIT
Cambridge, MA, USA
kraska@mit.edu

ABSTRACT

With advances in large language models (LLMs), researchers are creating new systems that can perform AI-driven analytics over large unstructured datasets. Recent work has explored executing such analytics queries using semantic operators—a declarative set of AI-powered data transformations with natural language specifications. However, even when optimized, these operators can be expensive to execute on millions of records and their iterator execution semantics make them ill-suited for interactive data analytics tasks. In another line of work, Deep Research systems have demonstrated an ability to answer natural language question(s) over large datasets. These systems use one or more LLM agent(s) to plan their execution, process the dataset(s), and iteratively refine their answer. However, these systems do not explicitly optimize their query plans which can lead to poor plan execution. In order for AI-driven analytics to excel, we need a runtime which combines the optimized execution of semantic operators with the flexibility and more dynamic execution of Deep Research systems. As a first step towards this vision, we build a prototype which enables Deep Research agents to write and execute optimized semantic operator programs. We evaluate our prototype and demonstrate that it can outperform a handcrafted semantic operator program and open Deep Research systems on two basic queries. Compared to a standard open Deep Research agent, our prototype achieves up to 1.95x better F1-score. Furthermore, even if we give the agent access to semantic operators as tools, our prototype still achieves cost and runtime savings of up to 76.8% and 72.7% thanks to its optimized execution.

1 INTRODUCTION

Enabling (complex) analytics over large unstructured data lakes has been a long-standing goal of data systems research. Traditional OLAP databases [8, 11, 22, 29], while great for structured queries, have struggled to support these workloads. SQL as the primary language is not sufficient to interact with the vast variety of unstructured data. To address this limitation, researchers — mostly from the database community – have proposed semantic operators—AI-powered analogs to relational operators inspired by classical query optimization techniques. Systems like Palimpzest [17, 24], LOTUS [20], DocETL [26], Galois [25], and others [1, 18, 30] enable developers to apply operations such as AI-driven maps, filters, joins, and aggregations—specified in natural language—over large unstructured datasets. Early research shows that these semantic operators can be effectively optimized for a variety of tasks, including information extraction, summarization, ranking, and more [24].

In contrast, the AI-community went down a different route: Deep Research systems [3, 9, 16, 19, 21, 28, 32], which are able to create (Python) code on the fly to query structured and unstructured data alike. Open Deep Research systems, like HuggingFace's SmolAgents implementation [28], use so-called "CodeAgents" [27] that can reason, write code, and use tools in an iterative fashion to execute a natural language instruction.

When compared at a high-level, OLAP systems, semantic operator systems, and Deep Research systems are all remarkably similar. Given a user's declarative query, each system creates a query plan, executes it over a dataset, and returns the final result. All systems have access to many query plans, but ideally prefer ones which are fast, cheap, and accurate. Finally, in each setting, predicting plan performance is non-trivial, and optimizing the query plan is critical for good performance. However, Deep Research systems are in many ways much more flexible; not only do they take natural language as an input but also often used techniques like self-reflection and incremental planning, which are best compared to adaptive query processing on steroids.

Challenges. However, semantic operator systems and Deep Research systems each have their own weaknesses which make neither framework fully adequate for answering unstructured analytics queries. For example, the left-hand side of Figure 1 shows a query from the recently published Kramabench [15] benchmark. The dataset consists of 132 files with statistics on fraud, identity theft, and other consumer reports, and the query asks the analytics system to compute the ratio of identity theft reports over a two year span. A hand-crafted semantic operator program tries to answer the query by filtering for files with statistics on identity theft reports, before using a map to compute the ratio.

The main issue with this program is that semantic operators' execution semantics can only process one file at a time. In addition to being expensive and slow (the majority of files in this dataset have state-level statistics which can be ignored for this query) this makes writing a correct program for this query difficult. A handful of files contain 2024 statistics and inferring which file is correct for this query requires reasoning across multiple files simultaneously. As a result, this program is expensive, slow, and often produces incorrect output(s).

While Deep Research systems can process the previous query with relative ease, they also struggle on other queries which are simple for semantic operators. For example, the right-hand side of Figure 1 shows a query from the Enron email dataset [7]. This query asks the analytics system to return all emails with firsthand discussion of certain business transactions. The Deep Research

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that you attribute the original work to the authors and CIDR 2026. 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, Chaminade, USA

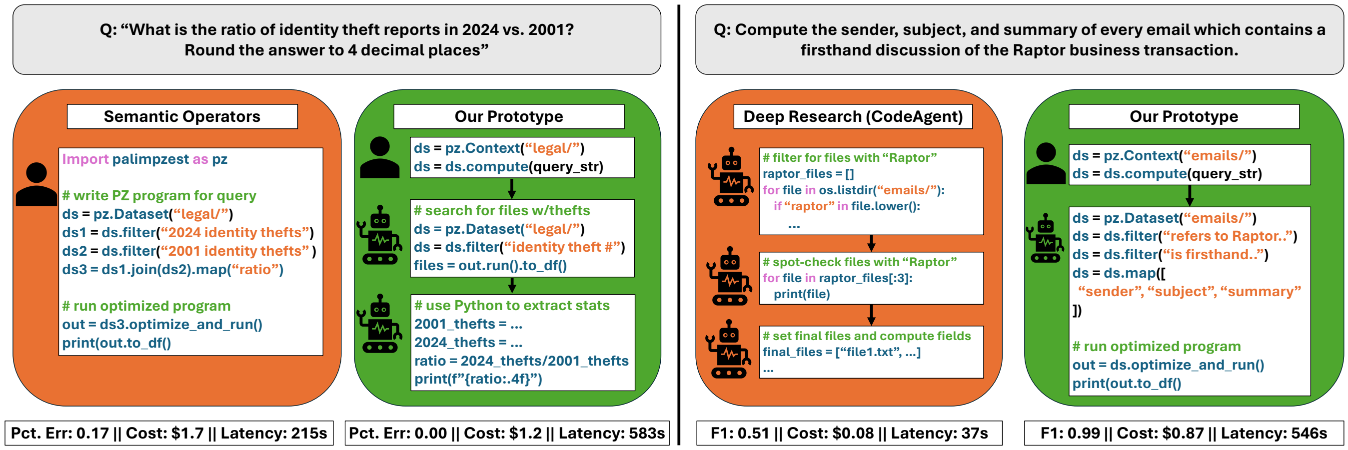

Figure 1: (Left) An example query from the Kramabench dataset which a handcrafted semantic operator program struggles to perform well on. Our prototype iterates between executing optimized semantic operator programs and writing Python code to identify the correct statistics and compute the final ratio. (Right) An example query on the Enron email dataset. An open Deep Research system achieves low recall (and F1-score) because it filters the data using simplistic Python code and only returns the few emails which it manually validates. Our prototype writes an optimized semantic operator program to process the entire dataset thus improving recall and F1-score.

system achieves high precision and low recall by writing Python code to search for certain keywords and then manually reading and verifying a few of those emails. By comparison, a handcrafted semantic operator program can answer this query almost perfectly.

While the Deep Research system's high-level query plan—filter for potentially relevant emails and verify that they satisfy the query—is not unreasonable, its execution of the query plan is poor. Especially when processing large datasets, we observe that Deep Research agents have a tendency to "take shortcuts" and do not consider expensive strategies such as reading every file to apply the filter predicate. Another common failure mode involves the agent terminating prematurely before it has fully executed its plan, which has also been observed elsewhere [4].

Goal. Hence we argue, that an ideal runtime for AI-driven analytics would utilize the strengths of Deep Research, semantic operators, and OLAP analytics systems. This runtime would share Deep Research's flexibility to write an initial query plan, iteratively execute Python code and use tools, and dynamically update its query plan in response to observations. It would also share semantic operator systems' ability to optimize their query plans automatically, using cost-based optimization [24], query rewrites [26], and physical operator optimizations [20]. Finally, this runtime should leverage structured information, possibly generated from unstructured data, which it can then query using SQL. This functionality is especially important in analytics settings where many queries are issued against the same data lake and future queries can reuse structured tables which are generated to answer previous queries.

Our Approach. As a first step towards building a unified runtime for AI-driven analytics, we created a prototype which aims to combine the flexibility of Deep Research with the optimized execution of semantic operators and SQL. Our prototype extends the Palimpzest framework [17] in three key ways. First, we introduced two new semantic operators: `compute` and `search`, which are physically implemented with agents that can plan their execution, write code, and use tools. Crucially, each operator is provided with a tool that can execute a natural language instruction with an optimized semantic operator program. This imbues each agent with the ability to execute optimized query plans as part of its more dynamic planning and execution.

For example, on the right-hand side of Figure 1, our prototype implements the Enron email query by passing the query string to a `compute` operator. The operator performs a few data exploration steps (not shown) before invoking its tool for writing a semantic operator program. That program ultimately executes with near-perfect accuracy, and—as shown in our evaluation in Section 4—is more efficient than providing an open Deep Research system with unoptimized semantic operators as tools.

Second, we extended Palimpzest's `Dataset` abstraction, to support access methods beyond iteration over a set of records. Specifically, we added support for users to provide their own indexing and top-k methods on top of their custom datasets. This more general abstraction for data access, which we call a `Context`, also contains a natural language description of the data it encompasses. Finally, users may also add their own tools to the `Context` to enable agents to interact with data in more bespoke ways. Such tools can include additional data access methods (e.g., a web search tool or a secondary index over a data lake), as well as methods for data cleaning, data visualization, etc.

Third, inspired by the use of materialized views in OLAP databases, we introduce a mechanism for indexing `Contexts` that are materialized during query execution. Each time a `compute` or `search` operator executes an instruction it generates a new `Context`, which is akin to a materialized view over the original `Context`. In an effort to leverage past query execution(s) to optimize new queries, we enabled Palimpzest's query optimizer to retrieve previously materialized `Contexts` which achieve high similarity to new `compute` and `search` instructions.

Contributions. In this paper we present a vision for building a new runtime for AI-driven analytics that combines the efficiency of SQL and semantic operators with the power and flexibility of Deep Research systems. In particular, we:

- Extend Palimpzest to include `search` and `compute` operators which support iterative execution patterns commonly found in Deep Research. (Section 2.)
- Introduce a new `Context` abstraction, which enables more dynamic access patterns in Palimpzest. (Section 2.)
- Describe physical and logical optimizations for `search` and `compute` operators. (Section 3.)
- Demonstrate that our prototype can execute an evaluation query that semantic operators struggle to process. (Section 4).
- Demonstrate that our prototype achieves 1.95x better F1-score than an open Deep Research system on a query from Kramabench, and achieves cost and runtime savings of 76.8% and 72.7%, respectively, relative to a Deep Research system which uses semantic operators as tools. (Section 4).

2 OVERVIEW

We present an overview of our prototype. First, we provide a quick background on implementing analytical queries with semantic operators and Deep Research systems. Then we describe our new `Context` abstraction as well as the `search` and `compute` operators which enable agents to execute optimized semantic operator programs. Finally, we discuss the addition of a `ContextManager` which indexes materialized `Contexts` so that they may be used to optimize future queries.

2.1 Analytical Queries with Semantic Operators and Deep Research systems

Recent work has focused on using semantic operators to execute analytical queries over unstructured datasets [1, 17, 18, 20, 25, 26, 30]. Unlike relational operators which are specified with SQL expressions, these operators are specified in natural language and are useful for a range of document processing tasks [17, 20, 24]. However, similar to their relational counterparts, semantic operators have iterator execution semantics which make them inefficient for many queries. For example, in the semantic operator program on the left-hand side of Figure 1, the first semantic filter will process all 132 files in the dataset even after it has found the number of identity thefts in the year 2024.

By contrast, Deep Research systems are capable of writing execution plans which enable them to more efficiently access the data they need. For example, on the same query a SmolAgents CodeAgent will list all 132 files and use information in the filename to determine which file(s) to read. However, while Deep Research agents are adept at writing query plans, their execution of these plans is often suboptimal. For example, an agent may generate a plan to read every file until it finds the file with identity thefts in 2024, and then give up on reading the dataset after the fourth or fifth file. These observations led us to consider whether we could combine the optimized execution of semantic operators with the planning and dynamic execution of Deep Research agents.

2.2 Context Class

In order to support more efficient query execution patterns, we created a new `Context` abstraction in Palimpzest. This abstraction, illustrated in Figure 2, inherits from Palimpzest's `Dataset` class which means it automatically supports iterator execution. Additionally, the `Context` class also exposes an index method which allows programmers to define key-based point lookups and/or vector-based search on their own dataset(s). Programmers can also define custom tools which are relevant for their dataset. For example, a programmer working with time-series data might expose tools for accessing sequence data at different sample frequencies. Finally, the `Context` class also has a description field ("desc"), which contains a natural language description of the dataset's contents.

While supporting various access methods, tools, and a description may seem arbitrary, these features are essential for enabling agents to interact with the dataset. By exposing iterator and index-based access methods, agents can make informed decisions about which access pattern to use when executing a given query. In the example from before, an agent may decide to use an embedding-based lookup to search for files with the number of identity thefts in 2024. Then, based on the result of its search, it can either use the information that was returned or resort to iterating over the entire dataset. Furthermore, detailed descriptions can assist agents in deciding which file(s) within a `Context` are likely to contain the information they need.

2.3 Compute and Search Operators

While the `Context` abstraction may be useful for agents, leveraging this abstraction requires building an operator which actually uses an agent. To address this, we set out to create a logical operator which could be compiled by Palimpzest and physically implemented with an agent. Ultimately, our experience trying to execute various queries from Kramabench [15] led us to create two logical operators: one for `compute` and one for `search`. Each operator takes a `Context` as input, but they have slightly different semantics: the `compute` operator seeks to generate a specific output, whereas the `search` operator tries to find information which can be used to enrich a `Context`'s description. Finally, given the widespread adoption of SmolAgent's CodeAgent, we decided to have a CodeAgent to serve as our physical implementation for each logical operator.

We illustrate the functionality of these new operators in Figure 2. The program on the left-hand side of the diagram creates an initial `Context` object, which includes a natural language description of the data along with support for indexing and tool use. The `search` operator takes the `Context` as input, and uses its description, tools, and data access methods to search for information on identity thefts. The operator outputs a new `Context` and updates its description to contain (a summary of) its search execution trace. This `Context` is then fed into a `compute` operator, which reads the description and uses the tools and access methods to compute the number of thefts in 2024.

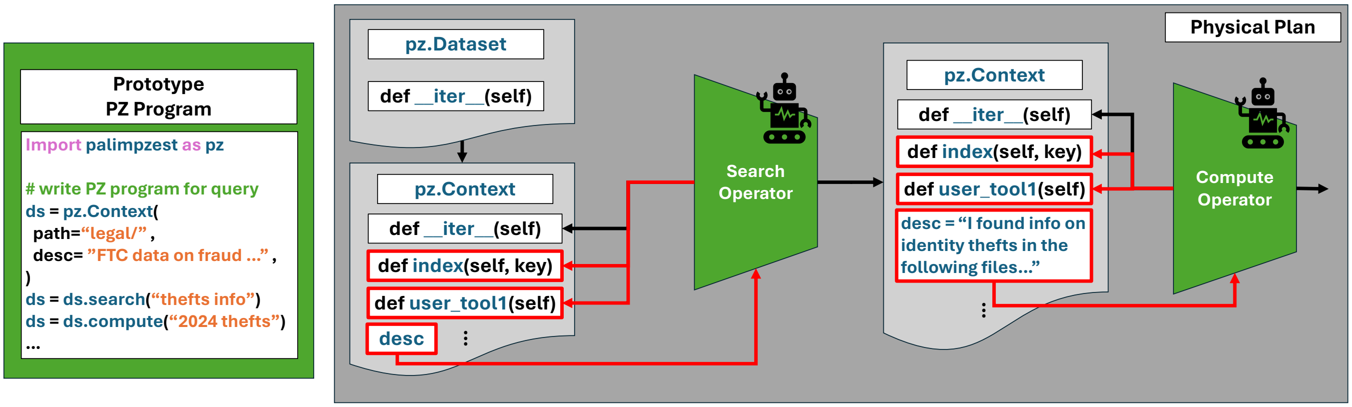

Figure 2: Overview of a Palimpzest (PZ) program and its physical plan. The program creates an initial `Context` object, which the search operator uses to look for information on identity thefts. The search operator outputs a new `Context` object with an updated description reflecting the results of its search. Finally, the `compute` operator uses this intermediate `Context` to compute the number of identity thefts in 2024.

2.4 Context Management & Maintenance

One of the key challenges with query processing using LLMs is the high cost and latency associated with invoking the models. For instance, processing 1,000 emails to extract their sender and subject is orders of magnitude more expensive and time consuming than executing a SQL query over a structured table with sender and subject columns. Prior work on optimizing LLM inference has shown that a simple way reduce LLM computation is to reuse previous execution results [33]. Similarly, databases have long used materialized views to reduce unnecessary computation [5]. Thinking along these lines, we created a `ContextManager` which embeds and caches the descriptions of materialized `Contexts`. This enables Palimpzest's optimizer to reuse these `Contexts` to help with answering future queries, which we discuss further in the next section.

3 OPTIMIZATION

The abstractions presented in the previous section provide systems like Palimpzest with a number of opportunities for (cost-based) optimization. In this section, we describe an initial set of logical and physical optimization which we have already implemented in our prototype, or plan to implement in the near future.

Logical Optimizations. Inspired by work from DocETL [26], an obvious avenue for logical optimization is to rewrite the query plan to better scope `search` and `compute` directives which are underspecified or overly complex. DocETL has demonstrated success at rewriting data processing pipelines in this fashion, and we believe Palimpzest's query optimizer could similarly apply an LLM judge to determine when a `search` or `compute` operator needs to be split into smaller operations. Alternatively, the query optimizer may also be able to improve the efficiency of the query plan by identifying opportunities to merge `search` and `compute` operations which are similar, or likely to process the same set of physical data. Finally, for `compute` operations which repeatedly fail during query execution, a dynamic query optimizer could insert a logical `search` operator at runtime before the `compute` in an attempt to provide it with a richer `Context`. All of the aforementioned optimizations are currently future work, and outside the scope of our prototype.

Physical Optimizations. Similar to work in Abacus [24], one direction for physical optimization includes allowing the query optimizer to select the model (and other parameters) used by the `compute` or `search` operator's agent. Building on ideas from materialized views research [5], another opportunity for optimization is to reuse previously cached `Context(s)` to augment (or replace) the `Context` for a `compute` operator. For example, if a query computes the identity theft reports in 2001, and then a second query seeks to compute the identity thefts in 2024, a detailed `Context` from the first query might help the second query execute more efficiently. We have implemented a preliminary version of this physical optimization in our prototype, although it is currently experimental.

4 EVALUATION

We evaluate our prototype system on two example queries. The first query comes from the `legal` workload in the Kramabench [15] benchmark. The workload's dataset consists of 132 CSV and HTML files which contain recent statistics on fraud, identity theft, and other consumer reports. The query asks the analytics system to compute the ratio between the number of identity theft reports in the years 2024 and 2001. The ground truth for this query is found in a single CSV file, which contains a breakdown of the fraud, identity theft, and other reports for the years 2001 to 2024 inclusive.

The second query asks the analytics system to filter a subset of emails from the Enron email dataset [7] for ones which contain firsthand discussion of one or more specific business transactions. Similar to prior work [17], we evaluate on a subset of 250 Enron emails in order to keep execution costs reasonable. The query also asks the system to extract a sender, subject, and summary of each email, but to simplify our evaluation we simply compute the precision, recall, and F1-score of the emails returned by each system.

Our Prototype Outperforms Semantic Operators. For our first experimental claim, we sought to demonstrate that the dynamic execution semantics of our prototype's `compute` operator

Table 1: `compute` achieves lower error than a handcrafted semantic operator program written in Palimpzest. In trials where the semantic operator program returned multiple output ratios, we averaged the percent errors for each ratio.

System	Pct. Err.	Cost ($)	Time (s)
Sem. Ops	17.00%	1.66	215.2
CodeAgent	27.56%	0.03	77.0
PZ `compute`	**0.02%**	1.17	583.0

could outperform a handcrafted semantic operator program on an interactive data analytics tasks. For this evaluation, we use the `legal-easy-3` query from the Kramabench [15] dataset. For our baselines, we handcrafted a Palimpzest program and a SmolAgents CodeAgent equipped with tools for listing and reading files. For our prototype, we execute the `compute` operator by simply passing in the query string as the natural language instruction.

We ran each system on the evaluation query three times and report the average percent error, cost, and runtime. The results from our evaluation are shown in Table 1.

Overall, our prototype's `compute` operator achieved the lowest percent error across all three systems. The semantic operator program was able to compute the correct ratio in all three trials, however in two trials it also computed a second ratio due to an errant file returned by one of its semantic filters. The CodeAgent often struggled to find the correct file in the dataset, and would return spurious ratios which it computed from the information it found in non-ground truth files. Finally, our prototype was able to effectively answer the query by writing optimized PZ programs to search for information on identity thefts in 2024 and 2001, before computing the final result in Python.

Our Prototype Outperforms Open Deep Research. For our second experimental claim, we sought to demonstrate that our prototype's `compute` operator could leverage its ability to write optimized PZ programs to outperform a SmolAgents CodeAgent. For the evaluation, we recreated a document processing task from [17] in which each system has to filter for emails matching two natural language predicates. For our baselines, we use two CodeAgents: the first CodeAgent is equipped with tools for listing and reading files, while the second agent (CodeAgent+) is additionally equipped with tools for applying (unoptimized) semantic filters and maps. For our prototype, we execute the `compute` operator by passing in the query string as the natural language instruction.

We ran each system on the evaluation query three times and report the average quality, cost, and runtime. We implemented each system with GPT-4o and used GPT-4o in CodeAgent+'s semantic operator tools as well. The results from our evaluation are shown in Table 2. Overall, our prototype's `compute` operator achieved near perfect output quality while saving 76.8% on cost and 72.7% on runtime relative to CodeAgent+ which manually invokes semantic operators. We observe that the CodeAgent without semantic operators is cheap and fast, but fails to achieve good quality (i.e. F1-score) on the objective. The agent's low quality—driven by poor recall of relevant emails—is a result of its tendency to rely on simple filter heuristics (e.g., regular expressions) and manual reading to identify emails that are relevant.

Table 2: `compute` writes optimized Palimpzest programs that achieve higher quality than a naive CodeAgent. `compute` achieves the same quality as a CodeAgent with semantic operators as tools (CodeAgent+), however `compute`'s optimized execution results in $> 70\%$ cost and runtime savings.

System	F1	Recall	Prec.	Cost ($)	Time (s)
CodeAgent	50.53%	46.15%	88.89%	0.08	37.0
CodeAgent+	**98.67%**	97.44%	100%	3.76	1,999.9
PZ `compute`	**98.67%**	97.44%	100%	0.87	546.2

Providing CodeAgent+ with semantic operators as tools helps the agent overcome the aforementioned recall issue. By invoking semantic map and filter operations, CodeAgent+ guarantees that every email is read and processed by an LLM. This leads to a significant boost in quality, however it also increases the cost and runtime of the system. Unfortunately, much of the increase in the cost and runtime is due to inefficient use of semantic operators. For instance, CodeAgent+ often executed multiple semantic filters in sequence without checking the output of the first semantic filter before executing the subsequent one(s).

Finally, our `compute` operator wrote correct Palimpzest programs in all three trials. These programs were immediately more efficient than those written by CodeAgent+ because Palimpzest's execution engine did not perform redundant computation on filtered emails. Furthermore, Palimpzest's query optimizer was able to use cheaper models for some of the semantic operators, thus yielding additional cost and runtime savings.

5 RELATED WORK

Deep Research Systems—which answer natural language questions over large datasets using a combination of LLM planning, tool use, code execution, and reasoning—have become increasingly popular for data analytics tasks. In a span of five months, OpenAI, Anthropic, Google, Perplexity, and xAI [3, 9, 19, 21, 32] all released private offerings to their combined hundreds of millions of users. Since then, researchers in industry and academia have sought to replicate the performance of these systems.

Some of the more prominent open source offerings include LangChain's Open Deep Research [16] and HuggingFace's SmolAgents Open Deep Research [28]. LangChain's system uses LLMs (agents), search tools, and MCP server(s) in three modular phases to scope, research, and write a final report for the user's question. HuggingFace's system uses its CodeAgents [27] to iteratively plan, use tools, and execute Python code in order to answer the user's question. Recent work has also focused on building agentic systems for data science [6, 10, 31]. While our prototype similarly makes use of agents to answer analytics queries, it places an emphasis on the use of cost-based optimization by (1) providing these agents with a tool for writing optimized Palimpzest programs and (2) enabling the query optimizer to reuse previously materialized `Contexts`.

Another recent line of work focuses on optimizing and executing analytics queries over unstructured data with semantic operator systems [1, 17, 18, 20, 25, 26, 30]. Palimpzest [17, 24] uses a multi-armed bandit sampling algorithm to gather statistics on operator performance before using cost-based optimization to optimize the semantic operator system. LOTUS [20] uses proxy methods to optimize semantic join, filter, group-by, and top-k operators while providing statistical guarantees with respect to a reference model. DocETL [26] uses LLMs to apply (and validate) query rewrites to semantic operator systems. Finally, Galois [25] introduced new logical and physical optimizations for answering queries with LLM-based operators. Our work extends these systems to support Deep Research queries requiring more dynamic execution semantics.

While traditional OLAP databases [8, 11, 22, 29] have limited support for AI-driven analytics over unstructured data, there is considerable work on Approximate Query Processing (AQP) which aims to support these queries. Some early work focused on adding machine learning classifiers to analytics systems for tasks including object detection, sentiment analysis, and more [2, 12–14, 23]. One limitation of these systems was they struggled to answer queries which did not align well with the task the classifier was trained for. While our prototype and AQP share a similar tradeoff between efficiency and execution accuracy, the generality of foundation models and dynamic execution semantics of `compute` and `search` enable our system to answer a broader range of analytics queries.

6 FUTURE WORK

While our prototype demonstrates promising initial results, more work is needed to fully implement our vision of an ideal runtime for AI-driven analytics. Our current prototype is tightly coupled with the Palimpzest framework, but an ideal runtime could make use of semantic operators from any semantic programming framework. Furthermore, the use of cost-based optimization within our prototype is largely limited to optimizing semantic operator programs written by agents. In the future, we intend to apply cost-based optimization at a higher level of abstraction. Specifically, we imagine this runtime will optimize query plans whose logical (and physical) operators consist of semantic operators as well as heretofore "agentic" operations including code execution, tool use, and LLM-based reasoning. Defining this space of logical operators and their physical counterparts is an area for future research. Additionally, applying cost-based optimization to agentic operators is challenging, as these operations' quality, cost, and latency are difficult to predict and may involve side effects (especially in the case of tool use and code execution). Overall, we believe that there are ample opportunities to apply ideas from declarative optimization (and data systems research more broadly) to this new paradigm.

7 CONCLUSION

We present a vision and a prototype for a new runtime for AI-driven analytics over large unstructured datasets. In particular, we propose a runtime which blends the optimized execution of semantic operator systems with the flexibility and dynamic execution semantics of Deep Research systems. To this end, we extend the Palimpzest framework to include two new semantic operators which leverage CodeAgents to execute a natural language instruction. To support these operators, we create a new `Context` abstraction in Palimpzest which enables dynamic data access methods such as indexing and search. Finally, we demonstrate our prototype's ability to leverage the strengths of Deep Research and optimized semantic operator execution on two evaluation queries for interactive data analytics and document processing, respectively.

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] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A. Shah, Benjamin Sowell, Dan Tecuci, Vinayak Thapliyal, and Matt Welsh. 2025. The Design of an LLM-powered Unstructured Analytics System. CIDR.

[2] Michael R. Anderson, Michael J. Cafarella, Germán Ros, and Thomas F. Wenisch. 2018. Physical Representation-Based Predicate Optimization for a Visual Analytics Database. *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2018), 1466–1477. https://api.semanticscholar.org/CorpusID:48362547

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

[4] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why Do Multi-Agent LLM Systems Fail? arXiv:2503.13657 [cs.AI] https://arxiv.org/abs/2503.13657

[5] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. 1995. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*. 190–200. doi:10.1109/ICDE.1995.380392

[6] Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. 2025. SheetAgent: Towards a Generalist Agent for Spreadsheet Reasoning and Manipulation via Large Language Models. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 158–177. doi:10.1145/3696410.3714962

[7] William W. Cohen. 2015. Enron Email Dataset. https://www.cs.cmu.edu/~enron/

[8] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. doi:10.1145/2882903.2903741

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

[10] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-agent: automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 668, 36 pages.

[11] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1917–1923. doi:10.1145/2723372.2742795

[12] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc.*

VLDB Endow. 13, 4 (Dec. 2019), 533–546. doi:10.14778/3372716.3372725

[13] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597. doi:10.14778/3137628.3137664

[14] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating approximate aggregation queries with expensive predicates. *Proc. VLDB Endow.* 14, 11 (July 2021), 2341–2354. doi:10.14778/3476249.3476285

[15] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Sivaprasad Sudhir, Om Chabra, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Michael J. Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. 2025. KramaBench: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. arXiv:2506.06541 [cs.DB] https://arxiv.org/abs/2506.06541

[16] LangChain. 2025. *Open Deep Research.* Retrieved July 31, 2025 from https://blog.langchain.com/open-deep-research/

[17] 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.

[18] Duo Lu, Siming Feng, Jonathan Zhou, Franco Solleza, Malte Schwarzkopf, and Uğur Çetintemel. 2025. VectraFlow: Integrating Vectors into Stream Processing. CIDR.

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

[20] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. arXiv:2407.11418 [cs.DB] https://arxiv.org/abs/2407.11418

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

[22] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1981–1984. doi:10.1145/3299869.3320212

[23] Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. 2023. Accelerating Aggregation Queries on Unstructured Streams of Data. *Proc. VLDB Endow.* 16, 11 (July 2023), 2897–2910. doi:10.14778/3611479.3611496

[24] 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

[25] Dario Satriani, Enzo Veltri, Donatello Santoro, Sara Rosato, Simone Varriale, and Paolo Papotti. 2025. Logical and Physical Optimizations for SQL Query Execution over Large Language Models *(SIGMOD '25)*. Association for Computing Machinery, New York, NY, USA, 28 pages. doi:10.1145/3725411

[26] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[27] SmolAgents. 2024. *Introducing smolagents, a simple library to build agents.* Retrieved August 1, 2025 from https://huggingface.co/blog/smolagents

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

[29] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2005. C-store: a column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases* (Trondheim, Norway) *(VLDB '05)*. VLDB Endowment, 553–564.

[30] Matthias Urban and Carsten Binnig. 2023. CAESURA: Language Models as Multi-Modal Query Planners. arXiv:2308.03424 [cs.DB] https://arxiv.org/abs/2308.03424

[31] Jiayi Wang and Guoliang Li. 2025. AOP: Automated and Interactive LLM Pipeline Orchestration for Answering Complex Queries. CIDR.

[32] xAI. 2025. *Grok 3 Beta — The Age of Reasoning Agents.* Retrieved July 31, 2025 from https://x.ai/news/grok-3

[33] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs.AI] https://arxiv.org/abs/2312.07104