iPDB: SQL with ML and LLM Predicates (Towards a Database Engine for AI)

Udesh Kumarasinghe
Tyler Liu
ukumaras@purdue.edu
liu3450@purdue.edu
Purdue University

Ahmed Mahmood
amahmoo@google.com
Google Inc.

Chunwei Liu
Walid G. Aref
chunwei@purdue.edu
aref@purdue.edu
Purdue University

ABSTRACT

Structured Query Language (SQL) has remained the standard query language for databases, and is highly optimized for processing structured data. However, it is inefficient for applications that leverage the capabilities of large language models (LLMs) to comprehend and extract semantic information from structured and unstructured data. This results in complex engineering and multiple data migration operations that transfer data between the data source and the LLM inference platform to couple them. We demonstrate iPDB, a system that supports in-database LLM inference using an extended declarative SQL syntax and new optimizations that result in efficient query processing of LLM-enabled SQL queries that outperform state-of-the-art systems.

PVLDB Reference Format:
Udesh Kumarasinghe, Tyler Liu, Ahmed Mahmood, Chunwei Liu, and Walid G. Aref. iPDB: SQL with ML and LLM Predicates. PVLDB, 19(12): 4782 - 4785, 2026.
doi:10.14778/3827998.3828121

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

1 INTRODUCTION

Recent advances in Artificial Intelligence (AI) have paved the way for significant breakthroughs in applications such as question answering [10] and information retrieval [11]. With large amounts of data being stored in databases, it is natural to consider pushing these AI advances into the inside of database engines and to its query language. Structured Query Language (SQL) has remained the standard and most commonly used querying language for databases. However, SQL is highly optimized for querying tables with structured attributes, e.g., integers, strings, and dates, but is inefficient on workloads that require extracting semantic information from unstructured data, which are highly relevant in today's application development. iPDB demonstrates an extended SQL syntax to support operations that involve inference through large language models (LLMs) [12] , which are referred to as semantic queries.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828121

Recently, several studies have proposed systems that extract and filter information from unstructured documents [6]. Although these systems demonstrate the potential and applications of semantic query processing, they either introduce an imperative query language or only support unstructured data. Hence, we extend standard SQL queries by introducing LLM capabilities. Specifically, we augment SQL queries with LLMs that enable context-aware, natural-language Relational Algebra operations (e.g., semantic selects, semantic projects, and semantic joins) extending beyond traditional relational operations and predicates. For example, given a table of products, we can find the list of compatible motherboards and CPUs, by self joining with a natural language condition.

```
SELECT c.name, m.name FROM Product AS m JOIN Product AS c
ON LLM o4mini (PROMPT 'is CPU  {{c.name}} {compatible BOOLEAN}
     with motherboard {{m.name}}')
WHERE m.kind = 'Motherboard' AND c.kind = 'CPU';
```

Integrating LLM inference into SQL queries establishes a declarative syntax for inference, supporting a broad range of tasks, including classification, sentiment analysis, summarization, and text generation. Having LLM capabilities implemented natively within SQL enables the query language's functionality and expressivity while allowing to create complex operations rather than simply feeding rows of unprocessed data into an LLM. Finally, iPDB can identify patterns in semantic queries and perform logical and physical optimizations on them, allowing iPDB to execute large-scale inference tasks efficiently. These optimizations are vital as LLM workloads are expensive both in runtime and monetary costs [6].

In this demonstration, we showcase iPDB with its extended semantic SQL in three motivating use-cases. Audience members will interact with the demonstration by running editable queries live in a provided graphical interface. The demonstration interface allows each query to run with each optimization enabled or disabled to emphasize their importance. The audience will view the query plan and performance statistics for each query. Finally, we will demonstrate the performance of iPDB in contrast to state-of-the-art systems that support semantic operators.

2 ENABLING SEMANTIC SQL QUERIES

We introduce a model upload statement to import models to the system and a generalized LLM calling expression to infer them.

2.1 Model Uploading

Given the variety of models, it is crucial to define and manage the available LLMs. Model upload syntax imports and verifies the model and its metadata into the database, and stores the metadata in a specialized model catalog.

```
CREATE LLM MODEL o4mini PATH 'o4-mini'
ON PROMPT API 'https://api.openai.com/v1/';
```

where the MODEL argument is the given name (i.e., o4mini) within iPDB, the PATH argument is the model identifier (i.e., 'o4-mini') in the vendor API. The API argument is the base URL for the API. Similarly, a locally stored LLM can be uploaded by specifying the LLM path in the PATH parameter and omitting the optional API parameter. The ON PROMPT clause indicates to the database that the prompt for the LLM call will be provided at query time. The prompt enables the use of structured placeholders to define input columns and typed output columns, allowing the user to reuse the same model for different tasks by modifying the prompts.

2.2 LLM Inference

Users write semantic queries using the LLM clause that can appear in various locations inside the SQL query. For example, the LLM clause can be used for table inference, generation, or scalar inference.

A table inference will produce a predicted table with one or more predicted columns. The LLM clause must appear in a FROM clause, and the predicted columns can be used in consecutive traditional SQL operations (e.g., projections).

```
SELECT state, sum(total_amount)
FROM LLM o4mini (PROMPT 'extract the {state VARCHAR} from the {{
     address}}', Customer) AS c
JOIN Orders AS o ON o.cid = c.cid
GROUP BY state;
```

Within the FROM clause, an LLM clause is defined with the o4mini model, the prompt and input relation are given as two arguments. Notice that in the PROMPT, the input column is defined using the {{...}}. Similarly, output columns are defined with {...} placeholders but with an additional output type (i.e., {state VARCHAR}).

Additionally, users may drop the input relation in use cases where LLM is used for relation generation (e.g., as in Query Q3 in Table 2). In this case, the LLM call behaves similar to a scan, where the system declaratively extracts structured data from the LLM.

Furthermore, with scalar inference, it can be used directly in traditional SQL expressions, where placeholders define the input columns and a single output column. The input columns can originate from any of the resolved relations.

3 SYSTEM OVERVIEW

This demonstration of iPDB will demonstrate extended semantic SQL that leverages a fork of DuckDB [8] as the base engine. iPDB introduces the LLM clause and the CREATE MODEL statement to the parser by modifying it to support the extended semantic SQL language. The native logical and physical LLM operators introduced model the execution of semantic queries. Additionally, the system catalogs, logical and physical planners, and the query optimizer of the base engine is extended to realize iPDB. The query processing stages of a semantic query are given in Figure 1.

3.1 Semantic Query Processing

Upon reading a semantic SQL statement, the parser transforms the LLM clause to a logical PREDICT structure. The planner retains the model name, the prompt, and any input tables in the operator's current state. Next, a verification step ensures the validity of the requested model by referring to the model catalog. Then, the input and output columns are extracted by parsing the provided prompt. It is ensured that the input columns are resolved by the source relations (i.e., the sub-trees of the logical plan). The extracted output columns and their types are informed to the planner, allowing them to be resolved by the subsequent operations (i.e., the ancestors of the PREDICT operator in the query's logical execution plan). Once the logical PREDICT operator is created, the optimizer is allowed to apply existing base engine optimizations and new optimizations offered by iPDB. Finally, the logical PREDICT operators in the optimized plan are transformed into physical PREDICT operators.

During execution, Operator inference conforms to the vectorized execution of the base engine. The operator is provided with a combined data chunk of the input relations containing a set of rows stored in column vectors. Only the input column is probed, extracting the input values for each row and appending them to the prompt as key-value pairs. Prompts are rewritten to ensure structured output. The populated prompt is passed to the LLM to generate results. Finally, the structured output values are extracted from the generated LLM output and are populated in the output data chunk in the respective predicted column(s). On inference failure or unstructured output, iPDB falls back to per-tuple execution and retries, returning null only for tuples that continue to fail.

3.2 Compatibility with LLMs

iPDB supports both local and remote LLMs. The locally stored models (i.e., Google's open-source model gemma) in GGUF format are supported by LLaMa.cpp[1] inference platform. Remote LLMs, whether commercial or free are supported through API calls that are compatible with the OpenAI API (e.g., OpenAI's o4-mini model).

4 DEMONSTRATION SCENARIO

We demonstrate several use cases that leverage the power of semantic SQL queries and highlight the opportunities enabled by the cohesion of LLM inference with traditional relational operations.

Interaction: Audience members will interact with a GUI (Figure 2) that allow writing semantic SQL queries, view results, visualize query plans and toggle individual optimizations. Users will be given a dataset (listed in Table 1) collected from PC Components[2] and supplemented with reviews from SNAP Amazon Datasets[3].

Part 1 **Use Case** - We let users run a set of queries on a use case, named 'Understanding Customers without leaving SQL'. We showcase the semantic SQL query results, execution plan, and performance statistics; demonstrating how iPDB parses and plans the execution of the semantic query.

Part 2 **Optimizations** - Then, we toggle optimizations and rerun the queries to showcase the impact of individual optimizations. The performance statistics and the changing query plans will show the impact of logical and physical optimizations. Additionally, user can run the same query on a comparative system [7] and compare the performance.

Part 3 **Exploration** - Attendees run the queries listed in Table 2 with and without optimization. These queries showcase

[1] https://github.com/ggml-org/llama.cpp
[2] https://www.pccomponents.com/
[3] https://snap.stanford.edu/data/web-Amazon.html

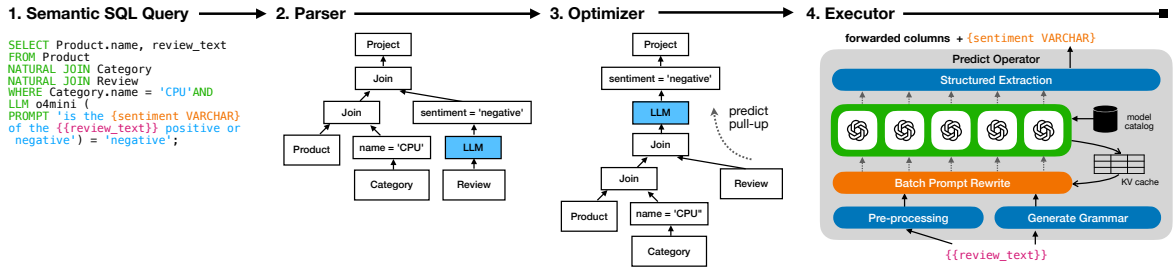

Figure 1: Query Execution Process in iPDB to support semantic operators

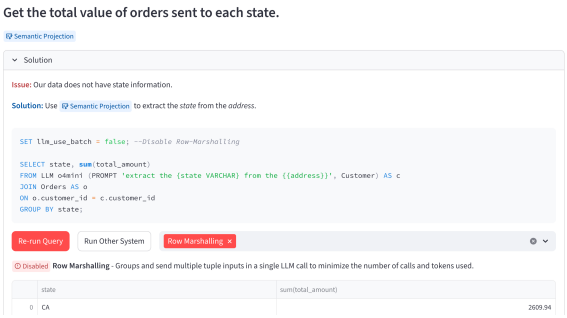

Figure 2: Screenshot of demonstration GUI.

varying semantic operators and optimization capabilities of iPDB. Additionally, audience will be invited to write their own ad-hoc semantic queries on the dataset.

Table	Column	Rows
Category	**cid**, kind	4
Warehouse	**wid**, name, address	9
Product	**pid**, name, description, price, cid	208
Review	**pid, rid**, review_text	946
Inventory	**wid, pid**, list_price, quantity	1112

Table 1: Tables and statistics of the dataset

5 OPTIMIZING SEMANTIC SQL QUERIES

One key reason for the widespread adoption of relational databases is the declarative nature of SQL. Users are not required to write optimal queries; it is the database system's responsibility to generate an optimal execution plan for the given query. The same ideology applies to iPDB. The system aims to optimize and execute declarative semantic SQL queries with minimal latency and cost while maintaining high quality results.

Prompt Deduplication. Duplicate values in a relational database table may exist in a stored database table or be created during an intermediate relation of an operation, e.g., a join on a many-to-many relationship. Asking the LLM the same prompt and inputs is redundant. Thus, iPDB utilizes a de-duplication mechanism with a key-value cache. The operator maintains a cache of input values and the LLM's parsed output throughout the lifetime of the predict operator. The operator returns a cached value preemptively when it is a hit for the given input values.

Multi-row Prompt Marshalling. Inferencing the LLM on every tuple is inefficient and infeasible. Thus, iPDB prompts several tuples as a batch, reducing the overall number of calls. The LLM is instructed to produce an array of predictions for each tuple. Multi-row prompt marshalling in iPDB is used in conjunction with prompt de-duplication. If an input value is in the cache, the tuple is not considered for the batch; instead, the cached value is returned along with the other tuples in that batch.

Semantic Predicate Ordering. Traditional optimizers assume predicates as zero-latency operations and push them to the closest query subtree that contains the predicate's relation set. This optimization is invalid for the semantic predicates [3]. Although semantic predicates are selective, they are expensive. Thus, naively pushing them down result in redundant LLM calls that are removed by other selects or joins. iPDB pulls predictive operations that contain a predicate on the predicted columns up the query tree.

6 RESULTS

We evaluate and compare iPDB with the state-of-the-art: (1) LOTUS [7] that has the highest coverage of semantic operations but lacks a declarative SQL interface, (2) EvaDB [4] that supports LLM calls in SQL as UDFs, (3) Flock [1] that implements a DuckDB extension with scalar and aggregate LLM functions. All the queries are evaluated using the `o4-mini` model by OpenAI.

Table 3 compares the performance of each approach for the evaluated queries. Values not included in the table are workloads not supported by that particular system. Compared to LOTUS and EvaDB, iPDB outperforms in runtime, the number of LLM calls it makes and the number of tokens used. The state-of-the-art relies solely on parallelism, whereas iPDB attempts to fit the optimal number of tuples into a single LLM call and executes these calls in parallel. This, along with iPDB's logical optimizations, minimizes the number of LLM calls made and the number of tokens sent. Flock also applies row-marshalling, resulting in fewer tokens and calls. However, the results are unstructured and of lower quality. $Q2$, $Q4$ and $Q5$ are incompatible as table inference, generation, and semantic joins are not supported. Optimizations in iPDB do not degrade results' quality. iPDB's relatively higher F1 scores are a result of structured output parsing and fallback strategies.

	Query	Operation	Type
Q1	`SELECT name, review_text FROM Product NATURAL JOIN Category NATURAL JOIN Review WHERE kind = 'CPU' AND LLM o4mini (PROMPT 'is the {sentiment VARCHAR} of the {{review_text}} positive or negative') = 'negative';`	Selection	Scalar
Q2	`SELECT state, state_tax FROM LLM o4mini (PROMPT 'get all the {state VARCHAR} and their {state_tax DOUBLE} pairs in the US');`	Generation	Table
Q3	`SELECT name, price, LLM o4mini (PROMPT 'has the {{price}} of {{name}} gone up or {gone_down BOOLEAN} from the release date dollar price') FROM Product NATURAL JOIN Category WHERE kind = 'CPU';`	Projection	Scalar
Q4	`SELECT name, avg(price) FROM LLM o4mini (PROMPT 'extract the {vendor VARCHAR} from the {{name}}', Product) GROUP BY vendor;`	Projection	Table
Q5	`SELECT C.name, M.name FROM (Product NATURAL JOIN Category) AS M NATURAL JOIN (Product NATURAL JOIN Category) AS C ON LLM o4mini (PROMPT 'is CPU {{C.name}} {compatible BOOLEAN} with motherboard {{M.name}}') WHERE M.kind = 'Motherboard' AND C.kind = 'CPU';`	Join	Scalar

Table 2: Demonstrated queries with the corresponding semantic operation and inference type

System	D1:Q1			D1:Q2			D1:Q3			D1:Q4			D1:Q5		
	Lat. (s)	C / T	F1	Lat. (s)	C / T	F1	Lat. (s)	C / T	F1	Lat. (s)	C / T	F1	Lat. (s)	C / T	F1
LOTUS	147.20	946 / 243K	0.980	N/A	N/A	N/A	75.4 s	Except.	Except.	67.50	208 / 106K	0.490	414.70	Except.	Except.
EvaDB	487.50	946 / 203K	0.777	N/A	N/A	N/A	105.7 s	41 / 21K	0.414	N/A	N/A	N/A	N/A	N/A	N/A
Flock	21.98	12 / 73K	0.688	N/A	N/A	N/A	35.63	**3 / 3K**	0.536	N/A	N/A	N/A	N/A	N/A	N/A
iPDB	**14.52**	**12 / 25K**	**0.996**	**16.18**	**1 / 2K**	**1.000**	**15.98**	3 / 5K	**0.951**	**9.30**	**13 / 12K**	**0.947**	**283.41**	**120 / 103K**	**1.000**

Table 3: Performance results of demonstrated queries comparing the latency, number of calls/tokens and the F1 score

7 RELATED WORK

Recent studies aim to address the execution of semantic queries. Palimpzest [6] and DocETL [9] focus on document processing, lacking tight integration with relational operators and the declarative abstractions of SQL. EvaDB [4] offers LLM inference in SQL with UDFs, thus limiting generalizability or opportunities for optimizations. LOTUS [7] enables semantic operators on dataframes; but it lacks a declarative interface and executes LLM inference eagerly, limiting logical optimizations. Flock [1] implements an LLM extension for DuckDB. However, as a limitation of being an extension, it is incapable of semantic join, table generation, and table inference that produce multiple typed columns. Although, Google BigQuery [2] offer AI functions, they are limited to proprietary LLMs and lack the logical optimizations. In contrast, iPDB combines the strengths of relational dbmss with flexible LLM inference. It supports inference of scalar and table generation directly in SQL queries, enabling the use of LLM-based tasks on both structured and unstructured data.

8 CONCLUSION

We introduce iPDB, a system for semantic SQL processing by LLM inference. We demonstrate a series of use-case queries that illustrate how this integration provides a practical, declarative approach for combining traditional SQL with semantic operations. iPDB supports typed data extraction, and logical and physical optimizations. iPDB's optimizations in the logical planning and the internals of the inference operator outperform the state-of-the-art in terms of latency and cost, while maintaining correctness.

ACKNOWLEDGMENTS

This research was conducted externally and was not supported by Google Inc. Some results presented were obtained using the Chameleon testbed [5] supported by the NSF.

REFERENCES

[1] 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 (Aug. 2025), 5415–5418.

[2] Jian He and Vaibhav Sethi. 2025. SQL reimagined for the AI era with BigQuery AI functions | Google Cloud Blog — cloud.google.com. https://cloud.google.com/blog/products/data-analytics/sql-reimagined-for-the-ai-era-with-bigquery-ai-functions.

[3] Joseph M. Hellerstein and Michael Stonebraker. 1993. Predicate migration: optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., USA). Association for Computing Machinery, New York, NY, USA, 267–276.

[4] Gaurav Tarlok Kakkar, Jiashen Cao, Aubhro Sengupta, Joy Arulraj, and Hyesoon Kim. 2025. Aero: Adaptive Query Processing of ML Queries. *Proc. ACM Manag. Data* 3, 3, Article 174 (June 2025), 27 pages.

[5] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbah, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 219–233.

[6] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*.

[7] Liana Patel, Siddharth Jha, Parth Asawa, Melissa Pan, Carlos Guestrin, and Matei Zaharia. 2024. Semantic Operators: A Declarative Model for Rich, AI-based Analytics Over Text Data. arXiv:2407.11418 [cs.DB]

[8] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 1981–1984.

[9] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[10] Murong Yue. 2025. A Survey of Large Language Model Agents for Question Answering. arXiv:2503.19213 [cs.CL] https://arxiv.org/abs/2503.19213

[11] ChengXiang Zhai. 2024. Large Language Models and Future of Information Retrieval: Opportunities and Challenges. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, USA, 481–490.

[12] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223