

SemBench: A Benchmark for Semantic Query Processing Engines

Jiale Lao
Cornell University
jjale@cs.cornell.edu

Andreas Zimmerer
University of Technology Nuremberg
andreas.zimmerer@utn.de

Olga Ovcharenko
BIFOLD & TU Berlin
ovcharenko@tu-berlin.de

Tianji Cong
University of Michigan
congtj@umich.edu

Matthew Russo
MIT CSAIL
mdrusso@mit.edu

Gerardo Vitagliano
MIT CSAIL
gerarvit@mit.edu

Michael Cochez
Vrije Universiteit Amsterdam
m.cochez@vu.nl

Fatma Özcan
Google
fozcan@google.com

Gautam Gupta
Google
gautamguptag@google.com

Thibaud Hottelier
Google
tbh@google.com

H. V. Jagadish
University of Michigan
jag@umich.edu

Kris Kissel
Google
kriskissel@google.com

Sebastian Schelter
BIFOLD & TU Berlin
schelter@tu-berlin.de

Andreas Kipf
University of Technology Nuremberg
andreas.kipf@utn.de

Immanuel Trummer
Cornell University
itrummer@cornell.edu

ABSTRACT

We present a benchmark targeting a novel class of systems: semantic query processing engines. Those systems rely inherently on generative and reasoning capabilities of state-of-the-art large language models (LLMs). They extend SQL with semantic operators, configured by natural language instructions, that are evaluated via LLMs and enable users to perform various operations on multi-modal data.

Our benchmark introduces diversity across three key dimensions: scenarios, modalities, and operators. Included are scenarios ranging from movie review analysis to medical question-answering. Within these scenarios, we cover different data modalities, including images, audio, and text. Finally, the queries involve a diverse set of operators, including semantic filters, joins, mappings, ranking, and classification operators.

We evaluated our benchmark on three academic systems (LOTUS, Palimpzest, and ThalamusDB) and one industrial system, Google BigQuery. Although these results reflect a snapshot of systems under continuous development, our study offers crucial insights into their current strengths and weaknesses, illuminating promising directions for future research.

Benchmark Availability:

The benchmark data, queries, and results are available at <https://www.sembench.org>.

1 INTRODUCTION

With the advent of LLMs and their inherent generative and reasoning power, a novel class of data processing systems has emerged in academia [10, 20, 31, 38, 49] and is already experiencing widespread adoption in industry [7, 12]: semantic query processing engines (SQPEs). SQPEs extend relational algebra by *semantic operators* that perform tasks described in natural language on a multitude of data types, including standard SQL types as well as unstructured data types such as images or audio data (stored in the cells of database tables). Such operators are enabled by the latest generation of LLMs, able to process multimodal inputs in *zero-shot mode*, i.e., processing novel tasks based on a task description alone (without requiring any task-specific training data). The following query can be processed by several of the recently released SQPEs.

Example 1.1. Consider a table `Cars` with a column `pic` containing pictures of cars (each table row contains one picture). The query below, formulated in semantic SQL, counts pictures of red cars:

```
1 SELECT COUNT(*)
2 FROM Cars
3 WHERE AI.IF(pic, 'the picture shows a red car');
```

The query above uses a semantic operator (`AI.IF`) to process images according to natural language instructions (“the picture shows a red car”). While we use BigQuery’s syntax in this and the following examples, the query above can be processed (after slight rewriting) by many other SQPEs as well (e.g., LOTUS, Palimpzest, and ThalamusDB).

Semantic operators are evaluated using LLMs, changing query optimization in two significant ways. First, when using LLMs, per-byte processing costs are higher by many orders of magnitude compared to traditional, relational processing. Hence, as soon as LLMs are invoked during query evaluation, the focus in cost optimization

shifts from optimizing relational processing to minimizing costs associated with LLM invocations. Second, LLMs results are inherently stochastic, and do not always produce 100% correct results. Hence, accuracy of results becomes another important metric to optimize. For SQPEs, the optimization problem can be stated as “minimize the latency and cost of LLMs, while maximizing accuracy”.

Current benchmarks for analytical data processing (e.g., TPC-H and TPC-DS) do not contain queries with semantic operators. Over many decades, these benchmarks have driven innovation in areas that contribute to efficient relational processing, e.g., relational operator implementations, query optimization, and table indexing. However, they are unsuitable for guiding the development of techniques that minimize the number of LLM calls, the number of input tokens, or the size of the LLMs used for specific tasks during query processing. However, in the context of SQPEs, such techniques are crucial for efficient processing.

We introduce a new benchmark, SemBench, that focuses on query processing with semantic operators. The goal of this benchmark is to foster innovation in areas that contribute to minimizing costs associated with LLM invocations, while maximizing accuracy. Our benchmark refers to an extended relational data model that is nowadays adopted by most SQPEs: data are stored in relational tables, but cells may contain images or audio files in addition to the usual SQL data types. Semantic operators can be applied to images, text, audio files, or combinations of the aforementioned types. For instance, our benchmark features queries with semantic joins that cross data modalities (e.g., a join connecting images with associated text).

Our benchmark encompasses multiple scenarios, each characterized by a scenario-specific database and a set of queries. To create benchmark data, we combine existing data sets from Kaggle that come with manual labels. For instance, to create a database for a wildlife monitoring scenario, we use a data set from Kaggle with animal sound recordings, labeled by the associated species, as well as a dataset with camera trap images, each image labeled with the animal species that appear in it. By combining both data sets and generating additional metadata (e.g., to represent a recording location), we obtain a database that enables complex queries. For instance, we count locations at which specific animal species co-occur, based on detections inferred from audio or visual data. To generate ground-truth results for such queries, we leverage the manual labels of the original data sets.

Our benchmark queries cover a wide range of semantic operators (as well as traditional relational operators). It contains queries that are supported by all or most of the current SQPEs, as well as queries containing semantic operators only available in a subset of evaluated systems. Our goal is to motivate SQPE developers to expand their scope in terms of supported queries, as well as to evaluate the impact of various query processing approaches on performance and result quality.

Our benchmark evaluates systems according to multiple metrics. First, we evaluate the quality of the generated results by comparison to the ground truth. Depending on the type of query, we use different metrics to assess the similarity of ground truth and generated results. For aggregation queries (i.e., queries containing SQL aggregates), we measure quality as the relative error. For retrieval queries, we measure quality via F1 score (based on whether or not

generated result rows appear in the ground truth). Second, we evaluate the processing overheads using metrics such as execution time and monetary execution fees (paid for LLM invocations).

We present experimental results for SQPEs from academia and industry. From academia, we evaluate LOTUS [38], Palimpzest [31], and ThalamusDB [20]. From industry, we evaluate Google’s BigQuery system [12], now offering support for semantic operators (“AI Functions”). All evaluated systems are currently undergoing rapid changes, and our results should be understood merely as snapshots. We publish an online leaderboard for our benchmark¹ that will be regularly updated as new results become available. We analyze our initial experimental results to derive insights regarding the strengths and weaknesses of the evaluated systems, the impact of specific performance optimization techniques, as well as avenues for future research. In summary, our original scientific contributions in this paper are the following:

- We introduce a new benchmark that focuses on the emerging class of semantic query processing systems. This benchmark features queries with semantic operators on multimodal data. The benchmark contains 5 scenarios, and a total of 55 queries, covering analysis across three modalities: text, image, and audio.
- We present an initial experimental study of the benchmark for a variety of semantic query processing engines, including systems from industry as well as from academia.
- We analyze these experimental results, link performance differences to specific query properties, and study the impact of different performance optimization techniques.

The remainder of this paper is organized as follows. In Section 2, we provide background and discuss related work. In Section 3, we discuss the principles underlying our benchmark design. Next, in Section 4, we describe the specific benchmark scenarios we created in detail. Section 5 introduces the systems used in our evaluation. Section 6 reports experimental results for different benchmarks and systems. Finally, Section 7 summarizes our experimental results and discusses future directions.

2 BACKGROUND

We discuss developments that led towards semantic query processing engines. Also, we discuss prior benchmarks and the differences.

2.1 Crowdsourced Data Processing

The idea of semantic operators is not new. The vision of expanding SQL with operators configured in natural language originated in the early 2010’s in the area of crowdsourced database systems. Systems such as CrowdDB [15], Deco [37], or Qurk [35] use human crowdworkers, hired on platform such as Amazon Mechanical Turk [2], to perform tasks on multimodal data according to user instructions. Many of those systems support SQL-style query interfaces that enable users to include instructions (automatically conveyed to crowd workers during processing) as a part of their queries. At the time, only human workers were flexible enough to perform diverse tasks with various types of data according to instructions formulated in natural language. The high cost and latency that comes with

¹<https://sembench.ngrok.io/>

using crowdworkers has inspired a large body of research, aimed at making crowdsourced database systems more efficient (e.g., via specialized operator implementations [8, 33, 34] or query optimization variants [11, 28, 37]). Despite those advances, processing data with crowdworkers remains expensive and, due to the limited number of crowdworkers, the scalability of the approach is limited.

2.2 Large Language Models

Large language models (LLMs) have enabled stunning advances in multiple areas of computer science over the past years. Those advances have been fueled by two key ideas: the Transformer architecture [50], making it easier to scale models up to large parameter counts, and the idea of transfer learning [41], i.e., training models on generic tasks for which large amounts of (unlabeled) training data are readily available, hoping for a knowledge transfer to other tasks where training data is sparse. For instance, current LLMs are often trained on the task of predicting the next word (or, more precisely, token, the atomic unit at which language models represent text) in arbitrary web text [40]. To perform well on that task, LLMs must develop capabilities akin to a rudimentary level of natural language understanding, as well as commonsense knowledge. Those skills are useful for many other tasks as well. While early-stage LLMs required fine-tuning for specific tasks, recent LLMs trained on large-scale generic corpora can address new tasks [6]. They can do so based on a natural language description alone (zero-shot learning), and their performance can further improve when provided with a few examples (few-shot learning) or when leveraging their reasoning ability. Those capabilities have enabled a new generation of systems, discussed next.

2.3 Semantic Query Processing Engines

Crowdsourced database systems introduced semantic operators. However, their scalability is limited due to their reliance on crowdworkers. Over the past two years, a new generation of database systems have appeared that support semantic operators but rely on LLMs to evaluate them. These systems include several research prototypes from academia [10, 20, 31, 38, 49], as well as several industry systems by major Cloud providers, in particular Google’s BigQuery [12] and Snowflake’s platform [7], now supporting semantic operators as well. A more detailed description of those systems is given in Section 5.

2.4 Related Benchmarks

Our benchmark focuses on analytical data processing. In that, it relates to other popular benchmarks in the database community, for instance, TPC-H [47], TPC-DS [46], and the Join Order Benchmark [25]. However, all of the aforementioned benchmarks focus on purely relational data processing. In that context, overheads due to data movements and relational operator evaluations are dominant. Instead, SemBench focuses on queries with semantic operators. For such queries, the overheads due to LLM calls for semantic operator evaluations are dominant. Hence, our benchmark motivates a very different set of techniques to optimize performance.

SemBench focuses on multimodal data processing via LLMs. In that, it relates to prior work proposing benchmarks for multimodal question answering [4, 27, 32]. However, prior benchmarks in this

domain typically aim at evaluating multimodal models, focusing on output quality rather than cost and execution time. Instead, SemBench aims at evaluating systems for semantic query processing, measuring processing overheads, along with quality. It considers complex queries mixing semantic operators (evaluated via language models) with traditional SQL operators.

3 BENCHMARK DESIGN PRINCIPLES

We discuss guidelines that were followed in the benchmark design.

3.1 Data

For our benchmark, we assume a relational data model with an extended type system. Beyond the SQL standard data types, columns may also contain images or audio files. For our benchmark, we store the path to the associated (image or audio) files in the corresponding column. All evaluated SQPEs support this data format.

Our benchmark data is based on manually annotated data sets, created for tasks such as image, text, or audio data classification. We combine such data sets (obtained primarily from the Kaggle platform), in some cases enriched by randomly generated data, to generate databases for our benchmark scenarios. By leveraging ground truth labels that come with the original datasets, we can obtain ground truth results for our benchmark queries.

Most of our benchmark databases contain data of multiple modalities (e.g., images and text). However, we also include scenarios that focus on one single modality, in particular text, in addition to tabular data. This enables us to evaluate SQPEs that support only a subset of data types.

3.2 Queries

We generate at least ten queries for each benchmark scenario. Each of our queries contains semantic operators, possibly mixed with standard SQL operations. For all of our queries, processing overheads due to semantic operators dominate total computation costs. This reflects the focus of our benchmark on techniques that reduce the overheads of semantic query processing.

Our queries cover a wide range of semantic operators. Table 1 summarizes the semantic operators supported by SemBench. These operators include semantic filters, which select multimodal data based on natural language instructions; semantic joins, which identify item pairs that satisfy specified matching conditions; semantic maps, which transform data according to natural language descriptions; semantic ranks, which order items based on text-described criteria; and semantic classify, which assigns items to predefined categories. The set of supported semantic operators differs across SQPEs. By analyzing available engines, we identified operators that are supported by most current SQPEs, as well as operators for which support is sparse. Our goal in creating the benchmark queries was to include queries that can be evaluated by the majority of SQPEs, as well as some queries that require more “exotic” operators (possibly motivating the inclusion of such operators in future system versions). Table 1 defines all semantic operators used in our benchmark, along with examples.

Table 1: Summary of key semantic operators covered by SemBench. T is a relation, X, Y are tuple types, t is a specific tuple, l is a natural language expression, and M is a model processing tuples based on l . Different systems offer various ways to define semantic operators. We adopt the syntax from [38] for its simplicity and clarity, and we redefine and extend several operators to support better generalization.

Operator	Definition	Example
<code>sem_filter($l : X \rightarrow Bool$)</code>	$\{ t \in T \mid M(l)(t) = 1 \}$	Select patients with sick lung based on X-ray images
<code>sem_join($l : (X, Y) \rightarrow Bool$)</code>	$\{ (t_i, t_j) \mid M(l)(t_i, t_j) = 1, t_i \in T_1, t_j \in T_2 \}$	Match two images of the same animal
<code>sem_map($l : X \rightarrow Y$)</code>	$\{ M(l)(t) \mid t \in T \}$	Extract brand names from product descriptions
<code>sem_rank($l : T[X] \rightarrow Seq[X], k$)</code>	$\langle t_1, \dots, t_k \rangle$ ranked by $M(l)$	Rank reviews by positivity
<code>sem_classify($l : X \rightarrow Y$)</code>	$\{ (t, M(l)(t)) \mid t \in T \}$	Classify animals by species based on images

3.3 Metrics

We evaluate SPQEs according to processing overheads and according to result quality. For the former, we consider run time, the number of LLM calls, the number of input and output tokens, and the monetary processing fees. Among those metrics, we consider processing fees as the most important one, since it derives from token consumption and, additionally, takes into account the size of the model used.

To assess result quality, we compare results generated by the SPQEs to the ground truth. We generate ground truth by exploiting the manual labels that come with our data sets (e.g., created for tasks such as classification). By executing queries on our data, assuming that LLM calls return data that is consistent with manual labels, we obtain ground truth for our query results. The metric used to measure the result accuracy depends on the query type. For aggregation queries, calculating SQL aggregates such as counts or sums, we measure the relative distance between the ground truth aggregate value and the value returned by an SQPE. For retrieval queries, we calculate the F1 score, considering recall as the ratio of ground truth result values contained in the generated result, and precision as the ratio of rows in the generated result that appear in the ground truth result as well. For ranking queries, we use Spearman’s rank correlation coefficient to measure the strength of correlation between the ranking produced by an SQPE and the ranking derived from ground-truth scores. For queries involving grouping operations—such as classification, simple mappings, or certain types of joins—we use the Adjusted Rand Index (ARI), a standard clustering similarity metric, to assess accuracy.

4 BENCHMARK SCENARIOS

In the following subsections, we describe each of our benchmark scenarios in more detail, focusing on both, workload and data. In Section 4.6, we compare all scenarios according to multiple criteria.

4.1 Movies

This scenario analyzes and compares movie reviews using sentiment analysis. Since text is the most broadly supported modality, the scenario is constructed to evaluate a wide range of systems and semantic operators using only tabular and textual content. We use movie review data from Kaggle [3] and construct two tables:

- 1 `Movies(id text, title text)`
- 2 `Reviews(id text, reviewId text, reviewText text)`

The `Movies` table contains metadata for each movie, and the `Reviews` table stores critic reviews. In this scenario, our queries are designed to evaluate a large number of systems with various semantic operators, including semantic filter (e.g., selecting five positive reviews for a given movie), semantic join (e.g., checking whether two reviews express the same or opposite sentiment), semantic classification (e.g., classifying movie reviews into different sentiments), and semantic ranking (e.g., ranking reviews based on the degree of positivity).

Example 4.1. The following query counts the number of positive reviews for the movie `taken_3`.

- 1 `SELECT COUNT(*) AS positive_review_cnt`
- 2 `FROM Reviews R`
- 3 `WHERE R.id = 'taken_3'`
- 4 `AND AI.IF(R.reviewText, 'this review expresses a positive sentiment');`

4.2 Wildlife

This scenario aims at identifying the presence and co-occurrence of animal species, based on camera trap pictures (we use a Kaggle data set containing camera trap pictures with species labels [42]) and audio recordings (similarly, we use a Kaggle data set [18, 36] containing animal sound recordings with associated species labels). We generate a database with two tables:

- 1 `ImageTable(image img, city text, stationID text)`
- 2 `AudioTable(audio audio, city text, StationID text)`

Both tables combine the image or audio recording with a randomly generated station ID and city (five possible cities and four possible stations). In this scenario, our queries focus in particular on semantic filters on audio and image data.

Example 4.2. The following query determines cities where elephants are present, indicated either by a corresponding picture or audio recording.

- 1 `SELECT DISTINCT city`

```

2 FROM (SELECT city FROM ImageData I WHERE
3   AI.IF(I.image, 'the picture shows an elephant'))
4 UNION (SELECT city FROM AudioData A WHERE
5   AI.IF(A.audio, 'this sounds like an elephant'));

```

4.3 E-Commerce

The E-Commerce scenario is inspired by an online fashion retail store, with the underlying dataset being available on Kaggle [1]. The dataset covers textual data, e.g., product description, structured data, as well as image data for each item, and overall comprises 44446 items, which also corresponds to the maximum scale factor. Products include various clothes as well as shoes, accessories, personal care, and home equipment. The ground truth for each of the queries is established using the columns with structured data, e.g., the primary color of the item, the type of item etc.

```

1 styles_details(id text, productDescription text,
   productImage img ...)

```

The queries of the E-Commerce scenario can be split into two groups, each with a separate goal: Single operator queries, which allow testing individual operators in isolation, as well as a set of complex queries with multiple operators.

Queries 1-9 assess individual semantic operator performance in an isolated way across systems on textual and image modality. Specifically, this includes the operators SEM_FILTER, SEM_MAP, and SEM_CLASSIFY on textual and image data respectively, as well as SEM_JOIN on text-to-text, text-to-image, and image-to-image joins. This allows establishing a fundamental understanding of operator performances between different systems as well as assessing the impact of new ways of implementing operators in a standardized and easy-to-understand setting. The prompts in these queries are deliberately kept simple with respect to modern LLM-standards to primarily focus on how efficiently a system can interact with a model and not on model-performance itself.

Queries 10-14 are more complex, requiring multiple different semantic operators on multiple modalities, oftentimes two different modalities as input to a single operator, and include many opportunities for complex cross-operator optimizations and other query optimization techniques, including prompt simplification and join ordering.

Currently, not all systems support all of these complex queries—in parts due to not supporting all required operators, in other parts due to excessive query runtimes even with a small scale factor (500).

Example 4.3. The following query finds product IDs of Reebok backpacks using semantic filtering on product descriptions.

```

1 SELECT id
2 FROM styles_details
3 WHERE AI.IF('This is a backpack from Reebok',
4   productDescription);

```

4.4 Medical

This scenario simulates a multi-modal hospital Electronic Health Records (EHR) system to assist healthcare professionals with investigating disease presence and co-occurrence through diverse data modalities. We construct five interconnected tables based on four publicly available, anonymized datasets from Kaggle [5, 22, 23, 48] and Mendeley [13, 14]:

```

1 Patient(PatientId int, Age int, Gender text,
2   SmokingHistory text, DidFamilyHaveCancer bool)
3 SymptomsText(PatientId int, SymptomId int,
4   Symptom text)
5 XrayImage(PatientId int, XrayId int,
6   XrayImage image)
7 SkinMoles(PatientId int, SkinImage_id int,
8   SkinImage image)
9 LungAudio(PatientId int, AudioId int,
10  Location text, FiltrationType text, Audio audio)

```

The medical scenario contains all four modalities covered in our benchmark (tables, text, images and audio). In particular, it integrates tabular patient data with diverse diagnostic modalities: first-person symptom descriptions (text), human lung sound recordings (audio), chest X-rays (imaging), and skin mole mapping (dermatological imaging). Each patient may have zero to multiple diagnoses, while diagnostic data can indicate either pathological conditions or healthy states. Patients can be subject to zero or multiple diseases. When feasible, diseases are represented across compatible modalities, e.g., a pneumonia patient may have both abnormal chest X-rays and pathological lung sounds. Alternatively, patients may have multiple distinct conditions.

To construct the data for the scenario, we synthesize patient data and randomly assign conditions to patients, adhering to specific rules for disease co-occurrence. For evaluation, we utilize the original labels from the source datasets as the ground truth (which are not exposed to the evaluated system). We want to stress that the purpose of this scenario is to evaluate the performance of semantic query processing systems, and that no medical insights can or should be derived from this dataset.

Example 4.4. The following query determines all sick patients.

```

1 (SELECT PatientId FROM SymptomsText WHERE
2   AI.IF('The patient is sick according to the
3     symptoms.', Symptom))
4 UNION DISTINCT
5 (SELECT PatientId FROM LungAudio WHERE
6   AI.IF('The patient is sick according to the
7     audio.', Audio))
8 UNION DISTINCT
9 (SELECT PatientId FROM XrayImage WHERE
10  AI.IF('The patient is sick according to the X-ray
11    image.', XrayImage))
12 UNION DISTINCT
13 (SELECT PatientId FROM m.SkinMoles WHERE

```



```
11 AI.IF('The mole or skin patch is malignant/sick
    according to the image.', SkinImage));
```

4.5 MMQA

The MMQA scenario simulates a multi-modal question answering system and is built on top of the MultiModalQA dataset [45]. It covers three modalities including tables, texts and images. For this scenario, we construct five tables from Wikipedia tables and texts within the MultiModalQA dataset, along with an additional meta-data table for the images from the same dataset. We also curate 11 questions that involve various AI operators such as semantic filters, joins, extraction and summarization.

```
1 ap_warrior(id int, finish varchar, race varchar,
    distance varchar, track varchar, ...)
2 ben_piazza(year int, title varchar, role varchar,
    notes varchar)
3 ben_piazza_text_data(row_id int, title varchar, url
    varchar, id varchar, text varchar)
4 lizzy_caplan_text_data(row_id int, title varchar,
    url varchar, text varchar)
5 tampa_international_airport(row_id int, airlines
    varchar, destinations varchar, airport varchar)
6 images(row_id int, image_filename varchar,
    image_filepath varchar)
```

Example 4.5. The following query joins airlines with images containing their logos.

```
1 SELECT t.Airlines, i.uri
2 FROM mmqa.tampa_international_airport t,
    mmqa.images i
3 WHERE AI.IF("You will be provided with an airline
    name and an image. Determine if the image
    shows the logo of the airline. Airline: ",
    t.Airlines, ", Image: ", i.uri);
```

4.6 Comparison of Scenarios

Table 2 compares all of the scenarios introduced before. The scenarios are complementary in terms of the data types to which semantic operators are applied. The Medical scenario is most diverse in terms of data modalities, applying semantic operators to text, images, and audio files. On the other hand, the Movies scenario is more restricted and applies semantic operators only to analyze text data. The Movies scenario enables developers to use our benchmark in parts for evaluating SQPEs that only support a limited set of modalities (typically, text).

Different scenarios focus on different semantic operators. The semantic filter operator, supported by all of the SQPEs we tested, is used in all scenarios. The E-Commerce scenario is the most join-heavy, using 9 semantic join operators in its 10 benchmark queries. Three scenarios use semantic classification, whereas two scenarios use semantic ranking and the semantic map operator, respectively.

Additionally, Table 2 shows the number of labeled items for each data modality (text, images, and audio files) that appear in each benchmark data set. Note that data processing via semantic operators is much more expensive than data processing with relational operators. With current systems, processing even a small part of our benchmark data sets is prohibitively expensive for many of our queries. Therefore, we only use a small part of our data (a few hundred to a few thousand rows, depending on the scenario) for the experiments. While the number of rows may seem small, compared to the row counts typically used to evaluate systems on benchmarks such as TPC-H, it is largely sufficient to evaluate SQPEs.

5 BENCHMARK SYSTEMS

We describe the systems used for our experiments in detail.

5.1 LOTUS

LOTUS [38] is a query engine that integrates LLMs to process both structured and unstructured data. It offers a declarative programming interface based on the DataFrame abstraction, along with an optimized execution engine that reduces cost and latency by approximating a predefined “gold algorithm”. This gold algorithm is a physical implementation of semantic operators using LLMs. Approximations can take various forms, such as using a less expensive model (e.g., replacing a powerful but expensive model gpt-4o with a smaller model gpt-4o-mini), or using embedding similarity (e.g., applying a pre-trained transformer model like e5-small-v2 [51] to compute embeddings and then evaluating similarity scores to determine whether items satisfy a join condition). The main objective of LOTUS is to reduce monetary and computational cost while maintaining accuracy guarantees with respect to the gold algorithm. LOTUS supports all semantic operators listed in Table 1, and it is capable of processing both text and image data. However, at the time of writing, LOTUS does not support audio data.

5.2 Palimpzest

Palimpzest [31] is a Python programming framework and query engine which enables developers to write declarative AI programs with semantic operators. The framework features lazy execution semantics with an interface based on the DataFrame abstraction. Palimpzest’s cost-based query optimizer [43]—modeled after the Cascades relational query optimizer [16]—samples a multitude of physical operators and leverages ground truth labels and/or LLM-based judging in order to optimize the physical query plan. If the query optimizer is not provided with ground truth labels or an LLM judge, it instead uses prior beliefs about physical operators’ cost, latency, and quality to optimize the plan.

In addition to its physical optimizations, Palimpzest also supports logical optimizations including filter pushdown and join re-ordering. Palimpzest is capable of optimizing query plans for quality, cost, or latency subject to a constraint on zero or more of the other dimensions. Finally, Palimpzest supports all semantic operators listed in Table 1 and is capable of processing text, image, and audio data.

Table 2: Comparison of different scenarios.

Scenario	#Queries	Modalities				Number of Semantic Operators					Modality Sizes (#rows)		
		Table	Text	Image	Audio	Filter	Join	Map	Rank	Classify	Text	Image	Audio
Movie	10	✓	✓	–	–	4	3	–	2	1	1,375,738	–	–
Wildlife	10	✓	–	✓	✓	17	–	–	–	–	–	8,718	650
E-Commerce	14	✓	✓	✓	–	12	9	3	1	2	44,446	44,446	–
MMQA	11	✓	✓	✓	–	5	3	4	–	–	5,000	1,000	–
Medical	10	✓	✓	✓	✓	12	–	–	–	1	1,200	10,012	336
Total	55	✓	✓	✓	✓	49	15	7	3	4	1,426,384	64,176	986

5.3 ThalamusDB

ThalamusDB [20] is a semantic query processing engine designed to handle multimodal data. It employs approximate query processing (AQP) to enable users to balance query quality and execution costs. During query execution, ThalamusDB presents partial results with error bounds, allowing users to obtain approximate answers with low latency. To control processing costs, ThalamusDB supports user-defined termination conditions that set thresholds on approximation error, execution time, token usage (i.e., monetary execution fees), or the number of LLM invocations. In addition, ThalamusDB adopts batch processing to reduce latency and lower token consumption. At the time of writing, ThalamusDB supports semantic filtering and joins, and is capable of processing text, image, and audio data.

5.4 BigQuery

BigQuery [12] is a fully managed, serverless, and highly scalable data warehouse on Google Cloud. BigQuery offers a suite of AI functions such as `AI.CLASSIFY`, `AI.IF`, and `AI.SCORE` to enable semantic data processing tasks. Under the hood, BigQuery is tightly integrated with Vertex AI, Google Cloud’s model hosting platform, and can leverage models such as Gemini for inference. BigQuery’s AI functions support row-level inference, for example extracting structured information from text, filtering records, and performing cross-modal semantic joins between text, images, and audio within a single SQL query. BigQuery supports all the semantic operators listed in Table 1 and all the modalities covered by SemBench (text, image, and audio). For cost accounting, BigQuery reports per-query token usage, including per-modality input tokens, thinking tokens from reasoning models, and output tokens, which allows users to compute monetary costs based on these consumptions.

5.5 Other Semantic Query Processing Engines

We considered including several other SQPEs in our experiments. FlockMTL [10] supports semantic query processing on DuckDB. However, the system does not report token usage, preventing us from calculating token consumption and processing fees.

CAESURA [49] proposes using LLMs for generating Python code for data processing (which may, in turn, involve calls to LLMs). It assumes natural language queries as input. We transformed our SQL queries on the Movies scenario into natural language versions. After that, CAESURA was able to generate executable code for the

first two queries in the Movies scenario, while exceeding the pre-set number of retries for the remaining ones. CAESURA took 119 seconds to process the first query with a cost of 41 Cents, resulting in an F1 score of 60%. For the second query, it produced executable code within 117 seconds with a cost of 56 Cents but obtained a recall of 0. By default, CAESURA uses the GPT-4 model. This shows that the approach adopted by CAESURA can, in principle, work in our scenario. However, as SemBench focuses primarily on semantic SQL queries while CAESURA processes natural language input, a comparison with the other systems would not be fair.

6 EXPERIMENTAL RESULTS

In Section 6.1, we describe our general experimental setup. In Section 6.2, we provide details on how each of the evaluated systems was tuned. Next, in Section 6.3, we report and analyze our experimental results.

6.1 Experimental Setup

Hardware Settings. All experiments are conducted on an EC2 instance of type `g4dn.2xlarge`, configured with 32 GB of RAM, 8 vCPUs, an NVIDIA T4 GPU with 16 GB of memory, and 250 GB of disk storage. The GPU is used solely for embedding computation, and we do not deploy LLMs locally. It should be noted that the major bottleneck of SQPE are (remote) model-involutions; by using the described machine, we made sure that query execution does not hit any other hardware limits.

Model Settings. For all main experiments presented in the paper in Section 6.3, we use `gemini-2.5-flash`, which belongs to the latest Gemini-series, as the backing LLM across all systems. The choice of the model has the biggest influence on system performance—hence it is crucial to evaluate all systems with the same model. We chose `gemini-2.5-flash` because it is available for use in all evaluated systems and supports all required modalities in the benchmark. Our experiments also showed that the model strikes a good balance between cost and output quality in the context of large-scale multimodal data processing. Additionally, We disable the reasoning capability of the LLM in the main experiments due to long processing times and high monetary cost and, as we see in the Appendix, not necessarily better results. We further set the model temperature to 0 for applicable systems for reproducibility. Results for additional models can be found on our online leaderboard at <https://semlench.ngrok.io/>.

Table 3: Scale Factors for Different Scenarios

Scenario	Available Range	Scale Factor Used
Movie	1–1,375,738	2,000
Wildlife	1–8,718	200
E-Commerce	50–44,446	500
Medical	1–11,112	11,112
MMQA	25–1000	200

Degree of Parallelism. We set the degree of parallelism for LLM invocation to 20 for all applicable systems (LOTUS, Palimpzest, and ThalamusDB). BigQuery does not give users the option to control this setting.

Scenario Settings. As shown in Table 2, the benchmark corpus includes over 1.4 million text rows, approximately 93 thousand image rows, and nearly one thousand audio rows. These data volumes support meaningful evaluation of LLMs and semantic query processing engines in terms of both processing time and cost. To control database sizes systematically, we define a scale factor (SF) for each scenario, corresponding to the maximum number of rows in that scenario. Table 3 lists the available SF ranges per scenario, along with the specific SFs used in our main experiments.

Evaluation Metrics. We report monetary cost, result quality, and query latency in our evaluation. For quality metrics, we use the metrics described in Section 3.3 in our experiments. All metrics are normalized to the range $[0, 1]$, where higher values indicate better performance. This normalization ensures consistency in both visualization and comparison. Specifically, the F1 score naturally falls within $[0, 1]$. For relative error, we apply the transformation $1/(1 + \text{relative_error})$ to map it into $[0, 1]$. Spearman’s rank correlation [9] and the Adjusted Rand Index (ARI) originally range from $[-1, 1]$, where -1 indicates opposite ranking or clustering similarity, 1 indicates identical outcomes, and 0 corresponds to random behavior. In our experiments, all evaluated systems produce rank correlation and ARI values greater than 0 . Therefore, we do not perform additional normalization for these two metrics.

Evaluation Settings. Each experiment is repeated five times. We report the average performance of monetary cost, quality, and latency per query, as well as the overall averages across all queries. In the tables, we also report the standard deviation across all queries. Due to space limitations, the per-query deviations are omitted, but are available in our online leaderboard.

6.2 System Settings

All systems are under active development, and we did not tune any system to achieve its best possible performance. Hence, the reported results are only a snapshot in time that provide directions for future improvements. We encourage authors of systems to submit optimized versions to the SemBench leaderboard.

LOTUS. We use LOTUS version 1.1.3 for our experiments. LOTUS provides options for optimized semantic filter, join, and rank. In evaluation, we fix the model to `gemini-2.5-flash`. For semantic join, we use the optimized operator from LOTUS, where embedding similarity scores are used as approximations to LLM calls. Following the recommendations of the authors, we use `e5-base-v2` for text

embeddings and `clip-ViT-B-32` for image embeddings. For ranking, LOTUS provides semantic map and semantic top-k operators. In our experiments, we find that semantic map and semantic top-k achieve similar ranking quality, but semantic top-k incurs significantly higher costs. Therefore, we use the semantic map operator for ranking queries.

Palimpzest. We use Palimpzest version 0.8.2 with the Abacus optimizer [43] turned off. We apply the MaxQuality objective, which configures Palimpzest to use its default “best” operator implementation(s) based on prior belief. We use the parallel execution strategy with the degree of parallelism set to 20. Model selection is disabled since a fixed model is used in the evaluation.

ThalamusDB. We use version 0.1.15 and the default settings for all parameters. In particular, we use a degree of parallelism of 20 and set the batch size for the join operator to ten (i.e., ten rows from both input tables are integrated into the same prompt). Specifically for the Medical scenario, we slightly rewrote queries to eliminate SQL WITH clauses, materializing the query results described in the WITH clause as temporary tables.

BigQuery. In the evaluation, we use a default reservation of 2000 slots and fix the model to `gemini-2.5-flash`. Scalability parameters such as LLM parallelism or batch size are managed by the engine and are not controllable by users.

6.3 Benchmark Results

Table 4 reports the results for the Movies scenario. The colors indicate the relative performance of the different systems for each query. For cost and latency, we normalize the values using $(v - \min)/(\max - \min)$, and classify the systems into three categories: the top 33% are labeled *Better*, the middle 33% are labeled *Middle*, and the bottom 33% are labeled *Worse*. For quality, we use absolute thresholds: values in $[0.8, 1]$ are labeled *Better*, values in $[0.6, 0.8)$ are labeled *Middle*, and values in $[0, 0.6)$ are labeled *Worse*. The left-most column shows the query number together with the semantic operators used in the query.

For the Movies scenario, most systems support all queries. ThalamusDB does not support the last two queries, requiring semantic rank operators, which are not supported by ThalamusDB. Despite a small SF=500, we find some queries (e.g., Q7) require several minutes of processing time, along with non-negligible monetary cost, for most of the compared systems. This shows that even relatively small amounts of data compared to database standards pose significant challenges in semantic query processing.

Table 4 shows per-query average values for each compared system, along with the relative standard deviation. Note that the standard deviation is usually small for cost and quality, compared to run time. Cost and quality variance are due, primarily, to randomization of the LLM output. However, the compared systems generally try to minimize randomization of the LLM output, e.g., by choosing a low temperature whenever possible. Run time, on the other hand, is influenced primarily by the latency variance of LLM calls.

Comparing only systems supporting all queries, BigQuery, the only commercial system in our evaluation, achieves optimal performance according to all metrics, followed closely by LOTUS (in

Table 4: Experimental Results of SemBench for the Movie Scenario. Applied operators: F - semantic filter, J - semantic join, R - semantic rank, C - semantic classify, L - LIMIT clause. The colors express relative performance (Better than Average, Average, Worse than Average, and X for not supported).

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
Q1: F L	\$0.09	1.00	33.1 s	$1 \cdot 10^{-3}$	1.00	3.8 s	$4 \cdot 10^{-4}$	0.95	4.2 s	\$0.05	1.00	26.3 s
Q2: F L	\$0.01	1.00	2.1 s	\$0.01	1.00	29.7 s	$2 \cdot 10^{-3}$	0.92	1.9 s	$3 \cdot 10^{-3}$	1.00	9.5 s
Q3: F	$5 \cdot 10^{-3}$	0.64	2.1 s	\$0.01	0.64	4.6 s	$2 \cdot 10^{-3}$	0.74	3.1 s	$3 \cdot 10^{-3}$	0.64	11.0 s
Q4: F	$5 \cdot 10^{-3}$	0.64	2.8 s	\$0.02	0.74	4.4 s	$2 \cdot 10^{-3}$	0.74	3.8 s	$3 \cdot 10^{-3}$	0.64	11.4 s
Q5: J L	\$2.38	0.59	536.5 s	\$0.01	0.39	1.9 s	$1 \cdot 10^{-3}$	1.00	2.3 s	\$1.01	0.89	54.5 s
Q6: J L	\$1.81	0.67	432.4 s	\$0.01	0.83	2.3 s	$9 \cdot 10^{-4}$	0.84	1.7 s	\$1.00	0.69	54.5 s
Q7: J	\$1.81	0.21	431.8 s	\$7.72	0.68	1056.1 s	\$0.15	0.57	636.9 s	\$3.31	0.70	198.3 s
Q8: C	$4 \cdot 10^{-3}$	0.93	2.3 s	\$0.02	0.86	4.3 s	$5 \cdot 10^{-3}$	0.83	6.8 s	$3 \cdot 10^{-3}$	0.76	10.9 s
Q9: R	\$0.02	0.75	4.9 s	\$0.05	0.78	5.7 s	X	X	X	\$0.02	0.78	13.3 s
Q10: R	\$0.13	0.40	30.9 s	\$0.38	0.42	39.2 s	X	X	X	\$0.13	0.44	32.1 s
Avg	\$0.62	0.68	147.9 s	\$0.82	0.73	115.2 s	\$0.02	0.82	82.6 s	\$0.55	0.75	42.2 s
Std Dev	$\pm 1.2\%$	$\pm 1.5\%$	$\pm 17.4\%$	$\pm 2.2\%$	$\pm 3.4\%$	$\pm 12.7\%$	$\pm 0.8\%$	$\pm 0.1\%$	$\pm 9.2\%$	$\pm 0.4\%$	$\pm 1.6\%$	$\pm 6.4\%$

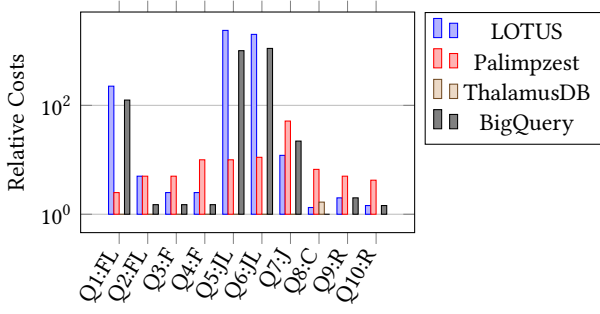


Figure 1: Relative costs in the Movies scenario.

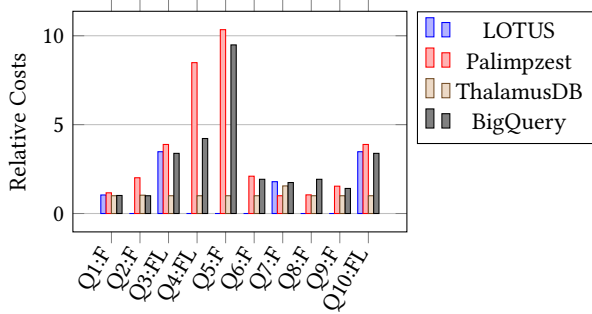


Figure 2: Relative costs in the Wildlife scenario.

terms of cost) and Palimpzest (in terms of quality). While ThalamusDB only supports a subset of queries in this scenario, it achieves optimal costs for each supported query (eight out of ten).

Interestingly, there are several queries for which the relative cost difference between different systems is significant. Figure 1 shows the relative cost of all systems, dividing processing costs for each

query by the minimal costs (over all compared systems). For Queries Q1, Q5, and Q6, processing costs vary by more than a factor of 100× across different systems. All three queries use a LIMIT clause. We found that the treatment of LIMIT clauses differs between the compared systems. For most SQPEs, the LIMIT clause is applied only in a post-processing step, after applying semantic operators to all data. SQPEs that terminate evaluation via semantic operators early, whenever the required number of rows to satisfy the LIMIT clause is available, achieve significantly better performance for those queries. Query Q2 uses a LIMIT clause as well, but restricts the scope to reviews of one specific movie (using SQL standard equality predicates, which are evaluated efficiently). In this case, applying semantic operators to all relevant data is less costly than for the other three LIMIT queries (which consider all reviews).

Among all queries without LIMIT clause, Q7 is the one with the most significant cost differences. ThalamusDB achieves minimal average processing costs, followed by LOTUS. This is explained by the fact that these two systems use implementations of the semantic join operators that are aimed at reducing execution costs. LOTUS implements relational joins by matching items first based on their embedding vectors, evaluating the join condition via LLM calls only on those row pairs whose embedding vectors are within a certain distance range. ThalamusDB uses batching to include multiple items from both tables into the prompt, instructing the LLM to find all matches regarding the join condition. On the other hand, those cost optimization strategies come at the expense of quality. BigQuery and Palimpzest achieve the highest average quality.

Table 5 shows results for the E-Commerce scenario. This scenario features the most diverse selection of semantic operators, making it challenging to implement it for all systems even though all generally support the required data modalities. LOTUS and BigQuery are the only systems that allow a complete implementation of this scenario, with Palimpzest being only missing out on one query because it does not support SEM_RANK and also doesn't allow emulating SEM_RANK

Table 5: Experimental Results of SemBench for the E-Commerce Scenario. Applied operators: F - semantic filter, J - semantic join, M - semantic map, R - semantic rank, C - semantic classify. The colors express relative performance (Better than Average, Average, Worse than Average, and X for not supported).

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
Q1: F	\$0.06	1.00	12.2 s	\$0.08	1.00	12.2 s	\$0.03	1.00	34.8 s	\$0.04	0.59	21.2 s
Q2: F	\$0.18	0.87	166.1 s	\$0.38	0.83	57.5 s	\$0.31	0.67	100.8 s	\$3.96	0.21	55.7 s
Q3: M	\$0.07	0.97	17.1 s	\$0.12	0.98	16.5 s	X	X	X	\$0.12	0.97	21.2 s
Q4: M	\$0.14	0.45	156.7 s	\$0.24	0.53	335.0 s	X	X	X	\$0.37	0.69	31.0 s
Q5: C	\$0.04	0.99	7.4 s	\$0.07	0.98	6.6 s	X	X	X	\$0.17	0.98	25.7 s
Q6: C	\$0.12	0.89	114.8 s	\$0.43	0.89	143.7 s	X	X	X	\$0.35	0.88	34.9 s
Q7: J	\$1.33	0.75	199.4 s	\$1.79	0.92	287.6 s	\$0.08	0.51	97.7 s	\$0.86	0.83	45.4 s
Q8: J	\$4 $\cdot 10^{-3}$	1.00	4.1 s	\$0.01	1.00	3.9 s	\$0.30	0.00	713.0 s	\$18.23	0.29	126.2 s
Q9: J	\$0.06	0.55	243.4 s	\$0.10	0.49	44.9 s	\$0.31	0.00	872.2 s	\$0.21	0.58	48.6 s
Q10: F J	\$0.21	0.00	519.5 s	\$1.02	0.06	1192.5 s	X	X	X	X	X	X
Q11: F J	\$0.27	0.78	158.7 s	\$0.61	0.73	132.4 s	X	X	X	X	X	X
Q12: F M	\$0.10	0.60	36.4 s	\$0.14	0.00	31.9 s	X	X	X	\$0.10	0.97	31.1 s
Q13: F	\$0.24	0.74	274.8 s	\$0.44	0.74	238.3 s	X	X	X	\$0.38	0.70	22.4 s
Q14: F J R	\$0.23	0.87	178.2 s	X	X	X	X	X	X	\$4.26	0.37	73.6 s
Avg	\$0.22	0.75	149.2 s	\$0.42	0.70	192.5 s	\$0.21	0.44	363.7 s	\$2.42	0.67	44.7 s
Std Dev	$\pm 41.5\%$	$\pm 2.0\%$	$\pm 27.4\%$	$\pm 0.1\%$	$\pm 0.9\%$	$\pm 70.0\%$	$\pm 0.0\%$	$\pm 0.5\%$	$\pm 31.2\%$	$\pm 1.1\%$	$\pm 7.7\%$	$\pm 20.5\%$

with a SEM_MAP together with a ORDER_BY as the system does not support classical sorting. ThalamusDB only supports 5 out of the 14 queries due to only supporting SEM_FILTER and SEM_JOIN together with the limitation that only a single input column for semantic operators is supported. While Q10 and Q11 are implementable in BigQuery, they do not complete within a reasonable execution time. Notably, LOTUS consistently achieves the lowest cost across all queries, being outperformed only by ThalamusDB on Q1 and Q7, while still achieving the best average accuracy in this scenario.

Table 6 shows results for the Wildlife scenario. This scenario uses semantic operators on audio data and images. LOTUS does not currently support analysis of audio data. Hence, it only supports four out of ten queries in this scenario. The other systems support all ten queries. Among the systems supporting all queries, ThalamusDB achieves the lowest average costs (followed closely by Palimpzest) while BigQuery maximizes quality.

Figure 2 shows the relative costs of all systems, scaled to the minimal costs (over all systems) for each query. Comparing Figure 2 with Figure 1, we find that cost differences are generally smaller (reaching only a relative cost difference of up to ten, rather than more than a factor of 100). The Wildlife scenario uses only the semantic filter operator, but no semantic joins, which tend to lead to larger cost differences between the compared systems. Analyzing the compared open-source systems, we find significant differences in the prompt templates used to implement the semantic filter. All systems use simple and compact prompt templates, especially for output tokens to reduce costs. Palimpzest uses relatively long prompt templates, containing examples for few-shot learning and detailed instructions. This increases quality while increasing costs

associated with reading input tokens. ThalamusDB uses concise prompt templates that reduce costs at the expense of quality.

Table 7 presents the experimental results for the MMQA scenario. Only Palimpzest fully supports MMQA queries. LOTUS can technically cover all semantic operators but across runs, it can throw internal processing errors. These errors occur when the LLM responses fail to adhere to the specified output schema in the prompt and the following code fail to handle unexpected outputs. This explains the “n/a” entry for Q4 despite LOTUS’s support for the semantic map operator. ThalamusDB only supports a small number of queries, in particular due to the lack of a semantic map operator and a semantic join operator across modalities. While BigQuery can perform one-to-one semantic mapping, they lack the many-to-one semantic map operator, which is essential for summarizing values across multiple rows within a single column.

Among the systems that successfully handled at least nine out of ten queries, BigQuery proved to be the most cost-effective solution. In contrast, Palimpzest achieved the highest average quality, with LOTUS following closely behind. Across all queries, the most challenging and expensive operations—in terms of both monetary cost and latency—were semantic joins between tables and images. This difficulty is illustrated by the results for Q7, which required 40,000 LLM calls for join predicate evaluation. This single query took Palimpzest over 35 minutes and LOTUS nearly 40 minutes to complete. The monetary cost for this query alone was more than 1,000 times that of a filtering query. The inherent complexity of these joins is also reflected in their consistently lower quality scores compared to other operations like semantic filtering and mapping.

Table 6: Experimental Results of SemBench for the Wildlife Scenario. Applied operators include: F - semantic filter, L - LIMIT clause. The colors express relative performance (Better than Average, Average, Worse than Average, and X for not supported).

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
Q1: F	\$0.11	0.79	92.4 s	\$0.13	0.79	32.8 s	\$0.11	0.79	19.6 s	\$0.11	0.79	32.0 s
Q2: F	X	X	X	\$0.01	0.17	2.8 s	\$0.01	0.14	4.5 s	\$0.01	0.19	9.4 s
Q3: F L	\$0.11	1.00	99.4 s	\$0.13	0.00	22.7 s	\$0.03	0.00	4.3 s	\$0.11	0.00	25.7 s
Q4: F L	X	X	X	\$0.01	0.00	2.7 s	$1 \cdot 10^{-3}$	0.00	1.3 s	\$0.01	1.00	9.4 s
Q5: F	X	X	X	\$0.13	0.75	13.5 s	\$0.01	0.75	2.3 s	\$0.12	0.75	19.2 s
Q6: F	X	X	X	\$0.13	0.00	19.3 s	\$0.06	0.50	13.2 s	\$0.12	0.20	24.3 s
Q7: F	\$0.23	1.00	188.3 s	\$0.13	1.00	43.9 s	\$0.20	1.00	28.8 s	\$0.22	1.00	24.6 s
Q8: F	X	X	X	\$0.13	0.75	34.9 s	\$0.12	0.75	23.8 s	\$0.23	0.75	35.5 s
Q9: F	X	X	X	\$0.13	0.57	19.1 s	\$0.08	0.67	17.2 s	\$0.12	0.59	37.6 s
Q10: F L	\$0.11	1.00	87.8 s	\$0.13	0.00	19.6 s	\$0.03	0.00	4.4 s	\$0.11	0.00	40.3 s
Avg	\$0.14	0.95	117.0 s	\$0.10	0.40	21.1 s	\$0.07	0.46	11.9 s	\$0.11	0.53	25.8 s
Std Dev	$\pm 0.0\%$	$\pm 0.0\%$	$\pm 8.0\%$	$\pm 0.0\%$	$\pm 0.0\%$	$\pm 48.7\%$	$\pm 0.0\%$	$\pm 0.0\%$	$\pm 14.1\%$	$\pm 0.0\%$	$\pm 18.0\%$	$\pm 59.6\%$

Table 7: Experimental Results of SemBench for the MMQA Scenario. Applied operators include: F - semantic filter, J - semantic join, M - semantic map. The colors express relative performance (Better than Average, Average, Worse than Average, and X for not supported).

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
Q1: M	\$0.02	1.00	7.4 s	$3 \cdot 10^{-3}$	1.00	4.5 s	X	X	X	\$0.01	1.00	14.1 s
Q2a: J	\$0.89	0.83	169.6 s	\$1.04	1.00	135.9 s	X	X	X	\$0.08	0.00	53.8 s
Q2b: J	\$0.89	0.83	166.8 s	\$1.04	1.00	133.8 s	X	X	X	\$0.12	0.00	38.4 s
Q3a: F	\$0.01	0.83	4.2 s	\$0.02	0.80	4.5 s	\$0.01	0.75	11.0 s	\$0.01	0.72	19.6 s
Q3f: F	\$0.01	1.00	4.2 s	\$0.02	1.00	8.0 s	\$0.01	1.00	7.0 s	\$0.01	0.67	22.3 s
Q4: M	X	X	X	$5 \cdot 10^{-3}$	0.54	1.2 s	X	X	X	$2 \cdot 10^{-3}$	0.60	9.7 s
Q5: M	$1 \cdot 10^{-3}$	1.00	0.48 s	$1 \cdot 10^{-3}$	1.00	0.50 s	X	X	X	X	X	X
Q6a: F	\$0.01	1.00	4.4 s	\$0.02	1.00	4.0 s	$2 \cdot 10^{-3}$	0.33	5.6 s	$4 \cdot 10^{-3}$	0.03	18.9 s
Q6b: F	\$0.01	1.00	3.8 s	\$0.02	1.00	4.1 s	$2 \cdot 10^{-3}$	1.00	5.5 s	$4 \cdot 10^{-3}$	0.04	17.3 s
Q6c: F	\$0.01	1.00	4.6 s	\$0.02	1.00	4.7 s	$2 \cdot 10^{-3}$	0.53	13.4 s	$4 \cdot 10^{-3}$	0.13	17.4 s
Q7: J	\$13.61	0.32	2311.4 s	\$15.65	0.31	2101.6 s	X	X	X	\$1.18	0.00	91.7 s
Avg	\$1.41	0.88	243.4 s	\$1.62	0.88	218.4 s	$5 \cdot 10^{-3}$	0.72	8.5 s	\$0.14	0.32	30.3 s
Std Dev	$\pm 0.0\%$	$\pm 0.2\%$	$\pm 15.5\%$	$\pm 0.0\%$	$\pm 0.2\%$	$\pm 12.1\%$	$\pm 0.0\%$	$\pm 0.0\%$	$\pm 29.0\%$	$\pm 0.3\%$	$\pm 1.5\%$	$\pm 30.1\%$

Table 8 presents the results for the ten queries of the medical scenario, which simulates a multi-modal Electronic Health Record system. Four key limitations emerge from these results. First, current systems demonstrate poor performance in medical audio processing. Similar to the Wildlife scenario, queries Q2, Q5, and Q6, which rely on semantic filtering of audio data, are either unsupported by LOTUS or perform poorly across systems. Second, existing systems offer limited operator support. ThalamusDB lacks semantic classification required for Q10 and inconsistently supports common table expressions, necessitating the manual materialization of temporary tables for Q6 instead. Palimpzest does not support relational

joins and requires us to hardcode the join order and execute the joins with Pandas instead, e.g., for Q6 and Q7. Third, Q4, Q6, Q7, Q8, and Q10 exhibit similar quality across all evaluated systems, while the execution costs differ significantly, with BigQuery being the most expensive. The most significant cost differences appear in Q3, Q5, Q6, and Q8, which process image data, again causing BigQuery costs to increase substantially. We attribute this to the higher computational cost of image semantic operators in BigQuery relative to text-only queries. Fourth, in the case of semantic operators for textual symptom detection, queries targeting patients

Table 8: Experimental Results of SemBench for the Medical Scenario. Applied operators include: F - semantic filter, J - semantic join, C - semantic classify, L - LIMIT clause. The colors express relative performance (Better than Average, Average, Worse than Average, and X for not supported).

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
Q1: F	\$0.06	0.40	18.8 s	\$0.10	0.31	21.1 s	\$0.02	0.28	32.8 s	\$1.01	0.31	37.2 s
Q2: F	X	X	X	\$0.12	0.08	9.4 s	\$0.15	0.04	25.6 s	\$0.25	0.00	45.4 s
Q3: F L	\$0.73	0.60	235.5 s	\$4·10 ⁻³	0.60	3.0 s	\$0.02	0.91	3.2 s	\$5.94	1.00	56.0 s
Q4: F	\$0.06	0.99	18.3 s	\$0.10	0.96	20.7 s	\$0.02	0.94	32.9 s	\$0.65	0.98	29.2 s
Q5: F	X	X	X	\$4·10 ⁻³	1.00	2.5 s	\$0.41	0.50	80.5 s	\$5.38	0.50	72.6 s
Q6: F	X	X	X	\$0.07	0.48	10.7 s	\$0.03	0.00	6.2 s	\$0.80	0.53	26.1 s
Q7: F	X	X	X	\$4.24	0.86	617.1 s	\$0.25	0.14	74.6 s	\$14.46	0.87	118.8 s
Q8: F L	\$0.11	0.73	149.6 s	\$0.08	0.65	19.6 s	\$0.09	0.61	81.1 s	\$2.81	0.56	43.0 s
Q9: F	\$0.32	0.57	174.5 s	\$0.43	0.38	67.2 s	\$0.48	0.00	3432.5 s	\$1.50	0.55	53.7 s
Q10: C	\$0.16	0.55	22.1 s	\$0.26	0.51	27.1 s	X	X	X	\$3.78	0.58	46.2 s
Avg	\$0.24	0.64	103.2 s	\$0.54	0.58	79.8 s	\$0.16	0.38	418.8 s	\$3.66	0.59	52.8 s
Std Dev	±0.0%	±0.1%	±4.9%	±1.5%	±0.3%	±9.3%	±0.0%	±15.1%	±10.9%	±5.2%	±0.7%	±28.9%

with more specialized diseases (e.g., Q4 - acne, Q10 - disease classification) yield higher quality results compared to those involving more generic conditions (e.g., Q1 - allergy). We hypothesize that large language models perform better when the problem space is narrower and more well-defined. The results indicate that none of the evaluated systems are yet capable of adequately addressing all queries in the specialized medical scenario.

Interestingly, when we tested different language models, some of them (e.g., gemini-2.5-pro) initially refused to perform skin mole mapping unless the prompt explicitly stated that the results would not be used for real-world medical decisions. This model behavior seems to be highly data-dependent though, since this limitation did not occur with X-ray images, textual symptom descriptions, or medical audio data. Unlike in traditional databases, the quality of results produced by semantic operators is highly dependent on the domain of the data being processed, and systems need to be aware of this fact and actively manage it.

7 CONCLUSIONS AND OUTLOOK

We introduced SemBench, a benchmark evaluating a novel class of systems: semantic query processing engines. Despite using only a small part of the available benchmark data for our experiments, we find that each of the evaluated systems incurs non-negligible processing overheads (cost and time) at least for some of the queries. Therefore, we believe that our benchmark will remain useful for stress-testing SQPEs in the coming years.

We identified several factors with a significant impact on the relative performance of SQPEs in our experiments. First, the implementation of semantic operators matters, for instance, for LIMIT and JOIN operators, where we observed significant differences in terms of costs as well as result quality. Second, prompt design matters for semantic operator implementations. Different SQPEs seem to aim at different cost-quality tradeoffs by using either relatively compact prompts, lowering costs, or more detailed prompts

that tend to improve quality. In one scenario, we had to manually expand our instructions to obtain answers on medical topics (a topic for which the LLM refused answers by default). An interesting extension to existing SQPEs would be automated prompt design strategies, optimizing cost-quality tradeoffs, or automatically rewriting prompts to fix situations in which the LLM refuses to generate answers. At this point, none of the evaluated systems implement such strategies.

Several optimizations, currently not or not widely supported by SQPEs, seem useful to improve performance in our benchmark. For instance, fusing multiple semantic operators into one single LLM call offers potential for performance improvements in several scenarios (e.g., in Q10 and Q11 of the E-Commerce scenario). As our scenarios entail multiple queries on overlapping data subsets, some of them with equal or similar predicates, caching strategies would be useful to reduce processing overheads further.

At this point, none of the evaluated systems supports all of our benchmark queries due to limitations either in terms of the supported operators or data modalities. On the other hand, our benchmark includes scenarios in which almost all evaluated systems support all queries. This gives developers of SQPEs the opportunity to start evaluating early-stage versions of their systems, supporting a limited set of features, on easier scenarios, while expanding the scope to more SemBench scenarios as new features are added. As SQPEs expand the range of supported semantic operators over the coming years (e.g., semantic aggregation), we plan to add more scenarios in future SemBench instantiations.

ACKNOWLEDGMENTS

The idea of a benchmark for semantic query processing originated at the Dagstuhl Seminar 25182 on Challenges and Opportunities of Table Representation Learning. We thank the seminar organizers for bringing this group of people together.

REFERENCES

- [1] Param Aggarwal. 2019. Fashion Product Images Dataset. <https://doi.org/10.34740/KAGGLE/DS/139630>
- [2] Amazon. 2018. Amazon Mechanical Turk. <https://www.mturk.com/mturk/welcome>.
- [3] andreza. 2023. Clapper: Massive Rotten Tomatoes Movies & Reviews. Kaggle. <https://www.kaggle.com/datasets/andreza/clapper-massive-rotten-tomatoes-movies-and-reviews>
- [4] Seongsu Bae, Daeun Kyung, Jaehae Ryu, Eunbyeol Cho, Gyubok Lee, Sunjun Kweon, Jungwoo Oh, Lei Ji, Eric Chang, Tackeun Kim, and Edward Choi. 2023. EHRXQA: A Multi-Modal Question Answering Dataset for Electronic Health Records with Chest X-ray Images. In *NeurIPS*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., Red Hook, NY, USA, 3867–3880. https://proceedings.neurips.cc/paper_files/paper/2023/file/0c007ebef1d11fd48da6ce4f54687db6-Paper-Datasets_and_Benchmarks.pdf
- [5] Niyar R Barman, Faizal Karim, , and Krish Sharma. 2023. Symptom2Disease Dataset. Diseases and Natural Language Symptom Descriptions. <https://www.kaggle.com/datasets/niyarbarman/symptom2disease/>
- [6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS* (Vancouver, BC, Canada) (*NIPS '20*). Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.
- [7] Benoît 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, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, New York, NY, USA, 215–226. <https://doi.org/10.1145/2882903.2903741>
- [8] Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Alon Halevy. 2014. Crowd-powered find algorithms. In *2014 IEEE 30th International Conference on Data Engineering*. Ieee, Chicago, IL, USA, 964–975. <https://doi.org/10.1109/ICDE.2014.6816715>
- [9] Yadolah Dodge. 2008. *Spearman Rank Correlation Coefficient*. Springer New York, New York, NY, 502–505. https://doi.org/10.1007/978-0-387-32833-1_379
- [10] 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 (Sept. 2025), 5415–5418. <https://doi.org/10.14778/3750601.3750685>
- [11] Ju Fan, Meihui Zhang, Stanley Kok, Mei Yu Lu, and Beng Chin Ooi. 2015. CrowdOp: Query optimization for declarative crowdsourcing systems. *TKDE* 27, 8 (2015), 2078–2092. <https://doi.org/10.1109/ICDE.2016.7498417>
- [12] Sérgio Fernandes and Jorge Bernardino. 2015. What is BigQuery?. In *Proceedings of the 19th International Database Engineering & Applications Symposium, Yokohama, Japan, July 13-15, 2015*, Bipin C. Desai and Motomichi Toyama (Eds.). ACM, 202–203. <https://doi.org/10.1145/2790755.2790797>
- [13] Luay Fraiwan, Omnia Hassanin, Mohammad Fraiwan, Basheer Khassawneh, Ali M. Ibnian, and Mohanad Alkhodari. 2021. Automatic identification of respiratory diseases from stethoscopic lung sound signals using ensemble classifiers. *Biocybernetics and Biomedical Engineering* 41, 1 (2021), 1–14. <https://doi.org/10.1016/j.bbe.2020.11.003>
- [14] Luay Fraiwan, Omnia Hassanin, Mohammad Fraiwan, Basheer Khassawneh, Ali M. Ibnian, and Mohanad Alkhodari. 2021. A dataset of lung sounds recorded from the chest wall using an electronic stethoscope. <https://doi.org/10.17632/jwyy9np4gv.3>
- [15] MJ Franklin and D Kossman. 2011. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*. 61–72. [http://www.cs.berkeley.edu/~sim\\$rxin/papers/crowddb_sigmod2011.pdf](http://www.cs.berkeley.edu/~sim$rxin/papers/crowddb_sigmod2011.pdf)
- [16] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data(base) Engineering Bulletin* 18 (1995), 19–29. <https://api.semanticscholar.org/CorpusID:260706023>
- [17] Noah Hollmann, Samuel Müller, and Frank Hutter. 2023. LLMs for Semi-Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. *arXiv:2305.03403 [cs.AI]*
- [18] hypnotu. 2023. DSAIL-Porini. Kaggle. <https://www.kaggle.com/datasets/hypnotu/dsail-porini>
- [19] jessicali9530. 2018. Labelled Faces in the Wild (LFW) Dataset. Kaggle. <https://www.kaggle.com/datasets/jessicali9530/lfw-dataset>
- [20] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proc. ACM Manag. Data* 2, 3 (2024), 186. <https://doi.org/10.1145/3654989>
- [21] jutrrera. 2018. Stanford Car Dataset by classes folder. Kaggle. <https://www.kaggle.com/datasets/jutrrera/stanford-car-dataset-by-classes-folder>
- [22] Kaggle. [n.d.]. Skin Cancer: Malignant or Benign. <https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign>
- [23] Kaggle. [n.d.]. X-ray Lung Diseases Images. <https://www.kaggle.com/datasets/fernando2rad/x-ray-lung-diseases-images-9-classes>
- [24] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *The Twelfth International Conference on Learning Representations*.
- [25] Viktor Leis, Andrey Gubichev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *PVLDB* 9, 3 (2015), 204–215.
- [26] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).
- [27] Chuhan Li, Ziyao Shanguan, Yilun Zhao, Deyuan Li, Yixin Liu, and Arman Co-han. 2024. M3SCIQA: A Multi-Modal Multi-Document Scientific QA Benchmark for Evaluating Foundation Models. In *EMNLP 2024 - 2024 Conference on Empirical Methods in Natural Language Processing, Findings of EMNLP 2024*. 15419–15446. <https://doi.org/10.18653/v1/2024.findings-emnlp.904>
- [28] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. 2017. CDB: Optimizing Queries with Crowd-Based Selections and Joins. In *SIGMOD*. 1463–1478. <https://doi.org/10.1145/3035918.3064036>
- [29] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*. PMLR, 19730–19742. <https://doi.org/10.1145/3035918.3064036>
- [30] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *ACL* (2021).
- [31] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpsest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*. 2.
- [32] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. 2025. MMBench: Is Your Multi-modal Model an All-Around Player?. In *Lecture Notes in Computer Science*, Vol. 15064 LNCS. 216–233. https://doi.org/10.1007/978-3-031-72658-3_13 arXiv:2307.06281
- [33] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. 2012. Counting with the crowd. *VLDB* 6, 2 (2012), 109–120. <https://doi.org/10.14778/2535568.2448944>
- [34] Adam Marcus, Eugene Wu, and David Karger. 2011. Human-Powered Sorts and Joins. *PVLDB* 5, 1 (2011), 13–24. <https://doi.org/10.14778/2047485.2047487> arXiv:arXiv:1109.6881v1
- [35] Adam Marcus, Eugene Wu, Samuel Madden, and Rob Miller. 2011. Crowdsourced Databases: Query Processing with People. In *CIDR*. 211–214. <http://dspace.mit.edu/handle/1721.1/62827>
- [36] Lorna Mugambi, Jason N Kabi, Gabriel Kiarie, and Ciira wa Maina. 2023. DSAIL-Porini: Annotated camera trap image data of wildlife species from a conservancy in Kenya. *Data in Brief* 46 (2023), 108863.
- [37] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. 2012. Deco: Declarative Crowdsourcing. In *Information and Knowledge Management (Maui, Hawaii, USA) (CIKM '12)*. Association for Computing Machinery, 1203–1212. <https://doi.org/10.1145/2396761.2398421>
- [38] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. *CoRR abs/2407.11418* (2024). <https://doi.org/10.48550/ARXIV.2407.11418> arXiv:2407.11418
- [39] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2024. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. *arXiv preprint arXiv:2407.11418* (2024).
- [40] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2020. Language Models are Unsupervised Multitask Learners. *OpenAI Blog* 1, 8 (2020), 1–9. http://static.cs.brown.edu/courses/cs146/assets/papers/language_models_are_unsupervised_multitask_learners.pdf
- [41] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer Learning in Natural Language Processing. In *ACL: Tutorials*. 15–18.
- [42] rushibhalajiputthewad. 2024. Sound Classification of Animal Voice. Kaggle. <https://www.kaggle.com/datasets/rushibhalajiputthewad/sound-classification-of-animal-voice>
- [43] Matthew Russo, Sivaprasad Sudhir, Gerardo Vitagliano, Chunwei Liu, Tim Kraska, Samuel Madden, and Michael J. Cafarella. 2025. Abacus: A Cost-Based Optimizer

- for Semantic Operator Systems. *CoRR* abs/2505.14661 (2025). <https://doi.org/10.48550/ARXIV.2505.14661> arXiv:2505.14661
- [44] 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 (Sept. 2025), 3035–3048. <https://doi.org/10.14778/3746405.3746426>
- [45] Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2021. MultiModalQA: complex question answering over text, tables and images. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- [46] TPC Council. [n.d.]. TPC-DS Homepage. <https://www.tpc.org/tpcds/>
- [47] TPC Council. [n.d.]. TPC-H Homepage. <https://www.tpc.org/tpch/>
- [48] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. 2018. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data* 5, 1 (14 Aug 2018), 180161. <https://doi.org/10.1038/sdata.2018.161>
- [49] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. <https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf>
- [50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 5999–6009. arXiv:1706.03762
- [51] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2024. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv:2212.03533 [cs.CL] <https://arxiv.org/abs/2212.03533>