SemBench: A Benchmark for Semantic Query Processing Engines

Jiale Lao
Cornell University
jiale@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 multimodal data.

Our benchmark introduces diversity across three key dimensions: scenarios, modalities, and operators. Included are scenarios ranging from movie review analysis to car damage detection. 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.

PVLDB Reference Format:
Jiale Lao, Andreas Zimmerer, Olga Ovcharenko, Tianji Cong, Matthew Russo, Gerardo Vitagliano, Michael Cochez, Fatma Özcan, Gautam Gupta, Thibaud Hottelier, H. V. Jagadish, Kris Kissel, Sebastian Schelter, Andreas Kipf, and Immanuel Trummer. SemBench: A Benchmark for Semantic Query Processing Engines. PVLDB, 19(8): 1754 - 1767, 2026.
doi:10.14778/3811243.3811249

Proceedings of the VLDB Endowment, Vol. 19, No. 8 ISSN 2150-8097.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://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 [12, 22, 34, 42, 54] and is already experiencing widespread adoption in industry [9, 14]: 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:

```
SELECT COUNT(*)
FROM Cars
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

doi:10.14778/3811243.3811249

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 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 result quality 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, monetary execution fees (paid for LLM invocations), and memory usage. Finally, we evaluate and compare the scalability of different systems.

We present experimental results for SQPEs from academia and industry. From academia, we evaluate LOTUS [42], Palimpzest [34], and ThalamusDB [22]. From industry, we evaluate Google's BigQuery system [14], 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 website for our benchmark[1] 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. It features queries with semantic operators on multimodal data. The benchmark contains 5 scenarios and 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 reports experimental results for different benchmarks and systems. Finally, Section 6 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 [16], Deco [41], or Qurk [39] use human crowdworkers, hired on platform such as Amazon Mechanical Turk [2], to perform tasks on multimodal data according to user instructions.

[1] https://sembench.org

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 using crowdworkers has inspired a large body of research, aimed at making crowdsourced database systems more efficient (e.g., via specialized operator implementations [10, 37, 38] or query optimization variants [13, 31, 41]). 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 [55], making it easier to scale models up to large parameter counts, and the idea of transfer learning [46], 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 [44]. 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 [12, 22, 34, 42, 54], as well as several industry systems by major Cloud providers, in particular Google's BigQuery [14] and Snowflake's platform [9], now supporting semantic operators as well.

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 [52], TPC-DS [51], and the Join Order Benchmark [28]. 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, 30, 35]. 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

SemBench is designed to use data modalities, operators, and scenarios that are already adopted by many systems in academia and industry. Each query can be processed by at least two semantic query processing engines [9, 12, 14, 22, 34, 42, 54]. While some systems currently support only a subset due to missing operators or data modalities, they are under active development. The selected scenarios are diverse, covering common evaluation settings used in academia, and are inspired by workloads observed in industry.

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

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. We adopt the syntax from [42] for its simplicity and clarity, and we redefine and extend several operators for better generalization.

Operator	Definition	Example
`sem_filter`$(l : X \to Bool)$	$\{ t \in T \mid M(l)(t) = 1 \}$	Identify damaged cars based on car images
`sem_join`$(l : (X, Y) \to Bool)$	$\{(t_i, t_j) \mid M(l)(t_i, t_j) = 1,\ t_i \in T_1,\ t_j \in T_2\}$	Match two images of the same animal
`sem_map`$(l : X \to Y)$	$\{ M(l)(t) \mid t \in T \}$	Extract brand names from product descriptions
`sem_rank`$(l : T[X] \to Seq[X],\ k)$	$\langle t_1, \ldots, t_k \rangle$ ranked by $M(l)$	Rank reviews by positivity
`sem_classify`$(l : X \to Y)$	$\{ (t, M(l)(t)) \mid t \in T \}$	Classify animals by species based on images

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.

3.3 Metrics

We evaluate SQPEs according to processing cost, result quality, and scalability. Processing cost includes execution time, the monetary cost of invoking LLMs, and memory usage. In our experience, costs according to at least one of these three metrics are the primary factors preventing us from using SQPEs on larger datasets. Result quality must also be evaluated, since LLM outputs may contain errors. Jointly measuring processing cost and result quality is similar to evaluation practices in existing work on approximate query processing (AQP) [19, 36, 57]. This similarity arises because semantic query processing can be viewed as a form of AQP, since LLM outputs are inherently uncertain. As we evaluate analytical data processing engines, we focus on scaling with respect to the size of the processed dataset (rather than the number of users or concurrent queries), similar to prior benchmarks for analytical workloads (e.g., TPC-H).

To assess result quality, we compare results generated by the SQPEs 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). We generate ground truth query results by substituting LLM calls with manual labels, provided in the data sets that our scenarios are based upon. For instance, to generate ground truth for a query counting positive movie reviews, we execute an SQL query that replaces the semantic predicate in the SemBench query, using the LLM to filter out negative reviews, with a simple SQL predicate, returing true if the review is assigned to a positive label in the underlying Kaggle data set (containing movie reviews with manual sentiment labels).

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:

```
Movies(id text, title text)
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`.

```
SELECT COUNT(*) AS positive_review_cnt
FROM Reviews R
WHERE R.id = 'taken_3'
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 [47]) and audio recordings (similarly, we use a Kaggle data set [20, 40] containing animal sound recordings with associated species labels). We generate a database with two tables:

```
ImageTable(image img, city text, stationID text)
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.

```
SELECT DISTINCT city
FROM (SELECT city FROM ImageTable I WHERE
 AI.IF(I.image, 'the picture shows an elephant'))
UNION (SELECT city FROM AudioTable A WHERE
 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.

```
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.

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

4.4 Cars

This scenario simulates a multi-modal car damage system to assist manufacturers and insurance companies with investigating damage presence and co-occurrence through diverse data modalities. We construct four interconnected tables based on four publicly available, anonymized datasets from Kaggle [7, 8, 27, 45]:

```
Cars(CarId int, Mileage float, FuelType text,
     TransmissionType text, VIN int, RegistrationDate
     text, Country text, NumberPlate text, PreviousOwners
     int)
ComplaintsText(CarId int, ComplaintId int,
  Complaint text)
CarImage(CarId int, ImageId int, Image image)
DiagnostAudio(CarId int, AudioId int, Audio audio)
```

The cars scenario spans all four modalities available in our benchmark (tables, text, images, and audio). It combines synthetic tabular vehicle records with complementary real-world multi-modal data: first-person problem descriptions (text), diagnostic recordings (audio), and images of intact and damaged cars. A given car may exhibit zero, one, or multiple issues. Multi-modal inputs can, therefore, reflect either damage or a normal state. When possible, the same damage is represented across compatible modalities, e.g., a car with brake damage may include both abnormal brake sounds in the diagnostic audio and visible brake-related damage in images. Cars may also present distinct damages across modalities, e.g., a dent visible in images and an airbag deployment mentioned in text.

To construct the tabular component, we synthesize non-visual vehicle attributes and randomly assign damages under a set of co-occurrence constraints. For each car, we sample a manufacturer year (2000–2025), mileage (0–200k), fuel type (e.g., gasoline, hybrid, electric), transmission (automatic, manual, CVT), and a country from a broad list. We then generate identifiers and registration metadata, including a 17-character alphanumeric VIN and a registration date. License plates are produced using country-specific format templates, including localized character sets where applicable. Finally, we assign the number of previous owners using a weighted probabilistic model that increases the expected owner count for older and higher-mileage vehicles. For the non-tabular modalities, we utilize existing real-world datasets. During evaluation, we treat

the original labels from these source datasets as ground truth that is not exposed to the evaluated system.

Example 4.4. The following query determines all damaged cars.

```
(SELECT CarId FROM ComplaintsText WHERE
 AI.IF('The car is damaged according to the symptoms.',
      Complaint))
 UNION DISTINCT
(SELECT CarId FROM DiagnostAudio WHERE
 AI.IF('The car is damaged according to the audio.',
      Audio))
 UNION DISTINCT
(SELECT CarId FROM CarImage WHERE
 AI.IF('The car is damaged according to the image.',
      Image));
```

4.5 MMQA

The MMQA scenario simulates a multi-modal question answering system and is built on top of the MultiModalQA dataset [50]. 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 metadata 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.

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

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

```
SELECT t.Airlines, i.uri
FROM mmqa.tampa_international_airport t, mmqa.images i
WHERE AI.IF("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. Since LLMs perform differently across domains, we select five representative scenarios, based on common academic datasets and inspired by workloads observed in industry, to cover a broad range of data domains. As SQPEs expand their support for semantic operators and data modalities, and as LLMs face new challenges in emerging data domains, we plan to release additional scenarios in future SemBench instantiations.

The Cars 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 EXPERIMENTAL RESULTS

In Section 5.1, we describe our general experimental setup. In Section 5.2, we discuss how each of the evaluated systems was tuned. In Section 5.3, we report and analyze our experimental results. In Section 5.4, we present scalability experiments. In Section 5.5, we compare and discuss different systems.

5.1 Experimental Setup

Hardware Settings. The experiments in Section 5.3 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. The scalability experiments are conducted on an Ubuntu server with two Intel Xeon Gold 5218 CPUs and 384 GB of RAM. We use a different server because current academic systems require significantly more memory than the EC2 instance can provide when scaling up (more than 200 GB for some systems). More details are discussed in Section 5.4.
Model Settings. For all experiments presented in the paper, we use `gemini-2.5-flash` 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. We disable the reasoning capability of the LLM in the main experiments due to long processing times and high monetary cost. We further set the model temperature to 0 for applicable systems for reproducibility.

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	–
Cars	10	✓	✓	✓	✓	12	–	–	–	1	157,376	30,131	1,387
Total	**55**	✓	✓	✓	✓	**50**	**15**	**7**	**3**	**4**	**1,582,560**	**84,295**	**2037**

Table 3: Scales for Different Scenarios.

Scenario	Available Range	Scale Used	
		Base	Scalability
Movie	1–1,375,738	2,000	1,000–16,000
Wildlife	1–8,718	200	100–1,600
E-Commerce	50–44,446	500	250–4,000
Cars	1–157,376	19,672	9,836–157,376
MMQA	25–1,000	200	100–800

Degree of Parallelism. We set the LLM invocation parallelism to 20. BigQuery does not expose this setting.

Scenario Settings. To control the database size, we define a *Scale* for each scenario as the number of rows in the primary entity table, with other table sizes defined as functions of it. This is similar to the scaling factors in TPC-H [52] and other benchmarks [51] for studying system behavior at different scales. Table 3 lists the available Scale ranges and the Scales used in our experiments. Base scales are chosen based on scalability tests to balance system stress with feasible monetary cost for future benchmark users. We also evaluate larger Scales with scalability experiments in Section 5.4, where a single query can exceed one hundred dollars, 200 GB main memory, and current systems do not complete within a reasonable time. In total, the cost of LLM calls when running the scalability experiments is $9,935.80, taking approximately 18 days to execute. As shown in Table 2, the full corpus contains over 1.5 million text rows, 84 thousand image rows, and two thousand audio rows, making the largest Scales impractical today. As systems improve, higher Scales can be used in future evaluations.

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 [11] 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.

5.2 System Settings

All systems build the final prompt by combining prompt templates with a natural language operator configuration provided as part of the semantic query. The prompt templates define operator semantics and vary across systems to reflect different design choices, such as reducing token usage or improving output quality. SemBench does not change the default templates used by different systems. Hence, even when using the same operator configurations in the queries sent to different systems, the prompts sent to the LLM may differ due to different prompt templates. We use default settings or author-recommended configurations for all tuning parameters.

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.

Palimpzest. We use Palimpzest version `0.8.2` with the Abacus optimizer [48] turned off. We apply the MaxQuality objective, which configures Palimpzest to use its default "best" operator implementation(s) based on prior belief. 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 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 Cars 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 evalutaion, 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.

5.3 Detailed Results per Scenario

Tables 4 reports the results. 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. It also reports per-query averages for each system along with the absolute standard deviation of the average (i.e., denoting by σ_i the standard deviation of query i out of n queries, we report $\sqrt{\sum_i \sigma_i^2/n}$). The standard deviation is small for cost and quality compared to run time. Variance in cost and quality mainly comes from randomness in LLM outputs, which systems often reduce by using a low temperature when possible. In contrast, run time variability is largely due to fluctuations in LLM call latency.

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 scale of 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.

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 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. 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, but the systems handle it differently. In LOTUS, the LIMIT clause is applied only as a post-processing step after semantic operators are evaluated on all data. In contrast, SPQEs can terminate semantic operator evaluation early once enough rows are produced to satisfy the LIMIT clause, which leads to significantly better performance for these 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 semantic 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 4 (b) 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` 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 supporing `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. Note that the cost of LOTUS demonstrates the highest variance across all systems in this scenario. The reason lies in the implementation of the semantic join in LOTUS. LOTUS analyzes a data sample to determine thresholds on the distance between embedding vectors, within which candidate join pairs are evaluated via the LLM. Comparing different runs, that sample, and therefore the thresholds, differ, meaning that the number of join candidates processed by the LLM varies significantly.

Table 4 (c) 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. Note that the standard deviation for cost is close to zero in this scenario. This is due to the fact that queries use exclusively semantic filter operators. For this operator and all compared systems, the cost is very stable across different runs. Unlike other systems, BigQuery shows a relatively high quality variance across different runs. This behavior arises because the model temperature is set to zero for the other systems, whereas model-related parameters in BigQuery are controlled internally and are not exposed to users. As a result, BigQuery exhibits higher output variability. BigQuery achieves the highest average quality among all systems supporting all queries (with a gap of over three standard deviations).

Cost differences are generally smaller in the Wildlife scenario, with relative differences of up to a factor of ten, compared to more than two orders of magnitude in the Movie scenario. The Wildlife uses only the semantic filter operator and no semantic joins, which typically cause larger cost differences across 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 4 (d) presents the experimental results for the MMQA scenario. Only Palimpzest fully supports MMQA queries. LOTUS can technically support all semantic operators, but it may produce internal processing errors across runs. These errors occur when LLM responses do not follow the output schema specified in the prompt

Table 4: SemBench Results. Operators: F - semantic filter, J - semantic join, R - semantic rank, C - semantic classify, L - LIMIT clause. Colors: Better than Average, Average, Worse than Average, and ✗ for not supported.

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency
(a) Movie Scenario												
Q1: F L	$0.09	1.00	33.1 s	$1·10⁻³	1.00	3.8 s	$4·10⁻⁴	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	$2·10⁻³	0.92	1.9 s	$3·10⁻³	1.00	9.5 s
Q3: F	$5·10⁻³	0.64	2.1 s	$0.01	0.64	4.6 s	$2·10⁻³	0.74	3.1 s	$3·10⁻³	0.64	11.0 s
Q4: F	$5·10⁻³	0.64	2.8 s	$0.02	0.74	4.4 s	$2·10⁻³	0.74	3.8 s	$3·10⁻³	0.64	11.4 s
Q5: J L	$2.38	0.59	536.5 s	$0.01	0.39	1.9 s	$1·10⁻³	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	$9·10⁻⁴	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	$4·10⁻³	0.93	2.3 s	$0.02	0.86	4.3 s	$5·10⁻³	0.83	6.8 s	$3·10⁻³	0.76	10.9 s
Q9: R	$0.02	0.75	4.9 s	$0.05	0.78	5.7 s	✗	✗	✗	$0.02	0.78	13.3 s
Q10: R	$0.13	0.40	30.9 s	$0.38	0.42	39.2 s	✗	✗	✗	$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	±$0.02	±0.01	±5.0s	±$3·10⁻⁵	±0.01	±12.2s	±$6·10⁻⁵	±0.00	±1.3s	±$0.01	±0.01	±6.5s
(b) E-Commerce Scenario												
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	✗	✗	✗	$0.12	0.97	21.2 s
Q4: M	$0.14	0.45	156.7 s	$0.24	0.53	335.0 s	✗	✗	✗	$0.37	0.69	31.0 s
Q5: C	$0.04	0.99	7.4 s	$0.07	0.98	6.6 s	✗	✗	✗	$0.17	0.98	25.7 s
Q6: C	$0.12	0.89	114.8 s	$0.43	0.89	143.7 s	✗	✗	✗	$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	$4·10⁻³	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	✗	✗	✗	✗	✗	✗
Q11: F J	$0.27	0.78	158.7 s	$0.61	0.73	132.4 s	✗	✗	✗	✗	✗	✗
Q12: F M	$0.10	0.60	36.4 s	$0.14	0.00	31.9 s	✗	✗	✗	$0.10	0.97	31.1 s
Q13: F	$0.24	0.74	274.8 s	$0.44	0.74	238.3 s	✗	✗	✗	$0.38	0.70	22.4 s
Q14: F J R	$0.23	0.87	178.2 s	✗	✗	✗	✗	✗	✗	$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	±$0.03	±0.01	±35.4s	±$4·10⁻⁴	±0.01	±101.0s	±$1·10⁻⁵	±0.00	±71.0s	±$0.01	±0.03	±3.4s
(c) Wildlife Scenario												
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	✗	✗	✗	$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	✗	✗	✗	$0.01	0.00	2.7 s	$1·10⁻³	0.00	1.3 s	$0.01	1.00	9.4 s
Q5: F	✗	✗	✗	$0.13	0.75	13.5 s	$0.01	0.75	2.3 s	$0.12	0.75	19.2 s
Q6: F	✗	✗	✗	$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	✗	✗	✗	$0.13	0.75	34.9 s	$0.12	0.75	23.8 s	$0.23	0.75	35.5 s
Q9: F	✗	✗	✗	$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	±$7·10⁻¹⁸	±0.00	±6.2s	±$5·10⁻¹⁸	±0.00	±5.2s	±$1·10⁻¹⁸	±0.00	±0.51s	±$2·10⁻¹⁸	±0.02	±7.6s
(d) MMQA Scenario												
Q1: M	$0.02	1.00	7.4 s	$3·10⁻³	1.00	4.5 s	✗	✗	✗	$0.01	1.00	14.1 s
Q2a: J	$0.89	0.83	169.6 s	$1.04	1.00	135.9 s	✗	✗	✗	$0.08	0.00	53.8 s
Q2b: J	$0.89	0.83	166.8 s	$1.04	1.00	133.8 s	✗	✗	✗	$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	✗	✗	✗	$5·10⁻³	0.54	1.2 s	✗	✗	✗	$2·10⁻³	0.60	9.7 s
Q5: M	$1·10⁻³	1.00	0.48 s	$1·10⁻³	1.00	0.50 s	✗	✗	✗	✗	✗	✗
Q6a: F	$0.01	1.00	4.4 s	$0.02	1.00	4.0 s	$2·10⁻³	0.33	5.6 s	$4·10⁻³	0.03	18.9 s
Q6b: F	$0.01	1.00	3.8 s	$0.02	1.00	4.1 s	$2·10⁻³	1.00	5.5 s	$4·10⁻³	0.04	17.3 s
Q6c: F	$0.01	1.00	4.6 s	$0.02	1.00	4.7 s	$2·10⁻³	0.53	13.4 s	$4·10⁻³	0.13	17.4 s
Q7: J	$13.61	0.32	2311.4 s	$15.65	0.31	2101.6 s	✗	✗	✗	$1.18	0.00	91.7 s
Avg	**$1.41**	**0.88**	**243.4 s**	**$1.62**	**0.88**	**218.4 s**	**$5·10⁻³**	**0.72**	**8.5 s**	**$0.14**	**0.32**	**30.3 s**
Std Dev	±$8·10⁻⁷	±0.00	±2.3s	±$2·10⁻⁵	±0.00	±5.3s	±$0.00	±0.00	±2.7s	±$5·10⁻⁵	±0.00	±3.2s

Table 4 continued from previous page.

(e) Cars Scenario

Q1: F	$1.74	0.90	550.0 s	$2.44	0.69	465.6 s	$1.37	0.81	829.7 s	$1.44	0.71	61.7 s
Q2: F	✗	✗	✗	$4·10^{-3}$	0.00	4.1 s	$0.01	0.09	14.1 s	$0.01	0.08	14.1 s
Q3: F L	$0.60	0.90	456.2 s	$0.01	0.92	6.1 s	✗	✗	✗	$1.66	1.00	36.4 s
Q4: F	$1.71	0.99	822.0 s	$2.41	0.99	443.6 s	$1.34	1.00	768.8 s	$1.41	0.99	68.7 s
Q5: F	✗	✗	✗	$0.01	1.00	6.3 s	$1.61	1.00	483.7 s	$1.47	1.00	58.9 s
Q6: F J	✗	✗	✗	$2.51	0.96	427.3 s	$1.96	0.97	775.4 s	$2.00	0.96	44.3 s
Q7: F	✗	✗	✗	$4.47	0.56	882.7 s	$3.06	0.58	2146.2 s	$3.17	0.45	86.0 s
Q8: F L	$1.78	0.45	1349.8 s	$2.02	0.29	268.7 s	$0.21	0.20	68.0 s	$1.69	0.24	38.2 s
Q9: F	✗	✗	✗	$2.05	0.00	308.1 s	$1.61	1.00	335.8 s	$0.03	0.00	13.6 s
Q10: C	$3.09	0.41	618.1 s	$4.69	0.51	594.9 s	✗	✗	✗	$2.70	0.57	62.0 s
Avg	**$1.78**	**0.73**	**759.2 s**	**$2.06**	**0.59**	**340.7 s**	**$1.24**	**0.71**	**603.2 s**	**$1.56**	**0.60**	**48.4 s**
Std Dev	$\pm\$4·10^{-17}$	±0.00	±92.5s	$\pm\$2·10^{-3}$	±0.01	±55.5s	$\pm\$5·10^{-17}$	±0.00	±14.3s	±$0.07	±0.00	±1.6s

and the subsequent code does not handle unexpected outputs. This issue explains the "n/a" entry for Q4, despite LOTUS supporting 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 4 (e) presents the results for the Cars scenario. First, current systems demonstrate poor performance in car diagnostics audio processing. 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. Palimpzest does not support relational joins and requires us to hardcode the join order and execute the joins with Pandas (Q6 and Q7). Third, Q5, Q6, and Q7, which process image data, exhibit similar quality across all evaluated systems, while the execution costs differ significantly, with Palimpzest being the most expensive. We attribute this to the higher computational cost of image semantic operators relative to text-only queries. Fourth, in the case of semantic operators for textual damage detection, queries targeting cars with more specialized damages (e.g., Q4 - engine problems) yield higher quality results compared to those involving more generic conditions (e.g., Q1 - crash/accident/collision). We hypothesize that LLMs 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 cars scenario. Unlike in traditional databases, the quality of results produced by semantic operators is highly dependent on the domain and modality of the data being processed, and systems need to be aware of this fact and actively manage it.

5.4 Scalability Experiments

We evaluate system scalability by creating multiple databases for each scenario that vary the data size around a scenario-specific Base Scale: {0.5× Base, Base, 2× Base, 4× Base, 8× Base}, and the scales used are summarized in Table 3. For each system and scenario, evaluation starts from the smallest scale with a per-query timeout of one hour. If any query exceeds this limit at a given scale, all remaining queries at that scale are still executed, but the system is not evaluated at larger scales. Systems are not forcibly terminated upon timeout, and executions that complete beyond one hour are still recorded. At each scale, results are averaged over the set of common queries supported by all systems. As shown in Figure 1, we report average execution time, monetary cost, and result quality as functions of the scale factor (the bottom row reports memory consumption in a separate scalability experiment and is discussed later). Error bars indicate standard deviations across five runs, and the one-hour timeout is marked using cross markers. We provide the full per-query results for all scales at https://sembench.org.

For the Movie scenario, nearly all systems complete each query within 1 hour for all scales. Unlike other scenarios that require more resources to process images and audio data, this scenario uses only text data. LOTUS reaches the timeout of one hour at scale 16K due to Q6, which uses a semantic join with a LIMIT clause. Unlike other systems, LOTUS does not terminate early after collecting the number of rows required by the LIMIT clause, but continues processing data. This increases cost and time overheads for Q6.

For the E-Commerce and MMQA scenarios, all systems except BigQuery fail to scale to large scales. BigQuery scales to these scales because it has an internal mechanism to adjust resources, such as the number of parallel LLM invocations, based on input size. This mechanism is not available in the other systems. The two scenarios include expensive multi-modal semantic join operators, and the number of LLM calls for these operators grows quadratically with table size. For example, for Q7 in the MMQA scenario, a scale of 400 results in 160,000 LLM calls, which leads to monetary costs of about $50 for LOTUS and $60 for Palimpzest. Similarly, Q8 in the E-Commerce scenario uses a semantic join over text and images. Only BigQuery can scale to the largest scale of 4000 for this query,

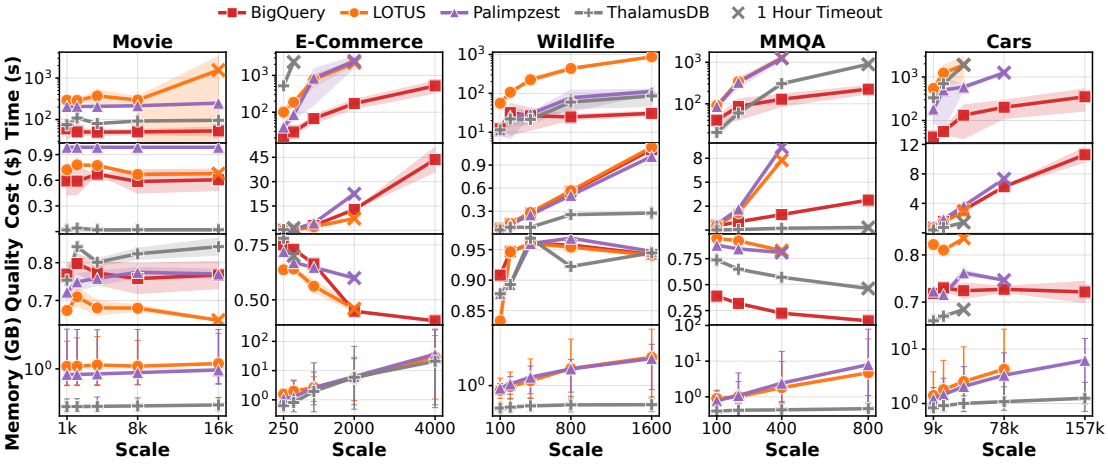

Figure 1: Scalability across scenarios. We report average execution time, monetary cost, result quality, and memory usage over a fixed set of queries that are supported by all systems. Error bars for execution time, cost, and result quality show standard deviations over five runs. For memory usage, error bars show the minimum and maximum values. Cross markers indicate a per-query timeout of 1 hour. After a timeout, the system is treated as failed and is not evaluated at larger scales.

with a cost of about $140. These results point to an important optimization problem: how to scale multi-modal semantic joins over large tables. In addition, as data sizes increase, result quality consistently decreases across all systems. Therefore, scaling processing while maintaining high result quality remains an open problem.

For the Wildlife scenario, all systems scale to the largest scale because this scenario does not include semantic join operators, but only semantic filters over images and audio data. First, the monetary cost of BigQuery increases linearly with the scale, while the execution time does not. As discussed earlier, this behavior is due to BigQuery adjusting resources for each query based on input size, a feature not available in other systems. Second, ThalamusDB achieves much lower monetary cost because it avoids unnecessary LLM evaluations during query execution. Taking Q5 from the Wildlife scenario as an example, Q5 asks for cities that have either images or audio recordings of elephants. Once ThalamusDB finds evidence that a city satisfies this condition, it stops evaluating additional rows for that city, since further evaluations cannot change the final result. In contrast, other systems continue to evaluate all relevant rows. Third, we find that result quality increases slightly for some queries as we scale up. Taking Q7 as an example, this query asks for cities where zebras and impalas co-occur. As the scale factor increases, more data is sampled, and the number of rows supporting each city also increases. With more supporting rows, systems have more chances to successfully evaluate at least one correct row for each city, which increases the likelihood of returning the correct result.

Cars is similar to Wildlife in terms of operators and modalities, as it does not include semantic joins but uses semantic filter and classification operators over text, image, and audio data. For Cars, we are able to increase the scale to cover all data. In this scenario, BigQuery, Palimpzest, ThalamusDB, and LOTUS rank from best to worst in execution time. Execution time and monetary cost increase nearly linearly with scale, while BigQuery achieves lower execution time due to its resource adjustment mechanism. The result quality remains stable across different scales.

We conduct one run of a separate experiment without a timeout to measure memory usage at scale, and Figure 1 (fourth row) reports the average, minimum, and maximum peak memory usage per query for each scenario. BigQuery memory usage is not reported because this information is not available to users. The average memory usage increases linearly with the scale factor. In contrast, the maximum memory usage varies widely across scenarios. It is below or around 10 GB in the Movie, Wildlife, and Cars scenarios, but reaches hundreds of gigabytes in the E-Commerce and MMQA scenarios. For example, Q8 in the E-Commerce scenario requires about 250 GB, 150 GB, and 100 GB of memory for ThalamusDB, LOTUS, and Palimpzest, respectively. These systems create image pairs to evaluate semantic joins and send them to remote LLMs. The memory used to store these image pairs is not released, so memory usage accumulates as more pairs are processed. This leads to significant resource waste and can cause server crashes. In addition, the memory usage of LOTUS on the Cars dataset at a scale of 157k is not reported. LOTUS does not handle LLM invocation failures robustly and may stall for over 48 hours at large scale due to network issues, unstable API, or rate limiting. This result indicates that systems should explicitly handle LLM invocation failures.

5.5 System Comparison

Table 5 shows aggregated results (for the base scale) for query groups that contain specific semantic operators. All comparative claims are based on a gap between average values of at least three standard deviations. For Filter, Map, Score, and Classify, where current implementations mainly differ in prompt design, LOTUS and Palimpzest attain the best quality by using detailed prompt templates, but this choice leads to higher costs due to increased input tokens. ThalamusDB uses concise prompt templates that result in the lowest costs ($0.19 for Filter) but lower quality (0.698 for Filter).

Table 5: Aggregated Results by Operator Type. Colors indicate relative performance (Better than Average, Average, Worse than Average). For each operator, only queries supported by all available systems are included.

	LOTUS			Palimpzest			ThalamusDB			BigQuery		
	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost	Quality	Latency	Cost
Filter	0.896	194.4s	$0.34	0.777	72.2s	$0.40	0.698	101.1s	$0.19	0.574	26.7s	$0.36
Join	0.614	541.1s	$2.74	0.703	454.3s	$3.29	0.487	387.3s	$0.14	0.601	69.4s	$1.02
Map	0.805	60.4s	$0.07	0.762	89.3s	$0.09	✗	✗	✗	0.820	14.3s	$0.06
Score	0.576	17.9s	$0.07	0.600	22.4s	$0.21	✗	✗	✗	0.607	22.7s	$0.07
Classify	0.763	246.8s	$1.08	0.733	186.6s	$1.30	✗	✗	✗	0.785	25.5s	$0.73
Avg (All Operators)	**0.731**	**212.1s**	**$0.86**	**0.715**	**165.0s**	**$1.06**	**0.592**	**244.2s**	**$0.17**	**0.677**	**31.7s**	**$0.45**
Avg (Filter + Join)	**0.755**	**367.7s**	**$1.54**	**0.740**	**263.2s**	**$1.84**	**0.592**	**244.2s**	**$0.17**	**0.587**	**48.1s**	**$0.69**

For semantic joins, cost reduction strategies introduce additional trade-offs. Palimpzest uses a naive semantic join implementation that generates item pairs for LLM evaluation in a nested loop manner. LOTUS applies embedding-based pre-filtering to reduce the number of LLM calls. This reduces the monetary cost to $2.74, compared to $3.29 for Palimpzest. However, pre-filtering can miss valid matches and leads to lower quality, with a score of 0.614 compared to 0.703. ThalamusDB uses batching to process multiple row pairs within a single prompt and asks LLMs to decide which row pairs satisfy the join condition. This reduces the cost to $0.14, but it yields lower quality of 0.487, compared to 0.703 for Palimpzest. Overall, the systems achieve different quality–cost trade-offs through different design choices. ThalamusDB achieves the lowest average cost, at only $0.17 on average, but with lower average quality. BigQuery achieves the best latency, with an average latency of 48.1s, and provides a good balance between quality and cost. LOTUS and Palimpzest achieve higher quality, but at moderately higher cost.

We report several lessons learned from our experiments. First, a system should exploit LIMIT clauses efficiently, terminating early once the required number of rows is generated. LOTUS does not support this and must process the entire data set before applying post-processing, increasing cost. Second, a system should decide during query execution whether additional rows need to be evaluated. ThalamusDB supports early termination when further processing cannot change the final result. Third, a system should adjust the degree of parallelism at runtime based on the sizes of the input relations, as done in BigQuery. Fourth, LLM invocation failures are common due to network issues, unstable APIs, or rate limits: systems should handle these failures explicitly by using retries with exponential backoff. Finally, SQPEs should limit memory consumption when processing multimodal data. For instance, LOTUS, Palimpzest, and ThalamusDB all encounter memory pressure when processing images with semantic joins, limiting their scalability.

6 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. 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 present, no evaluated system supports all benchmark queries due to limitations in 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. We thank Alon Halevy, Sam McVeety, and Tomas Talius for their valuable feedback and support. This material is based upon work supported by the National Science Foundation under Award No. 2239326. Michael Cochez's work on this publication is in part based discussions in COST Action CA24121 - Knowledge Graphs in the Era of Large Language Models (KGELL) supported by COST (European Cooperation in Science and Technology, https://www.cost.eu).

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] andrezaza. 2023. Clapper: Massive Rotten Tomatoes Movies & Reviews. Kaggle. https://www.kaggle.com/datasets/andrezaza/clapper-massive-rotten-tomatoes-movies-and-reviews

[4] Seongsu Bae, Daeun Kyung, Jaehee 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/niyarrbarman/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] Samuel Cavazos. 2025. NHTSA Customer Complaints. https://www.kaggle.com/datasets/alshival/nhtsa-complaints

[8] Hendrichs Cullen. 2023. VehiDE Dataset: Automatic vehicle damage detection. https://doi.org/10.34740/KAGGLE/DS/3685629

[9] 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

[10] 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

[11] 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

[12] 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

[13] Ju Fan, Meihui Zhang, Stanley Kok, Meiyu 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

[14] 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

[15] 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

[16] MJ Franklin and D Kossmann. 2011. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*. 61–72. http://www.cs.berkeley.edu/$\sim$rxin/papers/crowddb_sigmod2011.pdf

[17] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data(base) Engineering Bulletin* 18 (1995), 19–29. https://api.semanticscholar.org/CorpusID:260706023

[18] 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]

[19] Aaron Hurst, Daniel E. Lucani, and Qi Zhang. 2024. PairwiseHist: Fast, Accurate and Space-Efficient Approximate Query Processing with Data Compression. *Proc. VLDB Endow.* 17, 6 (Feb. 2024), 1432–1445. https://doi.org/10.14778/3648160.3648181

[20] hypnotu. 2023. DSAIL-Porini. Kaggle. https://www.kaggle.com/datasets/hypnotu/dsail-porini

[21] jessicali9530. 2018. Labelled Faces in the Wild (LFW) Dataset. Kaggle. https://www.kaggle.com/datasets/jessicali9530/lfw-dataset

[22] 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

[23] jutrera. 2018. Stanford Car Dataset by classes folder. Kaggle. https://www.kaggle.com/datasets/jutrera/stanford-car-dataset-by-classes-folder

[24] Kaggle. [n.d.]. Skin Cancer: Malignant or Benign. https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign

[25] Kaggle. [n.d.]. X-ray Lung Diseases Images. https://www.kaggle.com/datasets/fernando2rad/x-ray-lung-diseases-images-9-classes

[26] 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*.

[27] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3D Object Representations for Fine-Grained Categorization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*.

[28] Viktor Leis, Andrey Gubichev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *PVLDB* 9, 3 (2015), 204–215.

[29] 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).

[30] Chuhan Li, Ziyao Shangguan, Yilun Zhao, Deyuan Li, Yixin Liu, and Arman Cohan. 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

[31] 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

[32] 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.

[33] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *ACL* (2021).

[34] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*. 2.

[35] 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

[36] Qingzhi Ma, Ali M Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned approximate query processing: Make it light, accurate and fast. In *Conference on Innovative Data Systems,(CIDR21)*.

[37] 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

[38] 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

[39] 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

[40] 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.

[41] 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

[42] 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

[43] 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).

[44] 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

[45] Malak Ragaie. 2025. Car Diagnostics Dataset. https://www.kaggle.com/datasets/malakragaie/car-diagnostics-dataset

[46] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer Learning in Natural Language Processing. In *ACL: Tutorials*.

15–18.

[47] rushibalajiputthewad. 2024. Sound Classification of Animal Voice. Kaggle. https://www.kaggle.com/datasets/rushibalajiputthewad/sound-classification-of-animal-voice

[48] 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

[49] 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

[50] 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.

[51] TPC Council. [n.d.]. TPC-DS Homepage. https://www.tpc.org/tpcds/

[52] TPC Council. [n.d.]. TPC-H Homepage. https://www.tpc.org/tpch/

[53] 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

[54] 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

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 5999–6009. arXiv:1706.03762

[56] 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

[57] Yuxuan Zhu, Tengjun Jin, Stefanos Baziotis, Chengsong Zhang, Charith Mendis, and Daniel Kang. 2025. PilotDB: Database-Agnostic Online Approximate Query Processing with A Priori Error Guarantees. *Proc. ACM Manag. Data* 3, 3, Article 198 (June 2025), 28 pages. https://doi.org/10.1145/3725335