Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS

Liana Patel
Stanford University
lianapat@stanford.edu

Siddharth Jha
UC Berkeley
sidjha@berkeley.edu

Melissa Pan
UC Berkeley
melissapan@berkeley.edu

Harshit Gupta
Stanford University
gharshit@stanford.edu

Parth Asawa
UC Berkeley
pgasawa@berkeley.edu

Carlos Guestrin
Stanford University
guestrin@stanford.edu

Matei Zaharia
UC Berkeley
matei@berkeley.edu

ABSTRACT

The semantic capabilities of large language models (LLMs) have the potential to enable rich analytics and reasoning over vast knowledge corpora. Unfortunately, existing systems either empirically optimize expensive LLM-powered operations with *no performance guarantees*, or limit their support to simple batched-inference primitives. We introduce *semantic operators*, the first formalism with statistical accuracy guarantees for general-purpose AI-based operations with natural language parameters (e.g., filtering, sorting, joining or aggregating records using natural language criteria). Each operator can be implemented by multiple *AI algorithms*, which compose individual model invocations to orchestrate the model over the data. Our programming model specifies the expected behavior of each operator with a high-quality *reference algorithm*, and we develop an optimization framework that reduces cost, while providing accuracy guarantees for individual operators. Using this approach, we propose several novel optimizations to accelerate semantic filtering, joining, group-by and top-k operations by up to $1,000\times$. We implement semantic operators in the LOTUS system and demonstrate LOTUS' effectiveness on real, bulk-semantic processing applications, including fact-checking, biomedical multi-label classification, search, and topic analysis. We show that the semantic operator model is expressive, capturing state-of-the-art AI pipelines in a few operator calls, and making it easy to express new pipelines that match or exceed quality of recent LLM-based analytic systems by up to 170%, while offering accuracy guarantees. Overall, LOTUS programs match or exceed the accuracy of state-of-the-art AI pipelines for each task while running up to $3.6\times$ faster than the highest-quality baselines. LOTUS is publicly available at https://github.com/lotus-data/lotus.

PVLDB Reference Format:
Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. PVLDB, 18(11): 4171 - 4184, 2025.
doi:10.14778/3749646.3749685

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749685

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

1 INTRODUCTION

The powerful semantic capabilities of modern language models (LLMs) create exciting opportunities for building AI-based analytics systems that reason over vast knowledge corpora. Many applications require complex reasoning over large amounts of data, including both unstructured and structured data. For example a researcher reviewing recent ArXiv [2] preprints may want to quickly obtain a summary of relevant papers from the past week, or find the papers that report the best performance for a particular task and dataset. Similarly, a medical professional may automatically extract biomedical characteristics and candidate diagnoses from many patient reports [29]. Likewise, organizations wish to automatically digest lengthy transcripts from internal meetings and chat histories to validate hypotheses about their business [4].

Each of these tasks require a form of *bulk semantic processing*, where the analytics system must process large amounts of data and orchestrate models in complex patterns across a whole dataset. Supporting the full generality of these applications with efficient, easy-to-use analytics systems would have a transformative impact, similar to what RDBMSes had for tabular data. This prospect, however, raises two challenging questions: first, *how should developers express semantic queries*, and secondly, *how should we design the underlying data system to achieve high efficiency and accuracy*.

Unfortunately, existing systems are insufficient for bulk semantic processing, either limiting their expressiveness to simple batched inference primitives or providing no accuracy guarantees. First, several systems only support simple batched inference primitives [1, 8, 9, 14, 46, 47, 49, 50, 53, 67]. These systems do not support richer LLM-based operations, such as ranking, grouping or joining records, which require more complex *AI algorithms* that compose multiple model invocations to orchestrate the model over the data. Alternatively, more recent LLM-based analytics systems, such as DocETL [61] and UQE [25], study more complex AI operations, but empirically optimize these operations with *no accuracy guarantees*. These systems lack a formalism to define correct behavior, which hinders their robustness and usability, as we show in Section 5. These obstacles highlight a core challenge of integrating semantic-based processing within reliable query systems due to the inherent ambiguity of natural language instructions and LLM outputs.

We propose *semantic operators*, which extend the relational model with AI-based operations. Semantic operators provide the first formalism *with statistical accuracy guarantees* for general-purpose AI-based operations with natural language parameters, including semantic filters, joins, top-k rankings, aggregations, and projections. Each operator takes a concise natural language signature, given by the programmer, and its behavior is fully specified by a tractable, high-quality reference algorithm. Our optimization approach then exploits the rich design space of semantic operator execution plans to reduce cost, while providing *statistical accuracy guarantees* for individual operators with respect to the reference algorithm (i.e., ensuring that the output of the optimized operator will be similar to that of the reference algorithm). We propose several novel optimizations for the semantic filter, join, top-k and group-by operators. Our methods include approximation algorithms, building on statistical techniques used in prior works [37, 39] with novel and efficient proxy scores using small LLMs or semantic embeddings. We implement semantic operators in LOTUS (**L**LMs **O**ver **T**ables of **U**nstructured and **S**tructured data), an open source system that exposes these operators in a simple DataFrame-based API.

We systematically evaluate LOTUS on four real bulk-semantic processing applications: fact-checking, biomedical multi-label classification, search, and topic analysis. These applications include ones expressible and studied by recent LLM-based analytics systems (e.g., DocETL [61] and UQE [25]), where we compare against those systems, as well as others, where we compare against hand-designed pipelines from the AI literature. We demonstrate that the semantic operator model is effective, capturing state-of-the-art AI pipelines in a few operator calls that reliably match or exceed the quality of recent LLM-based analytics systems by up to 100%, while offering accuracy guarantees and running up to 3.6× faster than the best quality baselines. Furthermore, our reference algorithms provide high-quality results, and our optimizations speed up query execution by up to 1,000× with respect to them.

Specifically, our LOTUS program re-implements FacTool's recent state-of-the-art fact-checking pipeline [23], and achieves 12.5% higher accuracy with 28× lower execution time on the FEVER dataset [64], outperforming alternative LLM-powered analytics systems. In the extreme multi-label classification task on the BioDEX dataset [29], LOTUS reproduces state-of-the-art result quality [28] and the LLM-powered analytics baselines, while providing accuracy guarantees and 1,000× speedups compared to our reference algorithm. In the search application, LOTUS allows a simple composition of operators to achieve $1.03 - 2\times$ higher nDCG@10 than the next best performing baselines, while also providing query efficiency, with $1.67 - 10\times$ lower execution time than alternative high-quality algorithms. In the topic analysis task, LOTUS efficiently processes hundreds of ArXiv papers to discover a taxonomy of key topics in dozens of seconds, while once again providing configurable speedups with statistical accuracy guarantees.

Overall, our main contributions are:

- We propose semantic operators, the first formalism with statistical accuracy guarantees for general-purpose AI-based operations with natural language parameters. Specifically, our optimization framework specifies each operator's expected behavior and provides accuracy guarantees with respect to a high-quality reference algorithms.

```
def paper_digest(research_interest: str, baseline: str):
  return papers_df\
    .sem_search("abstracts", research_interest, K=100)\
    .sem_filter(f"the paper {{abstracts}} claim to
     outperform {baseline}")\
    .sem_agg(f"Write a digest summarizing {{abstracts}}
     and their relevance to {research_interest}")
```

Figure 1: Example LOTUS program using semantic operators to return a summary of relevant papers from the dataset papers_df. The user function, takes two strings, describing a research interest and a research baseline. The program searches over papers in the dataset, then filters based on whether each paper claims to outperform the baseline, and finally constructs a summary.

- We define optimizations with statistical accuracy guarantees for several useful operators, including semantic filters, joins, top-k and group-by, yielding up to 1,000× speedups.
- Across diverse real-world ML tasks, we evaluate the expressiveness of semantic operators and the benefit of our optimizations, including comparisons to prior work.

2 THE SEMANTIC OPERATOR MODEL

Semantic operators provide a declarative interface for semantic-based manipulation and access to data. These new *AI-based* operators extend the data independence model of relational systems [24] and introduce *model-data independence*: the separation of application logic from the underlying AI-based algorithm that specifies how data records are processed via individual model invocations. Unlike relational operators, semantic operators are parametrized by natural language expressions and rely on AI-based computation, making their behavior inherently ambiguous and imprecise. We address this key challenge by presenting a formalism for defining semantic operators, their behavior, and correct optimizations, in this section. In Section 3, we build on this formalism to provide optimized execution plans with accuracy guarantees, similar to relational query optimizers.

2.1 Example Semantic Operator Program

To begin to understand the capabilities of semantic operators, we examine a simple program written with the LOTUS API, shown in Figure 1. The function `paper_digest` takes two string parameters, a user's research interest and a research baseline, and returns a digest of relevant research papers that claim to outperform the baseline. To do this, the function uses 3 semantic operators to process the dataset of research papers, `papers_df`, which contains an *abstract* attribute. The program first performs top-k semantic search over the *abstract* attribute to find relevant papers, then semantically filters to find papers with abstracts claiming to outperform the baseline, and lastly creates a summary with a semantic aggregation.

Semantic operators perform familiar transformations, such as filters and aggregations, reminiscent of relational operators. The crucial difference lies in the ability of semantic operators to perform *transformations with reasoning-based, natural-language specifications*. For instance, the `sem_filter` operator takes the natural language parameter "the paper {abstracts} claim to outperform the baseline". The expected output of this operator is all records from the dataset with abstracts that pass this *natural language predicate*.

The reasoning capabilities of semantic operators significantly distend traditional relational operators and indices. Still, traditional

Table 1: Summary of Key Semantic Operators. T denotes a relation, X and Y denote arbitrary tuple types. l denotes a parameterized natural language expression ("langex" for short). Each operator may permit additional optional parameters, including accuracy targets.

Operator	Description	Definition	Reference Algorithm
$sem\_filter(l\colon X \to Bool)$	Returns the tuples that pass the langex predicate.	$\{t_i \mid t_i \in T \wedge l_M(t_i) = 1\}$	Compute $M(t_i, l), t_i \in T$
$sem\_join(t\colon T,\ \ l\colon (X,Y) \to Bool)$	Joins a table against a second table t by keeping all tuple pairs that pass the langex predicate.	$\{(t_i, t_j) \mid l_M(t_i, t_j) = 1, t_i \in T_1, t_j \in T_2\}$	Compute $M(\{t_i, t_j\}, l), t_i \in T_1, t_j \in T_2$
$sem\_agg(l\colon T[X] \to X)$	Aggregates input tuples according to the langex reducer function.	$l_M(t_1, ..., t_n) \forall t_1, .., t_n \in T$	Perform a hierarchical reduce, recursively computing $acc_{i,r} \leftarrow M(\{acc_{n_i,r-1}, ..., acc_{n_i+n,r-1}\}, l)$
$sem\_topk(l\colon T[X] \to Seq[X],\ \ k\colon int)$	Returns an ordered list of the k best tuples according to the langex ranking criteria.	$\langle t_1, ..., t_k \rangle$ st $\forall (t_i, t_j), i < j \implies l_M(t_i, t_j) = \langle t_i, t_j \rangle$	Perform quick-select top-k using pairwise comparisons, $M(\{t_i, t_j\}, l)$
$sem\_group\_by(l\colon X \to Y,\ \ C\colon int)$	Groups the tuples into C categories based on the langex grouping criteria.	$\underset{\{\mu_1,...\mu_C\},\mu_i \in V^{\mathbb{N}}}{\arg\max} \sum_{t_i \in T} \max_{j \in 1...C} l_M(t_i, \mu_j)$	Obtain centers $\mu_1, ...\mu_C$ with a clustering algorithm, and perform pointwise assignments $M(t_i, (l, \mu_1, ..., \mu_C))$,$t_i \in T$,
$sem\_map(l\colon X \to Y)$	Performs the projection specified by the langex.	$\{l_M(t_i) \mid t_i \in T\}$	Compute $M(t_i, l), t_i \in T$

non-AI primitives may be used to transparently optimize the execution of semantic operators. For instance, a naive execution of the semantic filter might process each record in a separate LLM invocation that prompts the model to evaluate the predicate. However, an optimized execution might leverage a light-weight model or vector index to speed up some predicate evaluations while reserving LLM-based predicate evaluations only when needed. Crucially, an efficient analytics system should provide such optimizations *transparently and adaptively* to reduce cost when possible, while also providing accuracy guarantees.

2.2 Defining Semantic Operators

Definition. *A semantic operator is a declarative transformation over one or more datasets, parameterized by a natural language expression. Each semantic operator can be implemented by potentially many AI-based algorithms, and its correct behavior is defined with respect to a given reference algorithm.*

Table 1 lists a core set of semantic operators, which cover common semantic transformations in real-world applications and mirror key transformations in relational operators. Specific systems may, of course, provide additional semantic operators beyond the ones we discuss here. Each semantic operator takes a *parameterized natural language expressions* (*langex* for short), which are natural language expressions that specify a function over one or more attributes. As Figure 1 demonstrates, the langex signature varies for different semantic transformations. While the `sem_filter` langex signature provides a natural language *predicate*, the `sem_agg` langex is a commutative, associative *aggregator* expression, which here indicates a summarization task over abstracts.

We provide a high-level definition of each semantic transformation with respect to the user langex and a world model[1], M, which captures a probability distribution over the vocabulary V. For example, semantic filter returns $\{t_i \mid t_i \in T \wedge l_M(t_i) = 1\}$, where T is the input relation and $l_M(t_i)$ represents a natural language predicate evaluated on tuple t_i with model M. Notably, the definition of each semantic operator can be implemented by multiple AI-based algorithms, and different decisions as to how to invoke the model over the relation have consequences on the algorithm's result quality. Thus, the correct behavior of each semantic operator is specified by a *reference algorithm*, a computable and tractable AI algorithm that produces results considered to be high quality. Each reference algorithm specifies a model access pattern over the relation T via an algorithm composed of model invocations $M(x, l)$, describing the subset of the data $x \subseteq T$, each model call is invoked over and the task-specific language expressions, given by l.

[1]In practice, the world model, M may be the strongest LLM a practioner has available.

2.3 Defining Correct Optimizations for Semantic Operators

Semantic operators create a rich design space of diverse execution plans. While reference algorithms provide high-quality implementations, they are often expensive, with the complexity of LLM calls scaling linearly or quadratically in the dataset cardinality. As such, we define correct optimizations for individual semantic operators below by considering alternate AI-based algorithms that can reduce cost while offering *close results*. In general, optimized semantic operator plans allow for both lossless optimizations and approximations of a reference algorithm. This formulation builds on approximate query processing and assumes some error is often tolerable, which we find (Section 5) is reasonable for achieving high-quality results for semantic processing, which is inherently non-exact.

Definition. *A correct optimization for a given semantic operator, and a reference algorithm for that operator, reduces cost while providing statistical accuracy guarantees with respect to the reference algorithm. Specifically, the optimization should ensure an accuracy target, γ, is met with probability $1 - \delta$.*

2.4 Core Semantic Operators

We now overview several core semantic operators, providing the operator's definition and a reference algorithm for each. We discuss design decisions for our reference algorithms based on state-of-the-art algorithms studied in the AI literature, well-known failure cases, such as long-context challenges [48], or our experimental observations. Our empirical evaluation (Section 5) confirms that the reference algorithms we present support state-of-the-art result quality in real ML applications; however, finding optimal reference algorithms for each operator remains an open research question.

Semantic Filter is a unary operator over the relation T and returns the relation $\{t_i \mid t_i \in T \wedge l_M(t_i) = 1\}$, where the langex provides a natural language predicate over one or more attributes.

Reference Algorithm. Our reference algorithm runs batched LLM calls over all tuples in relation T. Each model invocation, $M(t_i, l)$,

```
join_res = papers_df.sem_join(dataset_df, "The paper {
    abstract:left} uses the {dataset_name:right}.")
topk_res = papers_df.sem_topk("The paper has the
    funniest {title}", K=5)
grouped_res = papers_df.sem_group_by("What is the main
    research topic of the paper {abstract}", C=10)
```

Figure 2: Example usage of sem_join, sem_topk, and sem_groupby. papers_df contains fields for the "title" "abstract", and dataset_df, contains a field called "dataset_name".

prompts the LLM with a single tuple $t_i \in T$, the langex predicate, and an operator-specific instruction to generate a boolean value. This simple choice avoids well-studied long-context issues [48] by processing rows independently rather than in a single invocation.

Semantic Join provides a binary operator over relations T_1 and T_2 to return the relation $\{(t_i, t_j) | l_M(t_i, t_j) = 1, t_i \in T_1, t_j \in T_2\}$. Here the langex is parameterized by the left and right join keys and describes a natural language predicate over both.

Reference Algorithm. The reference algorithm implements a *nested-loop join pattern*, performing a single predicate evaluation for each pair of tuples, with model invocations $M(\{t_i, t_j\}, l), t_i \in T_1, t_j \in T_2$. This yields an $O(|T_1| \cdot |T_2|)$ LLM call complexity.

Semanic Top-k imposes a ranking[2] over the relation T and returns the ordered sequence, $\langle t_1, ..., t_k \rangle$ st $\forall (t_i, t_j), i < j \implies l_M(t_i, t_j) = \langle t_i, t_j \rangle$. Here, the langex signature is a general ranking criteria.

Reference Algorithm. Two important algorithmic design decisions for the semantic top-k include how to implement LLM-based comparison and how to aggregate ranking information from these comparisons. Our reference algorithm uses pairwise LLM comparisons for the former, and a quick-select top-k algorithm [33] for the latter. We briefly describe the reason for these choices and alternatives considered. Our decisions build on prior works, which have studied LLM-based passage re-ranking [27, 31, 45, 52, 56–59, 62] and ranking with noisy comparisons [20, 60] with the goal of achieving high quality results in a modest complexity of LLM calls or comparisons.

First, pairwise-prompting methods offer a simple and high-quality approach that feeds a single pair of tuples to each LLM invocation, $M(\{t_i, t_j\}, l)$, prompting the model to compare the two inputs and output a binary label. The two main classes of alternatives are point-wise ranking methods [27, 31, 45, 59, 69], and list-wise ranking methods [52, 56, 57, 62], both of which have been shown to face quality issues [27, 58, 62]. We verify these limitations in Section 5. In contrast, pairwise comparisons have been shown to be effective and relatively robust to input ordering [58].

In addition, we consider several possible rank-aggregation algorithms, including quadratic sorting algorithms, a heap-based top-k algorithm and a quick-select-based top-k ranking algorithm. Our evaluation in Section 5 demonstrates that each of these sorting algorithms yield high-quality results, comparable to one another. However, the quick-select-based algorithm offers an efficient implementation with at least an order of magnitude fewer LLM calls than the quadratic sorting algorithm and more opportunities for efficient batched inference, leading to lower execution time, compared to a heap-based implementation. The quick-select top-k algorithm proceeds in successive rounds, each time choosing a pivot, and comparing all other remaining tuples to the pivot tuple to determine the rank of the pivot. Because each round is fully parallelizable, we can efficiently batch these LLM-based comparisons before recursing.

Semantic Aggregation performs a many-to-one reduce over the input relation, returning $l_M(t_1, ..., t_n), \forall t_1, .., t_n \in T$. Here, the langex signature is a commutative, associative aggregation function[3], which can be applied over any subset of rows to produce an intermediate results. We note that the langex iteself is model-agnostic, assuming infinite context. Managing finite context limits of the underlying model M is an implementation detail of the system.

Reference Algorithm. Our reference algorithm uses a hierarchical reduce pattern. Our choice builds on the LLM-based summarization pattern studied by prior research works [16, 21, 68] and deployed systems [7, 13], which we briefly overview. Prior works primarily study two aggregation patterns. First, a fold pattern performs a linear pass over the data, iteratively updating the accumulated partial answer with the next tuple t_i, given by $acc_i \leftarrow M(\{acc_{i-1}, t_i\}, l)$. Alternatively, the hierarchical reduce pattern recursively aggregates n inputs in each round r and produce multiple partial answers, given by $acc_{i,r} \leftarrow M(\{acc_{n_i, r-1}, ..., acc_{n_i+n, r-1}\}, l)$ until a single answer remains. Both represent candidate reference algorithms, however, the hierarchical pattern has been shown to produce higher quality results for commutative, associative aggregation tasks, like summarization, in prior work [21] and allows for greater parallelism during query processing, making it our default choice.

Semantic Group-by takes a langex that specifies a projection from a tuple to an unknown group label, as well as a target number of groups, which specifies the desired granularity of group labels. As a example, a user might group-by the topics presented in a set of ArXiv papers, wishing to find 10 key groups. The group-by operator must *discover* representative group labels and assign a label to each each tuple. In general, performing the unsupervised group discovery is a clustering task, which is NP-hard [26]. Clustering algorithms over points in a metric space typically optimize the potential function tractably using coordinate descent algorithms, such as k-means. For the semantic group-by, the clustering task is over unstructured fields with a natural language similarity function specified by $l_M(t_i, \mu_j)$, which imposes a real-valued score between a tuple t_i and a candidate label μ_j. This operator poses the following optimization problem:

$$\underset{\{\mu_1, ... \mu_C\},\, \mu_i \in V^{\mathbb{N}}}{\arg\max} \sum_{t_i \in T} \max_{j \in 1...C} l_M(t_i, \mu_j)$$

where μ_i is a group labels, consisting of tokens in vocabulary, V.

Reference Algorithm. Since this operator, by definition, entails the NP-hard clustering problem [26], our reference algorithm uses a tractable clustering heuristic to discover group labels, then performs point-wise classification to assign each record to a discovered group label. Specifically, our LLM-based clustering algorithm discovers centers $\mu_1, ..., \mu_C$ by first performing a semantic projection, with model invocations $M(t_i, l), t_i \in T$, prompting the LLM to predict a candidate label for each input tuple. Then, we embed these candidate labels and perform an efficient vector clustering using k-means to construct C groups. For each group, we top-k

[2]This definition implies that l_M imposes a total and consistent ordering. However, this definition can also be softened to assume partial orderings and noisy comparisons with respect to model M.

[3]We note that the ordering of inputs within an LLM prompt invocation can in fact affect results quality for some tasks. To allow programmers to override the commutativity and associativity, LOTUS exposes a partioner function.

Algorithm 1: SEM-FILTER(T, l, $M(x)$, $A(x)$, γ_R, γ_P, δ)

Input: Relation T, langex predicate l, oracle model $M(x)$, proxy model $A(x)$, recall target γ_R, precision target γ_P, error probability δ
Output: Filtered relation T'

```
s ← sample_size
S ← ImportanceSample(T, A(x), s)
M_S ← {M(t) : t ∈ S}
A_D ← {A(t) : t ∈ D}
A_S ← {A(t) : t ∈ S}
τ+ ← PT_threshold_estimation(S, l, M_S, A_S, γ_P, δ/2)
τ− ← RT_threshold_estimation(S, l, M_S, A_S, γ_R, δ/2)
τ+ ← max(τ+, τ−)
T' ← ∅
// Evaluate predicate for each tuple
for t ∈ T do
    if A(t) ≥ τ+ then
        T' ← T' ∪ {t}
    else if A(t) ≥ τ− then
        if M(t) then
            T' ← T' ∪ {t}
end
return T'
```

sample by centroid-similarity scores, and perform a semantic aggregation to synthesize an appropriate label over each group. This provides a reasonable clustering heuristic similar to prior work [25], although alternative heuristics are possible. In the second stage, our reference algorithm uses the C generated labels, $\mu_1, ...\mu_C$, and performs point-wise assignments $M(t_i, (l, \mu_1, ..., \mu_C)), t_i \in T$. We choose point-wise classification to avoid the long-context scaling challenges studied in prior works [25, 48]. This algorithm yields $O(|T|)$ LLM call complexity.

3 OPTIMIZED EXECUTION PLANS FOR SEMANTIC OPERATORS

In this section, we present novel optimizations for several costly operators, including semantic filter, join, top-k and group-by. Each optimization provides statistical accuracy guarantees with respect to the reference algorithm, building from Section 2. While this paper focuses on novel optimizations for several expensive semantic operators, we envision a rich space of future work exploring new optimizations and applications of traditional optimizations for semantic operator programs. For example, several prior works demonstrate performance gains in both logical query plan optimizations (e.g. operator re-ordering [46, 47, 50, 51]) and other general LLM-approximation techniques (e.g. code synthesis [18, 47] and prompt adaptation [22]). We also note two core semantic operators, whose optimizations we do not study in this section: semantic maps and aggregations. Semantic maps reflect the general problem of batched LLM inference, which has been well studied by prior works. We instead focus the scope of this study on novel optimizations unique to semantic operators. Although semantic aggregations offer interesting optimization opportunities, we exclude an exploration here due to space limitations. We point the interested reader to the discussion in an extended version of this manuscript [54] and we leave a detailed quantitative analysis to future work.

3.1 Optimizing Semantic Filter

We provide an approximation for semantic filters that obtains recall and precision targets γ_R and γ_P with respect to the reference algorithm with probability $1-\delta$. We leverage the cheaper, but less accurate proxy model $A(t, l)$, which is configured by the user and can output a score indicating whether the tuple t passes the predicate. This idea is inspired by prior works [22, 36, 38, 65, 72, 72], which leverage model cascades for different problem settings. Specifically, several works study cascades in the video analytics setting with vision models, which have significantly different properties than LLMs, requiring different proxy scoring mechanisms. Other works study cascades for LLMs, but use heavy-weight scoring mechanisms to decide whether to use the proxy model and do not provide accuracy guarantees. In contrast, we focus on providing accuracy guarantees for the constrained problem of applying cascades for filters, which allows us to use lighter-weight scoring functions than the more general case studied in prior works.

In general, we cannot assume the proxy model is accurate. In fact, the proxy model may perform very poorly for some tasks and records. The goal of our algorithm is to automatically discover the quality of the proxy *at query time* by sampling and comparing to the reference algorithm with the oracle model. Our algorithm will exploit the proxy when it is likely to provide high-quality outputs, and otherwise defer the oracle model. To do this we must learn a decision rule, with learned thresholds on the proxy scores using sampling. As prior work [37] discusses, such sampling-based algorithms introduce multiple-hypothesis testing problems, requiring statistical corrections using confidence intervals, which we carefully apply in our algorithm.

We consider a small LLM as the proxy model and generate scores $A(t)$ using log-probabilities[4] corresponding to the `True` or `False` output tokens of the proxy model's predicate evaluations, re-scaling by the quantiles over all generated log-probabilities over the relation. Specifically, we construct q evenly-spaced quantiles in the range 0 to 100, and bin each computed proxy confidence score into one of these quantiles, reassigning the confidence score of each sample to its quantile number. In our experiments, we find that using $q = 50$ quantiles provides sufficient granularity while also maintaining efficiency. Algorithm 1 shows the full procedure for performing the approximate semantic filter. We begin by collecting a sample of tuples S, and labeling each sample with the oracle and proxy models. The sampling procedure uses importance sampling and defensively mixes a uniform sample following prior work [38]. Importance sampling chooses records t with replacement from the dataset D with weighted probabilities $w(t)$ then uses a reweighting identity to compute the expected value of a quantity $f(t)$, instead of uniformly sampling each record with probability $1/|D|$. Our weight function for sampling uses $\sqrt{(A(t))}$, following Kang et al. [38], and we refer the reader to Algorithm 4 of their paper for a detailed sketch. Additionally, the sample size is a hyperparameter, which can affect the convergence rate of probabilistic guarantees and has been studied by prior work on cascade-based optimizations. We set a default value by taking the maximum 1% of the dataset size and 100 samples, following prior work [38].

Using the central limit theorem and the normal approximation on the distribution of sample statistics, we then learn a decision rule by finding thresholds τ_+ and τ_-, which will respectively ensure

[4]Log-probabilities are available in common model providers, e.g., OpenAI, and serving systems, e.g., vLLM. In the absence of available log-probabilities, alternative uncertainty quantification methods, such as prompt-based methods, may be more suitable.

the precision target and recall target are met, each with an error probability of $\delta/2$. The statistical sub-procedures to choose each threshold follows prior work [38], but applies them to this new setting where we must ensure *both* targets are met, requiring a correction for hypothesis testing and multiple failure modes. Finally, using the learned decision rule, our algorithm proceeds to process each tuple in the relation. If the tuple's proxy score is at least τ_+, we mark it as passing the predicate. If the tuple's proxy score is less than or equal to τ_-, we mark it as failing the predicate, and for tuples with proxy scores between τ_+ and τ_-, we resort to the oracle model to provide a label.

3.2 Optimizing Semantic Join

Similar to semantic filters, we provide an approximation for semantic joins that obtains recall and precision targets γ_R and γ_P with respect to the reference algorithm with probability $1-\delta$. Due to the expensive quadratic scaling of the nested-loop join, rather than using a small LLM as the proxy model, we consider an even cheaper proxy based on semantic similarity scores using embeddings to reduce the LLM-call complexity. We leverage two possible proxy algorithms and dynamically choose the lowest-cost one.

The first approximation, **sim-filter** produces a proxy score $A_1(t_i, t_j)$, $t_i \in T_1, t_j \in T_2$ based on embedding similarity. We perform batched similarity search between the right and left join keys and re-calibrate similarity scores according to their quantiles to obtain proxy scores. We then use the proxy scores and a learned threshold to determine when to resort to the LLM for the join predicate evaluation between tuple pairs, and when to rely on the embedding-based proxy score. This approximation is likely to perform efficiently when tuple pairs with high semantic similarity scores between the right and left join key are more likely to pass the join predicate. This correlation phenomenon between predicate matches and embedding similarity has been studied by prior works [55] and is not always present. Thus, we introduce an alternative approximation, often suitable when correlation is not present.

The second approximation, **project-sim-filter** first performs a projection over the left join key and then uses the projected column to compute proxy scores $A_2(t_i', t_j)$ based on embedding similarities between the projected value t_i' and t_j. As before, we also re-calibrate the proxy scores taking their quantiles. Intuitively, performing the projection is useful when tuple pairs that pass the user's natural language predicate exhibit low semantic similarity. The projection step invokes the LLM over each tuple in left join table, prompting it to predict attributes values for the right join key, without specifying the domain of the right join key. Notably, this LLM projection is ungrounded, i.e., it is done without knowledge of the right join key's attribute domain and can thus be performed in a fully parallelized semantic map operation. Figure 2 provides an example of a semantic join between a table of papers and datasets, where the predicate evaluates whether a given paper abstract uses a specific dataset. Here, the map step would invoke the LLM over each abstract, instructing the model to output the dataset used, conditioned only on the abstract.

To dynamically choose between these two approximations, we follow a procedure similar to Algorithm 1 to determine the embedding thresholds for both the sim-filter and project-sim-filter approximations. We begin by importance sampling to collect the sample S and construct the set of oracle labels, O_S, over the sample. We then obtain embedding thresholds τ_+ and τ_- independently for each approximation algorithm using their respective proxy scores, A_1 and A_2. We use the learned thresholds for each candidate plan to then determine the exact oracle cost needed to execute either algorithm, and we take the least cost plan with its associated learned thresholds to proceed to evaluate the predicate for each tuple pair.

3.3 Optimizing Semantic Group-by

We provide an efficient approximation that guarantees a classification accuracy target, γ, is met with probability $1-\delta$. We follow the clustering algorithm in the first stage of the reference algorithm to discover centers $\mu_1, \ldots \mu_C$. We then use a proxy-based approximation for point-wise assignments in the second stage of the algorithm. Similarly to semantic joins, we leverage semantic similarity scores as a cheap proxy, although other proxy models are also feasible. Specifically, we leverage the embeddings constructed during the clustering stage and compute similarity scores between the candidate label of each tuple and the discovered centers $\mu_1, \ldots \mu_C$. The proxy score of the tuple t_i with a candidate label, t_i', for a discovered center μ_j is given by $A(t_i, \mu_j) = sim(t_i', \mu_j)$. Our goal is then to learn a threshold τ, such that if the proxy score between a tuple and the center is greater than τ, we return the center as the label for the tuple, and otherwise we resort to the more expensive LLM-based classification procedure. We accomplish this by uniform sampling and running the sub-procedure equivalent to the `PT_threshold_estimation` used in Algorithm 1 for semantic filters, where the metric we evaluate is classification accuracy over the C groups. Here we use uniform rather than importance sampling since the sample should consist of representative classes of the whole dataset, rather than a single, possibly rare, class of interest, as is the case for semantic filtering and joins.

3.4 Optimizing Semantic Top-k

We leverage the embedding similarity scores to optimize pivot selection for some queries, while incurring no accuracy loss. This optimization is useful when correlation exists between the ranking imposed by the user's arbitrary sorting criteria and the ranking imposed by semantic similarity scores. In this case, we can sort tuples based on embedding distances to the user's query, and select the $(k+\epsilon)$-th item, rather than a random item, as the first pivot. This can reduce the number of LLM comparisons required by subsequent rounds in the quick-select algorithm, leading to higher query efficiency at no accuracy loss. In the case of no correlation between the langex-based ranking and similarity-based ranking, this method amounts to random pivot selection, and in the worst case of an adversarial pivot, the algorithm will incur one extra pivot round, which can impact execution time, but not degrade quality.

4 THE LOTUS SYSTEM

In this section we describe the LOTUS system, which implements the semantic operator model as an extension of Pandas [11]. We chose a Pandas-like API in our initial system implementation to make it easy for users to integrate LOTUS with popular AI libraries in Python. However, semantic operators could also be added to a

```
papers_df.sem_sim_join(papers_df, left_on="abstract",
    right_on="abstract", K=10)
papers_df.sem_search(col="abstract", query="vector
    databases", K=10)
```

Figure 3: Example usage of sem_sim_join and sem_search.

variety of other data processing APIs and query languages, such as SQL. LOTUS leverages vLLM [43] to perform efficient batched inference, and uses FAISS [30, 34], by default, to support efficient vector search with indices stored and maintained locally on disk.

4.1 Datatypes

LOTUS' data model consists of tables with structured and unstructured fields (e.g., text or images), both of which can be passed as langex parameters. LOTUS also supports *semantic indices* over unstructured fields for optimized query processing. These indices leverage semantic embeddings over each entity in a column and capture semantic similarity using embedding distance metrics. Semantic indices can be created off-line with `sem_index` and a specified retriever model, and then loaded from disk using `load_sem_index`.

4.2 Semantic Operators in LOTUS

We now overview the semantic operators supported in the LOTUS API, which includes the core set of operators described in Section 2, as well as several additional variants provided for convenience. Each operator takes optional parameters to specify a target accuracy and error probability, which the optimizer will use to transparently perform optimizations. In general, the tradeoff curve between accuracy, latency and cost curve depends on the specific user task, dataset, and configured models. In practice, one can empirically analyze this tradeoff by benchmarking varied accuracy targets and choose the operating point most suitable for the given application.

Sem_filter, Sem_join, & Sem_sim_join. The LOTUS API supports `sem_filter` and `sem_join`, both of which take a langex predicate, as described in Section 2. In addition, LOTUS provides a join variant, `sem_sim_join`, where tuples are matched according to their *semantic similarity*, rather than an arbitrary natural-language predicate. Akin to an equi-join in standard relational algebra, the semantic similarity join is a specialized semantic join, which indicates additional optimization opportunities to the query engine by leveraging vector similarity search. Figure 3 provides an example of the `sem_join` compared to the `sem_sim_join`, where the user specifies the left and right table join keys, and a parameter K. The operator performs a left join such that for each row in the left table, the output table will contain K most similar rows from the right table.

Sem_topk & Sem_search. LOTUS supports a semantic top-k, which takes the langex ranking criteria, as described in Section 2. Programmers can optionally specify a group-by parameter. The groupings are defined using standard equality matches over the group-by columns. Additionally, as Figure 3 shows, LOTUS also provides a top-k variant, `sem_search`, which returns results ranked by similarity to the natural language query. LOTUS also exposes advanced relevance-based re-ranking functionality for search. Users can specify the `n_rerank` parameter, which will re-rank the top-K documents and return the top `n_rerank`.

Sem_agg. The LOTUS API supports semantic aggregations, following the description in Section 2. Similar to `sem_topk`, LOTUS also allows users to optionally specify a group-by parameter.

Sem_group_by. This operator creates groups over the input dataframe according to the langex projection and target number of groups. By default, the operator discovers suitable group labels, but the user can also optionally specify target labels. This operator is useful both for semantic clustering and classification tasks.

Sem_map & Sem_extract. Both these operators perform a natural language projection over an existing column. While `sem_map` projects to an arbitrary text attribute, `sem_extract` projects each tuple to a list of sub-strings from the source text. This is useful for applications, such as entity extraction, where finding snippets or verified quotes may be preferable to synthesized answers.

5 EVALUATION

We systematically evaluate LOTUS on four bulk-semantic processing applications from the AI and data literature: fact-checking, biomedical multi-label classification, search, and topic analysis. Some of these applications can be expressed by recent LLM-based analytics systems, like AI UDFs, UQE and DocETL, allowing us to compare against them; while other applications are not supported by the baseline systems, but LOTUS supports them. For each application, we additionally compare against strong hand-designed pipelines from the literature. We evaluate the LOTUS optimizer, comparing the performance of optimizations for semantic filters, joins, top-k and group-by with our reference algorithms. Lastly, we analyze the statistical accuracy guarantees provided by LOTUS' optimizations. Overall, we find the following:

- Compared to AI-based analytics systems, like AI UDFs, UQE, and DocETL, LOTUS attains similar or up to 170% higher accuracy, while running up to 3.6× faster and offering statistical accuracy guarantees that the other systems do not provide.
- Compared to hand-written pipelines from the AI literature for fact-checking, biomedical classification, and search, LOTUS matches state-of-the-art quality, or exceeds it by up to 100%, in programs involving a few semantic operators.
- The LOTUS optimizer can substantially reduce cost of semantic operator programs, by up to $1,000\times$ compared to high-quality reference algorithms, and provides accuracy guarantees, which hold across repeated trials.

To report controlled latency numbers, we run each baseline and experiment locally on 4 80GB A100 GPUs using Llama-3-70B [6] and E5 embeddings [66], with a batch size of 64 running on vLLM [43], unless otherwise stated. To run DocETL baselines and reproduce some of their benchmarks [61], we additionally use OpenAI [10] models, including GPT-4o-mini-2024-07-18, GPT-4o-2024-08-06 model, and text-embedding-3-small, where noted. For our experiments that use OpenAI models, we control for cost by limiting execution to 64-way thread parallelism. For reproducibility, we set temperature to $t = 0$ for all methods, unless otherwise stated.

5.0.1 Benchmarked Methods. We briefly overview the main methods we benchmark and tested parameters.

AI UDFs. Many database vendors [1, 8, 9, 14, 53, 67] now support AI UDFs, which include map-like, row-wise LLM operations and primitives for vector search. To control for serving infrastructure, we implement these programs using LOTUS running on vLLM, and restrict our API to `sem_map`, `sem_search` and `sem_sim_join`.

UQE. UQE [25] proposes an optimized LLM-powered filter method, which is relevant to this work. Since the code is not open-source, we implement the UQE filter in 100 line of python code. The UQE filter takes an LLM call budget and provides a best-effort embedding-based approximation. To provide a fair comparison between UQE and LOTUS, we normalize the *latency budget* when comparing against LOTUS programs that use LLM-based proxies, and we normalize *the LLM call budget* when comparing against LOTUS programs that use embedding-based proxies. We also hyperparameter tune UQE's minibatch size, used for its active learning algorithm.

DocETL. DocETL [61] uses an LLM agent as the query optimizer to perform rewrites to programs, allowing programs to specify separate LLMs for the optimizer and execution. Our experiments find that the optimizer often fails, and among runs where the optimizer succeeds, performance varies. We obtain three successful runs and report the average performance of these three successes. We also try using both Llama-70B as the DocETL optimizer, following our setup for each other baseline, and GPT-4o-mini as the DocETL optimizer, following the DocETL preprint [61]. However, we find both fail to produce a successful run in Section 5.2. We confirm a mistake in the pre-print with the authors and find that GPT-4o is required for the DocETL optimizer for the BioDEX experiments in Section 5.2. Since the codebase is an evolving artifact, we report the commit number we used here[5].

LOTUS. We set the default accuracy target, $\gamma = 0.9$ and failure probabilitiy, $\delta = 0.2$. We provide a detailed ablation, varying these parameters in Section 5.5. We also show several benchmarks using either the oracle model only ($\gamma = 1$), or the proxy model only ($\gamma = 0$). Additionally, our sample size, s, use 0.01% of the data and a minimum of $s = 100$ for smaller datasets, following prior work [37].

5.1 Fact-Checking

Fact-checking systems determine whether a given claim is correct, typically based on verifiability with a knowledge corpus. FacTool [23] is a recent open-source research work that provides a multi-step fact-checking pipeline. We consider the task of reconstructing the FacTool pipeline, which was originally written in over 750 lines of code. For our evaluation, we use the FEVER [64] dataset, a claim verification dataset based on a corpus of 5.5 million Wikipedia articles. Each claim is labeled with one of three reference labels, "Supported", "Refuted", or "NotEnoughInfo". We merge the latter two labels into a single class, "Not Supported", following prior work [23]. To control for our API spending budget, we sample 1,000 claims from the development dataset.

Baselines. For all baselines we use ColBERT [42] as the retreiver model[6] and Llama-70B served with vLLM. We run FactTool's open source codebase [5] to measure its performance. The AI UDF baseline uses a simple semantic map-search-map dataflow, following FacTool's pipeline. First each claim is mapped to two search queries. Then the generated queries are used for search over the Wikipedia corpus. Lastly, each claim, appended with retrieved context, is mapped to a truth label and reasoning chain. We use the same prompts found in FacTool [5], which include 3 demonstrations for generating search queries in the first semantic mapping, and chain-of-thought prompting in the second semantic mapping. The UQE baseline follows the first two steps of the AI UDF baseline, but replaces the final step with UQE's embedding-based LLM-powered filter. Our LOTUS program similarly follows a map-search-filter pipeline and uses a Llama-8B proxy model. To provide a fair comparison, we tune the UQE LLM call budget to normalize execution time with the optimized LOTUS program. Since DocETL does not provide search primitives, we cannot benchmark it in this task.

Table 2: Fact-checking Performance on the FEVER Dataset.

Method	Accuracy	ET (s), batching	ET (s), no batching	LoC
FacTool	80.9	N/A	5396.11	> 750
AI UDF: map, search, map	89.9	688.9	4,454.2	< 50
AI UDF map, search + UQE filter	66.0	184.4	738.3	150
LOTUS map, search, filter (unopt.)	91.2	329.1	989.0	< 50
LOTUS map, search, filter (opt.)	91.0	190.0	776.37	< 50

Results. We report each method's accuracy, an estimate of lines of code (LoC), and average execution time (ET) over 10 runs in seconds, both with and without batching to provide a fair comparison with FacTool, which provides a sequential implementation only. Overall, Table 2 demonstrates that the optimized LOTUS program reproduces state-of-the art accuracy, comparable to that of FacTool, with 7× faster unbatched execution and 28× faster batched execution, and in less than 50 LoC, while also outperforming baseline.

We begin by noting that the unoptimized LOTUS program achieves state-of-the-art accuracy; its improvement over the AI UDF and UQE baselines reflect the effectiveness of our reference algorithm. Comparing the un-optimized and optimized LOTUS program to the AI UDF baseline, we see that each achieves comparable accuracy in relatively few lines of code. Notably, the AI UDF baseline and LOTUS program differ solely in the last step of the pipeline, where LOTUS uses a semantic filter instead of a map. The LOTUS `sem_filter` can be more heavily optimized compared to the generic AI UDF map operation. This allows the LOTUS program to attain 3.6× faster execution, attributable to the filter's short LLM generations in the un-optimized LOTUS baseline, and our cascade-based optimization for filters in the optimized LOTUS program.

We also observe that the LOTUS program outperforms the UQE program by 38% accuracy at an equivalent latency budget. The UQE baseline differs from LOTUS solely in the way it performs semantic filtering. LOTUS provides a high quality un-optimized semantic filter, then optimizes the operation with accuracy guarantees by adaptively learning when to apply the proxy. On the other hand, UQE takes a best-effort approach with a fixed embedding-based proxy that is applied to all samples once the user's LLM budget is depleted. The UQE filter is unable to automatically learn that the embedding-based proxy performs poorly on this task.

Semantic Filter Optimization. Finally, we compare the LOTUS programs with and without the semantic filter optimization. Table 2 shows the optimized program achieves 99.8% accuracy relative to the un-optimized one, while reducing batched execution time by 1.7×. Figure 4 further highlights the diverse operating points

[5]https://github.com/ucbepic/docetl/commit/93050998077a9eb6fc1ee99dc96d1e0a222a987f

[6]FactTool's pipeline, by default, performs retrieval with a Google Search API [3]. We evaluate the pipeline with both the default retrieval API, and with the open-source ColBERT [42] index. We find that the results are similar, and we report the results using ColBERT for retrieval to hold the retriever model constant with the other baselines.

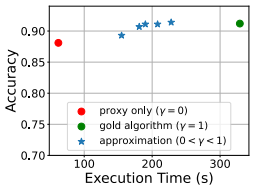


Figure 4: Accuracy vs. execution time (s) for the LOTUS fact-checking program on the FEVER dataset. We compare the performance using the proxy only (red circle), oracle only (green circle) and both models with varied the precision and recall targets, at $\delta = 0.2$ (stars) for the filter.

achieved by our `sem_filter` by varying the recall and precision target, from $\gamma = 0$ (red circle), to $\gamma = 1$ (green circle).

5.2 Biomedical Multi-label Classification

Biomedical classification entails processing complex, lengthy patient documents. We consider the extreme multi-label classification task on the BioDEX Dataset [29], which consists of a corpus of 65,000 biomedical articles, and ground-truth expert-created drug safety reports constructed from each article. The task is to classify the patient's drug reactions, given each medical article. Notably, there are 24,000 possible drug-reaction labels. Due to the large label set, leveraging an LLM to perform inference is difficult, and this setting has been studied in prior works [28]. We show below that this task can be efficiently modeled and optimized using the semantic join. Similar to prior work [29] we sample 250 patient articles for our evaluation to control for our API spending budget.

Baselines. The search baseline performs embedding similarity search over each patient article and the set of reaction labels. For the baseline LLM-based analytics systems, we implement a LLM-powered join over the patient articles and set of candidate reaction labels, followed by a ranking step, if supported. The AI UDF program entails a naive, nested-loop join pattern using row-wise LLM operators; however we note this is prohibitively costly and we report estimated latency. While UQE does not explicitly study joins, we implement a UQE join using its optimized filter. The DocETL program uses a join and reduce, which uses listwise ranking, following the preprint [61]. We note that we are unable to successfully run DocETL using Llama-70B or GPT-4o-mini for its optimizer, so results (Table 3, 4) are shown using GPT-4o for the DocETL optimizer. LOTUS supports both join and ranking. Although LOTUS provides a `sem_topk` operator, we limit our ranking step to list-wise ranking to maintain a fair comparison with DocETL, and revisit ranking in Section 5.3. Our main results are shown in Table 3, using Llama-70B and E5 embeddings and sampling size of 0.01%. We also provide an additional set of benchmarks in Table 4 comparing LOTUS and DocETL following the models, prompts and parameters used in the original DocETL preprint. The DocETL setup uses GPT-4o for the optimizer, GPT-4o-mini, text-embedding-3-small embeddings and a sampling size of 500. DocETL's join takes an LLM call budget, which we control to compare against the other baselines.

Results. Table 3 demonstrates the LOTUS programs consistently match or exceed the accuracy of all baselines, while also providing performance guarantees, unlike DocETL, which requires multiple reruns and manual selection of the optimizer LLM to produce high-quality results. The table reports the rank-precision@5 (RP@5) and rank-precision@10 (RP@10), following prior work [28], as well as execution time in seconds and the number of LLM calls. The LOTUS join with ranking program achieves higher rank-precision than the standalone LOTUS join program, and we show both.

Table 3: Biomedical Multi-label Classification Results on the BioDEX Dataset with Llama-70b

Method	RP@5	RP@10	ET (s)	# LM Calls
Search	0.106	0.120	2.91	0.00
AI UDF *	N/A	N/A	2,144,560	6,092,500
UQE	0.115	0.114	6,559	15,000
DocETL Join (avg of 3 successes)**	0.180	0.219	2050	13,185
DocETL Join + Rank (avg of 3 successes)**	0.262	0.282	2,342	13,433
LOTUS Join	0.212	0.213	2,340	5,644
LOTUS Join + Rank	0.265	0.280	2,503	5,869

Table 4: Additional Comparison Following DocETL's Original Benchmarks on the BioDEX Dataset with GPT-4o-mini

Method	RP@5	RP@10	ET (s)	# LM Calls
DocETL Join + Rank (avg of 3 successes)**	0.268	0.287	583	35,669
LOTUS Join + Rank	0.289	0.296	647	32,026

* AI UDF is infeasible to run and latency is estimated.
** The DocETL baselines requires using GPT-4o as its optimizer and still exhibit frequent failures, with, on average, 1 in 3 runs failing. We rerun the system multiple times and report results averaged only over 3 successful runs.

We begin by informally comparing these accuracy results to that of D'Oosterlinck et al. [28], a state-of-the art AI pipeline that composed a multi-step DSPy [41] program compiled using a combination of Llama-2-7b-chat, GPT-3.5-turbo, and GPT-4-turbo. D'Oosterlinck et al. report 24.73 RP@5 and 27.67 RP@10 for the compiled program, which is comparable to result quality achieved by the simple LOTUS join with ranking program.

Turning to the baselines measured in Table 3, we see that, first, the search baseline offers low results quality, with the LOTUS programs achieving over 2× higher RP@5 and RP@10 due to the stronger reasoning capability of LLM algorithms. Next, The AI UDF baseline, using row-wise LLM calls, is prohibitively expensive, and reflects the cost of our reference algorithm for joins. Its quadratic scaling of LLM call complexity with respect to dataset size leads to 1,000× higher estimated cost relative to the optimized LOTUS join program. UQE, which uses an embedding-based optimization, obtains similar accuracy to the search baseline, remaining well below the RP@5 and RP@10 of the LOTUS programs despite a generous LLM call budget, indicating that the UQE method has lower sample efficiency on this task. Next, we turn to a detailed comparison of DocETL and LOTUS, shown in Table 3 and Table 4. LOTUS consistently matches or exceeds the accuracy of DocETL's successful runs. Notably, the DocETL agentic optimizer requires GPT-4o and frequently fails, requiring multiple reruns, while LOTUS' optimizer provides statistical accuracy guarantees, which we analyze further in Section 5.5. Table 4 also shows that LOTUS and DocETL have similar numbers of LLM calls and execution time, however the token consumption per call of the DocETL programs are lower on average due to its optimized plan, which leads to modestly lower execution time despite modestly higher LLM calls.

Semantic Join Optimization. Lastly, we turn to Table 5 and study the candidate join plans produced by the LOTUS optimizer. Plan 1, the sim-filter, requires more LLM calls to meet the recall and

Table 5: Comparison of Candidate LOTUS Join Plans and Reference Algorithm on the BioDEX Dataset Using Llama-70b

Method	RP@5	RP@10	ET (s)	# LM Calls
LOTUS Join Plan 1	0.1541	0.170	12,563	27,687
LOTUS Join Plan 2 (chosen)	0.212	0.213	2,116	5,290
Reference Algorithm	N/A	N/A	2,144,560	6,092,500

precision targets, compared to Plan 2, the project-sim-filter plan. This reflects that Plan 2's proxy offers a stronger signal for predicate evaluations. We see that the LOTUS optimizer correctly selects Plan 2 to execute since it is lower cost. The selected pattern also results in significantly better accuracy due to its higher-quality proxy signal for this task. Specifically, the project-sim-filter pattern offers 37% higher RP@5 and 25% higher RP@10 compared to the sim-filter, with $1,000\times$ fewer LLM calls than the reference algorithm.

5.3 Search & Ranking

Relevance-based ranking has been widely studied in the context of information retrieval. In addition, our conversations with LOTUS users reveal a common need for ranking based on *complex natural language criteria* (e.g., ranking customer reviews based on how frustrated they sound). We assess LOTUS' ranking capabilities on both types of tasks using two datasets: BEIR's SciFact test set [63], a widely used benchmark for retrieval, and a new dataset, HellaSwag-bench, which we generated to assess more complex ranking tasks. For both, we report average nDCG@10, a standard ranking metric, and execution time (ET). The SciFact dataset consists of scientific claims and the task is to rank articles from a corpus by relevance to each claim. We sample 300 claims for our evaluation. HellaSwag-bench consists of 200 synthetic paper abstracts[7], each reporting an accuracy on the HellaSwag dataset [74], and the task is to rank abstracts by their reported accuracy. This dataset provides an objective ground truth, while focusing on a reasoning-based ranking criteria, rather than a relevance-based one[8]. We report results for $n = 20$ trials at temperature $t = 0.7$, similar to prior works [41].

Baselines. In addition to the standard search baseline, we add a re-ranker baseline using the MixedBread cross-encoder [15], a high quality re-ranker common in relevance-based retrieval. The AI UDF baseline performs a point-wise ranking by prompting the LLM to assign a relevance score to each row, similar to prior work [76]. The DocETL baseline uses the system's LLM `reduce` operation for ranking, as described by the pre-print [61]. Our LOTUS program uses the `sem_topk` operator with a semantic index on the document corpus of both datasets. For the reranker, AI UDF, DocETL, and LOTUS programs on SciFact, we perform search to retrieve 100 articles, before re-ranking them with each method.

Results. We evaluated the performance of the LOTUS program in comparison to the baseline systems, our reference algorithm, and alternative high-quality ranking algorithms we considered for our `sem_topk` reference algorithm.

First Table 6 demonstrates that LOTUS achieves $1.03 - 2\times$ higher accuracy than the next best-performing baselines on Scifact and HellaSwag-bench, reflecting the effectiveness of our LLM-based ranking algorithm using pairwise comparisons. Specifically, as expected, the re-ranker offers competitive accuracy compared to the LOTUS program on Scifact, which focuses on relevance-based retrieval, which is the supervised task on which the re-ranker model was trained. However, on HellaSwag-bench, a more complex ranking task, the LOTUS program significantly outperforms the re-ranker, reflecting the generalized capabilities of strong LLM-based reference algorithms. We also observe that the point-wise AI UDF method and the list-wise ranking, used by DocETL, offer substantially lower accuracy, unable to outperform the re-ranking baseline. This reflects the limitations of point-wise and list-wise ranking, both of which have been studied in prior work [58] and exhibit common failure modes involving calibration and long context. Notably, the DocETL baseline altogether fails to produce any successful execution plans using the GPT-4o optimizer on Scifact and Hellaswag-bench and using the Llama-70B optimizer on Scifact.

Table 6: Ranking Results on SciFact and Hellaswag-bench Datasets

Method	SciFact			HellaSwag-bench		
	nDCG@10	ET (s)	# LM Calls	nDCG@10	ET (s)	# LM Calls
Search	0.712	0.009	0	0.119	0.008	0
Reranker	0.741	2.64	0	0.461	2.36	0
AI UDF	0.457	9.83	100	0.091	19.7	200
DocETL - Llama 70B optimizer (avg. of 3 sucesses)*	N/A	N/A	N/A	0.246	14.1	3
DocETL - GPT 4o optimizer*	N/A	N/A	N/A	N/A	N/A	N/A
LOTUS	0.765	36.3	213.2	0.919	57.0	506.4

* DocETL is unable to acheive any successful runs using GPT-4o for its optimizer on SciFact and HellaSwag-bench, nor using Llama-70B for its optimizer on SciFact

Table 7: Comparison of LOTUS' Top-k to the Reference Algorithm and Other High-Quality Top-k Algorithms for Ranking

Method	SciFact			HellaSwag-bench		
	nDCG@10	ET (s)	# LM Calls	nDCG@10	ET (s)	# LM Calls
Quadratic Topk	0.768	801.2	4950	0.886	2116.6	19,900
Heap Topk	0.765	60.0	192.6	0.896	59.4	241.5
Quickselect Topk	0.771	40.5	237.0	0.893	49.4	448.1
LOTUS	0.765	36.3	213.2	0.919	57.0	506.4

Turning our attention to Table 7, we compare the quick-select top-k reference algorithm to several alternative candidates. The quadratic top-k, heap top-k, and quick-select top-k offer comparable accuracy on both datasets; however, the three candidate choices provide significant differences in efficiency. The quadratic algorithm requires $20 - 82\times$ more LLM calls and over $10\times$ higher execution time than the heap top-k and quick-select top-k. The quick-select top-k offers $16 - 32\%$ lower execution time than the heap-based sorting method, despite sometimes requiring more LLM calls. This is because the quick-select top-k implementation allows for efficient batched processing in each round of the algorithm, whereas the heap-based top-k incurs sequential LLM calls during heap updates.

Semantic Topk Optimization. Finally, the Table 7 compares the quick-select top-k with the LOTUS quick-select top-k using embedding-based pivot selection. We demonstrate that the optimization is lossless, as expected, and can decrease latency by 10%. Specifically,

[7]Each abstract in HellaSwag-bench is generated by prompting Llama-70b to write a research abstract that claims a accuracy value, randomly sampled from $0 - 100\%$.

[8]While the Hellaswag-bench ranking task can be solved with a sem_map, to extract accuracy values, followed by a structured sorting, our focus is on assessing semantic ranking algorithms. We find this benchmark useful for understanding performance trade-offs, which may generalize to a wider set of reasoning-based ranking queries.

(μ_1)	Advancements in Recommender Systems and Multimodal Data Integration
(μ_2)	Advancements in Generative Information Retrieval Systems
(μ_3)	Advancements in Large Language Models for Various Applications
(μ_4)	Advancements in AI Security and Malware Detection Techniques
(μ_5)	Advancements in Robotic Navigation and Manipulation Techniques

Figure 5: Discovered labels from LOTUS *sem_group_by("the topic of each {paper}", C=5)* over a dataset of recent ArXiv papers, scraped from the cs.DB, cs.IR, cs.CR and cs.RO domains.

this optimization is useful where document rank correlates with semantic similarity, which is likely on the Scifact dataset.

5.4 Topic Analysis on ArXiv Papers

A common unsupervised discovery task requires grouping a large document corpus by topics and assigning descriptive labels to each group. We consider this task over a dataset of 647 recent ArXiv articles, which we scraped from the database (cs.DB), information retrieval (cs.IR), cryptography and security (cs.CR), and robotics (cs.RO) domains. We aim to discover 5 key topic labels.

We can succinctly represent this task using the `sem_group_by` operator with LOTUS. AI UDFs, DocETL and standalone search systems do not serve operations for semantic group discovery. While UQE [25] studies optimizations for serving non-LLM aggregations, e.g., COUNT, with LLM-based group-bys, it does not focus on implementing, optimizing, and evaluating a standalone group-by operator, but suggests a standalone group-by implementation similar to our `sem_group_by` reference algorithm. Thus, in this section, we focus our analysis on understanding the performance of our `sem_group_by` reference algorithm and analyzing our optimization with respect to it. We use Llama-70B and E5 embeddings. For approximations, we use a sample size of 100 and $\delta = 0.2$.

Results. The `sem_group_by` consists of two sub-tasks: (a) discovering representative group labels, and (b) classifying each document. We first qualitatively analyze the labels discovered in the first sub-task. Figure 5 shows the discovered labels intuitively align with recent research topics in the cs.DB, cs.IR, cs.CR, and cs.RO ArXiv domains e.g., recommendation, retrieval, LLM applications, AI security, and robotics. Performing this label discovery sub-procedure took 44.03 seconds, representing a tractable algorithm.

Semantic Groupby Optimization. Turning to the classification sub-task, we compare the performance of the reference algorithm to our approximation, which uses an embedding-based proxy here. Intuitively, the LLM-based classification in the reference algorithm provides high quality by leveraging the LLM capabilities to reason about the main topic of each abstract, while the embedding-based model offers an approximation using semantic similarity between a topic and abstract. Figure 6 compares the execution time and classification accuracy of the oracle-based reference algorithm (green circle), our approximations using both the Llama-70B oracle LLM and embedding-based proxy at varied accuracy targets (blue stars), and the embedding-based proxy alone (red circle). The proxy-only baseline is 17.4× faster than the oracle, but about 39% less accurate. Our approximation allows users to declaratively interpolate between these two extremes by varying the accuracy target. We also note that the sampling-based procedure used to perform this optimization took less than 5 seconds, a small relative cost.

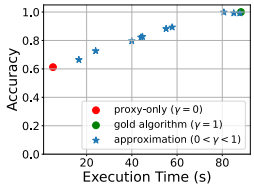


Figure 6: Classification accuracy vs. execution time (s) for the LOTUS sem_group_by over the ArXiv dataset. We compare performance of classifying each paper using the proxy only (red circle) or oracle only (green circle) to our approximation with varied accuracy targets, at $\delta = 0.2$ (blue stars).

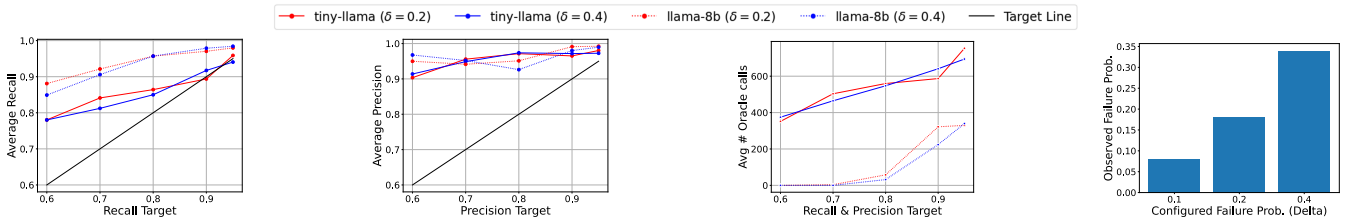
5.5 Accuracy Guarantees Evaluation

Our approximation methods are designed to adaptively learn when to leverage the proxy and guarantee the accuracy target is met. To demonstrate the accuracy guarantees provided by our optimizations, we vary the proxy models, recall and precision targets and failure probability parameters, studying the `sem_filter` performance on the fact-checking task. Figure 7a and Figure 7b demonstrate the average accuracy observed over 20 trials for varied recall and precision targets. We evaluate performance with both the Llama-8B proxy (solid lines) and a weaker proxy, TinyLlama-1B (dashed lines), with a failure probability of $\delta = 0.2$ (red) and $\delta = 0.4$ (blue). As expected, the average recall and precision values increase with higher target values for either metric. Figure 7c shows that decreasing the recall and precision target tends to reduce the number of oracle calls. As expected, at a fixed accuracy target, our approximation requires more oracle calls with the weaker proxy, TinyLlama. We also note that the cost, in number of oracle calls or latency, may not necessarily increase linearly with accuracy target, as the figure shows. Intuitively, the number of additional oracle calls required to achieve an ϵ increase in accuracy will depend on the target accuracy range. Finally, in Figure 7d we record the percentage of 50 trials that do not meet the recall and precision target, both set to 0.9, using the TinyLlama proxy. The results confirm the observed failure probabilities are lower than the configured failure probabilities, as expected due to our conservative implementation.

6 RELATED WORK

Data Systems with Batched Inference Primitives. Many prior works support a limited set of simple, batched inference primitives. First, AI UDF systems [1, 8, 14, 49, 53, 67] provide a low-level, non-declarative programming interface supporting batched inference LLM calls. These systems also integrate vector search, equivalent to our `sem_search` or `sem_sim_join`, and can combine AI UDFs with non-LLM operations (e.g., average, count). Alternatively, several recent works support domain-specific batched-inference operations [18, 46, 47, 50] for various tasks, including data cleaning, extract-transform-load (ETL) and conversational agents. In contrast to these prior works, semantic operators provide a *general-purpose query model* that goes beyond simple batched inference primitives to support semantic operations that require more complex *AI algorithms* (e.g., joins, aggregations, and ranking). Our detailed evaluation (Section 5) demonstrates the performance limitations of batched-inference LLM execution models.

Best-Effort LLM-Based Analytics Systems. Several recent systems [17, 25, 61] go beyond batched inference LLM primitives but provide no accuracy guarantees for their execution. Following a preprint [54] of this work, DocETL [61] proposes to use LLM agents as the query optimizer over LLM-powered operators. This is a



(a) Observed vs Target Recall

(b) Observed vs Target Precision

(c) # Oracle Calls vs Accuracy Target

(d) Observed vs Configured Failure Probability

Figure 7: We evaluate statistical accuracy guarantees with our semantic filter for fact-checking on the FEVER dataset. We show (a) average observed recall vs. configured recall targets (γ_R), (b) average observed precision vs. configured precision targets (γ_P), and (c) the number of oracle LLM calls vs. configured accuracy targets ($\gamma_R = \gamma_P$), using a Llama-70B oracle and either a Llama-8B proxy (dashed lines) or a TinyLlama proxy (solid lines), for two different failure probabilities ($\delta = 0.2, 0.4$). We also show observed vs. configured failure probabilities using the TinyLlama proxy and Llama-70B oracle, for accuracy targets $\gamma_R = \gamma_P = 0.9$.

promising direction for future work; however, our evaluation (Section 5) demonstrates that DocETL's agentic approach leads to high variance in performance, no accuracy guarantees, and requires multiple reruns to achieve comparable accuracy to ours, with approximately 1 in 3 runs failing due to optimizer errors, indicating that more work is needed to make these methods robust. UQE [25] also studies optimizations for LLM-powered operators, proposing an embedding-based filter approximation. In contrast to our methods, UQE provides best-effort performance with no accuracy guarantees. UQE also studies optimizations for *non-LLM aggregations* (e.g., average, count) combined with LLM-based filters and group-bys using stratified sampling, which is complementary to this work.

General ML-based Query Processing. Prior works study the use of non-LLM machine learning (ML) in databases. MADLib [32] extends SQL with abstractions for descriptive statistics and machine learning (e.g., regression and classification). NoScope [36], TASTI [40], SUPG [37], BlazeIt [35] and probabilistic predicates [51] propose methods to optimize queries involving expensive ML-based predicates (e.g., using object detectors) over large datasets, typically in video analytics. In contrast, our work integrates LLMs, which motivates our new formalisms for language-based operators and novel optimizations. Some optimizations proposed in these prior works, such as model cascades and predicate re-ordering, are also useful for optimizing LOTUS pipelines with language models.

Table Question Answering. A large body of work, including retrieval-augmented generation [44] and Text2SQL [70, 71, 73, 75], serves question-answering via natural language over databases. Typically, the LLM system invokes external tools, e.g., search APIs or SQL programs. Interestingly, these agentic workflows can instead invoke semantic operator programs as the underlying data-processing API. Recent work [19] demonstrates the promise of this approach to outperform baseline TableQA methods.

Adoption of Semantic Operators. Based on a preprint [54] of this work, Google recently implemented experimental semantic operators in BigQuery Dataframes [12], including `sem_map`, `sem_filter`, `sem_join`, `sem_agg`, `sem_topk`, `sem_search`, and `sem_sim_join`.

7 LIMITATIONS AND FUTURE WORK

Semantic operators open up exciting research directions for future work. We briefly outline the limitations of this work and compelling opportunities of future work towards robust, AI-enabled systems for semantic bulk processing. Firstly, the accuracy guarantees in this work are limited to individual semantic operators. Future work should build on the formalism introduced in this work to develop *end-to-end* accuracy guarantees for semantic operator queries. In this setting, the user could configure an accuracy target for the end-to-end semantic query, and the query optimizer will assign an error budget to each individual operator. Additionally, while this work explores simple cost-based optimizations for semantic operators, more work is needed to develop algorithms that tractably search over many possible execution plans employing various proxies (e.g., cheaper AI models, vector indexes, traditional indexes and operators, or code-generation methods), while rigorously considering the relative cost of each. Moreover, developing a framework of equivalence rules for semantic operator execution plans is an interesting direction towards principled optimizations. Lastly, while this work presents several high-quality reference algorithms for semantic operators, developing better reference algorithms remains an exciting open research question.

8 CONCLUSION

In this work, we proposed semantic operators to provide the first formalism with statistical accuracy guarantees for general-purpose, AI-based operations with natural language parameters. Our results across diverse applications, including fact-checking, biomedical multi-label classification, search, and topic analysis, demonstrate the generality, expressiveness and robustness of the semantic operator model as well as our optimization approach. For each task, we find that LOTUS programs capture state-of-the-art AI applications with low development overhead, match or exceed result quality of recent LLM-powered analytics systems, and substantially reduce cost, while ensuring accuracy guarantees. Since open-sourcing LOTUS, we have seen a growing user-base, as well as adoption of semantic operators among large database vendors. We believe both represent exciting steps towards powerful data systems that integrate AI reasoning capabilities over vast knowledge corpora.

ACKNOWLEDGMENTS

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project, including Meta, Google, and VMware, as well as Cisco, SAP, and a Sloan Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] [n.d.]. AI Functions on Databricks. https://docs.databricks.com
[2] [n.d.]. arXiv.org ePrint archive. https://arxiv.org
[3] [n.d.]. Custom Search JSON API | Programmable Search Engine. https://developers.google.com/custom-search/v1/overview
[4] [n.d.]. Discovery Insight Platform. https://www.findourview.com
[5] [n.d.]. GAIR-NLP/factool: FacTool: Factuality Detection in Generative AI. https://github.com/GAIR-NLP/factool
[6] [n.d.]. Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/
[7] [n.d.]. LangChain. https://www.langchain.com/
[8] [n.d.]. Large Language Model (LLM) Functions (Snowflake Cortex) | Snowflake Documentation. https://docs.snowflake.com/user-guide/snowflake-cortex/llm-functions
[9] [n.d.]. LLM with Vertex AI only using SQL queries in BigQuery. https://cloud.google.com/blog/products/ai-machine-learning/llm-with-vertex-ai-only-using-sql-queries-in-bigquery
[10] [n.d.]. OpenAI Platform. https://platform.openai.com
[11] [n.d.]. pandas - Python Data Analysis Library. https://pandas.pydata.org/
[12] [n.d.]. python-bigquery-dataframes/notebooks/experimental/semantic_operators.ipynb at main · googleapis/python-bigquery-dataframes. https://github.com/googleapis/python-bigquery-dataframes/blob/main/notebooks/experimental/semantic_operators.ipynb
[13] [n.d.]. Querying - LlamaIndex 0.9.11.post1. https://docs.llamaindex.ai/en/stable/understanding/querying/querying.html
[14] 2023. Large Language Models for sentiment analysis with Amazon Redshift ML (Preview) | AWS Big Data Blog. https://aws.amazon.com/blogs/big-data/large-language-models-for-sentiment-analysis-with-amazon-redshift-ml-preview/ Section: Amazon Redshift.
[15] 2024. mixedbread-ai/mxbai-rerank-large-v1 · Hugging Face. https://huggingface.co/mixedbread-ai/mxbai-rerank-large-v1
[16] Griffin Adams, Alexander Fabbri, Faisal Ladhak, Eric Lehman, and Noémie Elhadad. 2023. From Sparse to Dense: GPT-4 Summarization with Chain of Density Prompting. http://arxiv.org/abs/2309.04269 arXiv:2309.04269 [cs].
[17] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A. Shah, Benjamin Sowell, Dan Tecuci, Vinayak Thapliyal, and Matt Welsh. 2024. The Design of an LLM-powered Unstructured Analytics System. https://doi.org/10.48550/arXiv.2409.00847 arXiv:2409.00847 [cs].
[18] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. http://arxiv.org/abs/2304.09433 arXiv:2304.09433 [cs].
[19] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2SQL is Not Enough: Unifying AI and Databases with TAG. https://doi.org/10.48550/arXiv.2408.14717 arXiv:2408.14717 [cs].
[20] Mark Braverman and Elchanan Mossel. [n.d.]. Noisy sorting without resampling. ([n. d.]).
[21] Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. 2024. BooookScore: A systematic exploration of book-length summarization in the era of LLMs. http://arxiv.org/abs/2310.00785 arXiv:2310.00785 [cs].
[22] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. http://arxiv.org/abs/2305.05176 arXiv:2305.05176 [cs].
[23] I.-Chun Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, and Pengfei Liu. 2023. FacTool: Factuality Detection in Generative AI – A Tool Augmented Framework for Multi-Task and Multi-Domain Scenarios. https://doi.org/10.48550/arXiv.2307.13528 arXiv:2307.13528 [cs].
[24] E F Codd. 1970. A Relational Model of Data for Large Shared Data Banks. 13, 6 (1970).
[25] Hanjun Dai, Bethany Yixin Wang, Xingchen Wan, Bo Dai, Sherry Yang, Azade Nova, Pengcheng Yin, Phitchaya Mangpo Phothilimthana, Charles Sutton, and Dale Schuurmans. 2024. UQE: A Query Engine for Unstructured Databases. https://doi.org/10.48550/arXiv.2407.09522 arXiv:2407.09522 [cs].
[26] Sanjoy Dasgupta. [n.d.]. The hardness of k-means clustering. ([n. d.]).
[27] Shrey Desai and Greg Durrett. 2020. Calibration of Pre-trained Transformers. https://arxiv.org/abs/2003.07892v3
[28] Karel D'Oosterlinck, Omar Khattab, François Remy, Thomas Demeester, Chris Develder, and Christopher Potts. 2024. In-Context Learning for Extreme Multi-Label Classification. https://doi.org/10.48550/arXiv.2401.12178 arXiv:2401.12178 [cs].
[29] Karel D'Oosterlinck, François Remy, Johannes Deleu, Thomas Demeester, Chris Develder, Klim Zaporojets, Aneiss Ghodsi, Simon Ellershaw, Jack Collins, and Christopher Potts. 2023. BioDEX: Large-Scale Biomedical Adverse Drug Event Extraction for Real-World Pharmacovigilance. https://doi.org/10.48550/arXiv.2305.13395 arXiv:2305.13395 [cs].
[30] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. https://doi.org/10.48550/arXiv.2401.08281 arXiv:2401.08281 [cs].
[31] Andrew Drozdov, Honglei Zhuang, Zhuyun Dai, Zhen Qin, Razieh Rahimi, Xuanhui Wang, Dana Alon, Mohit Iyyer, Andrew McCallum, Donald Metzler, and Kai Hui. 2023. PaRaDe: Passage Ranking using Demonstrations with Large Language Models. https://doi.org/10.48550/arXiv.2310.14408 arXiv:2310.14408 [cs].
[32] Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library or MAD Skills, the SQL. http://arxiv.org/abs/1208.4165 arXiv:1208.4165 [cs].
[33] C. A. R. Hoare. 1961. Algorithm 65: find. *Commun. ACM* 4, 7 (July 1961), 321–322. https://doi.org/10.1145/366622.366647
[34] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. http://arxiv.org/abs/1702.08734 arXiv:1702.08734 [cs].
[35] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. http://arxiv.org/abs/1805.01046 arXiv:1805.01046 [cs].
[36] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. http://arxiv.org/abs/1703.02529 arXiv:1703.02529 [cs].
[37] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate selection with guarantees using proxies. *Proceedings of the VLDB Endowment* 13, 12 (Aug. 2020), 1990–2003. https://doi.org/10.14778/3407790.3407804
[38] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. Approximate Selection with Guarantees using Proxies. https://doi.org/10.48550/arXiv.2004.00827 arXiv:2004.00827 [cs].
[39] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating approximate aggregation queries with expensive predicates. *Proceedings of the VLDB Endowment* 14, 11 (July 2021), 2341–2354. https://doi.org/10.14778/3476249.3476285
[40] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. [n.d.]. Task-agnostic Indexes for Deep Learning-based Queries over Unstructured Data. ([n. d.]).
[41] 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. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. https://arxiv.org/abs/2310.03714v1
[42] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. https://arxiv.org/abs/2004.12832v2
[43] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. https://arxiv.org/abs/2309.06180v1
[44] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. https://doi.org/10.48550/arXiv.2005.11401 arXiv:2005.11401 [cs].
[45] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic Evaluation of Language Models. https://arxiv.org/abs/2211.09110v2
[46] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G. Parameswaran, and Eugene Wu. 2024. Towards Accurate and Efficient Document Analytics with Large Language Models. http://arxiv.org/abs/2405.04674 arXiv:2405.04674 [cs].
[47] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. http://arxiv.org/abs/2405.14696 arXiv:2405.14696 [cs].
[48] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. https://doi.org/10.48550/arXiv.2307.03172 arXiv:2307.03172 [cs].
[49] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E. Gonzalez, Ion Stoica, and Matei Zaharia. 2024. Optimizing LLM Queries in Relational

Workloads. http://arxiv.org/abs/2403.05821 arXiv:2403.05821 [cs].

[50] Shicheng Liu, Jialiang Xu, Wesley Tjangnaka, Sina J. Semnani, Chen Jie Yu, and Monica S. Lam. 2024. SUQL: Conversational Search over Structured and Unstructured Data with Large Language Models. https://doi.org/10.48550/arXiv.2311.09818 arXiv:2311.09818 [cs].

[51] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1493–1508. https://doi.org/10.1145/3183713.3183751

[52] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-Shot Listwise Document Reranking with a Large Language Model. https://arxiv.org/abs/2305.02156v1

[53] MotherDuck. [n.d.]. Introducing the prompt() Function: Use the Power of LLMs with SQL! - MotherDuck Blog. https://motherduck.com/blog/sql-llm-prompt-function-gpt-models/

[54] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. https://doi.org/10.48550/arXiv.2407.11418 arXiv:2407.11418 [cs] version: 1.

[55] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data* 2, 3 (May 2024), 120:1–120:27. https://doi.org/10.1145/3654923

[56] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. http://arxiv.org/abs/2309.15088 arXiv:2309.15088 [cs].

[57] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! http://arxiv.org/abs/2312.02724 arXiv:2312.02724 [cs].

[58] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2024. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. https://doi.org/10.48550/arXiv.2306.17563 arXiv:2306.17563 [cs].

[59] Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving Passage Retrieval with Zero-Shot Question Generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 3781–3797. https://doi.org/10.18653/v1/2022.emnlp-main.249

[60] Nihar B. Shah and Martin J. Wainwright. 2016. Simple, Robust and Optimal Ranking from Pairwise Comparisons. http://arxiv.org/abs/1512.08949 arXiv:1512.08949 [cs, math, stat].

[61] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. https://doi.org/10.48550/arXiv.2410.12189 arXiv:2410.12189 [cs].

[62] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. https://arxiv.org/abs/2304.09542v2

[63] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models. https://doi.org/10.48550/arXiv.2104.08663 arXiv:2104.08663 [cs].

[64] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a Large-scale Dataset for Fact Extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Marilyn Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, New Orleans, Louisiana, 809–819. https://doi.org/10.18653/v1/N18-1074

[65] P. Viola and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1. IEEE Comput. Soc, Kauai, HI, USA, I–511–I–518. https://doi.org/10.1109/CVPR.2001.990517

[66] 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. http://arxiv.org/abs/2212.03533 arXiv:2212.03533 [cs].

[67] WilliamDAssafMSFT. 2024. Intelligent Applications - Azure SQL Database. https://learn.microsoft.com/en-us/azure/azure-sql/database/ai-artificial-intelligence-intelligent-applications?view=azuresql

[68] Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively Summarizing Books with Human Feedback. https://doi.org/10.48550/arXiv.2109.10862 arXiv:2109.10862 [cs].

[69] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases. https://arxiv.org/abs/2404.13207v2

[70] Navid Yaghmazadeh, View Profile, Yuepeng Wang, View Profile, Isil Dillig, View Profile, Thomas Dillig, and View Profile. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (Oct. 2017), 1–26. https://doi.org/10.1145/3133887 Publisher: Association for Computing Machinery.

[71] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation. https://doi.org/10.48550/arXiv.1804.09769 arXiv:1804.09769 [cs].

[72] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2024. Large Language Model Cascades with Mixture of Thoughts Representations for Cost-efficient Reasoning. http://arxiv.org/abs/2310.03094 arXiv:2310.03094 [cs].

[73] John M Zelle and Raymond J Mooney. 1996. 1996-Learning to Parse Database Queries Using Inductive Logic Programming. (1996).

[74] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? https://doi.org/10.48550/arXiv.1905.07830 arXiv:1905.07830 [cs].

[75] Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. https://doi.org/10.48550/arXiv.2403.02951 arXiv:2403.02951 [cs].

[76] Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Bendersky. 2024. Beyond Yes and No: Improving Zero-Shot LLM Rankers via Scoring Fine-Grained Relevance Labels. http://arxiv.org/abs/2310.14122 arXiv:2310.14122 [cs].