Unstructured Data Analysis Using LLMs: A Comprehensive Benchmark

Qiyan Deng*
Beijing Institute of Technology
Beijing, China
qiyandeng@bit.edu.cn

Jianhui Li*
Beijing Institute of Technology
Beijing, China
lijianhuibit@bit.edu.cn

Chengliang Chai†
Beijing Institute of Technology
Beijing, China
ccl@bit.edu.cn

Ye Yuan†
Beijing Institute of Technology
Beijing, China
yuan-ye@bit.edu.cn

Jinqi Liu
Beijing Institute of Technology
Beijing, China
ljqbit@outlook.com

Junzhi She
Beijing Institute of Technology
Beijing, China
sjzbit2025@outlook.com

Kaisen Jin
Beijing Institute of Technology
Beijing, China
jks@bit.edu.cn

Zhaoze Sun
Beijing Institute of Technology
Beijing, China
sunzhaoze@bit.edu.cn

Yuhao Deng
Beijing Institute of Technology
Beijing, China
dyh18@bit.edu.cn

Jia Yuan
University of Arizona
Tucson, AZ, USA
jiayuan@arizona.edu

Yuping Wang
Beijing Institute of Technology
Beijing, China
wyp_cs@bit.edu.cn

Xu Zhou
Hunan University
Hunan, China
zhxu@hnu.edu.cn

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggr@bit.edu.cn

Lei Cao
University of Arizona
Tucson, AZ, USA
lcao@csail.mit.edu

ABSTRACT

The explosion of unstructured data has immense analytical value. By leveraging large language models (LLMs) to extract table-like attributes from unstructured data, researchers are building LLM-powered systems that analyze documents as if querying a database. These unstructured data analysis (UDA) systems differ widely in query interfaces, optimization, and operators, making it unclear which works best in which scenario. However, no benchmark currently offers high-quality, large-scale, diverse datasets and rich query workloads to rigorously evaluate them. We present Bench-U, a comprehensive UDA benchmark that addresses this need. We curate 6 datasets from different domains and manually construct a relational database view for each using 30 graduate students. These relational databases serve as ground truth to evaluate any UDA system, regardless of its interface. We further design diverse queries over the database schema that evaluate various analytical operators with different selectivities and complexities. Using this benchmark, we conduct an in-depth analysis of key UDA components—query interface, optimization, operator design, and data processing—and run exhaustive experiments to evaluate systems and techniques along these dimensions. Our main contributions are: (1) a comprehensive benchmark for rigorous UDA evaluation, and (2) a deeper understanding of the strengths and limitations of current systems, paving the way for future work in UDA.

PVLDB Reference Format:
Qiyan Deng, Jianhui Li, Chengliang Chai, Ye Yuan, Jinqi Liu, Junzhi She, Kaisen Jin, Zhaoze Sun, Yuhao Deng, Jia Yuan, Yuping Wang, Xu Zhou, Guoren Wang, and Lei Cao. Unstructured Data Analysis Using LLMs: A Comprehensive Benchmark. PVLDB, 19(9): 2398 - 2410, 2026.
doi:10.14778/3819518.3819559

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://bit-datalab.github.io/bench-u-page.

1 INTRODUCTION

Modern organizations store a large quantity of unstructured data such as clinical notes, legal contracts, financial reports, etc., which account for 80%-90% of global data based on IDC research [13]. These vast repositories of unstructured data, if analyzed appropriately, have immense value in various domains.

*Qiyan Deng and Jianhui Li are co-first authors.
†Chengliang Chai and Ye Yuan are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819559

As an example, healthcare providers often manage a corpus of hundreds of thousands of heterogeneous medical documents, including disease documents (detailing etiology, symptomatology, and progression patterns), drug documents (detailing indications, mechanisms of action, and contraindications) and documents of medical institutions. If there were a powerful unstructured data analysis (UDA) system, it could enable a provider to easily assist a patient who, for example, has symptoms of frothy urine and dull flank pain, by running queries to identify possible diseases and recommend public hospitals within certain distances of the patient's home that specialize in these diseases.

LLM-powered Unstructured Data Analysis. To support such needs, the database community is actively developing LLM-based systems for UDA [10, 17, 18, 21, 26]. These systems harness LLMs to generate information from multi-modal data lakes (encompassing text, images, etc.) and perform analytical operations such as filtering, aggregation, and join, thanks to the ever growing semantic comprehension and reasoning capabilities of LLMs.

Although these systems use different query interfaces, e.g., Python or SQL queries, they are all declarative systems, which, similar to relational databases, offer a set of logical operators for users to write a simple program to analyze data. These systems provide optimized implementation of these operators to ensure accuracy and reduce LLM cost. Typically, these systems feature a query optimizer that automatically transforms a user program to an optimized query execution plan, with optimizations such as ordering the filters, converting joins to filters, selecting an appropriate LLM for the task, etc. In this way, these systems abstract away time-consuming engineering details in UDA, while transparently optimizing accuracy and addressing the performance and scaling bottleneck of LLMs.

The Need For a Comprehensive Benchmark. Different systems have distinct implementations of operators and query optimization strategies. Furthermore, evaluations of these systems are small and do not cover diverse scenarios. As a result, it remains unclear which system works best in which scenario. For example, Palimpzest [18] demonstrates its cost optimization capabilities over a dataset of 1000 short emails. However, the short emails can not expose the system challenges regarding text chunking. DocETL [26] evaluates its multi-agent architecture using a single query over the CUAD dataset, which contains 510 lengthy legal contracts. However, the evaluation for DocETL completely ignores the necessity of token cost optimization. SemBench has a distinct and valuable goal: benchmarking semantic query processing over a mix of structured and unstructured data. However, SemBench does not target thoroughly evaluating the capability of the LLM-powered data systems in processing long, complex unstructured documents that raise significant challenges in cost and accuracy. Therefore, a comprehensive benchmark is still in desperate need to standardize the evaluation of LLM-powered UDA systems and guide future research in this field.

Design Goals. To fill this gap, in this work we propose to construct a benchmark guided by the following design goals:

(1) Realistic Large-scale Unstructured Data. To reflect real-world scenarios and evaluate system scalability and cost efficiency, the benchmark should include large-scale collections of unstructured documents. These documents should be not only numerous but also long-form (e.g., 10K+ tokens), and may contain substantial noise and irrelevant information.

(2) Dataset & Query Workload Diversity for Optimization. A benchmark that enables comprehensive evaluation of UDA systems must offers datasets and query workloads with diverse properties. Important dataset properties include domain diversity, data modality (e.g., text, images, tables), and document structure. Meanwhile, query workloads should cover diverse analytical operators (e.g., filter, join, aggregation), varied operator compositions and selectivities, and attribute combinations across modalities.

(3) High-quality Ground Truth for Fair Evaluation. To enable fair comparison across heterogeneous systems, a benchmark for unstructured data analysis should provide precise and high-quality labels as ground truth. Such ground truth enables consistent and reliable evaluation across systems with different interfaces and execution strategies.

(4) Easy to Evaluate Existing Systems. To comprehend the advantages/disadvantages of existing UDA systems and expose research opportunities, we have to implement them reliably, perform a thorough evaluation, and conduct a detailed analysis of the results.

Our Proposal. We construct Bench-U for unstructured data analysis that meets all the above goals.

Key Insight: A Relational Database w.r.t. Multi-modal Data Lake. To develop such a benchmark, our key insight is that constructing a relational database from the corresponding multi-modal data lake is sufficient to cover the evaluation needs of all existing systems, including accuracy, latency, and cost.

Example 1. *Still using the medical application as an example, given a category of files, e.g., disease documents, we can define a relational schema (i.e., a number of meaningful attributes such as disease name, etiology, symptoms, etc.) in advance. Then we extract the values of all attributes from the data lake as the ground truth. In this way, we obtain a relational database consisting of multiple tables, each corresponding to a specific category of files (i.e., disease, drug, medical institution, etc.). Consequently, any analytical queries concerning the aforementioned attributes can be evaluated based on corresponding data records in the constructed database, agnostic to their query semantics and execution strategies. For example, although the semantic operators in Lotus prefer natural language input, it can still compose queries about the attributes covered by the database to evaluate their implementation.*

This insight guides us to construct Bench-U that meets all the above goals.

To meet the first design goal, we collect **six** sets of unstructured documents, as shown in Table 2. In terms of the number of documents, all the datasets have at least hundreds of documents and, in particular, Healthcare has 100,000 long and complex documents, which is **100**× more than the existing benchmarks. In terms of the lengths of the documents, three of the datasets have an average length of more than 10,000 tokens. In particular, on Finance dataset, the length of the documents can be up to 838,418 tokens (≈100 pages). Effectively processing a large number of long documents that potentially incorporate much noisy information is prohibitively expensive and difficult. This poses significant challenges for document chunking, chunk retrieval, and filtering, which are critical to the performance of UDA systems.

For the second design goal of ensuring the diversity of the datasets and query workloads for effective optimization, we construct datasets from **six** representative domains, including healthcare, law, art, sports, science, and finance. Among them, art and science include images and text, while science additionally includes tables. We construct a workload of **608** queries (**10**× more than prior benchmarks), spanning five categories: `Select`, `Filter`, `Join`, `Agg`, and `Mixed` (other complex queries that are combinations of at least 3 operator types). These queries differ in multiple aspects, including different number of filters, various selectivities, binary or multiple joins, w/ or w/o aggregation operations, etc., forming a rich optimization space that enables systematic evaluation of logical and physical optimization techniques such as filter reordering, join planning, and model selection.

For the third goal of fair evaluation, we define **175** meaningful attributes derived from the six datasets, including categorical, numerical, and string types, exceeding existing benchmarks by **5**×. These attributes, represented in structured tables, lay the foundation for Bench-U to design a systematic evaluation framework that account for the uncertainty property of UDA systems, i.e., LLMs often deterministically producing output in diverse formats, thus ensuring fair comparison To ensure the reliability of this structured ground truth, a team of 30 graduate students performs extensive manual annotation, spending in total 4,110 hours (approximately 0.5 hours per document). The annotation follows carefully designed guidelines and is further validated through LLM-assisted cross-validation during the process. Finally, we quantify annotation consistency using inter-annotator agreement.

To achieve the last goal of easy evaluation, we have conducted extensive experiments to evaluate these systems on all datasets, measuring accuracy, cost, and latency across fine-grained query categories to capture differences in system performance. Furthermore, we analyze the experiment results from four perspectives: query interface, query optimization, operator design, and data processing, which are the key building blocks of such systems. This in depth, multi-faceted analysis offer actionable insights for future research.

Contributions. We make the following contributions:

(1) We present Bench-U, a benchmark for unstructured data analysis that includes large-scale and diverse datasets, as well as a rich set of queries; to the best of our knowledge, it is the first work that constructs a comprehensive benchmark and thoroughly evaluates existing LLM-powered UDA systems.

(2) We collect six sets of unstructured data from diverse domains, including more than 100,000 documents. We construct 608 queries that involve various analytical operators over the diverse attributes defined in the database.

(3) We provide comprehensive and cross-validated relational tables as ground truth, annotated by 30 graduate students over 4,110 hours, enabling objective and reproducible evaluation on UDA systems.

(4) We thoroughly evaluate seven representative UDA systems in accuracy, cost, and latency. Our in-depth analysis offers insights to guide future research in this field.

2 UNSTRUCTURED DATA ANALYSIS SYSTEMS

We present the architecture of a typical LLM-powered UDA system, which typically consists of 4 key modules: *the query interface*,

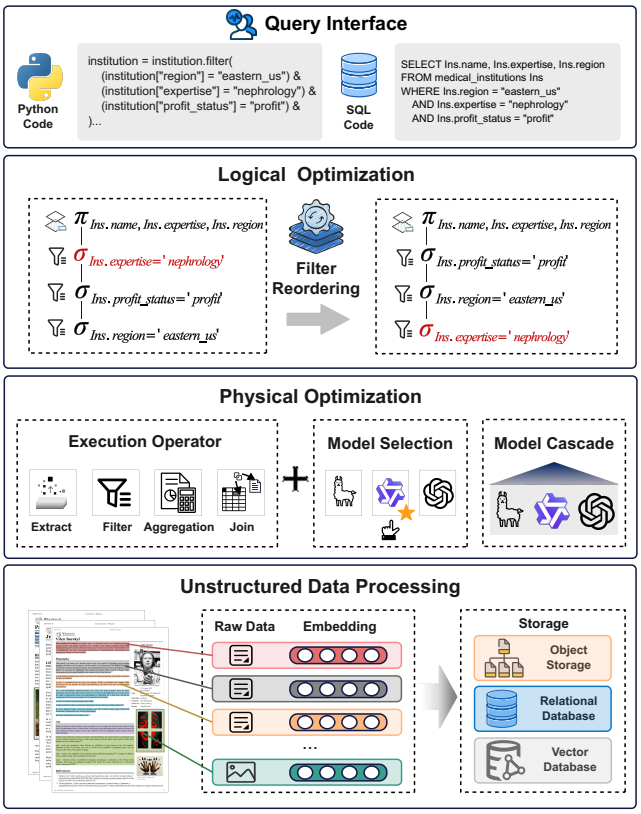

Figure 1: Architecture of Unstructured Data Analysis System.

logical optimization layer, *physical optimization layer*, and *data processing layer*. As shown in Fig. 1, a user submits a query via the interface, which describes the analytical task. The system parses the query into some analytical operators, such as `Extract`, `Filter`, `Join`, and `Aggregation`. Then, the logical optimization layer determines a logical query plan, such as pushing down predicates or ordering the operators. Then the physical optimization layer selects the appropriate physical implement w.r.t. each operator in the logical plan, producing an optimized physical execution plan. The data processing layer optimizes the data layout, supporting the aforementioned optimizations. In addition to storing the raw data in distributed storage systems (e.g., Amazon S3), it often segments documents into chunks which are subsequently transformed into embeddings and loaded into a vector database. This enables accurate and efficient retrieval to identify information relevant to a query, speeding up query execution and reducing cost.

Optimization Goals. Unlike traditional databases that focus mainly on optimizing latency, LLM-powered UDA systems have multiple optimization goals, i.e., *accuracy*, *cost*, and *latency*. Accuracy is critical in such systems for two reasons. (1) LLMs are prone to hallucinations, leading to potential inference errors; and (2) to reduce costs, a common practice is to only feed the LLM document chunks highly relevant to the query, rather than entire documents. However, if the retrieval misses relevant chunks, it will degrade the analysis accuracy. On the other hand, the LLM cost and inference

latency largely rely on the number of input and output tokens. Reducing them is typically achieved by minimizing the number of input tokens. However, the system still has to ensure that the LLM gets sufficient information to ensure the quality of the analysis.

We detail how existing systems use four modules to achieve the stated goals. Table 1 shows the modules of each system.

System	Data Processing			Operator				Query Optimization	
	Chunking	Embedding	Multi-modal	Extract	Filter	Join	Agg	Logical	Physical
Evaporate	✗	✗	✗	✓	✗	✗	✗	✗	✗
Palimpzest	✗	✗	✓	✓	✓	✓	✓	✓	✓
LOTUS	✗	✓	✓	✓	✓	✓	✓	✗	✓
DocETL	✓	✓	✗	✓	✓	✓	✓	✓	✓
ZenDB	✓	✗	✗	✓	✓	✓	✗	✓	✗
QUEST	✓	✓	✗	✓	✓	✓	✗	✓	✗
UQE	✗	✓	✓	✓	✓	✗	✓	✓	✗

Table 1: Overview of Existing UDA Systems.

2.1 Data Processing

The data processing component handles multimodal inputs (e.g., text and images) extracted from complex documents such as PDFs via OCR, and organizes them into different representations to support downstream analysis.

LOTUS [21], UQE [10], and DocETL [26] represent documents as semi-structured records (CSV or JSON), where each entry corresponds to a document. In CSV, text and image file paths are stored in separate columns, while JSON represents each document as an object with text and image fields. Subsequently, Palimpzest and UQE encode text into embeddings, while UQE further encodes images, storing these representations with the data. ZenDB [17] and QUEST [27] process plain text in relational databases and build indexes to support efficient retrieval and query analysis. ZenDB organizes documents into hierarchical semantic units, while QUEST constructs a two-level (document- and chunk-level) vector index by encoding summaries and chunks into embeddings to locate relevant content and reduce LLM usage cost. More details can be found in the technical report [7]. Evaporate [2] targets plain text stored in a folder for subsequent analysis. Palimpzest [18] organizes the plain text and images in each document into a directory. Both do not support indexing.

2.2 Query Interface & Operators

Query Interface. Each system provides a query interface for users to define analytical tasks. UQE, ZenDB, and QUEST use SQL-like language to support analysis over unstructured data. For example, in the Healthcare dataset, to find private institutions specializing in nephrology and located in the eastern United States, a user can write a query shown in Figure 1. DocETL, Evaporate, LOTUS and Palimpzest offer declarative Python APIs, corresponding to logical operators in relational databases, for users to compose a query as a Python program, as shown in Fig. 1.

Operators. Bench-U supports 4 common analytical operators.

Extract generates attributes from a set of documents D, formally defined as $\texttt{Extract}(D, A)$, where A specifies the attributes to be extracted from D and the output is a relational table T_D. For example, to extract the rating and locations of institutions, this operation can be expressed as Extract (Healthcare, [rating, location]). In an SQL-like interface, this corresponds to SELECT rating, location FROM Healthcare", while in the Python API, it could be written as *pz.addcolumns(*Healthcare*, [rating, location])"*. In addition, users can provide attribute descriptions as prompts, helping LLMs produce accurate answers. Evaporate employs LLMs to generate code to extract each attribute. Other systems feed the attribute (with optional user descriptions) and relevant document chunks (possibly the entire document) to LLMs for extraction.

Filter. Given a condition C, Filter operation selects a subset of documents that satisfy C from D, denoted by $\texttt{Filter}(D, C)$. A filter on documents D evaluates whether the relevant attributes of each document satisfy condition C. For example, *"lotus.sem_filter('the {document} satisfy {profit_status} is profit')"* is a filter in LOTUS. Existing systems adopt two strategies to implement such filters: (1) Palimpzest, QUEST, and ZenDB extract profit_status from each document and then evaluate whether it satisfies the filter; and (2) LOTUS, DocETL and UQE take the filter as part of prompts and leverage LLMs to determine whether the filter condition is satisfied.

Join is a cross-document operation, i.e., combining information from two sets of documents based on a join condition a, formally defined as $\texttt{Join}(D_1, D_2, a)$. D_1 and D_2 are two subsets of documents, and a is the join attribute. If the system extracts two tables (both contain the attribute a), the tables can be joined on a. For example, the system can extract a disease table from the disease subset and a medication table from the medication subset, which can be joined by disease name. This join allows users to identify diseases from symptoms and retrieve corresponding treatments. ZenDB and QUEST implement the Join by first extracting the disease table and the medication table, respectively, from two sets of documents and joining the two tables. On the other hand, LOTUS extracts the disease table, embeds the join key values, and uses them to retrieve relevant medication documents. It then extracts the medication table from this subset and joins the two tables to produce the result.

Aggregation is defined as $\texttt{Agg}(D, a, F)$, where a specifies the grouping attribute and F defines the aggregation functions to apply, such as Count, Sum, Avg, Min or Max. This operator supports analytical tasks like "computing the number of institutions grouped by expertise", i.e., Agg (Healthcare, 'expertise', Count), which can be represented as "*pz.GroupBySig(Healthcare, Count,'expertise')*" using the Python API of PALIMPZEST. In implementation, Evaporate, ZenDB, QUEST and Palimpzest extract the 'expertise' from all documents, group the values in a table, and then perform aggregation on the grouped data. LOTUS, UQE, and DocETL, on the other hand, first preprocess the documents by clustering them based on their embeddings, and then perform aggregation or batch inference within each cluster as an approximation to save costs. Besides, UQE supports grouping images according to their embeddings.

2.3 Logical Optimization

Given a user-specified query involving multiple operators, UDA systems generate an optimized logical plan to reduce LLMs costs and query latency. The optimizations adopted by existing systems mainly include filter reordering, filter pushdown, and join ordering.

Filter Reordering. Consider a query that selects artists and their birthdates with two filters, i.e., F_1 =filter(D, "lifespan is

less than 35") and F_2 =filter(D, "tone is warm")). Different systems employ different optimization strategies for reordering filters to reduce costs.

(1) Selectivity-only Strategy. Palimpzest and UQE adopt a selectivity-only filter reordering strategy that prioritizes a filter with low selectivity. This indicates that the attribute values that a document contains have a small probability to satisfy the predicates of this filter. This reduces the chance of extracting other attributes and evaluating the corresponding filters. These systems estimate the selectivity (denoted by $sel()$) by sampling. For example, if $sel(F_1) = 0.2$ and $sel(F_2) = 0.1$, applying F_2 first would leave ≈10% documents for F_1 to evaluate, potentially reducing the cost. However, only considering the selectivities tends to be suboptimal in minimizing the cost. For example, although $sel(F_2) < sel(F_1)$, if the document chunks that F_2 has to examine contain much more tokens than those of F_1, using this order might lead to higher costs than applying F_1 first.

(2) Selectivity-cost strategy. To address the above limitation, ZenDB ranks filters based on scores computed by $sel(F) \times cost(F)$ and prioritizes those with lower scores. Given the attribute a of a filter F and an unstructured document $d \in D$, it uses $d[a]$ to denote the chunk(s) that are highly relevant to a. The relevance can be computed by measuring the similarity between the embeddings of chunks and the attribute. Let $c(d[a])$ denote the number of tokens of $d[a]$. Then in ZenDB, $cost(F)$ corresponds to the average number of tokens of chunks relevant to a across all documents, i.e., $cost(F) = \frac{\sum_{d \in D} c(d[a])}{|D|}$. Therefore, it produces one single filter order w.r.t. all documents, similar to traditional databases. This is suboptimal, as different documents might contain different numbers of tokens w.r.t. an attribute. QUEST instead produces different orders for different documents considering both their selectivities and costs, Therefore, QUEST does not compute $cost(F)$ w.r.t. the whole document set D. Instead, for each document $d \in D$, it uses $cost_d(F) = c(d[a])$ to denote the number of tokens w.r.t. the attribute a in d. Then, it assigns d a specific optimal order which prioritizes the filter with lower values of $sel(F) \times cost_d(F)$.

Filter Pushdown. Given a query that joins two sets of documents with filters applied on them, the most straightforward way (ZenDB) is to first pushdown the filters to the two document sets respectively, extract the join key attribute, and then join. Traditional databases adopt this strategy because filters typically have a lower time complexity than joins and thus should be prioritized. However, in this unstructured data analysis scenario where LLM cost is the primary optimization goal, a join may not be more costly than a filter and potentially could have a higher priority. Inspired by this insight, QUEST proposes a join transformation strategy that first extracts the join attribute of one table and then uses the extracted values as filters to filter the other table, i.e., transforming a join into a filter. Treating this automatically generated filter equally to other filters, QUEST uses the cost model discussed above to order these filters. Therefore, QUEST might prioritize joins over filters to minimize the LLM cost, contradictory to filter pushdown in relational databases.

Join Ordering. For multi-join, ZenDB and QUEST dynamically and progressively decide the join order during query execution. More specifically, they first selects two tables to join based on their cost model, and it will determine the next join only after the first join finishes execution. This process iterates in a left-deep manner until all joins have been executed.

2.4 Physical Optimization

Next, the systems select an appropriate implementation w.r.t. each operator in the logical plan based on the properties of the data and user preferences, i.e., physical optimization. We introduce the typical physical optimizations in UDA below.

Model Selection. For each logical operator, the optimizer selects the most suitable model from a set of candidate based on users' preference, e.g., accuracy or cost. In Palimpzest, if a user wants to achieve target accuracy while at a relatively low cost, the system prefers a lightweight model for simple extraction tasks to reduce cost, e.g., using GPT-4.1-mini to extract "birthdate" from the Art dataset. Conversely, for complex semantic analysis where Llama fails to meet the target accuracy, it employs more advanced models, such as using GPT-4.1 to infer whether a case is a first-instance trial in the Legal dataset.

Model Cascade. In addition to selecting one model that is the most suitable for the operator, the optimizer in LOTUS applies the model cascade technique to save more cost while meeting the users' target accuracy. More specifically, it uses a sequence of different models to execute an operator. These models have various characteristics, such as diverse qualities, varying costs, and different levels of latency. For example, the model cascade can be {GPT-4.1-nano, GPT-4.1-mini, GPT-4.1}, which begins with the cheapest GPT-4.1-nano. LOTUS immediately returns the result if the GPT-4.1-nano output meets the target accuracy. If not, the input proceeds to the next model in the cascade, i.e., GPT-4.1-mini, until obtaining a satisfactory output or reaching the last model.

Parallel Execution. Leveraging efficient batched inference with vLLM [15], LOTUS and DocETL process multiple documents concurrently, allowing efficient operator execution over large-scale document collections.

Dataset	#-Domain	#-Attributes	#-Files	Tokens (Max/Min/Avg.)	Multi-modal
Art	1	19	1,000	1,665 / 619 / 789	✓
CSPaper	1	26	200	107,710 / 5,325 / 29,951	✓
Player	4	28	225	51,378/73/8,047	✗
Legal	1	19	566	45,437 / 340 / 5,609	✗
Finance	1	30	100	838,418 / 7,162 / 130,633	✗
Healthcare	3	51	100,000	63,234 / 2,759 / 10,649	✗

Table 2: Statistics of datasets.

3 THE BENCHMARK CONSTRUCTION

In this section, we first overview the construction process of Bench-U and then introduce each step in detail.

3.1 Overview

Bench-U consists of 6 datasets: Art, CSPaper, Player, Legal, Healthcare and Finance. We first collected and pre-processed the raw data. Next, we followed a semi-automatic, iterative process to identify the attributes of the relational tables. We then extract the values of these attributes from these datasets, i.e., labeling ground truth, combining iterative LLM prompting and human labeling. Finally,

we produce a comprehensive set of queries, which can be divided into 5 major categories and 42 sub-categories.

For each dataset, we provide benchmark results in a JSON file, where all documents are decomposed into separate modalities (text, tables, and images). Each document maps to a field in the JSON file, and the different modalities are stored in distinct entries. In this way, if a user wants to test her system using Bench-U, she can directly download the processed data, load the data into the system, run her queries, and compare the results with the ground truth table stored in the relational databases.

3.2 Datasets

Bench-U consists of six datasets with their statistics summarized in Table 2. Next, we describe these datasets below. For more details, refer to the technical report [7].

Art is collected from WikiArt.org [3], which covers artists and their artworks spanning from the 19th to the 21st centuries. Each document corresponds to an artist including biographical information, artistic movement, a list of representative works, and an image of a representative work.

CSPaper dataset is crawled from Arxiv[1] containing 200 research papers annotated with key attributes, including authors, baselines and their performance, the modalities of experimental datasets etc. In particular, some papers describe the performance of all baselines in the main text, while other papers only describe the best-performing baselines and leave other results in tables or figures, resulting in an analysis scenario with mixed-modal.

Player dataset is crawled from Wikipedia[19] that contains information about basketball including players, teams, team managers, etc., from the 20th century to the present, covering their basic and statistic information.

Legal is sourced from AustLII [5] with 570 professional legal cases from Australia between 2006 and 2009, covering different types such as criminal and administrative. Each case document typically includes evidence, charges, legal fee, etc.

Finance are collected from the Enterprise RAG Challenge [11], containing annual and quarterly financial reports published in 2022 by 100 listed companies worldwide with an average token length of 130,633. Each record typically includes mixed types of content like company name, net profit, total assets, etc.

Healthcare is obtained from MMedC [22], with healthcare documents since 2020. This dataset contains a massive amount of files (100,000), each file having 10,649 tokens on average. It covers various types of healthcare information, like drugs, diseases, medical institutions, news, interviews crawled from large-scale web corpora and open-access healthcare websites.

Summarization. These datasets show various characteristics. Compared to other datasets, the Art dataset is less complex due to its short documents and well-defined structure, but each document contains a corresponding image. CSPaper is a moderately complex academic dataset composed of research papers that contain text, tables, and images. Legal is more complex because it is a domain-specific dataset containing multiple attributes that require semantic reasoning to extract. Finance is another complex domain-specific dataset. The key challenge it introduces is the length of the documents – up to 100 pages. Lastly, Healthcare has the largest number of documents, containing rich information. Healthcare and Player contain multiple categories of files, e.g., disease, drug, medical institution, suitable for evaluating join queries.

Note that previous works [2, 8] also use the sources of Player, Art, and Legal in their experiments. We largely augment them by adding new documents, attributes and comprehensive query workloads, addressing the issue that the original datasets were curated primarily for extraction and offered limited query coverage.

Data processing. We design a unified data preprocessing pipeline to handle datasets with various features. For each dataset, we organize the collected raw data files into a main folder, with documents grouped into subfolders based on their domains. For the Healthcare and Player datasets, we partition documents into topic-specific domains, each mapped to a relational table. In Healthcare, which originally contains 680,000 healthcare documents, we sampled 100,000 and used LLMs to identify three main domains (disease, drug, and medical institution), covering 6,100 documents. We define and extract attributes only for these three domains. Other documents are kept to create a realistic large corpus where only some documents (the three domains above) are relevant to a query, allowing our benchmark to evaluate UDA retrieval capabilities. For Player, we identify 4 document subsets (players, teams, team managers, cities) suitable for evaluating join queries.

3.3 Ground Truth Labeling

To label the ground truth, we first identify a number of significant attributes from each dataset and then manually extract their values.

Attributes Identification. We hire 6 Ph.D. students from different majors (e.g., finance, law, medicine) at our university to carefully read these documents and identify attributes with varying extraction difficulty. For example, the attribute Judge name is easy to identify, as it usually appears at the beginning of each document in Legal. In Finance, Business cost is the sum of several components that differ by industry—for example, raw materials and wages for car manufacturers, or product sourcing and logistics for supermarkets. The labeler must identify these components from the document, extract their values, and sum them to obtain the final Business cost. Image files also contain easy-to-extract attributes like Tone, whose answers ("Neutral", "Bright", "Dark") are straightforward to categorize. In contrast, difficult attributes such as "Style" require art expertise for reliable extraction. To rigorously evaluate cross-modal reasoning, we also design complex attributes that require joint analysis of text, tables, and images. For example, in CSPaper, identifying the most critical ablation component requires first understanding the textual ablation setup, then locating the relevant tables or charts, and finally determining which component's removal causes the largest numeric drop.

Labeling. To ensure high-quality ground truth for Bench-U, we employ 30 graduate students to manually label these attributes, totaling 4,110 human hours. For each document, an ensemble of diverse LLMs independently retrieves candidate text chunks for each attribute. Human labelers then review the combined chunks to extract ground-truth values, using LLMs as assistants. If the attributes cannot be found in the retrieved chunks, labelers manually read the document to answer. To further ensure labeling quality, we use an LLM-as-a-judge to verify human annotations. When

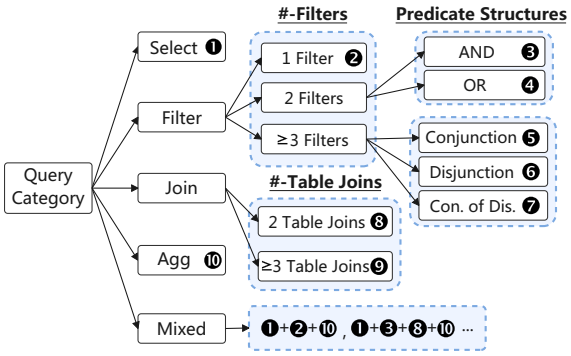

Figure 2: Query Categories.

the LLM flags an extracted attribute as incorrect, humans double-check the corresponding ground truth. However, for the large-scale dataset `Healthcare`, manually labeling all ground truth is impractical. We adopted a semi-automated iterative labeling strategy. Recall that `Healthcare` includes many documents outside the three major categories, which therefore rarely contain entities from them. We used LLMs to classify 100,000 documents as either belonging to one of the three categories or not. For documents in those categories, humans labeled attributes; otherwise, the ground truth is `NULL`. Remark that the LLM ensemble serves exclusively as auxiliary tools for retrieval, while humans label the ground truth. This ensures the ground truth reflects objective facts rather than LLMs outputs, preventing the benchmark from inheriting biases specific to certain prompting strategies or models.

Quality Assurance. We conducted an Inter-Annotator Agreement (IAA) evaluation to ensure labeling quality. We randomly sampled 191 documents across all domains and had three independent annotators extract attributes, yielding 3,445 attribute values for analysis. We use the pairwise agreement ratio [4] ($S = \frac{N_{agree}}{N}$), where N_{agree} is the number of instances in which two annotators extract the same value for a specific attribute in a document, and N is the total number of annotator pairs across the 3,445 cells. The results show strong agreement, with scores from 0.88 in specialized domains (`Finance` and `Healthcare`) to 0.98 in general domains (`Player`). Most inconsistencies arise from differences in domain expertise rather than factual errors. In specialized fields, non-expert annotators may systematically misinterpret complex implicit logic (e.g., inferring whether a legal case is a first-instance trial). Full details of quality assurance are provided in our technical report [7].

3.4 Query Construction

As shown in Figure 2, we divide the queries into five main categories based on the primary query operators, as different operators introduce different technical challenges: (1) `Select`, involving only `Extract` operations; (2) `Filter`, combining `Extract` with `Filter` operations; (3) `Join`, involving `Extract` and `Join` operations; (4) `Agg`, incorporating `Extract` with `Aggregation` operations; and (5) `Mixed`, including several categories of operators listed above. Note that every query includes an `Extract` step to return the attributes to be analyzed. We then further decompose each main category into subcategories to construct a broad range of queries that differ in modality, filter complexity in terms of the number of filters, predicate structures (the combination of conjunction or disjunction), selectivities, single or multiple joins, w/ or w/o aggregation, etc. These queries are in the form of both SQL-like queries and Python code, thus able to test systems with different interfaces, like `ZenDB` and `Palimpzest`. An example is shown below.

```
SELECT {Attribute}(s),
    {agg_func}({Attribute}(s))
FROM diseases
JOIN drug ON disease.name = drug.disease
WHERE {Attribute}{operator}{literal}
GROUP BY {group_by};
```

A SQL subcategory Example.

```
disease_doc = disease
drug_doc = drug
disease_doc = Filter(disease_doc,
    {Attribute}{operator}{literal})
disease_drug = Join(disease_doc, drug_doc,
    disease_doc.name = drug_doc.disease)
result = Extract(disease_drug,
    {Attribute}(s))
result = Aggregate(result, {group_by},
    {agg_func}({Attribute}(s))
```

A Code subcategory Example.

(1) *For the* `Select` *category*, for example, a query such as "SELECT {Attribute}(s)" represents a `Select` operation used to extract attributes. Here, "{Text Attribute}(s)" serves as a placeholder that can be instantiated with any number of available attributes with different modalities, which are randomly selected to generate concrete queries. These `Select` queries can clearly evaluate the system's capability of extracting attributes with different modalities.
(2) *For the* `Filter` *category*, we further classify it by the number of filters (1, 2, and ≥ 3 filters) and the way of combining the filters (conjunction only, disjunction only, and conjunction of disjunctions), resulting in a total of 6 `Filter` sub-categories. With varying complexities, these sub-categories of queries offer a large space for cost optimization (e.g., more filters bring a large optimization opportunity for filter reordering). As shown in the example, the pattern "{Attribute}{Operator}{Literal}" serves as the placeholder format for filter predicates. During query generation, we randomly sample both the filter attribute and the comparison operator (such as ≤, =, or ≥) with equal probability, and we further sample literal values to produce varying selectivities.
(3) *For the* `Join` *category*, to thoroughly evaluate the capacities of different systems in optimizing join queries, we curate two subcategories of join queries, i.e., binary joins and multi-table joins, where more tables incur higher computation cost. During query generation, we explicitly define a join graph w.r.t. each dataset. For example, in `Healthcare`, valid join paths include Disease ⋈ Drug, along with their corresponding join keys (e.g., Disease.name = Drug.disease). Similarly, in `Player`, we have Players ⋈ Teams ⋈ Managers, together with join keys that link these tables.

(4) *For the* Agg *category*, during query generation, we randomly sample the group-by attribute, the aggregated attributes, and the aggregation functions (such as Count) to populate the corresponding placeholders (i.e., {agg_func}({Attribute}(s)).
(5) *For the* Mixed *category*, it combines multiple types of operators in one query, e.g., first joining two tables and then run aggregation on the join results. It includes 32 subcategories, each of which is combined with at least three of the aforementioned main categories. The above example illustrates one such combination, i.e., selecting attributes with one filter, an aggregation and a 2-table join.

We then apply these subcategories to each dataset to construct queries. After generating a large pool of candidate queries, we ask human experts to select those that are both meaningful and representative. Specifically, for the first four main categories, we retain 10 queries per subcategory in each dataset. Since the Mixed category aims to evaluate the performance of combining different operators, a single query per template per dataset is sufficient.

4 EVALUATION

We evaluate existing systems on Bench-U to answer the questions:
- **RQ 1**: What is the overall performance of different systems on the benchmark? Which system achieves the best performance under different scenarios?
- **RQ 2**: How do different logical optimization strategies perform?
- **RQ 3**: How do different physical optimization strategies perform?

4.1 Experimental Settings

Systems for Evaluation. Our benchmark evaluates 7 existing UDA systems. (1) Evaporate is a table extraction system, so we employ Evaporate to extract structured tables from documents and subsequently execute SQL queries on the resulting tables. (2) Palimpzest provides Python API-based operators for unstructured data processing. We convert each SQL query into the corresponding Palimpzest code, execute it and obtain the results. (3) LOTUS also provides an open-source Python library, which we use to execute queries through its interface. (4) DocETL is an open-source project using Python code. We rewrite our queries with DocETL library and execute the Python programs. (5) QUEST provides SQL-like query interface for processing unstructured documents, and we directly use their code to execute queries. (6) ZenDB does not provide their code; therefore, we implement their SHT chunking and filter reordering strategies and evaluate them on Bench-U. (7) UQE does not release its code, so we reimplement its Filter and Agg operators, along with its logical optimizations, and run them on Bench-U. Since these two systems are closed-source, we have no choice but faithfully reproduce their core algorithms and configurations. Bench-U evaluates *system design choices* rather than official engineering optimizations, our results reflect the relative efficacy of these algorithmic designs rather than a definitive verdict on the original systems. All adaptation details are available in the technical report [7].

Evaluation Metrics. Following existing works [17, 27], we measure accuracy, cost, and latency w.r.t. all queries. For queries without aggregation (Select, Filter, Join), the outputs are relational tables, we follow [27] and report the average precision, recall, and F1 score. Given a query Q, the set of tuples returned by a method is $T(Q)$, and the ground truth is $GT(Q)$. For each element $t \in T(Q)$, we evaluate whether it can be matched to a corresponding tuple in $GT(Q)$. Since exact string matching fails on lexical variants (e.g., "Los Angeles Lakers" vs. "Lakers"), we employ an LLM-as-a-judge during evaluation to verify semantic equivalence, treating them as correct matches if they denote the same entity. Conversely, aggregation queries output numeric values. We classify the evaluation of these predicted values into two distinct categories based on the specific aggregation functions. For MIN and MAX, we use a precise exact match ($x = gt$). For SUM, AVG, and COUNT, we explicitly measure the numerical deviation between predicted values and ground truth. We follow the previous work [10] to calculate an accuracy score derived from the bounded relative estimation error for each group: $1 - \min(1, \frac{|x-gt|}{\min(|x|,|gt|)+\epsilon})$. Here, x represents the predicted value for a specific group, gt represents the ground truth value, and ϵ (10^{-5}) is a minimal constant added to prevent division by zero. The final accuracy score for the query is then calculated by averaging the scores across all groups. All our metrics are normalized to the range $[0, 1]$, where higher values indicate better performance. This normalization ensures consistency in both visualization and comparison throughout our benchmark evaluation. For LLM costs, we report the average number of tokens (in thousands) per document per query by using the LLM API, which counts the token consumption w.r.t. all data modalities, e.g., text and images. For latency, we report the mean execution time in seconds per document per query.

Environment. We implemented all experiments in Python 3.7 on an Ubuntu Server with four Intel(R) Xeon(R) Gold 6148 2.40GHz CPUs with 80 cores, 2 NVIDIA 4090 GPUs, 1TB main memory and 6TB SSD. The same environment ensures a fair comparison over systems. We adopt GPT-4.1-mini as the default LLMs for API calls, and Qwen3-Embedding-0.6B as the default embedding model.

4.2 Overall Performance Comparison (RQ1)

As shown in table 3, across the majority of evaluated datasets (Art, CSPaper, Legal and Finance), Palimpzest and LOTUS consistently achieve the highest accuracy, reaching approximately 0.58 in Art, 0.56 in CSPaper, 0.52 in Legal, and 0.39 in Finance. Specifically, in short documents (Art), Palimpzest and LOTUS achieve the lowest cost (2k tokens per document) at the same time. Their costs are lower than those of systems employing chunking strategies (e.g., ZenDB). The reason is that when processing short documents, multiple target attributes frequently appear within the same text chunk. Repeatedly feeding this shared chunk to the LLMs for different semantic operators ultimately consumes more tokens than directly processing the complete document a single time.

> **Insight I:** Chunking (or data retrieval) does not help on short documents, as most relevant information is already contained within a small context. In such cases, applying retrieval or chunk-level processing introduces unnecessary overhead, increasing cost without improving accuracy.

However, as the length of documents increases, the chunks relevant to semantic operators become increasingly separated. Consequently, chunking-based systems begin to achieve significant cost savings. Within the CSPaper, Legal, and Finance datasets, although Palimpzest and LOTUS achieve highest accuracy, they

Method	Art			CSPaper			Player			Legal			Finance			Healthcare		
	Acc.	Cost	Latency	Acc.	Cost	Latency	Acc.	Cost	Latency	Acc.	Cost	Latency	Acc.	Cost	Latency	Acc.	Cost	Latency
Evaporate	0.08	-	0.16	0.23	-	2.40	0.18	-	0.30	0.14	-	0.25	0.07	-	1.50	0.01	-	0.52
Palimpzest	0.58	1.99	1.40	0.56	27.47	7.70	0.69	9.22	3.83	0.52	8.83	3.07	0.39	176.35	10.55	0.24	14.55	4.76
LOTUS	0.57	2.52	4.13	0.55	34.49	8.22	0.68	12.16	4.18	0.51	12.41	2.84	0.38	256.98	15.87	0.22	21.75	5.55
DocETL	0.45	6.60	14.79	0.45	41.41	63.15	0.70	54.23	76.96	0.51	93.48	151.32	0.29	501.08	761.88	0.21	148.83	176.01
ZenDB	0.44	2.73	1.66	0.45	18.37	32.60	0.65	6.66	2.77	0.44	15.31	14.33	0.35	40.76	4.14	0.22	14.91	2.10
QUEST	0.44	2.05	1.38	0.44	10.63	2.39	0.66	3.83	1.52	0.45	6.21	1.50	0.35	32.65	3.49	0.27	7.08	1.91
UQE	0.41	0.75	5.79	0.34	29.25	5.91	0.53	5.39	5.01	0.37	7.06	5.18	0.29	78.65	5.91	0.17	9.30	0.63

Table 3: Overall Performance across 6 datasets.

incur costs that are five times higher than other systems (e.g., ZenDB and QUEST) that utilize text chunking and retrieval strategies.

Moreover, Palimpzest attains higher accuracy and lower cost than LOTUS on these datasets. This is because LOTUS asks the LLM to produce a boolean judgment from the entire document at once, whereas Palimpzest first extracts the target attribute and then checks whether it meets the condition. This two-step pipeline gives more explicit instructions, leading to more accurate results than direct logical execution.

Observation I: In the case of the filter operator, extracting the attribute first and evaluating the filter thereafter lead to more accurate results than directly executing it with LLMs. This is because decomposing the filter into the above two steps provides LLMs with more explicit instructions.

Palimpzest achieves lower operational costs than LOTUS, as Palimpzest implements specific logical optimization techniques. When evaluating multiple filters, Palimpzest prioritizes filters with low selectivity to discard irrelevant documents early in the execution pipeline. This filtering strategy substantially decreases the volume of operations required during subsequent processing stages. Therefore, if the user prioritizes accuracy, Palimpzest is the most suitable system, as it achieves the highest accuracy while maintaining relatively low costs.

Furthermore, QUEST and ZenDB achieve the lowest cost. They achieve substantial cost savings not only by partitioning documents but also by implementing logical optimizations. Furthermore, QUEST is more cost saving compared to ZenDB because it dynamically optimizes the execution order of filters for each individual document. This dynamical strategy prevents the computational waste compared to ZenDB. Specifically, ZenDB generates a unified logical plan that evaluates filters based on the average token cost across the entire dataset. However, the exact size of chunks corresponding to a specific attribute varies significantly among different documents.

Observation II: Implementing dynamic logical optimization at document level is an effective strategy for minimizing the operational cost of current systems.

In general, within CSPaper, Legal and Finance datasets, there currently exists no optimal solution that simultaneously achieves superior accuracy, reduced costs, and lower latency. Existing systems either process the complete document context to maximize extraction accuracy at the expense of cost and latency, or they employ retrieval or chunking strategies to minimize cost consumption while inevitably sacrificing accuracy.

Insight II: All existing systems struggle on long documents due to missing an effective data retrieval mechanism. That is, scanning the document with an LLM preserves complete context but is prohibitively expensive, while the systems that use data retrieval to filter data tend to erroneously prune the text chunks that contain valuable information.

However, in Player and Healthcare, Palimpzest and LOTUS no longer achieve the highest accuracy. Within the Player dataset, almost all systems achieve high accuracy, reaching approximately 0.68. This is because attributes in this dataset can be easily extracted using independent text chunks, which eliminates the necessity to scan the entire document. Therefore, both full document processing and chunking-based systems achieve high accuracy. DocETL achieves the highest accuracy of 0.70, since it leverages multiple agents to process these chunks. This collaborative reasoning capability enhances the overall execution accuracy. However, this marginal accuracy gain comes at an unacceptable price. The token cost of DocETL reaches 54.23k tokens per document and its latency jumps to 76.96 seconds per document. This is because the system repeatedly feeds chunks into the model and exhaustively evaluates all possible execution plans. Therefore, QUEST and ZenDB are recommended under such circumstances, as they achieve high accuracy while maintaining the lowest costs. And QUEST is the most cost-efficient, since its document-level logical optimization can further reduce cost by dynamically adapting execution plans for each document.

Insight III: When retrieval successfully captures all relevant information, systems can achieve high accuracy. However, different execution plans lead to different costs. In such cases, logical optimization strategies can identify more cost-efficient plans and save cost. Furthermore, document-level logical optimization can further reduce cost by dynamically adapting execution plans (e.g., filter ordering) for each document, thus yielding the most cost-efficient execution plan.

Within Healthcare, a lot of irrelevant and noisy information frequently misleads the LLMs. Consequently, systems (Palimpzest and LOTUS) that supply the entire document directly to the models exhibit poor performance. And QUEST offers the highest accuracy and lowest cost. This advantage is due to its hierarchical two-stage retrieval strategy. First, the system utilizes lightweight summaries

to entirely discard massive volumes of irrelevant documents. This document-level filtering substantially reduces unnecessary token consumption. Second, the system evaluates the remaining documents to process only on the relevant chunks. This chunk-level filtering process not only saves cost but also prevents noisy context from degrading the overall accuracy. However, this strategy relies heavily on the quality of retrieval, making the design of robust, highly effective retrieval mechanisms a critical direction for future research.

Insight IV: A good data retrieval strategy is the key to analyzing long and complex documents. In addition to reducing costs, it can even improve accuracy by preserving all relevant information while filtering out noisy or irrelevant content.

We also find that Palimpzest and LOTUS outperform other systems by over 10% accuracy on the Art and CSPaper datasets, which include multimodal information. Palimpzest and LOTUS use specialized semantic operators for image processing. In addition, the CSPaper defines cross-modal attributes, such as best performing baseline, that require complex integration of text with the corresponding tables or images. By providing the complete document directly to the models, Palimpzest and LOTUS preserve the cross modal structural integrity, ensuring that the critical associations between text, tables, and images remain complete, but at a high cost. Therefore, current systems lack advanced strategies for multi modal data chunking and corresponding retrieval. Developing such strategies could significantly advance cost optimization efforts in future works.

Insight V: Existing methods perform poorly on multi modal datasets, as they lack a retrieval method that can effectively prune the irrelevant data, while still preserve the correlation among the relevant data across different data modalities.

In general, operational cost correlates directly with execution latency because latency depends on input/output tokens and the number of LLM calls. DocETL is slowest, using a multi-agent approach with many LLM calls and the highest token usage. LOTUS and Palimpzest follow, as they also make multiple calls and pass the full document each time. UQE is more efficient but still processes entire documents per call; with filters, its regression model greatly improves efficiency. QUEST and ZenDB are efficient because they use fewer tokens and fewer LLM calls, while Evaporate is most efficient by generating extraction code.

4.3 Evaluation of Logical Optimization (RQ2)

Filter Reordering. We compare with four filter reordering strategies as follows. (1) Random: the filters are executed in random order; (2) Sel: the filters are ordered based on the selectivity; (3) Sel+Cost (ZenDB): the filters are ordered based on both the selectivity and the estimated average cost of extracting each attribute from the sampled documents; (4) (Sel+Cost)/doc (QUEST): each document has its own plan considering the selectivity and the estimated extraction cost w.r.t. this document. We compare the above strategies based on QUEST. To evaluate sufficiently, we conduct experiments on Player over different query subcategories w.r.t. different numbers of filters: S1 with one filter, S2 with 2 filters, and S3 with 3 or more. In Figure 3, for queries in S1, all baselines have nearly the same cost since each query uses a single filter and thus only one order. For queries with more filters, these methods are ranked as follows by the LLMs cost: (Sel+Cost)/doc < Sel+Cost < Sel < Random. The first two strategies save more cost because they optimize the order considering the cost. (Sel+Cost)/doc is the most cost-effective because it provides fine-grained optimization for each individual document.

Filter Pushdown. We compare with two strategies as follows. (1) DB-Pushdown: like in traditional databases, it pushes down filters respectively to the relevant tables before join. (2) Trans-Join (QUEST): it transforms a join to a filter operation and orders the filters using the above (Sel+Cost)/doc strategy to reduce the cost. We compare the above strategies using Mixed queries with filters on Player and Healthcare. We observe from Figure 4 that Trans-Join is more cost-effective than DB-Pushdown because given two tables to join, Trans-Join builds a cost model to judiciously determine which table will be extracted first and transformed to a filter on the other table, which provides the opportunity to run a join first if it incurs a smaller data extraction cost.

Join order. We evaluate three strategies as follows. (1) Pushdown: all filters are pushed down first, and then join is performed; (2) Random: tables (document subsets) are joined in random order, with each pair of tables joined using Trans-Join; (3) Dynamic (ZenDB, QUEST): it uses a cost model to dynamically identify two tables to join in a left-deep manner, with each pair of tables joined using Trans-Join. We compare the above strategies using Join queries involving three tables on Player and Healthcare. We observe from Figure 5 that Dynamic saves much cost because it dynamically selects the join operation that leads to the lowest cost.

4.4 Evaluation of Physical Optimization (RQ3)

Model Selection. We evaluate two strategies in Palimpzest as follows. (1) Model-Sel: We define a set of candidate models (GPT-4.1-nano (Nano), GPT-4.1-mini (Mini), and GPT-4.1 (Normal)) and set the selection objective as "minimize cost while achieving the target accuracy." (2) no-Model-Sel: The system always uses a specific model (Mini) for all queries. The user has to specify a desired accuracy. In our experiments, we set this value as the average accuracy obtained by running Filter queries with Mini on each dataset (Art and Legal). We evaluate both strategies using Filter queries on Art and Legal. The cost is calculated by multiplying the number of tokens by the model's price per token. As shown in Figure 6, on Art, Model-Sel exceeds the target accuracy, while also reducing cost by using a cheaper model. This is achieved by assigning Normal to harder attributes and Nano to easier ones, while Art has more easy attribute than hard attributes. On the challenging Legal dataset, Model-Sel outperforms No-Model-Sel in accuracy, but with higher cost due to more frequently using Normal.

Model Cascades. We evaluate two strategies in LOTUS as follows. (1) Cascades: we construct a model cascade using two models, GPT-4.1-nano (Nano) and GPT-4.1-mini (Mini). For each query, the system first uses the lower-cost Nano to execute the queries. If the accuracy does not meet the desired accuracy, the query is then forwarded to the larger Mini for further processing. (2) No-Cascades: The system uses only a single model, Mini to process all queries.

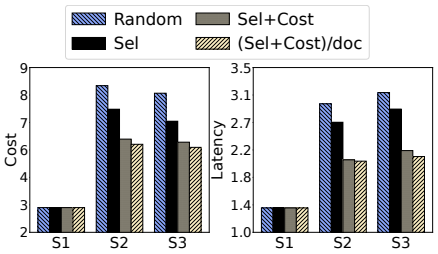

Figure 3: Evaluation of Filter Reordering.

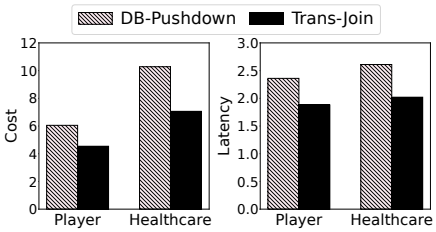

Figure 4: Evaluation of Filter Pushdown.

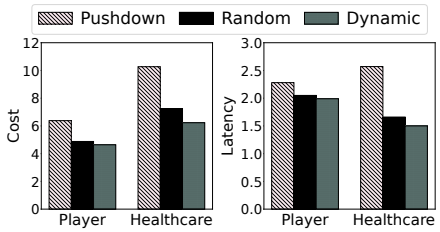

Figure 5: Evaluation of Join Order.

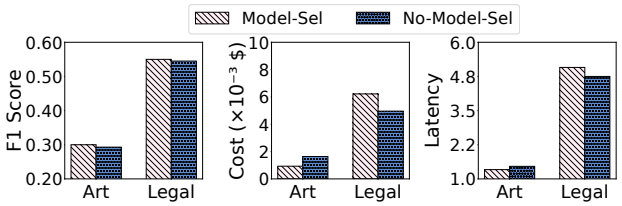

Figure 6: Evaluation of Model Selection.

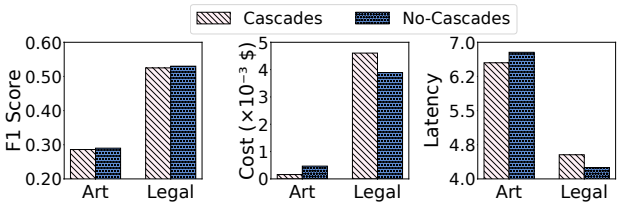

Figure 7: Evaluation of Model Cascades.

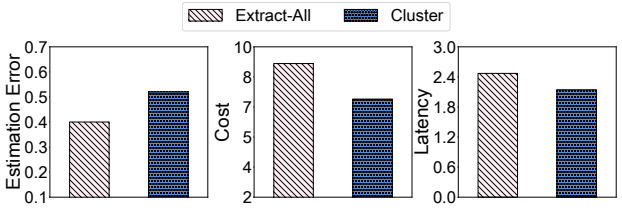

Figure 8: Aggregation Optimization.

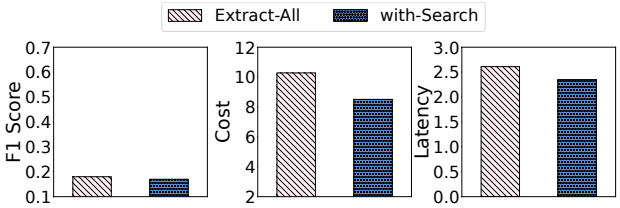

Figure 9: Join Optimization.

We evaluate both strategies on the same set of queries for each dataset and use the same cost and target accuracy settings as the above model selection experiment. As shown in Figure 7, on `Art`, `Cascades` achieves the target accuracy while reducing cost by primarily using the less expensive `Nano`. This is because, for most attributes in `Art`, the cheaper model is sufficient. On the challenging `Legal` dataset, `Cascades` perform similar as `No-Cascades`, but incurs higher costs. This is because almost every attribute eventually requires `Mini`, and all documents have to be first processed by `Nano` before being fed to `Mini`.

Opportunity I: Currently, there is no system supporting end-to-end query optimization, including all above logical and physical optimization strategies, which would be a promising direction to achieve high-efficacy query execution.

Optimization for Join. We evaluate two join strategies on `Healthcare`. (1) `Extract-All`: it extracts the join key attribute from the two tables and join them. (2) `with-Search`: it extracts each value of join key attribute from one table, leverages it as a semantic search key to prune many documents in the other table and then join. We can observe from Figure 9 that `with-Search` saves more cost than `Extract-All` since many documents in the other table are pruned without extraction. However, the accuracy is low because it incorrectly prunes documents that can be joined.

Optimization for Aggregation. We use the estimation error as the metric following [10], defined as the absolute difference between the estimated and true values divided by the true value. We evaluate two strategies on `Art` using `Agg` queries as follows. (1) `Extract-All`: it extracts the attributes involved in Groupby and Aggregation and then executes the query. (2) `Cluster`: it clusters the attribute-related chunks based on the attribute in Groupby, extracts the attribute in Aggregation within each cluster and executes the query. In Figure 8, `Cluster` is more cost-effective than `Extract-All` because it assumes all documents in a cluster share the same Groupby attribute value and thus avoids extracting that attribute. However, its relative error is high because high-quality clustering based solely on chunk embeddings is difficult.

Opportunity II: Another promising direction is to develop more effective strategies to align the embeddings of attributes with the embeddings of their chunks. In this way, the clustering in aggregation and pruning in join can be more accurate, and thereby the costs can be reduced more.

4.5 Evaluation of Chunking Strategies

We evaluate how different chunking strategies affect system performance: (1) `Fixed-length` [6]: fixed 512-token chunks. (2) `Heuristic` [25]: grammar-based chunks, up to 512 tokens. (3)

Semantic (QUEST): semantic chunks, 128–512 tokens. (4) Tree (ZenDB): SHT tree–based splitting. We run this experiment by changing the chunking strategy in QUEST while running Filter queries on Art and Legal. Semantic performs best by capturing rich semantic representations, yielding superior embedding similarity. Tree surpasses Heuristic on Art and Legal because it preserves more tokens per chunk, providing richer context at the cost of higher latency and expense. Fixed-length performs worst, as it can split sentences across chunks, leading incomplete information. We further evaluate different embedding models, multiple LLMs (closed- and open-source), various prompting strategies (zero-shot, few-shot, and CoT), and latency reduction through parallel execution. The complete experimental results, including all figures and detailed analyses, are provided in the technical report [7].

Scope and Limitations. First, Bench-U evaluates relational-style declarative querying over unstructured data, targeting systems (e.g., Palimpzest, LOTUS, UQE) that expose such data as relational views. While enabling rigorous evaluation of operators (Select, Filter, Join, Agg) and optimizers, it underrepresents non-tabular tasks such as open-ended QA or schema-free summarization. Second, results depend on design choices such as LLMs and prompting. Changing these may alter performance. Thus, Bench-U is not intended to provide a universal ranking, but a unified platform for comparing designs under strict settings, helping researchers uncover common observations and promising architectural directions within specific constraints.

5 RELATED WORK

UDA Benchmark. Recent years have witnessed substantial progress in benchmarking and structuring unstructured data. Pioneering works such as Doctopus and Evaporate focus on benchmarking the extraction of structured attributes from unstructured data to construct tables. SemRank focuses on benchmarking the retrieval and prioritization of relevant documents for a given query. Bench-U instead targets executing relational-style queries directly over unstructured data, moving beyond the simple extract operator and retrieval strategies. Other existing systems evaluate few queries to highlight specific capabilities, and thus provide limited coverage of the overall design space. For example, Palimpzest demonstrates cost optimization on a fraudulent email detection task over 1000 short emails, highlighting its ability to aggressively push down cheap filters. Similarly, LOTUS evaluates its execution engine using five queries across five tasks, focusing on semantic operators. For example, it tests the sem_filter operator via fact verification over 500 FEVER claims. However, the evaluations for LOTUS and Palimpzest do not expose the system challenges regarding text chunking. Furthermore, DocETL evaluates its architecture using one representative query per dataset across five datasets. For example, it runs a clause extraction task on CUAD (510 long legal contracts), extracting text spans for predefined clause types. These tasks highlight multi-agent reasoning, but do not consider token cost optimization. To address these limitations, Bench-U provides a diverse query workload that enables systematic evaluation of optimization strategies. Concurrent with our effort, SemBench has a distinct and valuable goal: benchmarking semantic query processing over a mix of structured and unstructured data. However, unlike Bench-U, SemBench does not target thoroughly evaluating the capability of the LLM-powered data systems in processing long, complex unstructured documents that raise significant challenges in cost and accuracy.

LLM-powered Unstructured Data Analysis Systems. Lotus [21] introduces semantic operators for unstructured data processing, including indexing, extraction, filtering and joining capabilities that enable the construction of complex analytical pipelines. It provides optimized physical implementation for each operator but lacks logical optimizations. Palimpzest [18] offers libraries for users to write declarative Python code to analyze unstructured data. It optimizes the logical plan of each program via filter reordering based on selectivities. DocETL[26] focuses on improving the accuracy of UDA using a multi-agent strategy, but it does not conduct logical optimization to save cost. ZenDB[17] uses semantic hierarchical trees to identify relevant document sections and applies filter ordering and predicate pushdown for optimization. However, it requires well-structured documents and is evaluated on just 221 documents and 27 queries. UQE [10] provides SQL-like analysis with sampling-based aggregation, while CAESURA [30] decomposes queries into operators to handle data in different modalities.

In addition, extracting data from unstructured sources is a key component in analyzing unstructured information, the strategies have evolved from rule-based methods [16, 20, 23, 24] to modern deep learning approaches [14, 31]. Nowadays, LLM-based systems seek to resolve these issues. Evaporate [2] generates extraction code through LLMs, balancing cost and quality through weak supervision. Similarly, early systems [9, 12, 28, 29] employ LLMs for document analysis but overlook cost optimization, which is a critical concern given the computational expense of LLM inference, and lack a thorough benchmarking.

6 CONCLUSION

In this paper, we build a comprehensive LLM-powered benchmark for unstructured data analysis. We collect six diverse unstructured datasets, define key attributes, and label them manually with the assistance of LLMs in cross-validation. We then construct hundreds of queries with various analytical operators, capable of comprehensively assessing the capability of existing systems, run the queries on the labeled datasets, and analyze their performance in depth.

ACKNOWLEDGMENTS

Chengliang Chai is supported by the NSFC (U25B2019, 62472031), the National Key Research and Development Program of China (2024YFC3308200), Beijing Nova Program, and Huawei. Qiyan Deng is supported by the BJNSF Qiyan Program (QY24174). Yuping Wang is supported by the NSFC (U23A20297, U23A20317). Xu Zhou is supported by the NSFC (U23A20317). Ye Yuan is supported by the NSFC (Grant Nos. 62225203, 62532001), Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China(Grant No. JYB2025XDXM108), Beijing Natural Science Foundation(Grant No. L241010). Guoren Wang is supported by the NSFC (62427808, U2001211), and the Liaoning Revitalization Talents Program (XLYC2204005). Lei Cao is supported by the NSF (DBI-2327954 and SHF-2106621), Amazon Research Awards, and Google.

REFERENCES

[1] [n.d.]. arXiv. https://arxiv.org. Accessed: 2025-02-17.

[2] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2025. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. arXiv:2304.09433 [cs.CL] https://arxiv.org/abs/2304.09433

[3] Art-.org. 2025. *Art-.org – Artists and Artworks (19th–21st C.).* https://www.art-.org/

[4] Ron Artstein and Massimo Poesio. 2008. Inter-coder agreement for computational linguistics. *Comput. Linguist.* 34, 4 (Dec. 2008), 555–596. https://doi.org/10.1162/coli.07-034-R2

[5] Australasian Legal Information Institute (AustLII). [n.d.]. AustLII – Australasian Legal Information Institute. https://www.austlii.edu.au/.

[6] Sinchana Ramakanth Bhat, Max Rudat, Jannis Spiekermann, and Nicolas Flores-Herr. 2025. Rethinking Chunk Size For Long-Document Retrieval: A Multi-Dataset Analysis. arXiv:2505.21700 [cs.IR] https://arxiv.org/abs/2505.21700

[7] BIT-DataLab. 2025. *Bench-U: A Benchmark for Unstructured Data Analysis.* Technical Report. BIT-DataLab Group. https://github.com/BIT-DataLab/Bench-U/blob/main/technical_report.pdf Technical Report, accessed 2025-12-01.

[8] Chengliang Chai, Jiajun Li, Yuhao Deng, Yuanhao Zhong, Ye Yuan, Guoren Wang, and Lei Cao. 2025. Doctopus: Budget-Aware Structural Table Extraction from Unstructured Documents. *Proc. VLDB Endow.* 18, 11 (July 2025), 3695–3707. https://doi.org/10.14778/3749646.3749647

[9] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Sam Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. https://www.cidrdb.org/cidr2023/papers/p51-chen.pdf

[10] 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. arXiv:2407.09522 [cs.DB] https://arxiv.org/abs/2407.09522

[11] Enterprise RAG Challenge. [n.d.]. Enterprise RAG Challenge. https://rag.abdullin.com/. Accessed 17 July 2025.

[12] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. , 26 pages. https://dl.acm.org/doi/10.1145/3654989

[13] K. V. Kanimozhi and M. Venkatesan. 2015. Unstructured Data Analysis–A Survey. *International Journal of Advanced Research in Computer and Communication Engineering* 4, 3 (2015), 223–225.

[14] Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 3748–3761. https://doi.org/10.18653/v1/2020.emnlp-main.306

[15] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody-Hao Yu, JosephE Gonzalez, Hao Zhang, and Ion Stoica. [n.d.]. Efficient Memory Management for Large Language Model Serving with PagedAttention. ([n. d.]).

[16] Taesung Lee, Zhongyuan Wang, Haixun Wang, and Seung-won Hwang. 2013. Attribute extraction and scoring: A probabilistic approach. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 194–205. https://doi.org/10.1109/ICDE.2013.6544825

[17] 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. arXiv:2405.04674 [cs.DB] https://arxiv.org/abs/2405.04674

[18] 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. arXiv:2405.14696 [cs.CL] https://arxiv.org/abs/2405.14696

[19] National Basketball Association – Wikipedia. [n.d.]. NBA – Wikipedia. https://en.wikipedia.org/wiki/National_Basketball_Association. Accessed 17 July 2025.

[20] Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A Survey on Open Information Extraction. arXiv:1806.05599 [cs.CL] https://arxiv.org/abs/1806.05599

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

[22] Pengcheng Qiu, Chaoyi Wu, Xiaoman Zhang, Weixiong Lin, Haicheng Wang, Ya Zhang, Yanfeng Wang, and Weidi Xie. 2024. Towards Building Multilingual Language Model for Medicine. arXiv:2402.13963 [cs.CL] https://arxiv.org/abs/2402.13963

[23] Swarnadeep Saha and Mausam. 2018. Open Information Extraction from Conjunctive Sentences. , 2288–2299 pages. https://aclanthology.org/C18-1194/

[24] Swarnadeep Saha, Harinder Pal, and Mausam. 2017. Bootstrapping for Numerical Open IE. , 317–323 pages. https://doi.org/10.18653/v1/P17-2050

[25] Roie Schwaber-Cohen and Arjun Patel. 2025. Chunking Strategies for LLM Applications. Pinecone Blog. https://www.pinecone.io/learn/chunking-strategies-for-llm-applications/ Retrieved 17 July 2025 from Pinecone website.

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

[27] Zhaoze Sun, Qiyan Deng, Chengliang Chai, Kaisen Jin, Xinyu Guo, Han Han, Ye Yuan, Guoren Wang, and Lei Cao. 2025. QUEST: Query Optimization in Unstructured Document Analysis. arXiv:2507.06515 [cs.DB] https://arxiv.org/abs/2507.06515

[28] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. 2021. From Natural Language Processing to Neural Databases. , 1033–1039 pages. https://doi.org/10.14778/3447689.3447706

[29] Matthias Urban and Carsten Binnig. 2023. Towards Multi-Modal DBMSs for Seamless Querying of Texts and Tables. arXiv:2304.13559 [cs.DB] https://arxiv.org/abs/2304.13559

[30] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, CA, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf

[31] Bowen Yu, Yucheng Wang, Tingwen Liu, Hongsong Zhu, Limin Sun, and Bin Wang. 2021. Maximal Clique Based Non-Autoregressive Open Information Extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9696–9706. https://doi.org/10.18653/v1/2021.emnlp-main.764