QUWARTS: Query Workload Aware Relational Table Synthesis from Unstructured Text

Aritra Mazumder
University of Utah
Salt Lake City, Utah, USA
aritra.mazumder@utah.edu

Whanhee Cho
University of Utah
Salt Lake City, Utah, USA
whanhee.cho@utah.edu

Anna Fariha
University of Utah
Salt Lake City, Utah, USA
afariha@cs.utah.edu

ABSTRACT

Despite the prevalence of structured databases, a lot of the enterprise data— such as clinical notes, financial reports, and legal contracts—remains unstructured and primarily text-based. This creates challenges for analytical queries over such unstructured documents, particularly those involving filtering, aggregation, or joins across documents. Existing methods that process queries directly over unstructured text (e.g., RAG systems) struggle to handle these structured queries effectively. One potential solution is to extract structured data from unstructured text on a per-query basis and execute queries over the extracted data. However, such per-query extraction incurs impractically high query-time latency. An alternative is to perform a one-time structured data extraction offline, independent of any specific query. While this reduces query latency, it often results in suboptimal extraction—such as misaligned schemas or inconsistent entity representations—which leads to inaccurate query results due to incorrect aggregations and failed joins.

Our key observation is that making the offline structured data extraction process aware of the incoming query patterns, which can be extracted from historical query workload, can strike a balance between query result accuracy and latency. We demonstrate QUWARTS, a Query Workload Aware system for Relational Table Synthesis from unstructured text to support analytical queries. Under the hood, QUWARTS leverages the query workload to (i) identify query-intent aligned schema, (ii) perform necessary data transformations to ensure joinability, and (iii) normalize entities to improve aggregation correctness. We will showcase how QUWARTS outperforms existing approaches in terms of query result accuracy and latency over several real-world datasets.

1 INTRODUCTION

An estimated 80–90% of global data is unstructured text, spanning emails, documents, and reports. Extracting insights from such data requires analytical queries involving filtering, aggregation (e.g., sums and counts), and joins across documents. While structured relational databases benefit from decades of advances in query processing and optimization, efficient and effective analytical querying over unstructured text remains underdeveloped.

Recent advances in Large Language Models (LLMs) for processing text create new opportunities for practical systems to support analytical queries over unstructured text documents. In this setting, three criteria are critical: (1) high *accuracy* of the query results, (2) low *latency* per query, and (3) low *cost*, in the number of tokens used to prompt the LLM. Existing approaches in this space fall into three paradigms: (1) direct retrieval, (2) per-query (online) structured data extraction, and (3) offline extraction of structured data. We summarize the trade-offs among these paradigms in Figure 1.

Paradigm	Systems	Accuracy ↑	Cost ↓	Latency ↓
Direct retrieval	RAG [6]	Low	Low	Low
Per-Query extraction	LOTUS [9], Palimpzest [8]	High	High	Moderate
	UQE [4]	Moderate	Low	Moderate
	ZenDB [7], QUEST [12]	Moderate	Low	Low
	DocETL [11], ReDD [3]	High	High	High
Offline extraction	SQUID [10], Evaporate [2], Map & Make [1]	Low	Low	Low
Workload-aware extraction	**QUWARTS**	**High**	**Low**	**Low**

Figure 1: Categorization of existing systems for analytical querying over unstructured text. An ideal system should achieve high query result accuracy with low latency and low cost. Existing approaches face tradeoffs: direct retrieval-based systems have low accuracy, per-query extraction systems incur high latency and cost, and query-unaware offline extraction offers low accuracy. QUWARTS achieves all three desirable properties by leveraging query workload during offline extraction.

Direct-retrieval systems [6] typically retrieve the top-k document chunks that are most similar—based on their embeddings—to the analytical query's embedding. Such approaches do not use any intermediate structured representation and, thus, often miss relevant information, resulting in poor query result accuracy. They are particularly ill-suited for aggregate queries, as reliance on partial top-k retrieval and implicit reasoning leads to incomplete enumeration (e.g., missing entities that are not salient in the embedding space). Moreover, complex joins that require systematically matching keys across documents are not naturally supported in this approach.

An alternative paradigm is *Per-Query extraction*, which aims to extract relevant information into a structured representation for each query. Most per-query extraction systems [3, 4, 8, 9, 11] scan all documents for every query and extract data specifically tailored to that query. While this approach yields high query-result accuracy, it incurs substantial latency and cost: each query is optimized mostly from scratch, requiring repeated document passes and expensive LLM invocations. Caching and re-using extraction plans can partially mitigate these overheads, but the fundamental inefficiency remains especially when the analytical queries are largely disjoint. While ZenDB [7] and QUEST [12] reduce cost and latency by avoiding full-document scanning via embedding-based retrieval, retrieval errors can still miss relevant content, reducing accuracy.

Finally, *Offline Extraction* is to extract data from all documents into a structured format (e.g., relational tables) during an offline phase, before any queries are observed. These systems [1, 2, 10] prioritize table extraction accuracy rather than query-specific performance. As a result, they achieve low latency and low cost, but often suffer from low accuracy, since the extraction is a one-shot process conducted without knowledge of the incoming queries. For example, due to lack of homogenization, they fail to join over keys indicating synonyms of the same entity (e.g., "USA" vs. "United States").

Our key observation is that the offline extraction phase, which effectively "converts" unstructured text documents into a structured format, should be guided by the expected query patterns, which can be obtained from historical query workloads. Such awareness allows (1) selection of an optimal schema with the necessary attributes for accurately answering queries and (2) extraction of relevant data, including essential post-processing steps such as homogenizing values and resolving equivalent entities into a canonical form. We illustrate the shortcoming of prior work in the three paradigms through Example 1 and then show how workload-aware offline extraction offers a better alternative in Example 2.

EXAMPLE 1. *Luna is a sports journalist who wants to figure out whether experienced players tend to play for more successful teams over 141 player documents and 30 team documents (Figure 2) and runs the following query:*

```
SELECT P.NAME, T.LOCATION, T.CHAMPIONSHIP
FROM PLAYER P
JOIN TEAM T
ON P.TEAM=T.NAME
WHERE P.AGE > 30;
```

The Player document stores birth year rather than age, and the team name in the player document ("Lakers") differs from the team name in the team document ("Los Angeles Lakers"). The expected result over the full dataset is:

Player	Location	Championship
LeBron James	Los Angeles	17
Kevin Durant	Phoenix	0
Chris Paul	San Francisco	7

We show how existing systems perform on Luna's query:

- ***RAG** [6] retrieves only the top-k documents most similar to the query. Since the age filter must be evaluated over every player and each qualifying player must be paired with their team document, RAG returns the following incomplete result:*

Player	Location	Championship
LeBron James	NULL	NULL

- ***ReDD** [3] generates relational tables on the fly by scanning all documents at query time. **DocETL** [11] runs a pipeline of LLM operations over document chunks at query time. Both return the correct result, but ReDD (DocETL) consumes over 7 million (800k) LLM tokens and takes around 50 minutes (250 seconds) per query. At scale, this per-query cost makes both systems impractical.*
- ***ZenDB** [7] and **QUEST** [12] extract each attribute from a small candidate region, which keeps cost and latency low. Since they examine only a limited text region per value, they miss mentions outside that region with the following result:*

Player	Location	Championship
LeBron James	Los Angeles	17
Kevin Durant	Phoenix	0

- **SQUID** *[10] pre-extracts data offline, generating one row per entity mention. Since player documents mention both past and current teams, each player can appear once for every team named in the text rather than only for the team that matches the query intent. This creates extra rows and leads to incorrect multi-row results, as shown below, even though latency is only 0.02 s.*

Player Document	***Team Document***
LeBron James was born in 1984. He currently plays for the Lakers and has won multiple MVP awards...	*The Los Angeles Lakers, based in Los Angeles, have won 17 NBA championships. The team was founded in...*

Figure 2: Example documents with highlighted information required for Luna's query: `PLAYER.NAME` ("LeBron James"), birth year (1984) to derive player's `AGE`, join keys player's `TEAM` ("Lakers") and team `NAME` ("Los Angeles Lakers"), `LOCATION` (city "Los Angeles"), and `CHAMPIONSHIP` count (17 NBA championships).

Player	Location	Championship
LeBron James	Los Angeles	17
LeBron James	Cleveland	1
Kevin Durant	Phoenix	0
Kevin Durant	San Francisco	7
Chris Paul	San Francisco	7
Chris Paul	Phoenix	0

EXAMPLE 2. *A workload-aware approach inspects the potential query patterns before extraction, noting filters on `PLAYER.AGE` and joins between `PLAYER.TEAM` and `TEAM.NAME`. During offline preprocessing, it derives each player's age from their birth year and applies entity resolution to normalize team names, storing both "Lakers" and "Los Angeles Lakers" as "Los Angeles Lakers," which yields the correct result. The synthesized tables are persisted, so any future query over player age, team names, or their join executes directly over them without re-reading any document.*

Example 1 and 2 show the main tradeoff between accuracy, cost, and latency. Systems that extract structured data at query time trade latency and cost for accuracy. In contrast, systems that preprocess documents once are efficient at query time, but without workload guidance they fail to derive required attributes, normalize values, and preserve relevant entities. A workload-aware system combines the advantages of both by performing extraction once offline while using the workload as guidance for optimal extraction.

QUWARTS. We present QUWARTS, a workload-aware system for analytical queries over unstructured text documents. QUWARTS takes as input a *reference workload*: a set of representative queries that the user has previously executed or intends to execute. It analyzes this workload upfront to determine what tables, attributes, and normalizations are needed, applies these steps once during offline preprocessing, and at query time executes queries directly over the synthesized tables—achieving accurate multi-table analytical processing with low cost and latency.

Related Work. Various approaches have been proposed to enable analytical queries over unstructured documents. *Retrieval-based systems* like RAG [6] answer directly from retrieved text, which is efficient but not well-suited to exhaustive analytical processing. *Query-time extraction* systems build structure when a query arrives: some read full documents for higher accuracy at high per-query cost and latency [3, 4, 8, 9, 11], while others extract from selected regions for lower cost and latency but lower accuracy [7, 12]. *Offline extraction* systems pre-build tables before queries arrive [1, 2, 10], giving low query-time latency, but without workload guidance they cannot determine how to normalize values or resolve entity mentions.

Demonstration. Our demonstration showcases QUWARTS's capabilities using several real-world datasets in a comparative analysis

setting. Participants will observe how workload-aware preprocessing enables fast, accurate analytical queries over unstructured text documents. We will demonstrate (1) query execution showing high accuracy with low-cost and low-latency answers, (2) handling of multi-table joins that existing systems struggle with, and (3) comparative analysis of accuracy, latency, and cost trade-offs against baselines ReDD [3] and SQUID [10].

We provide an overview of QUWARTS's architecture, optimization techniques, and present some preliminary results in Section 2. Then in Section 3, we present a concrete demonstration scenario on the UDA-Bench NBA dataset [5].

2 SYSTEM OVERVIEW

QUWARTS operates in two stages: an offline stage that analyzes the workload and prepares tables, and an online stage that serves queries. We first describe how the offline stage uses the reference workload to decide what information to extract and store, then explain how the online stage serves queries over these pre-built tables.

Offline Stage. During the offline stage, QUWARTS uses the reference workload to determine which tables to extract and how to populate them with data. This stage consists of two components, schema discovery—which discovers the structure of the tables to extract—and data population—which fills the tables with data.

Schema Discovery. To reliably support analytical operations like filtering, joining, and aggregating, QUWARTS extracts structured relational tables from unstructured text. QUWARTS analyzes the reference workload to determine what tables, attributes, and relationships to extract. For each table, it identifies a primary key column (e.g., player name, company name) that uniquely identifies entities.

Data Population. After determining the schema, data is extracted from the document corpus and loaded into the tables. By extracting data offline, queries can execute directly over pre-built tables at query time, achieving low latency and cost. To ensure high query accuracy, QUWARTS uses the workload to guide extraction.

Running LLM extraction on every document chunk is expensive. QUWARTS uses Evaporate [2] to generate a Python function for each table that identifies whether a chunk is relevant to that table. Only the chunks identified as relevant proceed to LLM for extraction.

To address entity normalization challenges discussed in Example 2, QUWARTS identifies the primary entity for each relevant chunk using the primary key column, then applies entity resolution to consolidate variations into a canonical form. We use BLINK [13] for its high accuracy on entity linking tasks, though QUWARTS can work with any entity resolution system. BLINK uses a bi-encoder to find candidate matches via embedding similarity, then a cross-encoder to refine these matches.

Extracted values must match the formats used in the query predicates. For example, if the workload contains queries that use `USA` in `WHERE country = 'USA'`, but extracted values use "United States", queries involving this predicate will not produce the correct results. QUWARTS extracts and normalizes attribute values to match the formats found in the workload.

New queries may reference attributes not in the training workload. Extracting all possible attributes offline would be very expensive since documents can mention many attributes. Instead, QUWARTS builds an attribute index during preprocessing that maps each chunk to the attributes it mentions. The index is kept in case new queries need attributes outside the reference workload, enabling efficient online extraction only when needed.

Online Stage. Queries that use attributes already in the database execute directly over the pre-computed tables, achieving low latency and cost. However, new queries may need attributes not seen in the reference workload. QUWARTS performs column augmentation for these cases while maintaining low latency and cost: it uses the attribute index to locate chunks mentioning that attribute, extracts values for all entities, and adds the new column to the table.

Evaluation. We evaluate QUWARTS on the NBA dataset from UDA-Bench [5], using 80% of the queries as the reference workload during the offline preprocessing stage and testing on the remaining 20% of queries. All systems use Qwen2.5:7b-instruct as the underlying LLM. We compare QUWARTS against state-of-the-art systems including ReDD [3], SQUID [10], DocETL [11], and Palimpzest [8]. QUWARTS achieves the best accuracy-cost-latency trade-off: on average per query, it uses only 7,175 tokens compared to 4.2M for ReDD and 862K for DocETL, while achieving 0.44 F1 score compared to 0.23-0.32 for other systems. Average query latency is 3.92 seconds, compared to 1,722 seconds for ReDD and 258 seconds for DocETL. Unlike systems that sacrifice accuracy for efficiency (e.g., SQUID), QUWARTS delivers both high F1 score and low cost through its workload-aware preprocessing approach.

3 DEMONSTRATION SCENARIO

We demonstrate QUWARTS using the NBA dataset from UDA-Bench [5]. We guide participants through the QUWARTS interface (annotated in Figure 3) impersonating Luna, a sports journalist analyzing veteran NBA players.

Steps (A1), (A2) & (A3) (Setup) Luna selects the NBA Dataset (from UDA-Bench [5]) in (A1). She then selects the NBA Players workload in (A2) — the workload of queries that QUWARTS analyzes upfront to determine what tables and attributes to extract during offline preprocessing. In (A3), Luna adds SQUiD and ReDD as baselines to compare against QUWARTS. She has the option to add other systems as well, such as DocETL.

Step (B1) (Custom query input) While Luna uses preset queries in this demo, the Custom Query panel in (B1) provides a way for her to write and enter her own new queries.

Step (B2) (Selecting a preset query) Luna clicks Query Q_1 from the Preset Workload panel shown in (B2). The preset workload panel is populated with test queries that were excluded from the reference workload and not seen during offline preprocessing, allowing Luna to evaluate QUWARTS on unseen queries.

Step (C) (System configuration) In the System Knobs panel, Luna chooses BLINK as the tool for matching and linking entity names, and she picks Qwen2.5:7b-instruct as the LLM to use for extracting information from the documents.

Step (D) (Viewing extracted tables) After executing the query, Luna inspects the results in (D). QUWARTS finds 20 rows in total, initially displaying three teams (Chicago Bulls in Chicago with 3 players, Boston Celtics in Boston with 7 players, Dallas Mavericks in Dallas with 1 player). Preprocessing took 1 hour 8 minutes for QUWARTS versus 2 hours 13 minutes for SQUID, while ReDD requires no preprocessing.

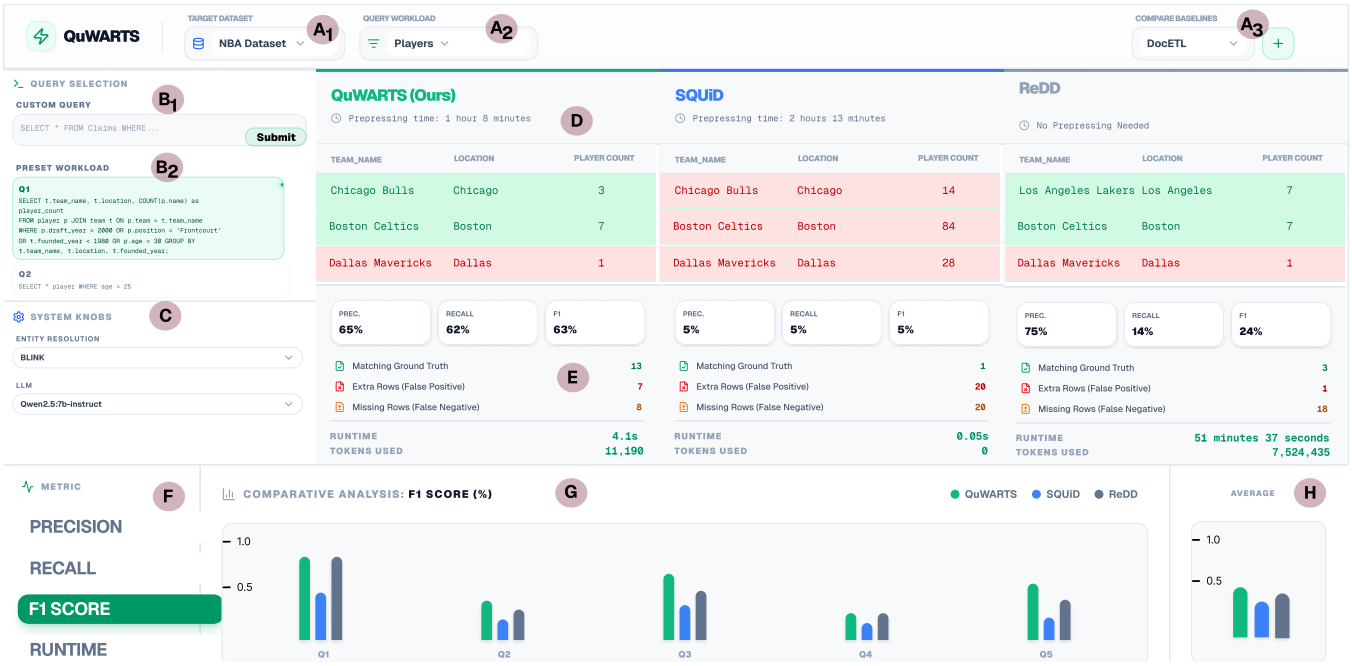

Figure 3: The QUWARTS interface: (A1) select target dataset, (A2) select query workload, (A3) add baselines for comparison, (B1) enter a custom query, (B2) choose a query from the preset workload, (C) adjust system knobs, (D) preview extracted tables and preprocessing time, (E) inspect detailed evaluation metrics and row breakdown for each query, (F) select a metric for comparison, (G) view comparative analysis across queries, (H) view overall average performance.

Step (E) (Analyzing quality metrics) In (E), Luna examines the metrics. QUWARTS achieves 63% F1 Score (65% Precision, 62% Recall) with 13 matching rows, completing in 4.1s using 11,190 tokens. SQUID shows only 5% F1 with 1 matching row in 0.05s, while ReDD achieves 24% F1 but takes 51 minutes 37 seconds and 7.5M tokens. QUWARTS balances the tradeoffs: it is significantly faster than ReDD while maintaining better accuracy, and achieves much higher F1 than SQUID with minimal latency increase.

Steps (F) & (G) (Comparative analysis) Luna selects F1 Score in (F) and views the comparative charts in (G), which shows that QUWARTS (green) consistently outperforms SQUID (blue) and ReDD (gray) across queries.

Step (H) (Overall performance) In (H), the average metrics confirm QUWARTS achieves the best performance.

Beyond the NBA scenario, we will demonstrate QUWARTS on other UDA-Bench datasets [5], spanning finance, healthcare, computer science literature, art, and legal documents, showing how workload-aware extraction supports analytical queries over diverse unstructured corpora. The key takeaway from our demo is that participants can customize the setup by selecting datasets, query workloads, baselines, and system knobs, while QUWARTS uses the chosen workload to steer offline extraction toward the query patterns analysts care about, making accuracy, latency, and cost tradeoffs easy to compare side by side.

REFERENCES

[1] N. Ahuja, F. D. Bardoliya, C. Baral, and V. Gupta. 2025. Map&Make: Schema Guided Text to Table Generation. In *ACL*.

[2] S. Arora, B. Yang, S. Eyuboglu, A. Narayan, A. Hojel, I. Trummer, and C. Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *PVLDB* (2023).

[3] D. Chao, K. Chen, N. Guan, and N. Koudas. 2025. Relational Deep Dive: Error-Aware Queries Over Unstructured Data. *CoRR* abs/2511.02711 (2025).

[4] H. Dai, B. Wang, X. Wan, B. Dai, S. Yang, A. Nova, P. Yin, P. M. Phothilimthana, C. Sutton, and D. Schuurmans. 2024. UQE: A Query Engine for Unstructured Databases. In *NeurIPS*.

[5] Q. Deng, J. Li, C. Chai, J. Liu, J. She, K. Jin, Z. Sun, Y. Deng, J. Yuan, Y. Yuan, G. Wang, and L. Cao. 2025. Unstructured Data Analysis using LLMs: A Comprehensive Benchmark. *CoRR* abs/2510.27119 (2025).

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[7] Y. Lin, M. Hulsebos, R. Ma, S. Shankar, S. Zeighami, A. G. Parameswaran, and E. Wu. 2025. Querying Templatized Document Collections with Large Language Models. In *ICDE*.

[8] C. Liu, M. Russo, M. J. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. J. Franklin, T. Kraska, S. Madden, R. Shahout, and G. Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *CIDR*.

[9] L. Patel, S. Jha, M. Z. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia. 2025. Semantic Operators and Their Optimization: Towards AI-Based Data Analytics with Accuracy Guarantees. *PVLDB* (2025).

[10] M. Sadia, Z. Yang, Y. Xiao, A. Chen, and A. R. Chowdhury. 2025. SQUiD: Synthesizing Relational Databases from Unstructured Text. In *EMNLP*.

[11] S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *PVLDB* (2025).

[12] Z. Sun, C. Chai, Q. Deng, K. Jin, X. Guo, H. Han, Y. Yuan, G. Wang, and L. Cao. 2025. QUEST: Query Optimization in Unstructured Document Analysis. *PVLDB* (2025).

[13] L. Wu, F. Petroni, M. Josifoski, S. Riedel, and L. Zettlemoyer. 2020. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP*.