Text2SQL is Not Enough: Unifying AI and Databases with TAG

Asim Biswal[1,*] Liana Patel[2,*] Siddharth Jha[1] Amog Kamsetty[1] Shu Liu[1]
Joseph E. Gonzalez[1] Carlos Guestrin[2] Matei Zaharia[1]
[1]UC Berkeley [2]Stanford University

arXiv:2408.14717v1 [cs.DB] 27 Aug 2024

ABSTRACT

AI systems that serve natural language questions over databases promise to unlock tremendous value. Such systems would allow users to leverage the powerful reasoning and knowledge capabilities of language models (LMs) alongside the scalable computational power of data management systems. These combined capabilities would empower users to ask arbitrary natural language questions over custom data sources. However, existing methods and benchmarks insufficiently explore this setting. Text2SQL methods focus solely on natural language questions that can be expressed in relational algebra, representing a small subset of the questions real users wish to ask. Likewise, Retrieval-Augmented Generation (RAG) considers the limited subset of queries that can be answered with point lookups to one or a few data records within the database. We propose Table-Augmented Generation (TAG), a unified and general-purpose paradigm for answering natural language questions over databases. The TAG model represents a wide range of interactions between the LM and database that have been previously unexplored and creates exciting research opportunities for leveraging the world knowledge and reasoning capabilities of LMs over data. We systematically develop benchmarks to study the TAG problem and find that standard methods answer no more than 20% of queries correctly, confirming the need for further research in this area. We release code for the benchmark at https://github.com/TAG-Research/TAG-Bench.

1 INTRODUCTION

Language models promise to revolutionize data management by letting users ask natural language questions over data, which has led to a great deal of research in Text2SQL and Retrieval-Augmented Generation (RAG) methods. In our experience, however (including from internal workloads and customers at Databricks), users' questions often transcend the capabilities of these paradigms, demanding new research investment towards systems that combine the logical reasoning abilities of database systems with the natural language reasoning abilities of modern language models (LMs).

In particular, we find that real business users' questions often require sophisticated combinations of domain knowledge, world knowledge, exact computation, and semantic reasoning. Database systems clearly provide a source of ***domain knowledge*** through the up-to-date data they store, as well as ***exact computation*** at scale (which LMs are bad at).

LMs offer to extend the existing capabilities of databases in two key ways. First, LMs possess ***semantic reasoning*** capabilities over textual data, a core element of many natural language user queries. For example, a Databricks customer survey showed users wish to ask questions like *which customer reviews of product X are positive?*, or *why did my sales drop during this period?*. These questions present complex reasoning-based tasks, such as sentiment analysis over free-text fields or summarization of trends. LMs are well-suited to these tasks, which cannot be modeled by the exact computation or relational primitives in traditional database systems.

Secondly, the LM, using knowledge learned during model training and stored implicitly by the model's weights, can powerfully augment the user's data with ***world knowledge*** that is not captured explicitly by the database's table schema. As an example, a Databricks internal AI user asked *what are the QoQ trends for the "retail" vertical?* over a table containing attributes for account names, products and revenue. To answer this query the system must understand how the business defines *QoQ* (e.g., the quarter over quarter trends from the last quarter to the current quarter or this quarter last year to this quarter this year), as well as which companies are considered to be in the *retail vertical*. This task is well-suited to leverage the knowledge held by a pre-trained or fine-tuned LM.

Systems that efficiently leverage databases and LMs together to serve natural language queries, in their full generality, hold potential to transform the way users understand their data. Unfortunately, these questions cannot be answered today by common methods, such as Text2SQL and RAG. While Text2SQL methods [26, 28, 31, 32] are suitable for the subset of natural language queries that have direct relational equivalents, they cannot handle the vast array of user queries that require semantic reasoning or world knowledge. For instance, the previous user query asking *which customer reviews are positive* may require logical row-wise LM reasoning over reviews to classify each as positive or negative. Similarly the question which asks *why sales dropped* entails a reasoning question that must aggregate information across many table entries.

On the other hand, the RAG model is limited to simple relevance-based point lookups to a few data records, followed by a single LM invocation. This model serves only the subset of queries answerable by point lookups and also fails to leverage the richer query execution capabilities of many database systems, which leaves computational tasks (e.g., counting, math and filtering) to a single invocation of the error-prone LM. In addition to being error prone and inefficient at computational tasks, LMs have also been shown to perform poorly on long-context prompts limiting their ability to reason about data at scale in the generation phase of RAG.

We instead propose ***table-augmented generation (TAG)*** as a unified paradigm for systems that answer natural language questions over databases. Specifically, TAG defines three key steps, as shown in Figure 1. First, the query synthesis step `syn` translates the user's arbitrary natural language request R to an executable database query Q. Then, the query execution step `exec` executes Q on the database system to efficiently compute the relevant data T. Lastly, the answer generation step `gen` utilizes R and T, where the LM is orchestrated, possibly in iterative or recursive patterns over the data, to generate the final natural language answer A. The TAG model is simple, but powerful: it is defined by the following three

*Both authors contributed equally to this research.

equations, but captures a wide range of previously under-studied interactions between LMs and databases.

$$\text{Query Synthesis:} \quad \text{syn}(R) \rightarrow Q \tag{1}$$

$$\text{Query Execution:} \quad \text{exec}(Q) \rightarrow T \tag{2}$$

$$\text{Answer Generation:} \quad \text{gen}(R,\ T) \rightarrow A \tag{3}$$

Notably, the TAG model unifies prior methods, including both Text2SQL and RAG, which represent special cases of TAG and serve only a limited subset of user questions.

While several prior works address these special cases of TAG, we provide the first end-to-end TAG benchmark composed of a broad set of realistic queries that require LM reasoning and knowledge capabilities. We demonstrate the significant research challenges posed by these types of questions, as well as the promise of efficient TAG implementations. Our evaluation analyzes the vanilla Text2SQL and RAG baselines as well as two stronger baselines, Text2SQL with LM generation and retrieval with LM-based re-ranking. Across a variety of query types, we find each baseline method consistently fails to achieve high accuracy, never surpassing 20% exact match on the benchmark. On the other hand, we implement hand-written TAG pipelines on top of the recent LOTUS runtime [21] and find they achieve up to $20-65\%$ higher accuracy compared to the baselines. This significant performance gap demonstrates the promise of building efficient TAG systems.

2 THE TAG MODEL

We now describe the TAG model, which takes a natural language request R and returns a natural language answer A grounded in the data source. We outline three main steps that TAG systems implement: query synthesis, query execution, and answer generation. We define TAG tractably as a single iteration of these steps, but one can consider extending TAG in a multi-hop fashion.

2.1 Query Synthesis

The syn function takes a natural language request R and generates a query Q to be executed by the database system. Given a user request, this step is responsible for **(a)** deducing which data is relevant to answering the request (e.g., using the table schema), and **(b)** performing semantic parsing to translate the user request into a query that can be executed by the database system. This query could be in any query language, but in our example we use SQL.

Figure 1 shows an example TAG instantiation for the user query which asks, *"Summarize the reviews of the highest grossing romance movie considered a 'classic'"*. Here, the data source contains information about each movie's title, revenue, genre, and an associated review. In this step, the system leverages the semantic reasoning abilities of the LM to generate a SQL query that uses attributes on `movie_title`, `review`, `revenue`, and `genre` from the data source. Note that in this example, the database API is able to execute LM UDFs within SQL queries, so this step also introduces calls to the LM for each row to identify classic films within the query.

2.2 Query Execution

In the query execution step, the exec function executes the query Q in the database system to obtain the table T. This step leverages the database query engine to efficiently execute the query over vast amounts of stored data. The database API can be served by a wide variety of systems, which we explore in Section 3. Some APIs may allow for LM-based operators [18–21], permitting the database engine to leverage the LM's world knowledge and reasoning capabilities within exec.

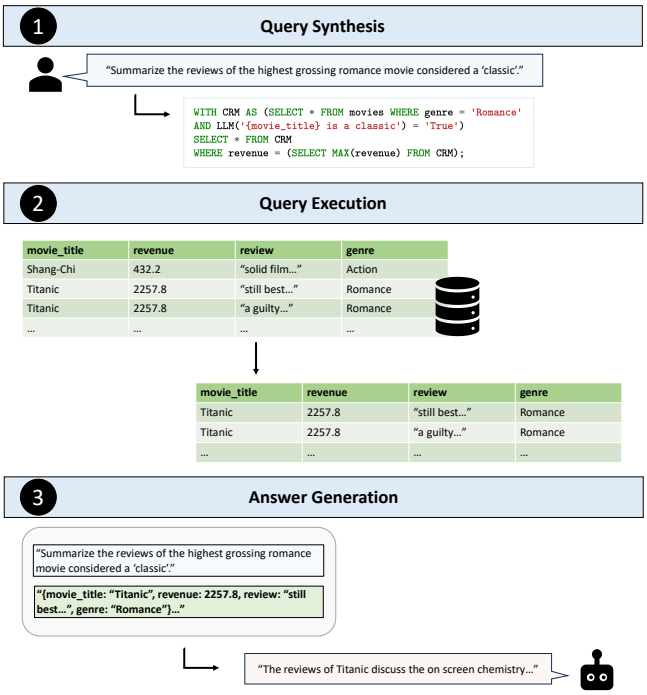

Figure 1: An example TAG implementation for answering the user's natural language question over a table about movies. The TAG pipeline proceeds in three stages: query synthesis, query execution, and answer generation

In the example shown in Figure 1, the database query is a selection and ranking query written in SQL, which returns a table containing relevant rows. The query performs the selection using an LM to assess which movies are classics according to their `movie_title`, as well as a standard filter on `genre` to find romance movies. The query also ranks the results based on `revenue` to find the highest grossing film. As the figure shows, the resulting table contains reviews for the movie "Titanic".

2.3 Answer Generation

The answer generation step in TAG mirrors the generation step in RAG. In this step, the gen function uses the LM to generate an answer A to the user's natural language request R, using the computed data T.

Figure 1 shows the final stage of the example TAG pipeline outputting a summary of the reviews on "Titanic" as the answer to the original user request. In the example, the relevant data T is encoded as a string for the model to process. The encoded table is passed to the LM along with the original user request, R. To obtain the answer, this step leverages the model's semantic reasoning capabilities over the `review` column to summarize the reviews.

3 TAG DESIGN SPACE

In this section, we explore the generality of the TAG model and describe the rich design space it produces, highlighting several under-studied opportunities for further research.

Query Types The TAG model is expressive enough to serve a broad range of natural language user queries. We consider two important query categorizations, according to **(a)** the *level of data aggregation* required to answer the query and **(b)** the *knowledge and capabilities* needed for answering the query. First, the TAG model captures both point queries, such as retrieval-based questions [3, 9, 15, 16, 25, 30] which require look-ups to one or a few rows of the database, as well as aggregation queries, such as summarization or ranking-based questions which require logical reasoning across many rows of the database. Secondly, the TAG model enables natural language queries with varying demands on the system to provide data or reasoning-based capabilities, including for tasks such as sentiment analysis and classification.

Data Model The underlying data model can take many forms. Our implementation uses relational databases to store and retrieve structured attributes for knowledge-grounding in the downstream question-answering task. Others may operate on more unstructured data (e.g., free-text, images, video, and audio) or semi-structured data, which may be stored with a variety of data models, such as key-value, graph, vector, document, or object stores.

Database Execution Engine and API The underlying system used to store the data can use many possible database execution engines. Text2SQL considers the setting of an SQL-based query engine for retrieving relational data for user queries. In this setting, `syn` will leverage the knowledge of the data source, such as the table schema, and return a SQL query to perform the retrieval step. In another common setting, retrieval systems over vector embeddings, `syn` transforms the natural language query into an embedding and `exec` performs similarity-based retrieval over the vector store.

While these two settings have been widely studied, several under-studied alternative settings present interesting opportunities for efficiently implementing TAG systems to serve a broader range of queries. For instance, recent works augment relational models with semantic operators [21], which provide a set of declarative AI-based operators (e.g., filtering, ranking, aggregating, and performing search with natural language specifiers) or LM user-defined functions [19], which provide a general-purpose LM function. Additionally, query languages like MADLib [10], Google's BigQuery ML [1], and Microsoft's Predictive SQL [24] augment SQL-based APIs with native ML-based functions. Leveraging these systems provides unique opportunities for executing optimized reasoning-based retrieval patterns. For instance, in the example shown in Figure 1, a TAG pipeline implemented with semantic operators [21] might use a `sem_filter` operator to filter rows based on whether they are a 'classic' during the query execution step.

LM Generation Patterns Given the table T of relevant data, `gen` can be comprised from a vast array of implementation decisions to produce the final natural language answer A in response to the user request R. While Text2SQL omits the final generation step and stops short after query execution, RAG pipelines typically leverage a single LM-call generation implementation where relevant data is fed in context. In this setting, several works study sub-problems related to table encoding [8], prompt compression [5], and prompt tuning [13] to optimize the in-context learning results.

More recent research, such as LOTUS [21], highlights the potential of composing iterative or recursive LM generation patterns for answering queries involving reasoning-based transformations, aggregations, or rankings across multiple data rows. Early work demonstrates the rich design space presented by these LM-based algorithms and promising results on several downstream tasks.

4 EVALUATION

In this section, we introduce the first TAG benchmark and evaluate a collection of baselines, aiming to address the following questions:

(1) How do existing methods for table question answering perform on queries requiring *semantic reasoning* or *world knowledge*?
(2) How does a hand-written implementation of the TAG model, which divides computational and reasoning steps across DBMS and LM operations, perform on these queries?

4.1 Benchmark Methodology

Existing benchmarks have explored how models perform on basic queries answerable entirely from data in the data source. We build upon prior work by modifying queries such that they require knowledge not directly available in the data source or semantic reasoning to answer. We select BIRD [17], a widely used Text2SQL benchmark on which LMs have been evaluated, for its large scale tables along with its variety of domains and query types.

Dataset Our queries span 5 domains selected from BIRD, each containing diversity in query types. We select *california_schools*, *debit_card_specializing*, *formula_1*, *codebase_community*, and *european_football_2* as the DB sources for our queries.

Queries The BIRD benchmark defines fundamental query types, including match-based, comparison, ranking, and aggregation queries. We select queries among these types from the BIRD benchmark and modify them to require either world knowledge or semantic reasoning for the model to answer. As an example of a modified query requiring world knowledge, in the *california_schools* DB, a modified query adds an additional clause asking for only schools in the Bay Area. This information is not in the table and requires the model's world knowledge to answer. Next, a modified query requiring LM reasoning asks for the top 3 most sarcastic comments on a particular post in the *codebase_community* DB. For evaluation on these queries, we rely on human-labeled ground truth. Our final benchmark consists of 80 modified queries, 40 requiring parametric knowledge and 40 requiring reasoning, with 20 of each of the 4 chosen BIRD query types.

Evaluation metrics We measure accuracy as the percentage of exact matches as compared to the labeled correct answer for the

match-based, comparison, and ranking query types. For aggregation queries, we provide qualitative analysis on results using each baseline. We also measure execution time in seconds for each query.

Experimental setup We use the instruction tuned variant of Meta's Llama-3.1 model [2] with 70B parameters as our LM for both Text2SQL and final output generation. We use SQLite3 as our database API for baselines involving SQL and use an E5 base embedding model [23] for our RAG baseline. We run Llama-3.1-70B-Instruct with vLLM [14] on 8 A100 80GB GPUs.

4.2 Baselines

Text2SQL In this baseline, the LM generates SQL code which is run to obtain an answer. For a given NL query, we construct an LM prompt containing table schemas for every table in the query's domain, using the same prompt format as in the BIRD work. We evaluate this baseline executing the generated SQL code in SQLite3 and measuring the number of incorrect answers, including instances where the model fails to generate valid SQL code.

Retrieval Augmented Generation (RAG) RAG style methods have been explored for table retrieval [6, 25], where tabular data is embedded into an index for search. For our baseline, we use *row-level embeddings*. A given row is serialized as "- col: val" for each column before being embedded into a FAISS [7] index. During query time, we perform vector similarity search to retrieve 10 relevant rows to feed in context to our model along with the NL question.

Retrieval + LM Rank We extend the RAG baseline by utilizing an LM to assign a score between 0 and 1 for retrieved rows to rerank rows before input to the model, as is done in the STaRK work [25]. We use Llama-3.1-70B-Instruct as our reranker.

Text2SQL + LM In this baseline, our model is first asked to generate SQL to retrieve a set of relevant rows to answer a given NL query. This is an important distinction from the Text2SQL baseline, where the model is asked to directly generate SQL code that alone provides an answer to the query when executed. Similar to the RAG baseline, relevant rows are fed in context to the model once retrieved.

Hand-written TAG We also evaluate hand-written TAG pipelines, which leverage expert knowledge of the table schema rather than automatic query synthesis from the natural language request to the database query. We implement our hand-written TAG pipelines with LOTUS [21]. The LOTUS API allows programmers to declaratively specify query pipelines with standard relational operators as well as semantic operators, such as LM-based filtering, ranking, and aggregations. LOTUS also provides an optimized semantic query execution engine, which we use to implement the query execution and answer generation steps of our hand-written TAG pipelines.

4.3 Results

Table 1 shows the accuracy, measured by exact match, and execution time of each method. As the table shows, across the selected BIRD query types, we find that our hand-written TAG baseline consistently achieves 40% exact match accuracy or better, where all other baselines fail to exceed 20% accuracy.

The Text2SQL baseline performs poorly on all baselines with an execution accuracy no higher than 20% but especially poorly on ranking queries with only 10% accuracy, as many of the ranking queries require reasoning over text. The Text2SQL + LM generation baseline has similar poor performance across baselines, but does worse on match-based and comparison queries with only 10% accuracy. On these query types, several context length errors occur trying to feed in many rows to the model after the executed SQL.

Turning our attention to the RAG baseline, we see that it fails to answer a single query correctly across all query types, highlighting its poor fit for queries in this space. Adding LM reranking allows Retrieval + LM rank to answer a query correctly among the comparison queries, however the baseline still performs worse than all others besides RAG.

Our hand-written TAG baseline answers 55% of queries correctly overall, performing best on comparison queries with an exact match accuracy of 65%. The baseline performs consistently well with over 50% accuracy on all query types except ranking queries, due to the higher difficulty in ordering items exactly. Overall, this method gives us between a 20% to 65% accuracy improvement over the standard baselines.

Additionally, Table 2 highlights the weaknesses of standard methods in answering the query types discussed in Section 3. Namely, vanilla Text2SQL especially struggles on queries requiring LM reasoning with 10% exact match accuracy, due to its omission of the answer generation step. Meanwhile, the RAG baseline and Retrieval + LM Rank baseline struggle on all query types, answering only one query correctly, due to their reliance on the LM to handle all exact computation over data. In contrast, the hand-written TAG baseline achieves over 50% accuracy on both queries requiring knowledge and queries requiring reasoning, emphasizing the TAG model's versatility in the queries it encapsulates.

Notably, along with offering superior accuracy, the hand-written TAG method offers an efficient implementation with up to 3.1× lower execution time over other baselines. The hand-written baseline takes an average of 2.94 seconds for all queries. This relatively low execution time highlights that an efficient TAG system can be designed by exploiting efficient batched inference of LMs.

Lastly, we qualitatively analyze the results of each baseline on aggregation queries. Figure 2 shows the results for the RAG, Naive TAG, and hand-written baselines on the example query *"Provide information about the races held on Sepang International Circuit."*. The RAG baseline is only able to provide information about some of the races, as most of the relevant races are not retrieved. On the other hand, the Text2SQL + LM baseline is not able to utilize any information from the DBMS, relying only on parametric knowledge and providing no further analysis. The hand-written baseline provides a thorough summary of all the races from 1999 to 2017 held at Sepang International Circuit. We observe a similar trend across other aggregation queries provided by the benchmark, with initial results highlighting the potential of TAG systems to successfully aggregate large amounts of data to provide informative answers. We leave quantitative analysis to future work.

Table 1: Accuracy and execution time (ET) for TAG benchmark queries, averaged over all queries and each query type: TAG significantly improves answer quality while achieving the fastest or nearly fastest execution time.

Method	Overall		Match-based		Comparison		Ranking		Aggregation	
	Exact Match ↑[1]	ET (s) ↓	Exact Match ↑	ET (s) ↓	Exact Match ↑	ET (s) ↓	Exact Match ↑	ET (s)	Exact Match ↑	ET (s) ↓
Text2SQL	0.17	5.63	0.20	4.72	0.20	4.01	0.10	7.26	N/A	6.53
RAG	0.00	3.23	0.00	3.73	0.00	**2.29**	0.00	**2.01**	N/A	4.89
Retrieval + LM Rank	0.02	4.82	0.00	6.20	0.05	4.19	0.00	3.42	N/A	5.46
Text2SQL + LM	0.13	9.08	0.10	11.25	0.10	3.89	0.20	11.80	N/A	9.38
Hand-written TAG	**0.55**	**2.94**	**0.60**	**1.70**	**0.65**	5.05	**0.40**	2.50	N/A	**2.50**

[1] Excludes aggregation since accuracy is not measured.

Query: Provide information about the races held on Sepang International Circuit

RAG	Text2SQL + LM	Hand-written TAG
Based on the given data points, the following information is available about the races held on Sepang International Circuit: The races held on this circuit are the Malaysian Grand Prix. The location of the circuit is Kuala Lumpur, Malaysia. The following races were held on this circuit: 2001 Malaysian Grand Prix, 2003 Malaysian Grand Prix, 2002 Malaysian Grand Prix, 2000 Malaysian Grand Prix.	The data points provided do not contain specific information about the races held on Sepang International Circuit. However, based on general knowledge, the Sepang International Circuit is a racing circuit in Sepang, Malaysia, and it has hosted the Malaysian Grand Prix, a Formula One World Championship event, from 1999 to 2017.	The Sepang International Circuit in Kuala Lumpur, Malaysia, hosted the Malaysian Grand Prix from 1999 to 2017. The races were held on the following dates: 1999: October 17 (15th round) 2000: October 22 (2nd round), 2001: March 18 (2nd round), 2002: March 17 (2nd round), 2003: March 23 (2nd round), 2004: March 21 (2nd round), 2005: March 20 (2nd round), ..., 2016: October 2 (16th round), 2017: October 1 (15th round).

Figure 2: Example Aggregation Results: The RAG baseline provides an incomplete answer to the query while Text2SQL + LM fails to answer the question using any data from the DB. The Hand-written TAG baseline provides the most thorough answer, synthesizing data from the DB and its own world knowledge.

Table 2: TAG benchmark results averaged over queries requiring Knowledge or Reasoning: TAG performs consistently well with above 50% exact match accuracy on both Knowledge and Reasoning query types.

Method	Knowledge		Reasoning	
	Exact Match ↑	ET (s) ↓	Exact Match ↑	ET (s) ↓
Text2SQL	0.20	5.23	0.10	5.52
RAG	0.00	**2.73**	0.00	2.58
Retrieval + LM Rank	0.03	4.97	0.00	3.87
Text2SQL + LM	0.10	10.27	0.20	6.39
Hand-written TAG	**0.53**	3.50	**0.60**	**2.24**

5 RELATED WORK

Text2SQL Text2SQL using LMs has been extensively explored in prior work. WikiSQL [33], Spider [29], and BIRD [17] are all popular datasets for cross-domain Text2SQL. These datasets contain structured data across many domains on which the task of converting natural language queries to SQL is evaluated. However, this direction does not utilize model capabilities beyond SQL generation, keeping queries that require reasoning or knowledge beyond a static data source out of scope.

Retrieval Augmented Generation Retrieval augmented generation (RAG) [16] enables LMs to extend beyond their parametric knowledge to large collections of text. SQuAD [22] and HotPotQA [27] focus on question-answering over single document and multiple document sources respectively. The dense table retrieval (DTR) model [11] extends RAG to tabular data, embedding tabular context to retrieve relevant cells and rows for a query. Join-aware table retrieval [6] adds a table-table similarity score term to the DTR model to improve performance on complex queries involving joined tables. In contrast to prior RAG work, the TAG model encompasses a larger field of queries users have over their data by leveraging LM capabilities in the query execution step and allowing DBMS operations for exact computation over large amounts of data.

NL Queries over Semi-structured Data Prior work has explored the relational information between table entities and unstructured entity fields in semi-structured data sources. STaRK [25] evaluates table retrieval methodologies across semi-structured knowledge bases (SKBs), including both structural and nonstructural information. SUQL [20] addresses the task of conversational search, where an LM is used as a semantic parser to handle unstructured components user queries over hybrid data. While these works primarily focus on natural language *search* queries over semi-structured data, we seek to explore a broader range of queries leveraging more LM capabilities for tasks beyond search and lookup.

Agentic Data Assistants Recent work has explored LM agents as data assistants [12]. Spider2-V [4] explores multimodal agent

performance in tasks involving code generation and GUI controls. Though we define the TAG model tractably as one iteration of the syn, exec, and gen functions, future work may explore extending this in an agentic loop.

6 CONCLUSION

In this work we proposed table-augmented generation (TAG) as a unified model for answering natural language questions over databases. We developed benchmarks to study two important types of queries: those that require world knowledge, and those that require semantic reasoning capabilities. Our systematic evaluation confirms that baseline methods are unable to make meaningful traction on these tasks. However, hand-written TAG pipelines can achieve up to 65% higher accuracy, demonstrating substantial research opportunities for building TAG systems.

7 ACKNOWLEDGEMENTS

This research was supported in past by affiliate members and supporters of the Stanford DAWN project and the Sky Computing Lab at Berkeley, including Accenture, AMD, Anyscale, Cisco, Google, IBM, Intel, Meta, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, NVIDIA, Samsung SDS, SAP, VMware, and a Sloan Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] [n. d.]. Introduction to AI and ML in BigQuery | Google Cloud. https://cloud.google.com/bigquery/docs/bqml-introduction

[2] 2024. https://ai.meta.com/blog/meta-llama-3-1/

[3] Raviteja Anantha, Svitlana Vakulenko, Zhucheng Tu, Shayne Longpre, Stephen Pulman, and Srinivas Chappidi. 2021. Open-Domain Question Answering Goes Conversational via Question Rewriting. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 520–534. https://doi.org/10.18653/v1/2021.naacl-main.44

[4] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor Zhong, Lu Chen, Kai Yu, and Tao Yu. 2024. Spider2-V: How Far Are Multimodal Agents From Automating Data Science and Engineering Workflows? arXiv:2407.10956 [cs.AI] https://arxiv.org/abs/2407.10956

[5] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. http://arxiv.org/abs/2305.05176 arXiv:2305.05176 [cs].

[6] Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is Table Retrieval a Solved Problem? Exploring Join-Aware Multi-Table Retrieval. arXiv:2404.09889 [cs.IR] https://arxiv.org/abs/2404.09889

[7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[8] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding – A Survey. https://doi.org/10.48550/arXiv.2402.17944 arXiv:2402.17944 [cs].

[9] Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. 2023. RARR: Researching and Revising What Language Models Say, Using Language Models. https://doi.org/10.48550/arXiv.2210.08726 arXiv:2210.08726 [cs].

[10] Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library or MAD Skills, the SQL. https://doi.org/10.48550/arXiv.1208.4165 arXiv:1208.4165 [cs].

[11] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. arXiv:2103.12011 [cs.CL] https://arxiv.org/abs/2103.12011

[12] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. arXiv:2401.05507 [cs.CL] https://arxiv.org/abs/2401.05507

[13] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. https://doi.org/10.48550/arXiv.2310.03714 arXiv:2310.03714 [cs].

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[15] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 6086–6096. https://doi.org/10.18653/v1/P19-1612

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. http://arxiv.org/abs/2005.11401 arXiv:2005.11401 [cs].

[17] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. http://arxiv.org/abs/2305.03111 arXiv:2305.03111 [cs].

[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. http://arxiv.org/abs/2405.14696 arXiv:2405.14696 [cs].

[19] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E. Gonzalez, Ion Stoica, and Matei Zaharia. 2024. Optimizing LLM Queries in Relational Workloads. https://doi.org/10.48550/arXiv.2403.05821 arXiv:2403.05821 [cs].

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

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

[22] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250 [cs.CL] https://arxiv.org/abs/1606.05250

[23] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2024. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv:2212.03533 [cs.CL] https://arxiv.org/abs/2212.03533

[24] WilliamDAssafMSFT. 2023. PREDICT (Transact-SQL) - SQL machine learning. https://learn.microsoft.com/en-us/sql/t-sql/queries/predict-transact-sql?view=sql-server-ver16

[25] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases. https://doi.org/10.48550/arXiv.2404.13207 arXiv:2404.13207 [cs].

[26] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proc. ACM Program. Lang.* 1, OOPSLA (Oct. 2017), 63:1–63:26. https://doi.org/10.1145/3133887

[27] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600 [cs.CL] https://arxiv.org/abs/1809.09600

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

[29] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL] https://arxiv.org/abs/1809.08887

[30] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. https://doi.org/10.48550/arXiv.2203.14465 arXiv:2203.14465 [cs].

[31] John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2 (AAAI'96)*. AAAI Press, Portland, Oregon, 1050–1055.

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

[33] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103 (2017).

A SAMPLE QUERIES

We detail the modifications made to BIRD queries for our benchmark. Each query is modified to require either LM knowledge or reasoning to answer. Sample queries are shown below.

Match-based
Original BIRD Query: *What is the grade span offered in the school with the highest longitude?* **Modified Query:** *What is the grade span offered in the school with the highest longitude in cities in that are part of the 'Silicon Valley' region?* **Analysis:** This query is modified to require LM knowledge of which cities are within the Silicon Valley region of California, information not available in the data source.
Comparison
Original BIRD Query: *Among the players whose height is over 180, how many of them have a volley score of over 70?* **Modified Query:** *Among the players whose height is over 180, how many of them have a volley score of over 70 and are taller than Stephen Curry?* **Analysis:** This query is modified to require LM knowledge of how tall Stephen Curry is.
Ranking
Original BIRD Query: *What are the titles of the top 5 posts with the highest popularity?* **Modified Query:** *Of the 5 posts wih highest popularity, list their titles in order of most technical to least technical.* **Analysis:** This query is modified to require LM reasoning over a textual field, the post's title.
Aggregation
Original BIRD Query: *Write all comments made on the post titled 'How does gentle boosting differ from AdaBoost?'* **Modified Query:** *Summarize the comments made on the post titled 'How does gentle boosting differ from AdaBoost?' to answer the original question.* **Analysis:** This query is modified to rely on LM reasoning over text on the textual comment fields to provide a summary.

B LM PROMPTS

We summarize the prompts used with instruction tuned Llama-3.1 80B for query synthesis and answer generation.

B.1 Query Synthesis

For the query synthesis step, in our case a Text2SQL step, we use the same table schema encoding and LM prompt as the original BIRD benchmark. An example prompt for query synthesis is shown below.

```
-- Example Prompt for Query Synthesis
CREATE TABLE frpm
(
    CDSCode TEXT not null primary key,
    Academic Year TEXT null,
    ...
)

CREATE TABLE satscores
(
    ...
    AvgScrRead INTEGER null,
    AvgScrMath INTEGER null,
    ...
)
```

```

CREATE TABLE schools
(
    ...
    District TEXT not null,
    School TEXT null,
    ...
)

-- External Knowledge: None
-- Using valid SQLite and understading External Knowledge, answer the following questions for the tables provided above.
-- Among the schools with the average score in Math over 560 in the SAT test, how many schools are in the bay area?
SELECT
```

B.2 Answer Generation

On the Text2SQL + LM and RAG baselines, the answer generation step requires the LM to answer a user question with the provided rows in context. We utilize a separate prompt for aggregation queries, while match-based, comparison, and ranking share the same prompt. We show both prompts below.

```
-- Example Prompt for Answer Generation for Match-based, Comparison, and Ranking Queries.

You will be given a list of data points and a question. Use the data points to answer the question. Your answer must be a list of values that is evaluatable in Python. Respond in the format [value1, value2, ..., valueN]. If you are unable to answer the question, respond with []. Respond with only the list of values and nothing else. If a value is a string, it must be enclosed in double quotes.

Data Point 1:
...
- School: <school name for first row>
- AvgScrMath: <average math score first row>
...
Data Point 2:
...
- School: <school name for second row>
- AvgScrMath: <average math score second row>
...

Question: Among the schools with the average score in Math over 560 in the SAT test, how many schools are in the bay area?
```

```
-- Example Prompt for Answer Generation for Aggregation Queries.

You will be given a list of data points and a question. Use the data points to answer the question. If a value is a string, it must be enclosed in double quotes.

Data Point 1:
...
- PostId: <post id for first row>
- Text: <comment text first row>
...
Data Point 2:
...
- PostId: <post id for second row>
- Text: <comment text second row>
...

Question: Summarize the comments made on the post titled "How does gentle boosting differ from AdaBoost?" to answer the original question.
```

C HANDWRITTEN PIPELINES

We use the LOTUS package to construct hand-written TAG pipelines. For each query in our benchmark, a pipeline consisting of a series of dataframe transformations and filters along with LOTUS semantic LM operators was written in Python. Example pipelines are visible below.

```
-- Match-based
query = "What is the grade span offered in the school with the highest longitude in cities in that are part of the 'Silicon Valley' region?"

schools_df = pd.read_csv("../pandas_dfs/california_schools/schools.csv")
unique_cities = pd.DataFrame(schools_df["City"].unique(), columns=["City"])
sv_cities = unique_cities.sem_filter("{City} is a city in the Silicon Valley region")
```

```
schools_df = schools_df[schools_df["City"].isin(sv_cities["City"])]
schools_df = schools_df.sort_values(by=["Longitude"], key=abs, ascending=False).head(1)
prediction = schools_df["GSoffered"].tolist()[0]

return prediction
```

```
-- Ranking
query = "Of the 5 posts wih highest popularity, list their titles in order of most technical to least technical."

posts_df = (
    pd.read_csv("../pandas_dfs/codebase_community/posts.csv").sort_values(by=["ViewCount"], ascending=False).head(5)
)

prediction = posts_df.sem_topk("What {Title} is most technical?", 5).Title.values.tolist()

return prediction
```

```
-- Aggregation
query = "Summarize the comments made on the post titled 'How does gentle boosting differ from AdaBoost?' to answer the
     original question"

posts_df = pd.read_csv("../pandas_dfs/codebase_community/posts.csv")
comments_df = pd.read_csv("../pandas_dfs/codebase_community/comments.csv")
posts_df = posts_df[posts_df["Title"] == "How does gentle boosting differ from AdaBoost?"]
merged_df = pd.merge(posts_df, comments_df, left_on="Id", right_on="PostId")

prediction = merged_df.sem_agg("Summarize the comments", all_cols=True)._output[0]
return prediction
```