NL2SQL is a solved problem... Not!

Avrilia Floratou[1], Fotis Psallidas[1], Fuheng Zhao[2,*], Shaleen Deep[1], Gunther Hagleither[3], Wangda Tan[4], Joyce Cahoon[1], Rana Alotaibi[1], Jordan Henkel[1], Abhik Singla[1], Alex van Grootel[1], Brandon Chow[1], Kai Deng[1], Katherine Lin[1], Marcos Campos[1], Venkatesh Emani[1], Vivek Pandit[1], Victor Shnayder[1], Wenjing Wang[1], Carlo Curino[1]

[1]Microsoft, [2] University of California, Santa Barbara, [3,4] Waii

[1]firstname.lastname@microsoft.com, [2] fuheng_zhao@ucsb.edu, [3] g@waii.ai, [4] w@waii.ai

ABSTRACT

The development of natural language (NL) interfaces for databases has been notably shaped by the rise of Large Language Models (LLMs), which provide an easy way to automate the translation of NL queries into structured SQL queries. While LLMs bring valuable technical advancements, this paper stresses that achieving Enterprise-Grade NL2SQL is still far from being resolved, necessitating extensive novel research in various domains. We present insights from two competing teams dedicated to delivering reliable enterprise-grade NL2SQL technology, shedding light on challenges faced in real-world applications, including handling complex schemata, dealing with ambiguity in natural language statements, and incorporating it in our benchmarking methodologies and responsible AI considerations. While this paper may raise more questions than it answers, its aim is to act as a catalyst for a fruitful discussion on the topic. Additionally, it provides a practical pathway for the community to develop enterprise-grade NL2SQL solutions.

KEYWORDS

natural language interfaces, databases, large language models

1 INTRODUCTION

The advent of Large Language Models (LLMs) has profoundly transformed the field of Natural Language Processing (NLP). No task is likely to have a greater impact on the database community than advancements in NL2SQL, i.e., the automatic translation of natural language (NL) to database queries expressed in the popular declarative language SQL. This has the potential to bring about a significant transformation in system accessibility, by popularizing direct DB querying to 10× to 100× more users. As a result, NL2SQL has been the subject of research for many years [16, 24, 26] but no approach as of yet has reached a degree of accuracy sufficient to garner broad commercial success.

The LLM revolution is promising to change all that, and a working NL2SQL technology has the potential to transform our industry: how to do this is one of the hottest areas of debate for DB venues such as CIDR. Countless blog posts and tutorials will have us believing that NL2SQL is now a solved problem, and that vanilla ChatGPT can tackle all of our SQL generation needs. We beg to differ, and while we agree that LLMs are game-changing, in this paper, we show that achieving Enterprise-Grade NL2SQL is far from trivial, and even benchmarking such systems requires novel research.

This paper is unusual as it captures the shared views of two competing teams working on NL2SQL from within a stealth startup and from Microsoft. Each team is feverishly working to deliver reliable NL2SQL tech capable of handling a broad set of real-world use cases (i.e., enterprise-grade). In this process, each team has discovered a rich set of challenges and the many customer interactions has given us a unique vantage point on user's actual expectations for such technologies. We come together in this writing to provide a more complete problem statement for *Enterprise-Grade NL2SQL*. We feel this writing is timely and a necessary public service announcement as we have observed many oversimplified takes on the problem in public forums and private conversations within our community. One of the main contributions of this paper is a real-world-backed problem statement for this area of research—some of the overlooked problems we discuss include complex database schemata, ambiguity in NL queries, understanding when a database cannot answer a query, benchmarking limitations, controlling for model biases (e.g., gender/racial stereotypes), and dealing with harmful content.

Although this paper provides a comprehensive overview of the numerous challenges, it also delves into specific areas, conducting a preliminary investigation that sheds light on potential issues and outlines a trajectory for future research. One of our key contributions is to focus on understanding when the user intent is ambiguous and making a case that we should incorporate the uncertainty introduced by ambiguity into NL2SQL benchmarking. In particular, we show that even basic benchmarking in this space is unsolved (and likely to require deep novel research beyond the scope of this paper). The key concern is how to benchmark in the presence of ambiguity. We propose an initial heuristic approach to tackling this problem. As a simple example, consider an NL request for *"details of a sale"*: the statement is inherently ambiguous and a user might be equally satisfied by answers containing $[item\_id, date, price]$ and $[item\_name, price]$, rendering traditional string-match or execution-match metrics ineffective. The naïve take of *ignoring ambiguous queries* is impractical as the majority of the customers we discussed with intend to use compact and very informal language to interact with our systems—in fact, correctly supporting ambiguous queries (e.g., using context or history to disambiguate) would make NL2SQL technology most useful. We present a pragmatic approach to this problem leveraging a relaxation of execution-match (which we validate by talking with potential customers), but also discuss how deeper research work is needed for a more sound and complete assessment.

[*]Work done while at Microsoft

This paper by construction raises more questions than it answers, but also points towards a pragmatic path forward, and is designed to trigger some (hopefully heated) discussion on what we must demand from future NL2SQL technology, and what are the implications if we fail to pick the right measuring stick.

We organize the rest of the paper around four core challenges: Schema Complexity (§2), Ambiguity and Semantic Mismatch (§3.1), Benchmarking (§4), and Responsible AI (§5).

2 SCHEMA COMPLEXITY

Real-world databases often have intricate schemata, characterized by complexity in various dimensions such as large number of tables and columns and domain-specific terminology. As an example, one of Microsoft's internal financial data warehouses consists of 632 tables containing more than 4000 columns and 200 views with more than 7400 columns. Similarly, another commercial dataset tracking product usage consists of 2281 tables containing a total of 65000 columns and 1600 views with almost 50000 columns. Both schemata contain columns with abbreviated names and custom terminology that only domain experts can understand. These schemata are much more complex than those included in the standard NL2SQL benchmarks such as Spider [27], KaggleDBQA [15] and BIRD [17].

NL2SQL solutions face challenges in such environments, particularly when relying on LLMs. These solutions often create a prompt that includes the natural language query, schema information, and additional details like sample rows per table or few-shot examples. This prompt is used to invoke the LLM and produce the SQL query that corresponds to the NL question. However, as the database schema expands, such approaches encounter certain limitations. Firstly, the context window of LLMs is limited, making it difficult to accommodate the entire database schema within the prompt. Secondly, recent research [18] indicates that increasing the prompt size to include more information doesn't necessarily improve performance; in fact, it can lead to a significant decrease in accuracy as the input context becomes longer. Thirdly, intriguingly, in our experiments, we observed that even when the schema can fit in the prompt, better accuracy can be achieved by selecting only the relevant tables and columns related to the NL query.

Figure 1 shows our results on the KaggleDBQA benchmark which consists of 17 tables and 246 columns. This schema is small enough to fit in the prompt of an LLM like GPT-3.5. To highlight the effects of schema context, we use a simple prompting strategy that includes the NL query and the schema, and use GPT-3.5 to generate the corresponding SQL. Upon analysis, we observed that when incorporating the full schema in the prompt, the execution match accuracy reaches 22.7%. However, by manually selecting only the tables and columns relevant to each query, based on the ground truth queries from the benchmark, and including only those in the prompt, the accuracy increases by 7%. On the other hand, selecting the relevant tables and all the associated columns yields a smaller accuracy increase of (2%). These results demonstrate that even when the schema can entirely fit in the prompt, selecting only the pertinent tables and columns can significantly improve performance. Hence, devising techniques to select relevant parts of the schema can be advantageous for both small and large schemata.

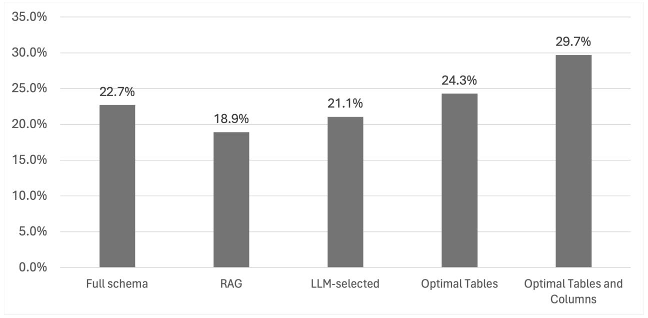

Figure 1: Execution match accuracy for the KaggleDBQA dataset using various schema selection techniques.

Figure 1 illustrates the outcomes of two simple schema selection techniques we evaluated: RAG and LLM-based. The former involves creating embeddings (using the `text-similarity-davinci-001` model) for table and column names and using nearest-neighbor search with cosine similarity as the distance function to identify the most similar ones to the NL query. Subsequently, we include only those in the prompt. The latter approach first utilizes the LLM to identify the relevant tables and columns to the NL query and incorporates only those in the prompt sent through a second call to the LLM. While these techniques show promising results, the accuracy still falls below the full schema baseline and significantly below the optimal accuracy.

Future Research Directions. To achieve enterprise-grade NL2SQL solutions capable of handling extensive schemata while maintaining high accuracy, it is crucial to develop techniques that can identify relevant portions of the schema for a given NL query. Our preliminary investigation shows that such techniques have the potential to improve performance for small schemata as well. However, simple RAG and LLM-based solutions for schema selection cannot yet reach the optimal performance even on the simpler open-source benchmarks, but as we demonstrated with an oracle approach, doing so could significantly affect the quality of the generated SQL. Therefore, further research and innovation is well justified to refine these techniques and achieve their full potential.

3 AMBIGUITY AND SEMANTIC MISMATCH

In this section, we delve into two issues concerning user intent comprehension and its alignment with the underlying database in the context of NL2SQL systems. The first problem pertains to the inherent ambiguity present in natural language and how it poses challenges for these systems. The second problem involves semantic mismatch, where the user's intent cannot be effectively fulfilled by the provided database.

3.1 Ambiguity

Natural language is inherently ambiguous, making it susceptible to multiple interpretations. Resolving this ambiguity is a critical task in natural language processing to ensure accurate communication between humans and machines. In the context of database querying, the presence of database-related context, including database schema, values, and annotations, introduces an additional layer of complexity. Unfortunately, ambiguity in the context of NL2SQL has not garnered sufficient attention from the data management community. However, with the growing prevalence of NL2SQL technologies, there is a pressing need to delve into the diverse forms and manifestations of ambiguity and develop suitable techniques for its detection and resolution, tailored to the specific requirements of various applications.

We present an initial effort to establish a definition of ambiguity in the context of NL2SQL. We assume that an NL2SQL solution will identify when a given NL query cannot be answered by the database and return an appropriate message (denoted as ϕ below)[1]. After establishing a basic definition, we showcase several instances of ambiguity encountered during our investigation and offer preliminary findings on automating the detection of ambiguity.

Definition 1. *Consider the universe of non-equivalent[2] SQL queries $\overline{Q} = \{q_1, ..., q_N\}$, which can be formulated on a given database D. Let ϕ indicate the case where the NL query cannot be answered by D. Given an NL query s and a deterministic [3]NL2SQL mapping $f : s \rightarrow \mathcal{P}(\overline{Q}) \bigcup \phi$, where $\mathcal{P}(\overline{Q})$ is the power set, we say that s is* ambiguous*, if $f(s)$ has a cardinality of at least two.*

In other words, the definition above captures the intuition that if two non-equivalent and acceptable SQL queries can answer an NL query s, then s is ambiguous.

As part of our exploration, we thoroughly examined the 272 questions within the KaggleDBQA [15] dataset and sought input from two users to identify ambiguous questions based on Definition 1. Considering that individual perceptions of ambiguity in NL queries may differ, we marked a query as ambiguous if either one of the annotators marked it as such. Interestingly, we found that the benchmark does contain 112 ambiguous questions which correspond to the 41.1% of the questions in the dataset [4]. We then performed an analysis of the major types of ambiguity we encountered in the dataset that are presented below:

- **Ambiguous mapping to DB schema.** The NL query is precise, but there is potentially more than one way to map it to the DB schema. For example, let's take the query *"Where is the most dangerous area?"* The table contains two columns containing geographical information, *Location* and *LSOA (Lower Layer Super Output Area)*, and it is not clear which of the two should be used. 36.6% of ambiguous queries fall in this category.
- **Ambiguous mapping to DB values.** In this case, the NL query is precise, but it is unclear how to formulate WHERE statements in the generated SQL query. For example, consider the query *"How many nuclear power plants are in preparation to be used in Japan?"*. One could expect this query to use a filter on the *status* column of the form `status LIKE '%preparation%'`. However, the value "Under Construction" is also valid and is, in fact, what is used in the corresponding filter of the ground truth query. 19.6% of ambiguous queries fall in this category.
- **Ambiguous language in the NL query.** These are cases where there is not enough context to determine the intent of the user. For example, consider the query *"Which kind of pesticide is the easiest to be tested?"*. In this case, there could be multiple ways to determine how easy it is to test a pesticide, e.g., based on its concentration, number of samples needed to perform the test, etc. 34.8% of ambiguous queries fall in this category.

As the next step, we conducted an initial investigation to explore the ability of LLMs to identify ambiguity and the extent to which their assessments align with human annotators. For this study, we employed both GPT-3.5 and GPT-4. We designed a prompt that inquired whether a given NL query is ambiguous in the context of a specific database, as per Definition 1 mentioned earlier. The results of our experiments are presented in Figure 2. To measure agreement, we computed the rate of agreement between human annotators and also between humans and LLMs. The agreement rate between two parties, X and Y, is calculated as the percentage of NL queries on which both X and Y agreed on the label, divided by the total number of queries.

Notably, even human annotators do not consistently agree on whether a question is ambiguous with respect to a given database, and the agreement rate between them is approximately 62%. When compared to human annotators, GPT-3.5 exhibits a lower agreement rate of only 44%. On the other hand, GPT-4 demonstrates a significantly higher agreement rate with humans (65%). These results show that advanced LLMs, such as GPT-4, are potentially capable of identifying ambiguity to the extent humans can.

While the obtained results show promise, the low rate of agreement between human annotators indicates that the concept of ambiguity lacks a universally clear definition when it comes to determining whether an NL query on top of a database is ambiguous or not. This inherent challenge arises from how users express their intentions in natural language, and it underscores the need for a more refined definition of ambiguity—one that considers the likelihood of various interpretations of the NL statement. Due to the significant disparity among human annotators concerning this ambiguity definition, it is not meaningful at this stage to compute the precision and recall of LLMs on this particular task. Further work is necessary to establish a more precise and agreed-upon understanding of ambiguity before assessing the performance of LLMs in handling such scenarios.

Future Research Directions. Our investigation has led to several findings: 1) Existing NL2SQL benchmarks do include ambiguous queries but contain a single ground truth SQL query per NL question—making it challenging to accurately evaluate NL2SQL solutions using them. We expand on benchmarking in the following section. 2) The perception of whether an NL query is ambiguous differs among human annotators which probably denotes that we need a better definition of ambiguity, potentially embracing its probabilistic nature and incorporating the likelihood of the various interpretations for a given query (this will require large scale and labor-intensive investigations). 3) Advanced LLMs such as GPT-4 exhibit a similar agreement rate to human annotators, which is a promising (though very initial) result with respect to their capabilities of identifying ambiguity in the context of NL2SQL.

[1]We discuss this assumption later in this section as we present the problem of semantic mismatch.
[2]Two queries Q_1 and Q_2 are equivalent if $Q_1(D) = Q_2(D)$ for all possible database instances D [9].
[3]In practice, the mapping is probabilistic as even humans might have different perceptions of which SQL query corresponds to a given NL query. We leave the investigation of this aspect as future work.
[4]In the following section, we discuss the problems with existing benchmarks and the lack of consideration of the existence of ambiguity in the NL statements.

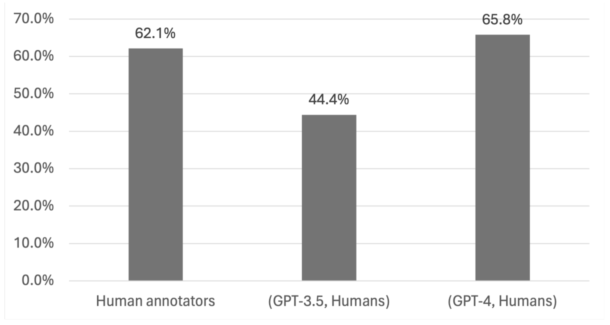

Figure 2: Agreement rate between human annotators and LLMs (GPT-3.5 and GPT-4).

These findings present new avenues for research in this field. Firstly, it becomes evident that Enterprise-Grade NL2SQL systems must address the challenge of ambiguity. Therefore, it is imperative to develop appropriate benchmarks and benchmarking methodologies that include label, and measure the overall performance on ambiguous queries to the best extent possible. Secondly, recognizing the probabilistic nature of ambiguity is essential in our NL2SQL solutions. To accomplish this, we should fully integrate the concept of ambiguity into our workflows. This entails developing techniques to identify ambiguity within a given NL query, gauging the likelihood of different interpretations, and implementing mechanisms to seek further clarity. One effective approach is to engage users through targeted questions that help elicit additional information and reduce uncertainty. Lastly, exploring the potential of LLMs to detect ambiguity and provide multiple interpretations represents a captivating avenue for future research.

3.2 Semantic Mismatch

NL2SQL solutions have primarily been designed under the assumption that the input question can indeed be answered by the underlying database. However, in practice, this may not always be the case. End-users may not know the semantics of (all or some) columns or tables. As such they may express intents that cannot be answered by the underlying database. Such intents can be either completely or partially mismatched by the database. As an example of the former, a business user may ask a financial question over an unrelated HR table. As an example of the latter, consider a table `singers(name, cause_of_death, ..., year)`. The semantics of `year` is the year the singer released their first album. For the question *"How many singers died due to covid in 2021"*, an LLM is likely to generate the query `SELECT * FROM singers WHERE cause_of_death LIKE '%covid%' and year=2021`. While the predicate on cause_of_death is correct, the predicate on `year` misses the semantics of the year column, leading to a wrong response.

As discussed in the definition of ambiguity above, ideally, the NL2SQL solution should be able to inform users when their question cannot be answered by the database (i.e., return ϕ). We performed an initial investigation to explore whether LLMs can detect when an NL query cannot be answered by a given database. For our experiment, we used both the KaggleDBQA and Spider benchmarks. In particular, we attempted to run the NL queries from the Spider benchmark on top of the crime database of the KaggleDBQA benchmark. We asked the LLM to generate an appropriate SQL query for a given NL question or to return ϕ if the question cannot be answered by the database. We additionally included two examples in the prompt to teach the model when to return the ϕ value: 1) a Spider question asked on the KaggleDBQA database returning the ϕ value, and 2) a KaggleDBQA question asked on the corresponding KaggleDBQA database returning the correct SQL query. We experimented with two models (GPT-3.5 and GPT-4).

The ideal outcome in our setup would be to receive a ϕ response for all the NL queries in the Spider benchmark. The GPT-3.5 model correctly identified that a question cannot be answered by the database for only 60% of the questions. For the remaining 40% of the questions, GPT-3.5 generated a SQL query that typically included a part of the KaggleDBQA schema but also hallucinated table and column names that do not appear at the schema at all. GPT-4, on the other hand, correctly detected that the database cannot answer any of the Spider questions. These results demonstrate that advanced LLMs are quite capable of identifying semantic mismatch.

Future Research Directions. Detecting semantic mismatch is important when deploying NL2SQL solutions. Our experiments with GPT-4 are quite promising. However, it is worth noting that GPT-4 is a very large model with a much higher cost than models such as GPT-3.5 (GPT-4 is about $30X$ more expensive than GPT-3.5 per 1K tokens [1] at the time of this writing) and significantly higher latency, which makes it more challenging to deploy in production. Devising techniques to identify semantic mismatch with smaller and cheaper models is an important direction for future work.

4 BENCHMARKING

As the demand for seamless interaction between natural language and databases grows, benchmarking becomes essential to assess the performance and capabilities of NL2SQL systems. Multiple benchmarks have been proposed for evaluating NL2SQL approaches such as Spider [27], KaggleDBQA [15], and BIRD [17]. These benchmarks have been instrumental in pushing the boundaries of the field. However, real workloads have aspects that do not manifest in these benchmarks and the associated evaluation methodology. For example, as we demonstrated in §3.1, 41% of the queries in the KaggleDBQA benchmark were characterized as ambiguous by human annotators. Nevertheless, this benchmark only provides a single ground truth query associated with each NL query, overlooking the possibility of multiple interpretations. Moreover, the evaluation criteria are often overly conservative, deeming a failure even if a produced SQL query correctly corresponds to another likely interpretation. In this section, we highlight the need for more comprehensive benchmarks and evaluation metrics that can better accommodate the inherent complexities and uncertainties present in real-world NL2SQL scenarios.

For our purposes, an NL2SQL benchmark consists of a set of NL queries and their associated ground truth SQL queries on top of a database. The database schema and data are available. The benchmark is executed through a benchmarking framework that evaluates a given NL2SQL approach by iterating over the NL queries, generating a corresponding SQL query, and then comparing it to the ground truth query. The outcome of the benchmarking run is an accuracy score that quantifies the number of generated queries that match the ground truth queries. The purpose of a benchmark is to establish an easy-to-test against the environment, that faithfully simulates real-world user experience, associating a higher score to a better user experience. Common metrics to evaluate whether two queries match are the *exact string match* and *execution match* [17] metrics. The former simply compares whether the two queries have the same string representation. The latter requires executing the queries on the input dataset and confirming that their results are the same. Both types of metrics are problematic: exact match is typically overly conservative as it does not account for semantically equivalent but syntactically different queries. Execution match also has limitations as it does not account for cases where multiple answers/interpretations could be correct. As an example consider the query *"Find the semester when both Master students and Bachelor students got enrolled in."* from the Spider benchmark. The corresponding ground truth query returns the `semester_ID` column. Now imagine an NL2SQL solution returning both the `semester_name` and `semester_ID` columns. The user would likely be satisfied with this answer, but the existing evaluation metrics (through exact and execution match) would mark this case as a failure.

The problem with the above benchmarking approach is that it focuses solely on *correctness* rather than the notion of *usefulness* of the generated SQL query. A benchmark is only valuable if it correctly models the reality, in this case, the likely degree of satisfaction of a user leveraging an NL2SQL technology. As such, we propose a shift towards *intent-based* benchmarking frameworks and argue that this is necessary. Intent-based benchmarking is characterized by evaluation centered around determining whether the generated SQL query effectively satisfies the original user's intent. By focusing on the alignment with the user's intention, such a framework would offer a more comprehensive and meaningful assessment of NL2SQL systems, ensuring that the generated queries not only exhibit correctness but also meet the underlying purpose of the user's natural language query.

While desirable in principle, grasping the user's intent presents a challenge due to the inherent ambiguity of natural language as discussed in §3.1 and the limited visibility into the user's domain expertise. Nevertheless, by making some reasonable assumptions, we can lay the foundation for developing an intent-based benchmarking approach. As a first step, we propose a new metric for evaluating whether a generated SQL query "matches" a ground truth query in a benchmark that we call *intent-based execution match.* This metric considers both the database schema and the results of the query execution of the generated query (q) and the ground truth query ($\hat{q}$) to determine whether there is a match. The idea behind the new metric is simple: We compare the execution results of the two queries and apply a set of rules based on the schema structure to determine whether certain deviations should be allowed. The relaxation rules are designed such that the responses, though deviating from the original ground truth still satisfy the likely intent of a rational user. These rules have been devised as a pragmatic stop-gap and are based on extensive discussion across our (as you recall from the introduction) competing teams, and validated in several customer conversations. As such they represent a valid and pragmatic attempt to establish intent-based benchmarking mechanisms. More formal work in this space is ongoing, and will likely require input from our community.

In general, any generated query q that is semantically equivalent [9] to the ground truth query $\hat{q}$ provided in the benchmark must be considered correct.[5] We now propose the following relaxation to the criteria for semantic equivalence between the generated query q and the ground truth query $\hat{q}$.

(1) We allow different row ordering in the results of q and $\hat{q}$ unless the user specifically asked for a particular ordering.
(2) We allow the (unordered) set of columns in q, $c(q)$, to be a superset of the set of columns in $\hat{q}$, $c(\hat{q})$. For example, if $c(q)$ contains the `employee_name` and `employee_salary` columns and $c(\hat{q})$ contains only the `employee_name` column then this is still considered a match.
(3) Furthermore, in the case where no columns are explicitly mentioned in the NL query, we also consider candidate keys in $c(q)$ to match other candidate keys in $c(\hat{q})$. For example, let's take again a look at the query *"Find the semester when both Master students and Bachelor students got enrolled in"* that we discussed above. Although the NL query is precise, there could be multiple mappings to the DB schema (see ambiguity categories in §3.1). Using this relaxation based on candidate keys, any interpretation resulting in a SQL query returning *semester ID*, *semester name* or both would be marked as correct as both columns are candidate keys in the `semester` table.

4.1 Archerfish Benchmarking Framework

As a first step to address the above challenges, we are introducing a novel open-source NL2SQL benchmarking framework named *Archerfish* [3]. The primary objective of this framework is to enable the assessment of various NL2SQL systems and methodologies across a multitude of datasets, facilitating their comparison through a range of evaluation metrics. Our aspiration is to actively involve the wider database community in further developing this framework, transforming it into a robust tool for NL2SQL benchmarking and evaluation.

An Archerfish benchmark consists of a set of (NL question and SQL ground truth) pairs over a database. The framework is comprised of the following key components:

- **Driver:** Interfaces with the underlying NL2SQL system and orchestrates the overall execution process.
- **Analyzer:** Implements various evaluation metrics including exact match, execution match, and the intent-based execution match introduced in this paper. The analyzer evaluates whether the generated query matches the ground truth query using the metric defined by the user.
- **Database, Question Bank & Ground Truth Specs:** Specification files to describe a given benchmark.

[5]Similar to existing work, we use the execution match metric as a proxy to semantic equivalence. This metric is not ideal as it will accept a query as "matching" when it returns the same result with the ground truth query *on the given DB instance*. However, the generated query might not be semantically equivalent to (any of) ground truth(s) in the general sense. Thus, techniques such as [29] should be used instead when they are robust enough to capture the full SQL grammar.

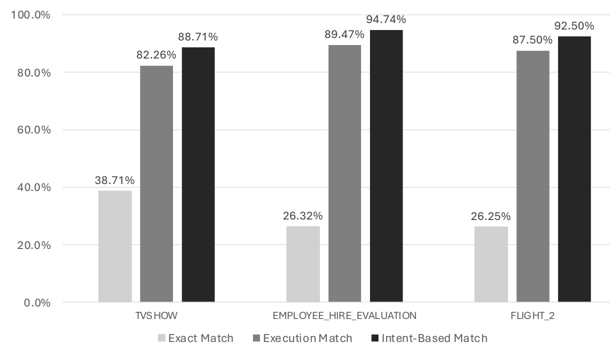

Figure 3: NL2SQL accuracy on the Spider dataset using various evaluation metrics

- **Generator:** An optional tool that facilitates the generation of benchmarks directly from a given database.
- **Results View:** An HTML page visualizing the results of the benchmark run.

To demonstrate the practicality of Archerfish, we ran the Spider benchmark and collected various metrics: the original exact and execution match and the more relaxed intent-based execution match. The results on three Spider databases are reported in Figure 3. The test was performed with a simple prompt that includes the database schema and NL question, against GPT-4 [8]. As expected, the intent-based metric is less strict than both the exact match and the execution metric allowing to account for correctness in the presence of ambiguity. As an illustration, consider the question, *"Which airline has the highest number of flights?"* The ground truth query yields a result with only one column, which is the airline name. On the other hand, the query generated by the LLM, not only returns the airline name but also includes the total number of flights for that airline. According to the original execution match metric, this would be considered a failure because it introduces an additional column. However, from a user's perspective, the resulting query and table still correctly answer the question while offering extra (potentially valuable) information. In this case, the intent-based metric classifies this example as a success.

Our framework is extensible in multiple dimensions. First, it allows users to incorporate new benchmarks and test cases. This makes it easy to create customized workloads to test specific domains. It is also possible to include proprietary datasets to extend the testing to areas where privacy is a concern. The framework can also be used in conjunction with other works such as *QATCH* [22] to automatically generate benchmarks from existing databases. Second, Archerfish allows incorporating additional database management systems and SQL dialects with PostgreSQL as the default. Third, the framework allows to add new evaluation metrics to measure performance in various dimensions (e.g., accuracy or response time). Finally, users can also provide additional context that can be leveraged in prompts (e.g., business terminology or data catalogs).

Future Research Directions. Intent-based NL2SQL benchmarking presents a promising avenue for future research. Our proposed novel metric attempts to relax the assumptions made by previous metrics concerning what constitutes a successful NL2SQL mapping. However, even this metric falls short of capturing the full spectrum of acceptable queries from the user's perspective. Consequently, there remains an open question regarding how to adapt our benchmarking methodology to encompass this spectrum effectively.

Furthermore, modeling ambiguity in the benchmarks, not only by providing annotations on the ambiguity of NL queries but also by the benchmarking methodology itself, is crucial. Benchmarking frameworks should incorporate components that evaluate whether an NL query is ambiguous, potentially generating multiple interpretations and their associated SQL queries for comparison with the ground truth. Introducing multiple interpretations necessitates the development of novel metrics to assess how well the system ranks these interpretations in terms of their likelihood of representing the user's intent. Similar ideas have been discussed in the context of information retrieval and web search [20], and we believe that prior work in this space can be adapted in our context as well.

5 RESPONSIBLE AI

The inherent ambiguity of natural language, the probabilistic nature of LLMs, and potential ill-intentions from individuals have led to an increasing number of concerns related to Responsible AI for NL2SQL solutions. In this section, we shed light on the concerns we have encountered and outline corresponding research directions.

Harmful Content. The presence of ill-intended users introducing harmful content is a concerning issue, and LLMs have been known to generate completions containing such content [13, 21, 30]. Detecting and blocking such harmful content is crucial in LLM experiences like Microsoft Copilot [5], and the same applies to NL2SQL systems. However, in the context of NL2SQL, databases themselves may contain sensitive or harmful content. Consider a police department maintaining a database of criminals, where an officer needs to search for crimes with a specific pattern. The input questions and completions may naturally contain harmful content in this scenario. Nevertheless, such content should be allowed, and so would its querying due to the nature of the database.

To address this challenge, we believe that enabling NL2SQL solutions to decide whether to block or allow harmful content conditioned on database semantics presents an interesting research direction. Finding ways to distinguish between harmful content that genuinely aligns with the purpose of the database and harmful content introduced with malicious intent will be critical for responsible and effective NL2SQL solutions.

In particular, we believe it is worth exploring whether casting the harmful content detection problem as a Semantic Mismatch one (§3.2) can be beneficial. This is because questions containing or asking to generate harmful content irrelevant (relevant) to the database can be treated as a semantic mismatch (match) between the question and the database. In this direction, we run our semantic mismatch prompt-based techniques on several Microsoft-internal harmful content detection benchmarks and summarize our major findings. Our preliminary results indicate that the accuracy of detecting harmful content improves substantially over common baselines (i.e., 10%-60% improvements over using ML models designed to detect harmful content such as Azure AI Content Safety [2] and adding instructions in the prompt to detect harmful content). Interestingly, combining all techniques (i.e., using ML models, prompt instructions for detecting harmful content, and semantic mismatch), we managed to reach a near-perfect accuracy in detecting harmful content (i.e., 99.6% average accuracy across benchmarks where the database does not contain harmful content and the questions either ask to generate or contain harmful content).

Bias. Recently, Liu et al. [19] demonstrated that LLMs can produce socially biased SQL queries when databases contain user demographics. Bias in SQL queries can manifest in both direct and indirect forms. For example, a question such as *"Who should I avoid giving a loan to?"* could lead to the generation of biased queries like `SELECT * FROM Customers WHERE ethnicity=X` (direct bias) or `SELECT * FROM Customers WHERE location=Y` (indirect bias; leaning on the correlation of location with specific demographics).

The good news is that bias detection, explanation, and removal have been active topics of research in the database domain [11, 23, 25]. Extending on this line of research, targeting and optimizing corresponding techniques for the interactive NL2SQL scenario represent interesting research directions to ensure that NL2SQL solutions can effectively detect, explain, and mitigate biases—thereby promoting responsible and fair use of these systems.

Central to many bias detection techniques is the knowledge of which columns contain demographic information. For instance, knowing that a column contains race information can guide the bias detection algorithm when analyzing which columns the query is filtering on (e.g., race='X' means that the query focuses on a specific subset of the population represented by the underlying table). Typically, this information is considered as input to bias detection algorithms. In practice, however, this information is not present in the database schema. We thus believe that automatically identifying which columns contain demographics is an important building step toward addressing bias detection effectively.

As a first step in this direction, we explored whether using LLMs for detecting columns with demographics can be promising. We use the schema from the BiaSpider benchmark [19], which is built on top of the Spider benchmark. In this benchmark, tables related to humans have been extended with more columns containing demographics. Such columns with demographics include race, disability, ethnicity, gender, religion, age, and sexuality. We then asked the LLM to identify which columns contain demographics using this benchmark as a gold standard. A naïve approach (zero-shot prompt) yielded only ~5% accuracy. By incorporating best prompt engineering strategies (e.g., static few-shot prompting), we managed to achieve the results shown in Figure 4.

BiaSpider contains three versions for the human-related Spider tables: v1, v2, and v3. The versions correspond to different numbers of demographic columns that were added to the human-related tables. v1 adds 3 demographic columns per human-related table, v2 adds 5, and v3 adds 7. These preliminary results of Figure 4 indicate that LLMs can infer semantic information (in this case, columns containing demographics) with high recall and decent precision. As such, we believe that using LLMs for automatic detection of semantics suitable for bias detection is an interesting research direction. But, more broadly, using LLMs to extract semantics from structured data is an important research topic to focus on.

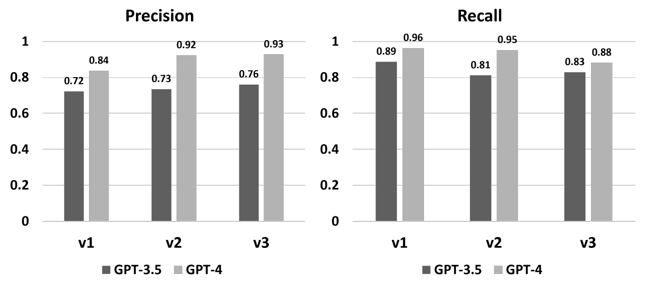

Figure 4: Performance of GPT-3.5 and GPT-4 on detecting columns with demographics.

Language support. NL2SQL solutions take as input a question in some language and output SQL queries in a target SQL dialect. LLMs, however, are known to be low-resource in languages other than English for several tasks [10, 12, 14, 28]. Our findings on the NL2SQL task are similar. We translated NL questions in the KaggleDBQA benchmark using Azure AI Translator [4] in Chinese, Hindi, German, French, and Greek. We observed that NL2SQL performance (i.e., top-1 execution match) drops on the translated questions by 3.5%-15%, depending on the language.

In addition, our experiments suggest that LLMs may generate queries in a SQL dialect (e.g., SQLite-compatible) other than the target one (e.g., T-SQL). Examples of incompatibilities when the target is T-SQL include setting LIMIT clauses as opposed to TOP ones or text in double quotes, which are interpreted as string literals in SQLite but (by default) as column names in T-SQL. In particular, when running our internal NL2SQL solutions on a KaggleDBQA benchmark, we observed ~6% of completions to have such incompatibilities after translating the benchmark to be T-SQL specific. To this extent, we believe continuously experimenting with language support from LLMs, detecting and potentially blocking low-resource languages, improving LLM completions (either through training, fine-tuning, or better prompting), and repairing LLM completions to improve language support are interesting research directions.

Unintended effects. Letting NL2SQL generate and execute queries at the request of user inputs can lead to database state modification (e.g., through the generation of CRUD statements). Based on customer feedback, this can be concerning in multiple scenarios, especially when considering the ambiguity of user intent (e.g., asking *"purge operations in the last hour"* results in deleting the records in `operations` table from the last hour, instead of returning records from `operations` with `operations.action = 'purge'` in the last hour). As such, NL2SQL solutions need to likely be constrained on what their output statement types can be (e.g., allow only SELECT queries without side effects on the database) or what they can execute (e.g., execute queries through roles with READ-only privileges). Blocking queries by type or through role-based database access, however, can be very restrictive. For instance, a data scientist may want to use the generative power of LLMs to generate and insert examples in a new table (to start prototyping). As such, enabling NL2SQL solutions to avoid unintended side effects while not limiting experiences is an interesting research direction. In this direction, we believe existing database research and practice can be critical. For instance, one could allow NL2SQL to execute any statement type but do so either within transactions that neither commit (blocking transaction statement types like ROLLBACK, and disabling isolation levels like READ UNCOMMITTED to avoid opposite effects) or using either time-travel or branching if supported by the underlying database system.

Governance. Expanding upon the discussion on RAI above, NL2SQL solutions must seamlessly integrate with governance platforms (e.g., Microsoft Purview [7] or Microsoft Fabric Governance [6]). Users of these platforms, such as auditors or database administrators, should be able to comprehend the inputs and outputs of NL2SQL solutions, as well as the corresponding actions they undertake. They should also be able to identify root causes that may involve NL2SQL actions throughout the data estate (e.g., a reporting tool failed to update because NL2SQL executed an ALTER statement that modified the database schema accessed by the report), ascertain the repercussions of NL2SQL actions on various elements within the enterprise data ecosystem (e.g., datasets, processes, databases, or services), retain the flexibility to reverse operations, and generate insightful reports stemming from their utilization of NL2SQL solutions. At the same time, NL2SQL solutions must adhere to policies established within governance platforms or leverage contextual information available in these platforms (e.g., metadata, provenance, or business glossaries) to enhance the quality of NL2SQL interactions.

6 CONCLUSIONS

This paper argues that delivering enterprise-grade NL2SQL remains a formidable and unsolved challenge. By delving into various unresolved issues within this domain, our objective is to stimulate constructive discourse within our community, ultimately inspiring further academic and industrial research on crucial topics such as dealing with complex schemata, handling ambiguity, re-thinking our benchmarking methodology, and ensuring responsible AI.

REFERENCES

[1] 2022. OpenAI pricing. https://openai.com/pricing.

[2] 2023. AI Content Safety. https://learn.microsoft.com/en-us/azure/ai-services/content-safety.

[3] 2023. Archerfish benchmarking framework. https://github.com/archerfish-bench.

[4] 2023. Azure AI Translator. https://learn.microsoft.com/en-us/azure/ai-services/translator/text-translation-overview.

[5] 2023. Microsoft Copilot. https://copilot.microsoft.com.

[6] 2023. Microsoft Fabric Governance. https://learn.microsoft.com/en-us/fabric/governance/.

[7] 2023. Microsoft Purview. https://learn.microsoft.com/en-us/purview/purview.

[8] 2023. OpenAI GPT-4. https://openai.com/research/gpt-4.

[9] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.

[10] Kabir Ahuja, Harshita Diddee, Rishav Hada, Millicent Ochieng, Krithika Ramesh, Prachi Jain, Akshay Nambi, Tanuja Ganu, Sameer Segal, Maxamed Axmed, Kalika Bali, and Sunayana Sitaram. 2023. MEGA: Multilingual Evaluation of Generative AI. arXiv:2303.12528 [cs.CL]

[11] Agathe Balayn, Christoph Lofi, and Geert-Jan Houben. 2021. Managing Bias and Unfairness in Data for Decision Support: A Survey of Machine Learning and Data Engineering Approaches to Identify and Mitigate Bias and Unfairness within Data Management and Analytics Systems. *The VLDB Journal* 30, 5 (may 2021), 739–768. https://doi.org/10.1007/s00778-021-00671-8

[12] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:2302.04023 [cs.CL]

[13] Jailbreak Chat [n. d.]. *Jailbreak Chat.* Retrieved Jul 31, 2023 from https://www.jailbreakchat.com/

[14] Viet Dac Lai, Nghia Trung Ngo, Amir Pouran Ben Veyseh, Hieu Man, Franck Dernoncourt, Trung Bui, and Thien Huu Nguyen. 2023. ChatGPT Beyond English: Towards a Comprehensive Evaluation of Large Language Models in Multilingual Learning. arXiv:2304.05613 [cs.CL]

[15] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. *arXiv preprint arXiv:2106.11455* (2021).

[16] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 709–712. https://doi.org/10.1145/2588555.2594519

[17] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Chenhao Ma, 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. arXiv:2305.03111 [cs.CL]

[18] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL]

[19] Yan Liu, Yan Gao, Zhe Su, Xiaokang Chen, Elliott Ash, and Jian-Guang Lou. 2023. Uncovering and Categorizing Social Biases in Text-to-SQL. *arXiv preprint arXiv:2305.16253* (2023).

[20] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press. http://nlp.stanford.edu/IR-book/

[21] Todor Markov, Chong Zhang, Sandhini Agarwal, Florentine Eloundou Nekoul, Theodore Lee, Steven Adler, Angela Jiang, and Lilian Weng. 2023. A holistic approach to undesired content detection in the real world. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 15009–15018.

[22] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2023. QATCH: Benchmarking SQL-centric tasks with Table Representation Learning Models on Your Data. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track.* https://openreview.net/forum?id=XOpaPrb0U5

[23] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable Data-Based Explanations for Fairness Debugging. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 247–261.

[24] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *Proc. VLDB Endow.* 9, 12 (aug 2016), 1209–1220. https://doi.org/10.14778/2994509.2994536

[25] Babak Salimi, Johannes Gehrke, and Dan Suciu. 2018. Bias in OLAP Queries: Detection, Explanation, and Removal. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1021–1035.

[26] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.* 13, 12 (jul 2020), 2747–2759.

[27] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).

[28] Wenxuan Zhang, Sharifah Mahani Aljunied, Chang Gao, Yew Ken Chia, and Lidong Bing. 2023. M3Exam: A Multilingual, Multimodal, Multilevel Benchmark for Examining Large Language Models. arXiv:2306.05179 [cs.CL]

[29] Qi Zhou et al. 2022. SPES: A Symbolic Approach to Proving Query Equivalence Under Bag Semantics. In *2022 ICDE*. 2735–2748.

[30] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. arXiv:2307.15043 [cs.CL]