Grounding LLMs for Database Exploration: Intent Scoping and Paraphrasing for Robust NL2SQL

Catalina Dragusin*
ETH Zurich
Zurich, Switzerland
catalina.dragusin10@gmail.com

Katsiaryna Mirylenka*
Zalando Switzerland AG
Zurich, Switzerland
katya.mirylenka@zalando.ch

Christoph Miksovic Czasch
IBM Research
Zurich, Switzerland
cmi@zurich.ibm.com

Michael Glass
IBM Research
Yorktown NY, Unites States
mrglass@us.ibm.com

Nahuel Defosse
IBM Research
Nairobi, Kenya
nahuel.defosse@ibm.com

Paolo Scotton
IBM Research
Zurich, Switzerland
psc@zurich.ibm.com

Thomas Gschwind
IBM Research
Zurich, Switzerland
thg@zurich.ibm.com

ABSTRACT

Large language models (LLMs) hold immense promise for democratizing data access through natural language interaction with complex databases. This paper addresses the challenge of accurately translating user intent into executable SQL queries (NL2SQL) by introducing a novel framework that enhances LLM-driven database exploration. Our approach centers on two key techniques: user intent scoping, which assesses a query's answerability against the database schema, and high-quality paraphrasing. User intent scoping classifies queries as fully answerable, partially answerable, or out-of-scope, providing crucial user guidance and preventing misinterpretations. High-quality paraphrasing augments training data with diverse, semantically equivalent query formulations. By selectively preserving critical entities while varying linguistic structure, this technique improves the robustness and generalizability of LLM-based NL2SQL systems. We evaluate our framework on two public cross-domain datasets (Spider and Bird) and a proprietary business intelligence dataset. Results demonstrate significant improvements in query accuracy and user guidance compared to baseline methods. Notably, paraphrasing enhances performance in fine-tuning and retrieval-augmented prompting scenarios. Scope detection effectively identifies and manages out-of-scope and partially answerable queries, facilitating more effective data exploration. This work paves the way for more reliable and user-friendly LLM-powered systems for navigating and querying tabular data with low annotation and medium engineering efforts.

VLDB Workshop Reference Format:
Catalina Dragusin*, Katsiaryna Mirylenka*, Christoph Miksovic Czasch, Michael Glass, Nahuel Defosse, Paolo Scotton, and Thomas Gschwind. Grounding LLMs for Database Exploration: Intent Scoping and Paraphrasing for Robust NL2SQL. VLDB 2025 Workshop: Applied AI for Database Systems and Applications (AIDB 2025).

*Work performed while Catalina D. was a Master's student and Katsiaryna M. was a staff research scientist at IBM Research, Zurich, Switzerland.

1 INTRODUCTION

The democratization of data access is crucial for informed decision-making across diverse domains [15, 27]. Large language models (LLMs) possess immense potential to facilitate natural language exploration of domain-specific tabular data within institutional data warehouses [11, 26, 51], contributing to the reliable conversational data analytics [1]. This empowers both technical and non-technical users to seamlessly interact with data, fostering data-driven insights. However, developing a reliable LLM-based system requires solving key challenges, such as effectively aligning user natural language intent with the specific domain of the underlying database schema [24]. This work aims to leverage the inherent capabilities of out-of-the-box LLMs, primarily in a few-shot setup, to assess the extent to which practitioners can rely solely on LLMs for building robust natural language applications.

Natural Language to SQL (NL2SQL) is a critical task that translates natural language questions into executable SQL queries [22]. Traditional domain alignment in NL2SQL often involves schema annotation, such as creating seed lexicons [42] or utilizing domain-specific dictionaries/ontologies [33]. These methods can be cumbersome and limit flexibility, particularly when user intent deviates from predefined vocabularies or schema annotations.

The advent of LLMs has revolutionized NL2SQL, offering compelling solutions due to their exceptional natural language understanding [11]. LLM-based NL2SQL approaches primarily involve fine-tuning [37] or in-context learning [7, 30]. Retrieval-Augmented

Proceedings of the VLDB Endowment. ISSN 2150-8097.

Generation (RAG) [21] further enhances performance by incorporating information from external knowledge bases, such as the database itself [48] or question-SQL pairs [8]. RAG improves grounding and reduces hallucinations, but its effectiveness is based on accurate semantic retrieval. Retrieval Augmented Prompting, a specific type of RAG, uses similar question-SQL pairs as few-shot examples [4].

Fine-tuning and Retrieval Augmented Prompting rely on domain-specific NL2SQL samples, often expert-crafted or synthetic. Augmentation, notably LLM-driven paraphrasing, boosts data diversity. Fine-tuning and Retrieval Augmented Prompting rely on domain-specific NL2SQL samples, often expert-crafted or synthetic. Augmentation, notably LLM-driven paraphrasing, boosts data diversity. However, while LLMs excel at general paraphrasing [45], a generic paraphrase can be detrimental for NL2SQL, as it may alter critical entities (e.g., IDs, numerical values) essential for a correct SQL query. Our work introduces a controlled paraphrasing technique specifically for data augmentation. It reliably preserves essential entities while generating diverse linguistic variations of the user's intent, creating a more robust training set for downstream NL2SQL models.

To enhance domain adaptation in NL2SQL systems, we incorporate LLM-based scope detection. This process, also known as user intent scoping [50], determines the coverage of a user's query by the database schema, assessing whether the database contains the necessary information to answer it. This classification, which divides queries into fully answerable, partially answerable, and out-of-scope categories, precedes SQL generation. By identifying and filtering out-of-scope queries, we mitigate error propagation and ensure schema adherence. Furthermore, for partially answerable queries, our approach enables proactive elicitation of user clarification, facilitating iterative refinement and improving the precision of data retrieval. This mechanism promotes a more robust and user-centric interactive data exploration paradigm.

Main Contributions. We propose a novel bilateral approach to enhance LLM-based SQL generation and domain adaptation consisting of:

(1) **User Intent Scoping** We introduce a novel method to classify user queries as in-scope, partially in-scope, or out-of-scope. This guides users and improves SQL generation accuracy by flagging unanswerable queries and suggesting answerable portions for partially in-scope questions.
(2) **High-Quality Paraphrasing**: We develop a paraphrasing technique incorporating named entity recognition (NER) and index classification to capture user intent across diverse natural language variations.

We evaluate our methods on three datasets: two public cross-domain benchmarks and a proprietary domain-specific dataset containing Business Intelligence data. The proprietary dataset presents additional challenges as it is not statistically covered by general LLM knowledge. The results demonstrate the efficiency of the scope detection method that improves the user experience by providing clear guidance on query answerability. Additionally, paraphrasing improves NL2SQL generation accuracy for both fine-tuned LLMs and RAG-assisted approaches. The code for this work is provided in supplementary material.

2 RELATED WORK

The NL2SQL task aims to bridge the gap between non-technical users and database systems by generating SQL queries from natural language questions [11]. Traditional methods, including rule-based approaches [23] and deep learning techniques [9, 40], struggle with complex queries [17].

Recent advancements in LLMs have provided promising alternatives due to their robust language understanding, flexibility, and conversational capabilities [6]. The potential for fine-tuning and in-context learning has been widely explored [7, 25, 37], with further improvements gained by adapting LLMs for specific structured tasks [3] and employing techniques like Reinforcement Learning from AI Feedback (RLAIF) to enhance generation quality [10].

The success of LLMs in NLP tasks, including NL2SQL, is largely attributed to the transformer architecture [39] and its self-attention mechanisms. These mechanisms enable efficient training on large datasets and effective handling of long-range dependencies. Fine-tuning these pre-trained models has demonstrated significant improvements in domain-specific applications such as NL2SQL [25].

Recognizing unanswerable questions is crucial for robust NL2SQL systems. Zeng et al. [50] proposed a system that classifies questions as translatable or not using a classifier. Similarly, Kochedykov et al. [18] incorporated an Out-Of-Domain question detection component. In this work, we propose a hybrid approach that combines information retrieval and LLM-based NER for robust scope detection, effectively guiding users towards available database information.

Named Entity Recognition (NER) plays a critical role in identifying and categorizing entities within text. Recent work, such as GPT-NER [41], has adapted NER to a text generation task. Building upon these ideas, we developed an LLM-based NER approach that associates named entities with database columns.

Synthetic data generation is crucial for efficient LLM fine-tuning. Techniques like Self-Instruct [43] utilize LLMs to generate training examples. In the context of NL2SQL, data augmentation is extensively employed [25, 34, 36, 37]. Sun et al. [37] generate synthetic data by creating diverse SQL queries. Li et al. [25] introduce a bi-directional data augmentation method involving Question-to-SQL and SQL-to-Question generation. Paraphrasing is also used for data augmentation in NL2SQL, aiming to introduce more variability in the training data [34, 36]. In this work, we introduce a controlled, high-quality user query paraphrasing that retains the entities of the initial question. A complementary approach to handle data scarcity is active learning, which focuses on intelligently selecting the most informative samples for annotation from a pool of unlabeled data [44].

Beyond data augmentation, paraphrasing is also leveraged for improved LLM evaluation by enhancing the diversity of reference texts [38]. However, evaluating the quality of paraphrases poses a challenge, necessitating a combination of human evaluation and automatic metrics [35], the approach that we also follow and enhance in this work.

3 PRELIMINARIES

In this paper, we establish the following definitions:

Entity is a real-world object such as a person, address, or country, that corresponds to a database column. Unlike the classical named

entities, each entity in this context refers to one column in the given database. Therefore, in this setting, the task of *NER* aims to identify the entities that can be associated with a column, instead of those that can be assigned to a predefined class.

Definition 1. *(Entity) Given a database $\mathcal{D}$, consisting of n tables $T_1, T_2, ..., T_n$, where each table T_i has the columns $C_{i1}, C_{i2}, ..., C_{im_i}$ where m_i represents the number of columns in table T_i. A term x that is part of a string of text symbols $X = \{x_1, x_2, ..., x_t\}$ can be identified as an **entity** ϵ if it corresponds to a column C_{ij} in $\mathcal{D}$, where $i \in \{1, 2, ..., n\}$ and $j \in \{1, 2, ..., m_i\}$: $\epsilon : x \sim C_{ij}$.*

Definition 2. *(Column Matching Logic) Considering the database $\mathcal{D}$ mentioned in Definition 1, we define three types of **matching logic** for column content: exact, approximate, and semantic. These types dictate the appropriate comparison strategy for values within a column.[1] The nature of these types is explained hereafter.*

The indexing and search strategy for columns will depend on their assigned matching logic. We consider three types:

Exact matching for column C means that a query q should be matched exactly with a value v in C: $I_e(q, C) = \{v \in C \mid v = q\}$.

Approximate matching for column C assumes that fuzzy matching is required, where a query q matches a value v if their edit distance $d(q, v)$ is below a threshold d_{max}: $I_a(q, C, d_{max}) = \{v \in C \mid d(q, v) \leq d_{max}\}$.

Semantic matching for column C refers to matching a query q with a value v based on its meaning, using a semantic similarity function $s(q, v)$. A match occurs if their similarity $s(q, v)$ is above a threshold t: $I_s(q, C, t) = \{v \in C \mid s(q, v) \geq t\}$.

4 PROBLEM FORMULATION

The NL2SQL task seeks to accurately map user queries to executable SQL. Cross-domain generalization remains a significant challenge due to variations in schema and user intent. In this paper we address two core subproblems for enhanced domain adaptation:

Scope Detection (Query Answerability): Given a user question q and a database D, the task of scope detection aims to decide whether the question q can be answered using the information in the database $\mathcal{D}$ (i.e. in scope of the given database $\mathcal{D}$), can be partially addressed using the information in the database $\mathcal{D}$ (i.e. partially in scope of the given database $\mathcal{D}$) or not at all (i.e. out of scope). Therefore, if $\mathcal{Q}$ is the set of all possible user questions q, the goal of this task is to determine a function $f : \mathcal{Q} \rightarrow \{0, 1, 2\}$ that would assign each user question $q \in \mathcal{Q}$ a label in the set of labels {0, 1, 2}, where 0 corresponds to not in scope, 1 to in scope and 2 to partially in scope, given the information in database $\mathcal{D}$.

High-quality paraphrasing: Given a user question q and a database $\mathcal{D}$, the task of high-quality paraphrasing requires to output a paraphrase P(q) that maintains the idea or intent of the original question q. Moreover, all identified named entities in question q should be preserved in their exact, approximate, or semantically equivalent forms depending on their type. Let the set of identified entities in the question q be $E = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}$. From Definition 1, we know that each entity $\epsilon_i \in E$ is associated with a column C_i in the database $\mathcal{D}$ and, from Definition 2, we know that each column C_i is associated with an index type I_i which is either exact, approximate or semantic ($I_i \in \{I_e, I_a, I_s\}$). If the index I_i is semantic ($I_i = I_s$), it is desirable that the entity ϵ_i is paraphrased so that the variability of the dataset increases. Otherwise, if the index I_i is not semantic ($I_i \in \{I_e, I_a\}$), the entity ϵ_i should be preserved in its original form.

[1]While the term 'index' in traditional SQL databases refers to a specific data structure for performance, in this context, we use these terms to describe the *matching logic* required for a given column's content, which informs subsequent search and processing strategies.

5 PROPOSED APPROACHES

This section details our framework, which consists of two primary capabilities—Scope Detection and Paraphrasing—built upon a foundation of three shared, auxiliary components: Column Type Classification, Named Entity Recognition (NER), and Entity Search. Figure 1 illustrates the integrated workflow, showing how foundational components provide essential information for the main user-facing capabilities.

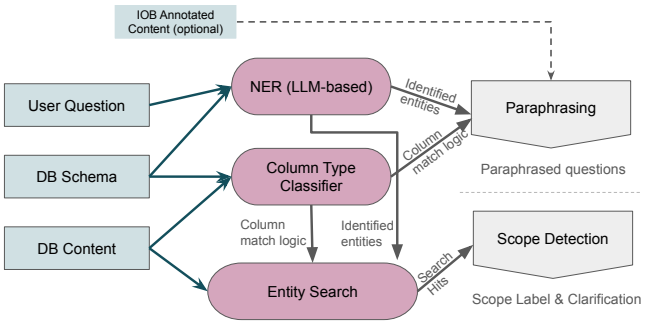

Figure 1: Natural Language Question processing pipeline for better SQL generation and user experience

5.1 LLM-based Named Entity Recognition

Given that only a limited amount of NL2SQL datasets contain annotated named entities within utterances along with their start and end positions (IOB), developing a NER method is instrumental for the paraphrasing approach. As noted in Definition 1, this task is slightly different from the classic NER as the entities have to correspond to database columns. Simultaneously, it partially resembles schema linking, as both tasks aim to map parts of a natural language question to elements of a database schema [16, 20]. The distinction lies in the fact that our task focuses exclusively on condition values (i.e. the literals used in the "WHERE" clause), whereas schema linking additionally aims to identify the column and table names mentioned in the utterance. This task is also related to the broader challenge of entity matching in databases, where various techniques, including graph-based neural networks, have been applied [19].

The decision to employ an LLM is motivated by the lack of suitably annotated datasets required to train a traditional NER tagger, and by the high variability of entity types (i.e., column names), which change for each database.

Inspired by the work on LLM-based NER such as GPT-NER [41], we propose a system that leverages a high-quality LLM in conjunction with domain knowledge from the database schema to extract

relevant entities without extensive additional training. This approach relies on the LLM's inherent ability to understand language and the provided schema information to reason about and identify essential entities.

The approach takes as input an utterance and a database schema and aims to identify potential named entities that correspond to columns in the schema. It, then, returns a list of these identified named entities in a specified JSON format which is provided in Appendix 9.4. We propose a NER component, leveraging the capabilities of an LLM, specifically Mistral-7B [13] that uses the following prompt:

```
### Input
Given the schema below, carefully consider and extract the named entities from the
user question:
user question = {utterance}
schema = {db_schema}
Produce the answer in the following json format {json_format}.
There can be one or more named entities.
Each entity is an individual entry in the response:
- put the entity value in the "entity_value" field
- put the corresponding schema column name in the "column_name" field
- put the corresponding schema table name in the "table_name" field
### Output:
```

Although the model receives a specific JSON format for the output, it may not always adhere to it. We implemented post-processing with regular expressions to parse the output. Each extracted triplet (entity value, column name, table name) is then validated: (a) the entity value must be a substring of the original utterance; (b) the table name must exist in the schema; (c) the column name must belong to the identified table. In cases of unrecoverable parsing errors (e.g., malformed JSON), the sample is considered a failure for that attempt. The proposed LLM-based NER workflow is illustrated in Figure 2.

5.2 Column Matching Logic Classification

The second auxiliary approach – column matching logic classification – aims to automatically assign the appropriate type (exact, approximate or semantic) to each database column. To achieve this, a form of data profiling [29] is conducted by computing different statistics for each column, which is subsequently used as features for training a machine learning classifier. These includes:

- *Textual characteristics*: average number of tokens and average value length to capture the textual complexity of the data (e.g., if the text is more complex, it would likely need a semantic comparison).
- *Length distribution*: variance of the column value length (e.g., if the variance is higher, a more flexible type of comparison may be needed for the column entities such as approximate or semantic).
- *Data type*: to identify inherent matching capabilities (e.g., a date should not be matched semantically, numbers should be matched exactly).
- *Uniqueness*: the number of unique values in a column and the ratio between the number of unique values in the column and the number of unique values in the entire table to understand the potential for exact matching (e.g., if all the values in a column are different, then, exact matching should be used such as in the case of ids).

In order to better capture the uniqueness of the text fields, TF-IDF and perplexity were first used in a series of experiments. As these measures did not lead to an improvement in classifier performance, they were omitted from the list of features. Manual feature selection was employed: all the selected features improved the performance of the classifier, while the ones that did not change or degraded the performance were discarded.

A Random Forest classifier [28] was trained on these features for the classification task. An index type classification algorithm was wrapped around the column type classifier, taking as input the database id, the name of the column and table of interest, and returning the predicted index type. The pseudo-code of the algorithm (and further proposed algorithms within this work) together with the hyperparameters for the classifier are available in Appendix.

5.3 Entity Search

The entity search component, an auxiliary subcomponent of the scope detection module, takes as input an entity query (which can be a word or multiple words) and a database and searches the query across all columns in that database. This functionality can be useful for determining whether the information required to answer a user question can be found within the provided database.

The search service first retrieves the appropriate column type (exact, approximate, or semantic) as determined by the column classifier. It then employs a different search strategy based on the type:

Exact: Performs exact matching by querying the database and checking if any column values match the input (e.g., phone numbers or product identifiers).

Approximate: Utilizes fuzzy search using Elasticsearch[2] to accommodate minor variations in the query (e.g., "Cisco Inc." vs. "Cisco Incorporated").

Semantic: Leverages a vector store to retrieve the most semantically similar results, using BAAI general embedding (BGE) [47] as the embedding model and the cosine similarity as a similarity function, filtered by a threshold, to handle situations where the searched term might be phrased differently.

5.4 Scope Detection

As described in Section 4, the scope can be classified into three categories illustrated in Figure 3:

In Scope: The query entirely refers to information present in the database (e.g., "Which is the item with the lowest price that I can order?").

Partially in Scope: The query seeks information that is partially available in the database (e.g., "Which is the email and birth date of the customer with id 123?" if the birth date information is not stored in the database).

Not in Scope: The query requests information not stored in the database (e.g., "How will the weather be tomorrow?").

We propose two approaches for scope detection:

(1) Easy Scope Detection: identifies the columns mentioned in the utterance by prompting an LLM (mixtral-8x7b-instruct-v01) and then verifies their presence in the database schema. If all the identified

[2]https://elasticsearch-py.readthedocs.io/en/v8.13.1/

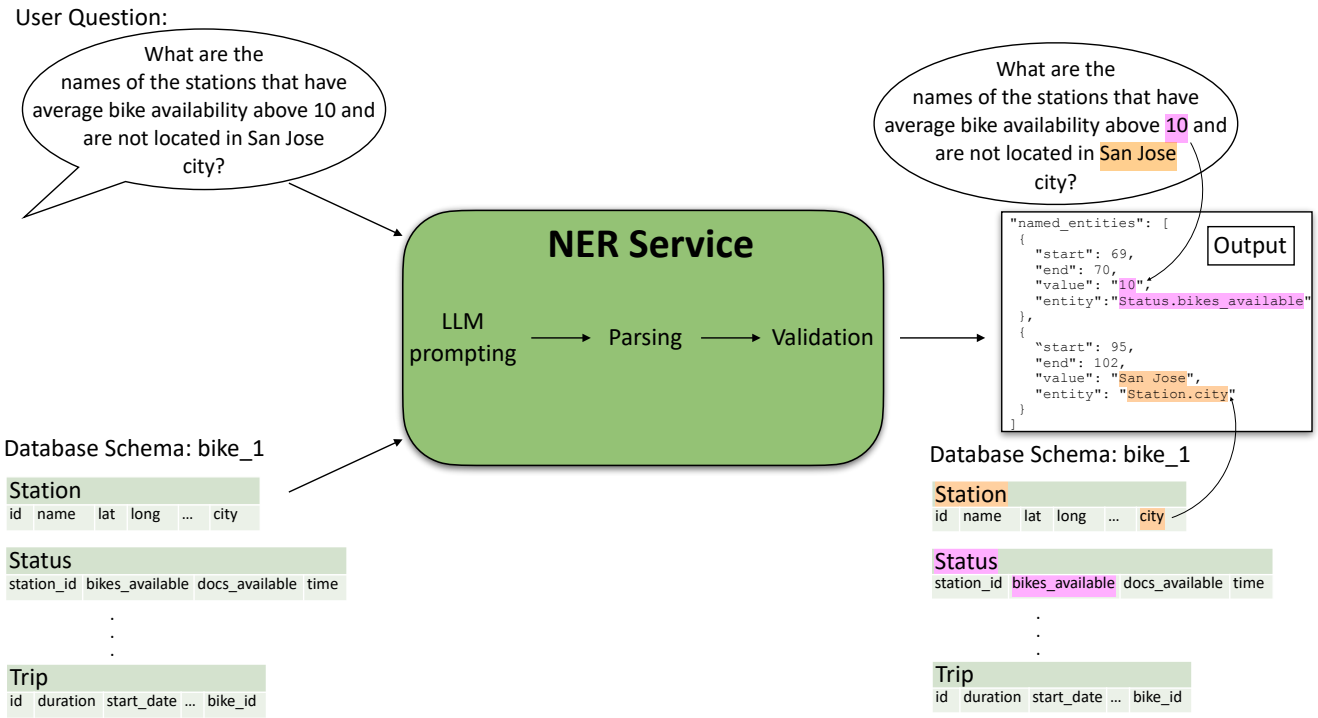

Figure 2: The proposed LLM-Based NER approach

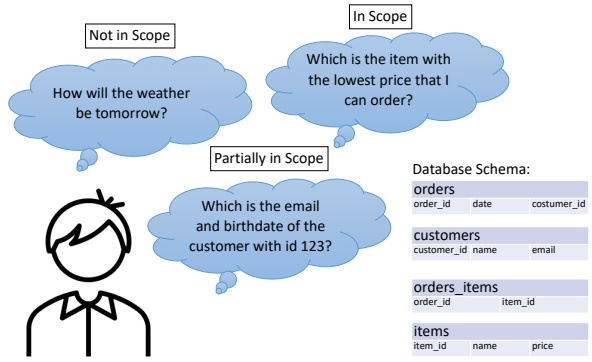

Figure 3: Categories of scope relevance of the user question

columns are not found in the schema, then the question is considered out of scope while if all of them are present in the schema - in scope, otherwise, the questions are labeled as partially in scope.
(2) Scope Detection with Search: This approach builds upon the "easy scope detection" method. After the entities are identified, they are searched in the database using the search service. The goal is to handle cases where column names are not explicitly mentioned, but the corresponding information is present. A question is labeled 'out-of-scope' if no corresponding columns are identified in the schema and no named entities are found in the database. It is labeled 'in-scope' if all identified columns are present in the schema. Otherwise, the question is considered 'partially-in-scope'. All the relevant prompts and algorithms can be found in Appendix.

5.5 Paraphrasing

The goal of our paraphrasing component is not to pre-process a query at inference time, but to perform offline data augmentation to create a richer and more diverse training dataset. While LLMs are proficient at general paraphrasing [45], naively applying them for NL2SQL data augmentation is risky. First, specific parts of the query are highly important for retrieving the right information and, therefore, must remain unchanged. For instance, in the queries: "Who is the advisor of student with ID 10045?", "Which are the mountains with a height in meters greater than 2500?". The ID "10045" and the height "2500" should remain, as the corresponding SQL query would have to include them in the WHERE statement (e.g., `WHERE student.ID = "10045"` or `WHERE mountain.height_in_m > "2500"`). If the ID "10045" or the height "2500" is omitted or altered by the LLM, the resulting SQL query will not return correct results. Secondly, other parts of the user questions require variation in order to enhance the robustness of the NL2SQL pipeline during training. These are notions that are semantically similar to certain column entries in the database. For example, if a query asks for products belonging to the "software" product category, the `product_category` column could be indexed semantically and "software" could be matched with similar entries within the product category column (e.g., `WHERE product.product_category = "Web Applications"`).

To achieve paraphrasing with these considerations, we propose to replace the identified entities in the user questions with placeholders and, after the rest of the utterance is paraphrased, to plug them back in, either additionally paraphrased, if the entity supports some variation, or not.

The proposed paraphrasing approach includes the following steps:

1. **Identify Entities** together with their corresponding columns and tables, using the NER approach proposed in Section 5.1 (if IOB is not available).
2. **Classify Column Types** to determines the type of entity column: exact, approximate or semantic as proposed in Section 5.2.
3. **Placeholder Replacement**: the entities are replaced with placeholders.
4. **Paraphrasing Entities**: Semantic entities are paraphrased using the flan-t5-xl model [31], [5] with k_1 attempts until a semantic similarity score of at least τ_1 is achieved.
5. **Paraphrasing Utterances** using the mixtral-8x7b [14] model with k_2 attempts until the following quality conditions are met: (a) a semantic similarity score of at least τ_2 but smaller than τ_3 is reached (excluding paraphrases which are too similar) ; (b) the number of placeholders in the paraphrase is equal to the number of placeholders in the original question and the values of the placeholders remain unchanged; (c) the length of the paraphrase is smaller than double the length of the original question.
6. **Placeholder Replacement** using paraphrased entities or original values.

The steps of the paraphrasing process together with an illustrative example are presented in Figure 4 and Figure 5.

To ensure paraphrasing quality, cosine similarity scores were computed for both entities and entire paraphrased utterances. The thresholds τ_i, where $i \in \{1, 2, 3\}$, are determined arbitrarily based on the similarity score distribution of "good" and "bad" paraphrase variations for both entities and utterances (Figure 6). The "good" and "bad" labels were assigned to paraphrased entities and utterances by a human evaluator. A paraphrased entity is considered "good" if it is a synonym of the original one and "bad" otherwise while a paraphrased utterance is considered "good" if the main idea is preserved and the placeholders remain unchanged and "bad" if this is not the case or if it is too similar to the original question. The distributions were derived from an annotated held-out set of the BI dataset by plotting the cosine similarity between the original samples and their paraphrases. Given the histograms, entity generations were filtered with $\tau_1 = 0.7$, and utterances with $\tau_2 = 0.75$ and $\tau_3 = 0.95$. The overlapping distributions of good and bad paraphrases for utterances may indicate that the cosine similarity is not the most suitable metric. A paraphrase which omits an important detail could still be semantically similar to the original user question. It is still useful to filter out the very low quality paraphrases or the paraphrases which are too similar to the original utterance. Other assessments of the quality of the paraphrase are performed such as verifying if all the placeholders for entities are preserved after the paraphrasing step or if the length of the paraphrase is no more than double the length of the original user question.

6 EXPERIMENTAL EVALUATION

This section evaluates the proposed scope detection and paraphrasing approaches, as well as their auxiliary subcomponents. We follow the approach hierarchy outlined in Figure 1 and assess their effectiveness on the following datasets:

1. Spider [49] - a big, complex and cross-domain dataset for the NL2SQL task, manually annotated by college students. The number of databases in Spider is 200 and they span across 138 domains. There dataset split is 7000/1034 question-SQL pairs for train and dev correspondingly. In this work, only the samples from train set are considered.

2. Bird [26] is a cross-domain dataset that is more challenging for the task of NL2SQL due to its 95 databases which are big and contain poorly formatted data, aiming to reflect real life data. Bird consists of over 12,751 question-query pairs, covering more than 37 domains. The train set contains 9428 question-SQL pairs while the dev set is made of 1534 samples.

3. Proprietary BI (Business Intelligence) dataset, covers a database with 5 tables and 37 columns. The corpus consists of 5000 question-query samples. The questions have the named entities annotated together with the column and table they belong to, as well as their start and end positions. The test set is made up of 113 human-annotated question query pairs and 500 generated held-out samples.

6.1 Evaluation of LLM-based NER

The NER subcomponent was evaluated on manually annotated entities for 70 randomly selected user questions of the Spider dataset. The manual annotation was done by two NLP and DB experts with quality control checks. Any disagreements were resolved through discussion to establish a ground truth. The performance of the NER component was measured by:

(A) Exact Accuracy - considers a prediction to be correct only if the model identifies all entities perfectly, including the values, tables, and columns.

(B) Overall Accuracy - considers a prediction partially correct if it identifies at least one entity value or a corresponding table and column name pair correctly. If all entities are correctly identified for a user question, the sample receives a full score as in the case of Exact Accuracy; otherwise, it receives a fraction of the score proportional to how many values, column names, and table names were correctly identified.

We focused on LLMs with permissive licenses to ensure our methods are accessible to practitioners. Table 1 compares the performance of the various LLMs that were used for the NER task. Mistral-7b-instruct-v02 performs the best according to both exact and overall accuracy, getting the highest number of correctly tagged user questions. The LLM-based NER service achieved an average exact accuracy of 53.2% and an overall accuracy of 67.3% across 20 repeated runs with a small standard deviation.

LLM-fine tuning for the NER task has potential to improve the performance, but this would require generation of training data that have the named entities annotated with the corresponding database columns. Such datasets annotated by humans are expensive and time consuming to produce. In addition many of the best performing LLMs cannot be used for training data generation due to the restrictive licensing for business needs.

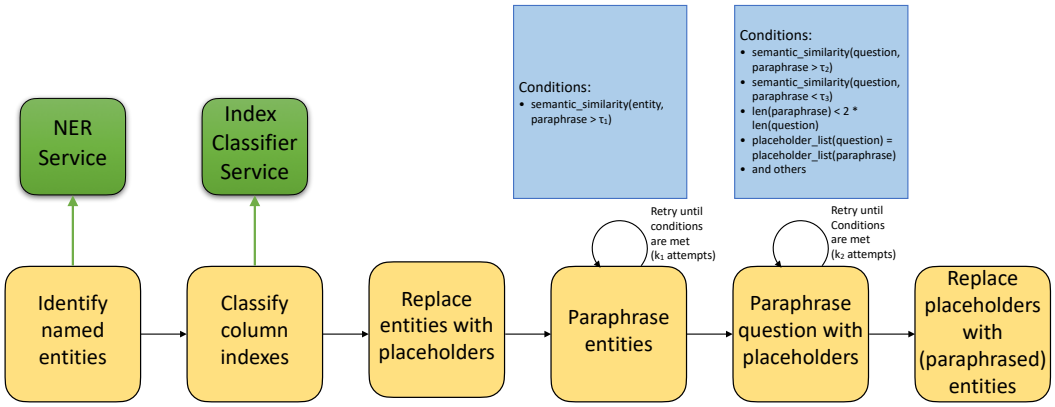

Figure 4: The steps of the paraphrasing process

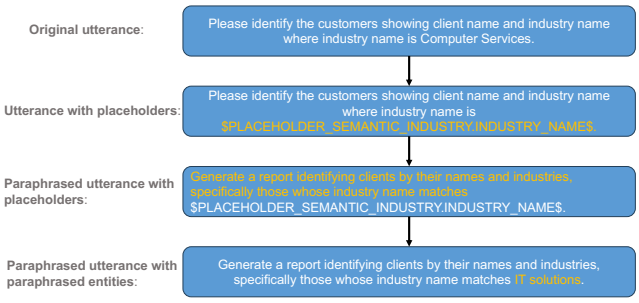

Figure 5: Example of the paraphrasing process

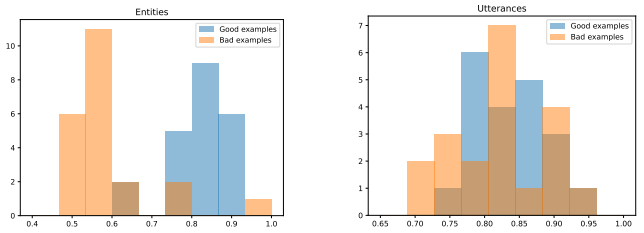

Figure 6: Semantic Similarity Histogram for Paraphrased Entities (left) and Paraphrased Utterances (right)

Table 1: Exact and Overall Accuracy of Different LLMs for the NER Task

LLM	llama-70b	mixtral-8x7b	falcon-180b	mistral-7b
Exact A.	24.1	22.5	42.9	**53.2**
Overall A.	48.3	54.0	56.0	**67.3**

6.2 Column Classifier Evaluation

We created a labeled dataset for the column classifier from the Spider dataset. After excluding databases without schema files, we processed 157 databases containing 837 tables. For each column in non-empty tables, we computed the statistical features described in Section 5.2. From this pool, we randomly selected 350 columns (304 textual and 46 date-based). A larger sample size was chosen here to ensure sufficient diversity in column statistics for training a robust classifier. Two authors manually labeled each column as 'Exact', 'Approximate', or 'Semantic' based on its content and likely usage in queries, with disagreements resolved by a third author. A stratified train/val/test split (60%/20%/20%) was used to maintain label distribution.

We compared the performance of various classifiers for this task, namely RandomForest, KNeighbors (XGBoost), and DecisionTree classifiers using micro-F1(accuracy) and macro-F1 scores (Table 2). The Random Forest achieved the best performance according to both scores, so it was used in the deployed system.

Table 2: Performance on Column Classification Task

Classifier	DecisionTree	XGBoost	RandomForest
Micro F1 (Acc.)	64.2	74.2	**80.0**
Macro F1	56.7	69.6	**75.4**

6.3 Scope Detection Evaluation

We identified 4 categories of interest for user utterances: **(A).** Completely unrelated to data exploration, for example, 'How are you?'. **(B).** Relevant for the domain database at hand: **(B1)** In the scope of a given database; **(B2)** Partially in the scope; **(B3)** Not in the scope.

Evaluation Dataset. To create a comprehensive evaluation set, we constructed a dataset of 400 samples, balanced across four categories, using the 9 most popular databases from the Spider and Bird dev sets.

- **Category A (Unrelated):** 100 questions were randomly sampled from the SQuAD dataset [32] and paired with a random database to simulate completely irrelevant queries. These were labeled as 'out-of-scope'.

- **Category B1 (In-Scope):** 100 questions were randomly sampled from the dev sets and paired with their correct database, labeled as 'in-scope'.
- **Category B2 (Partially In-Scope):** 100 in-scope questions were taken, but their corresponding database schemas were programmatically altered by removing one or more columns essential for answering the question. These were labeled as 'partially-in-scope'.
- **Category B3 (Out-of-Scope, Domain Mismatch):** 100 questions from the dev sets were paired with a randomly chosen database *other than* their correct one, simulating queries that are topically relevant but mismatched to the specific database. These were also labeled 'out-of-scope'.

This construction method provides a challenging benchmark that tests a system's ability to distinguish not just relevant from irrelevant queries, but also fully answerable from partially answerable.

Other works which present scope detection approaches [18, 50] do not state the results of the scope detection component separately, providing only the performance of their whole NL2SQL systems with other evaluation datasets. Because of this, we use as a baseline the results of the vanilla LLM (mixtral-8x7b-instruct-v01) prompt (please see Appendix) in zero-shot fashion.

Table 3: Scope Detection (SD) Results

	Spider Dev		Bird Dev	
Method	Accuracy	Macro F1	Accuracy	Macro F1
LLM baseline	39.25	39.58	34.50	34.65
Easy SD	**74.50**	69.29	**66.00**	**58.07**
SD with Search	74.25	**69.88**	56.00	54.10

The results for Spider dev and Bird dev are provided in Table 3. The most common problem for the pure LLM-based approach is that the 'not is scope' and the 'partially in scope' classes are heavily confused. Most probably this is because LLM is not fully aware of the domain of the used databases. The easy scope detection method performs the best while the scope detection with search approach is performing similarly for Spider but worse for Bird. We expect that it could be further improved by increasing index classification and search quality. Confusion matrices for scope detection methods are provided in Appendix.

6.4 Paraphrasing Evaluation

We used the following paraphrasing quality evaluation metrics:
Cosine Similarity Score (CSS). measures the semantic similarity between the original utterance and its paraphrase both being projected in the same embedding space.[3] As paraphrases that are too similar to the original are not informative, we penalize scores above a threshold (0.98) by setting them to zero.

[3]BAAI general embedding (BGE) [47] was used as the embedding model to calculate the cosine similarity. BGE is one of the best performing text embedding models [12].

Human evaluation score (HES). $HES = 0.5 \times paraphrase\_score + 0.5 \times named\_entity\_score$, where

$$paraphrase\_score = \begin{cases} 0 & \text{changed intent} \\ 0.5 & \text{partially good} \\ 1 & \text{same intent} \end{cases}$$

and $named\_entities\_score$ for each entity is defined as follows: (i) if it belongs to a column which should have a semantic index: $named\_entity\_score = \mathbb{1}$(if paraphrased) (ii) if it belongs to a column which should not have a semantic index: $named\_entity\_score = \mathbb{1}$(if kept the same). The final HES is the average of the paraphrase score and the average of all entity scores.

We compared the performance of several LLMs for paraphrasing with the results demonstrated in Table 4. The table includes the number of paraphrases generated for a set of 20 utterances and the time taken. Mixtral-8x7b-instruct-v0-1 without few-shots has the best trade-offs between quality (number of good paraphrases) and efficiency (inference time). This model was chosen for paraphrasing service deployment.

Table 4: Successful Paraphrasing generation over 20 trials

Model	# of Successes	Time
mistral-7b	18/20	3m50.76s
mixtral-8x7b	16/20	3m40.74s
mixtral-8x7b-gptq	15/20	23m19.68s*
mistral-7b w/o few-shots	19/20	3m25.98s
mixtral-8x7b w/o few-shots	20/20	2m57.12s

*The higher runtime may be due to overhead from the specific quantization library or suboptimal hardware configuration.

Table 5 presents results for cosine similarity score (using BGE embeddings) and human evaluation score (HES) for all the three datasets for the paraphrase generation task, showing that the proposed approach of high-quality paraphrasing outperforms the baseline approaches.

	cosine similarity (BGE)		Human Eval. Score (HES)	
Dataset	Our	Mixtral	Our	Mixtral
Spider	**88.1**	85.9	**92.5**	82.5
Bird	**89.8**	85.9	**96.3**	81.3
BI	**88.5**	88.0	**92.8**	89.0

Table 5: Paraphrasing assessment with different evaluation strategies

Fine Tuning results are provided in Table 6. We first use original train set of the benchmark for fine-tuning to get the generation accuracy results. Then, the training set is augmented with paraphrasing, the same model is fine-tuned with the augmented set and evaluated for comparison. For the BI dataset we used a Granite 20B model, while for the others we used a Deepseek 6.7B model. Models

have been trained using 8x NVIDIA A100 80GB cards as accelerators using the `SFTTrainer` from `TRL` and leveraging DeepSpeed ZeRO-3.

RAG. In this experiment, we use Qdrant[4] as a vector store and all-MiniLM-L6-v2[5] as for semantic embeddings. Two collections of embeddings are created and used for retrieving the three shots to a user question: one which contains all train samples and the other which contains all train samples plus their paraphrases. The execution accuracy results for the three datasets are provided in Table 6.

Table 6: Effect of paraphrasing added to SFT and RAG for SQL execution accuracy

Dataset	SFT	SFT + paraph.	RAG	RAG + paraph.
SPIDER	**81.1**	79.7	**54.6**	54.4
BIRD	54.1	**54.7**	54.7	**55.1**
BI	83.2	**84.4**	84.7	**85.3**

Paraphrases are indeed helpful for better SQL generations, producing accuracy increase in all the classes of SQL complexity for Bird and BI datasets. At the same time, paraphrasing was not helpful for the Spider benchmark. Apparently, most of the samples in Spider already have up to 6 human created paraphrases per SQL. This is not the case for the other two datasets, that initially have one unique utterance per SQL and do benefit from augmentation with paraphrasing.

7 CONCLUSION

In this paper, we introduced a framework to improve the reliability and domain adaptability of LLM-driven database exploration systems. Our contributions are twofold: a user intent scoping mechanism to classify query answerability, and a high-quality paraphrasing technique for data augmentation.

Our scope detection method effectively identifies in-scope, partially-in-scope, and out-of-scope queries. By filtering unanswerable questions before the SQL generation stage, this component can prevent the system from producing incorrect or nonsensical queries for invalid user intents – a common source of LLM 'hallucinations'. This aligns with the broader goal of improving system reliability. It touches upon the challenge of understanding model uncertainty, where recent work has explored consistency-based hypotheses to quantify uncertainty in both general LLMs [46] and specific Text-to-SQL applications [2]. Scope detection also provides a mechanism for interactive systems to request user clarification, leading to a more robust user experience.

Our controlled paraphrasing technique successfully augments training data, improving the performance of both fine-tuned and RAG-based NL2SQL models, especially on datasets that lack initial linguistic diversity like BIRD and our proprietary BI set.

8 LIMITATIONS AND FUTURE WORK

Our work has several limitations that open avenues for future research. First, the current components focus on mapping entities and scoping intent, but do not generate complex SQL constructs like multi-table joins or nested subqueries. Integrating our framework into a full end-to-end system that handles such complexity is a key next step. Second, the performance of our framework is dependent on the accuracy of its underlying LLM-based components (e.g., NER). Errors in these foundational steps can propagate. Future work could explore hybrid methods that combine LLM flexibility with rule-based checks for greater accuracy. Finally, our experiments used established open-source models. Evaluating the framework with newer, state-of-the-art LLMs could yield further performance improvements. We believe the principles of intent scoping and controlled paraphrasing will remain vital for building the next generation of trustworthy and user-friendly NL2SQL systems.

[4]https://qdrant.tech/

[5]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

REFERENCES

[1] Sihem Amer-Yahia, Jasmina Bogojeska, Roberta Facchinetti, Valeria Franceschi, Aristides Gionis, Katja Hose, Georgia Koutrika, Roger Kouyos, Matteo Lissandrini, Silviu Maniu, Katsiaryna Mirylenka, Davide Mottin, Themis Palpanas, Mattia Rigotti, and Yannis Velegrakis. 2025. Towards Reliable Conversational Data Analytics. In *28th International Conference on Extending Database Technology, EDBT 2025*. OpenProceedings. org, 962–969.

[2] Debarun Bhattacharjya, Balaji Ganesan, Michael Glass, Junkyu Lee, Radu Marinescu, Katsiaryna Mirylenka, and Xiao Shou. 2024. Consistency-based Black-box Uncertainty Quantification for Text-to-SQL. In *NeurIPS 2024 Workshop on Statistical Foundations of LLMs and Foundation Models*.

[3] Robin Chan, Katsiaryna Mirylenka, Thomas Gschwind, Christoph Miksovic, Paolo Scotton, Enrico Toniato, and Abdel Labbi. 2024. Adapting LLMs for Structured Natural Language API Integration. In *EMNLP*. 991–1000.

[4] Xiang Chen, Lei Li, Ningyu Zhang, Xiaozhuan Liang, Shumin Deng, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Decoupling knowledge from memorization: Retrieval-augmented prompt learning. *NeurIPS* 35 (2022), 23908–23922.

[5] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research* 25, 70 (2024), 1–53.

[6] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[7] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. arXiv:2308.15363 [cs.DB] https://arxiv.org/abs/2308.15363

[8] Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2023. Retrieval-augmented GPT-3.5-based Text-to-SQL Framework with Sample-aware Prompting and Dynamic Revision Chain. arXiv:2307.05074 [cs.IR] https://arxiv.org/abs/2307.05074

[9] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL*.

[10] Noah Hampp and Katya Mirylenka. 2024. Reward Modeling and RLAIF for Improved Natural Language to SQL Generation. In *MML, The Mathematics of Machine Learning Workshop, ETH Zurich*.

[11] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426 [cs.CL] https://arxiv.org/abs/2406.08426

[12] HuggingFace. 2024. Massive Text Embedding Benchmark (MTEB) Leaderboard. https://huggingface.co/spaces/mteb/leaderboard. [Online; accessed 21-May-2024].

[13] Albert Q. Jiang, Alexandre Sablayrolles, et al. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL]

[14] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[15] Yongyao Jiang and Chaowei Yang. 2024. Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured Query Language within a Spatial Database. *ISPRS International Journal of Geo-Information* 13, 1 (2024), 26.

[16] George Katsogiannis-Meimarakis, Katsiaryna Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. 2026. In-depth Analysis of LLM-based Schema

Linking. In *29th International Conference on Extending Database Technology (EDBT)*.
[17] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.
[18] Denis Kochedykov, Fenglin Yin, and Sreevidya Khatravath. 2023. Conversing with databases: Practical Natural Language Querying. In *EMNLP*. 372–379.
[19] Evgeny Krivosheev, Mattia Atzeni, Katsiaryna Mirylenka, Paolo Scotton, Christoph Miksovic, and Anton Zorin. 2021. Business Entity Matching with Siamese Graph Convolutional Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 15882–15883.
[20] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *EMNLP*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.).
[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeirIPS* 33 (2020), 9459–9474.
[22] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *arXiv preprint arXiv:2406.01265* (2024).
[23] Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84.
[24] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *AAAI*, Vol. 37. 13067–13075.
[25] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, and Renjie Wei. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. arXiv:2402.16347 [cs.CL]
[26] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *NeurIPS* 36 (2024).
[27] Yunyao Li et al. 2024. *Natural language interfaces to databases*. Springer.
[28] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.
[29] Felix Naumann. 2014. Data profiling revisited. *ACM SIGMOD Record* 42, 4 (2014), 40–49.
[30] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL] https://arxiv.org/abs/2304.11015
[31] Colin Raffel, Noam Shazeer, Adam Roberts, et al. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683 [cs.LG]
[32] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822* (2018).
[33] Diptikalyan Saha et al. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *VLDB* (2016), 1209–1220.
[34] Irina Saparina and Mirella Lapata. 2024. Improving Generalization in Semantic Parsing by Increasing Natural Language Variation.
[35] Lingfeng Shen, Lemao Liu, Haiyun Jiang, and Shuming Shi. 2022. On the evaluation metrics for paraphrase generation. *arXiv preprint arXiv:2202.08479* (2022).
[36] Fatemeh Shiri, Terry Yue Zhuo, Zhuang Li, Van Nguyen, Shirui Pan, Weiqing Wang, Reza Haffari, and Yuan-Fang Li. 2023. Paraphrasing Techniques for Maritime QA system.
[37] Ruoxi Sun and Sercan Ö. Arik et al. 2024. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (extended). arXiv:2306.00739 [cs.CL]
[38] Tianyi Tang, Hongyuan Lu, Yuchen Eleanor Jiang, Haoyang Huang, Dongdong Zhang, Wayne Xin Zhao, and Furu Wei. 2023. Not all metrics are guilty: Improving nlg evaluation with llm paraphrasing. *arXiv preprint arXiv:2305.15067* (2023).
[39] Ashish Vaswani et al. 2017. Attention is all you need. *NeurIPS* 30 (2017).
[40] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2021. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. arXiv:1911.04942 [cs.CL]
[41] Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. 2023. Gpt-ner: Named entity recognition via large language models. *arXiv preprint arXiv:2304.10428* (2023).
[42] Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a Semantic Parser Overnight. In *ACL*, Chengqing Zong and Michael Strube (Eds.). 1332–1342.
[43] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560* (2022).
[44] Lukas Wertz, Katsiaryna Mirylenka, Jonas Kuhn, and Jasmina Bogojeska. 2022. Investigating active learning sampling strategies for extreme multi label text classification. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. 4597–4605.
[45] Sam Witteveen and Martin Andrews. 2019. Paraphrasing with Large Language Models. In *ACL*.
[46] Quan Xiao, Debarun Bhattacharjya, Balaji Ganesan, Radu Marinescu, Katsiaryna Mirylenka, Nhan H Pham, Michael Glass, and Junkyu Lee. 2025. The Consistency Hypothesis in Uncertainty Quantification for Large Language Models. In *The Conference on Uncertainty in Artificial Intelligence (UAI)*.
[47] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597 [cs.CL]
[48] Wenbo Xu, Liang Yan, Peiyi Han, Haifeng Zhu, Chuanyi Liu, Shaoming Duan, Cuiyun Gao, and Yingwei Liang. 2024. TCSR-SQL: Towards Table Content-aware Text-to-SQL with Self-retrieval. arXiv:2407.01183 [cs.DB] https://arxiv.org/abs/2407.01183
[49] Tao Yu, Rui Zhang, 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).
[50] Jichuan Zeng, Xi Victoria Lin, Caiming Xiong, Richard Socher, Michael R Lyu, Irwin King, and Steven CH Hoi. 2020. Photon: A robust cross-domain text-to-SQL system. *arXiv preprint arXiv:2007.15280* (2020).
[51] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: Large language model meets database. *Data Science and Engineering* 9, 1 (2024), 102–111.

9 APPENDIX

9.1 Column Type Detection

A Random Forest classifier [28] mentioned in the paper was tuned using the hyperparameters listed in in Table 7. Hyperparameter tuning was performed using grid search.

Hyperparameter	Value
bootstrap	True
max_depth	10
max_features	'sqrt'
min_samples_leaf	1
min_samples_split	2
n_estimators	50

Table 7: Random Forest Classifier hyperparameter values

The index classifier algorithm for the column follows the steps shown in Algorithm 1.

Algorithm 1 Column Classification

Input: Column and table name in the format "table_name.column_name", $table\_column$ of interest, and the database id: db_id
Output: the predicted index type i of $table\_column$
procedure CLASSIFY_COLUMN($table\_column, db\_id$)
    $column\_values \leftarrow$ `get_column_values`($table\_column, db\_id$)
    ▷ Get column values by querying the database
    if `len`($column\_values$) = 0 **or** $column\_values$.`dtype` = `numeric` **then**
        $i \leftarrow$ "$Exact$"
    end if
    $statistics \leftarrow$ `get_column_features`($column\_values$)
    $i \leftarrow$ `classifier.predict`($statistics$)
    return i
end procedure

9.2 LLM fine-tuning Hyperparameters

Hyperparameters for LLM fine-tuning and inference are specified in Table 8.

Algorithm 3 Easy Scope Detection

Input: The user question q, and the database id, db_id
Output: the predicted scope label of the given question and database, $scope\_label$
procedure EASY_SCOPE_DETECTION(q, db_id)
    $identified\_columns \leftarrow$ `llm_identify_columns`(q, db_id) ▷ Prompt the LLM to identify the columns in the given question
    $db\_schema \leftarrow$ `get_db_schema`(db_id)
    $columns\_in\_schema \leftarrow [\ ]$
    $columns\_not\_in\_schema \leftarrow [\ ]$
    for $column$ in $identified\_columns$ **do**
        if `column_in_schema`$(column, db\_schema)$ **then**
            $columns\_in\_schema \leftarrow columns\_in\_schema$.append$(column)$
        else
            $columns\_not\_in\_schema \leftarrow columns\_not\_in\_schema$.append$(column)$
        end if
    end for
    if len$(columns_in_schema)$ = len$(identified_columns)$ **then**
        $scope\_label \leftarrow$ "in_scope"
    else if len$(columns_not_in_schema)$ = len$(identified_columns)$ **then**
        $scope\_label \leftarrow$ "out_of_scope"
    else
        $scope\_label \leftarrow$ "$partially\_in\_scope$"
    end if
    return $scope\_label$
end procedure

Hyperparameter	Value
Max_Input_Tokens	8k
LoRA r	16
LoRA α	8
LoRA dropout	0.05
Optimizer	AdamW
LR Weight	1e-5
Decay	0.1
Batch Size	16
Train Epochs	1
GPU	A100-SXM4-80GB

Table 8: Hyperparameters for model fine-tuning and inference

9.3 Entity Search Algorithm for Scope Detection Task

Algorithm 2 showcases the steps of the entity search subcomponent.

Algorithm 2 Search

Input: The search query q and the database id db_id
Output: the list of search results $search\_results$
procedure SEARCH(q, db_id)
    $search\_results \leftarrow [\ ]$
    $db\_schema \leftarrow$ `get_db_schema`(db_id)
    for $table$ in db_schema **do**
        for $column$ in $table$ **do**
            $result \leftarrow$ ""
            $table\_column \leftarrow table +$ "." $+ column$
            $i \leftarrow$ `classify_index`$(table\_column, db\_id)$
            if $i =$ "$Exact$" **then**
                $result \leftarrow$ `exact_search`$(q, table\_column, db\_id)$
            else if $i =$ "$Approximate$" **then**
                $result \leftarrow$ `approximate_search`$(q, table\_column, db\_id)$
            else
                $result \leftarrow$ `semantic_search`$(q, table\_column, db\_id)$
            end if
            if $result \neq$ "" **then**
                $search\_results \leftarrow search\_results$.append$(result)$
            end if
        end for
    end for
    return $search\_results$
end procedure

9.4 Scope Detection Algorithm and Prompts

Algorithm 3 represents the Easy Scope Detection method.

The following prompt was designed for identifying the possible columns in a user question for the Easy Scope Detection approach:

```
### Input:

    Detect the entities which could represent columns in a
database from the given user question.
   You are also given a database schema bellow. The entities
can correspond to columns in the schema below or be outside
the schema. If they correspond to the schema output the name
exactly like in the schema; if they do not correspond to the
schema output a name that would be suitable for that column.

   user question = {user_question}

   schema = {db_schema}

  Provide the answer in the following format: {json_format_columns}
   There can be one or more column names.
   Please ensure that the response is valid json.

   ### Output:
```

The model is also given a json format for returning the results:

```
json_format_columns = { "columns": [
          {
              "column_name": "",
          },
          {
              "column_name": "",
          },
]}
```

The following prompt was designed for identifying the possible named entities in a user question for the Scope Detection with Search approach:

```
### Input:

   Extract the named entities and/or the column, table they
belong to from the given user question.
    You are also given a database schema bellow. The named
entities can correspond to columns in the schema below or be
outside the schema.
```

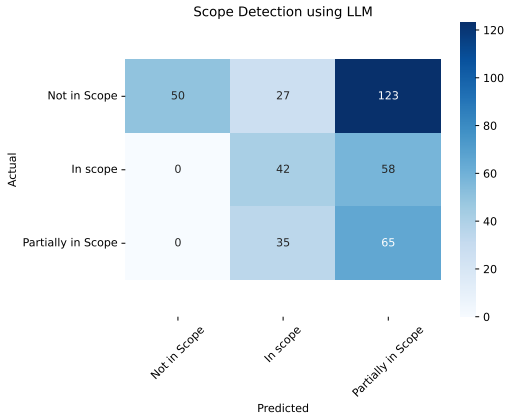

Figure 7: Scope Detection Using LLM

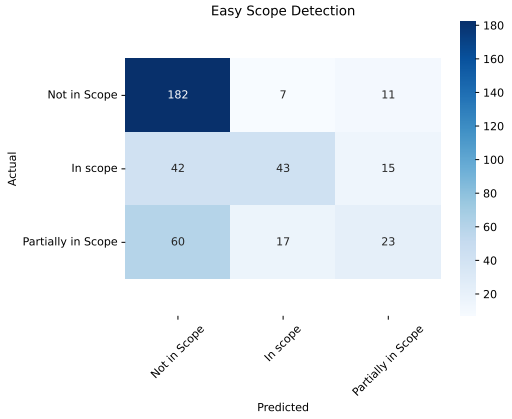

Figure 8: Easy Scope Detection

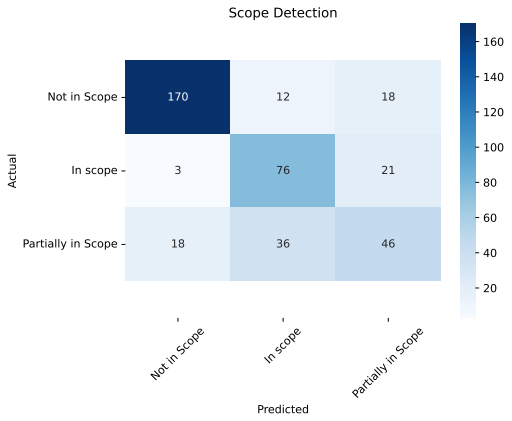

Figure 9: Proposed Scope Detection

```
schema = {db_schema}
```

```
    Provide answer in the following format: {json_format}
    Note that some column names or table names can be empty.
    There can be one or more named entities.
    Each entity is an individual entry in the response:
    - put the entity value  in the "entity_value" field
      - put the corresponding schema column name in the
"column_name" field
   - put the corresponding schema table name in the "table_name"
field

    Please ensure that the response is valid json.

    ### Output:
```

The model is also given a json format for returning the results:

```
    json_format = { "entities": [
                {
                    "entity_value": "",
                    "column_name": "",
                    "table_name": "",
                },
                {
                    "entity_value": "",
                    "column_name": "",
                    "table_name": "",
                },
    ]}
```

Benchmark scope detection prompt for LLM baseline:

```
    ### Input:

    Given the utterance and the database schema below, take
a deep breath and classify whether the utterance can be fully
answered using the database (in scope), partially answered
using the database (partially in scope) or not answered using
the database (not in scope):

    utterance = {utterance}
    db_schema = {db_schema}

    Output:
    - 0 for not in scope,
    - 1 for in scope,
    - 2 for partially in scope

    Output only your choice (0, 1 or 2):
    ###Output:
```

9.5 Paraphrasing Prompts

Here is the prompt designed for the utterance paraphrasing task:

```
### Input:

You are a helpful AI language model who helps me to paraphrase
the given query. The query contains placeholders that start
and end with '\$' and begin with the word 'PLACEHOLDER'. For
example, '\$PLACEHOLDER_S_PRODUCT.PRODUCT_CATEGORY\$',
'\$PLACEHOLDER_CLIENT.CLIENT_NAME\$'. Maintain the original
meaning, while ensuring the values of the placeholders and
their order remain unchanged in the paraphrased version. Keep
the answer short and to the point, no need for elaboration.
Only output the rephrased query, nothing more. DO NOT output
```

```
an answer to the query instead of the paraphrased query.
REMEMBER, placeholders must remain unchanged.

query = {query_with_placeholders}

{previous_responses}

###Output:
```

For multiple paraphrase generations, when the paraphrase is regenerated, the previous unsuccessful paraphrases are appended together with a message instructing the model to avoid repeating the same paraphrases and to focus on diversity.

Both the zero-shot and the few-shots approaches were employed while experimenting with paraphrasing. In the few-shot setting, the diversity of the paraphrases generated by the model was limited as it tended to heavily rely on the patterns observed in the examples. Therefore, the zero-shot approach yielded superior results in this case. However, in future work, we intend to explore different strategies for example selection for this task, including retrieval-augmented prompting. This would require creating a dataset of utterance-paraphrase pairs containing placeholders, specifically crafted for our use case.

For paraphrasing the entities, the model is provided with a few examples using the following prompt:

```
### Input:
You are given an entity name. Can you please give a synonym
for it? Please do not provide entire sentences that explain
what the word means.

I will give you a few examples:

"Healthcare" -> "Medical"
"Food and Beverage" -> "Culinary Industry"
"Transportation and Logistics" -> "Shipping, Transport, Supply
Chain"

entity = {entity}

###Output:
```

9.6 Scope detection.

The confusion matrices for various scope detection methods are provided in Figures 7, 8, and 9.