SUMMARY

This paper tackles a critical limitation in Natural Language to SQL (NL2SQL) systems, their inability to determine when a user's question cannot be answered given the database schema. While modern NL2SQL systems perform impressively on benchmarks, they often fail in real-world environments where databases are large, complex, and contain incomplete information. These failures typically result in models trying to generate SQL queries even when the data to answer the question is missing or misaligned with the user's intent. This produces misleading outputs and erode user trust, particularly in high-stakes domains like healthcare.

To address this, we propose SGAM, a Schema Guided Abstention Mechanism that enables NL2SQL systems to abstain from generating SQL when a query is unanswerable given the database schema. SGAM operates as a pre-generation filter and does not require fine-tuning on each individual database, making it scalable and practical for real-world applications. The system is composed of three main components: a preprocessing step that converts the database schema into a format called M-Schema and chunks it into smaller sub-parts of the schema for improved processing, a schema linking phase that extracts only the schema elements relevant to the user's question and outputs a reduced schema, and a final binary classification phase that predicts whether the question can be answered given the reduced schema.

Unlike previous approaches that rely heavily on task- or schema-specific fine-tuning and large labelled datasets, SGAM is trained using general-purpose datasets like Spider and a modified version of BIRD. This allows SGAM to perform in zero-shot settings and maintain strong generalisability across domains. Experiments show that SGAM outperforms state-of-the-art abstention models across modified BIRD, EHRSQL, and TrialBench, achieving better recall and F2-scores without defaulting to excessive abstention. The schema linker, evaluated independently, also demonstrates strong performance, effectively reducing schema size while retaining most relevant schema elements, which is an essential capability for dealing with large enterprise databases. However, the schema linker does not contribute, when testing the entire pipeline with abstention and SQL generation.

When integrated into a full NL2SQL pipeline, SGAM improves both execution accuracy and overall reliability. Interestingly, a variant of SGAM that omits schema linking performs better in pipeline integration, suggesting that further research on the schema linking needs to be conducted to improve the abstention task. SGAM offers an effective and generalisable method to make NL2SQL systems more reliable by training them to identify and abstain from answering infeasible questions. This marks a meaningful step toward making these systems more reliable for practical use.

This study is conducted in collaboration with Novo Nordisk A/S, who provided data and guidance in the field of clinical trial metadata.

Schema Guided Abstention Mechanism for NL2SQL

Claes Mortensen
cmorte19@student.aau.dk
Department of Computer Science,
Aalborg University
Aalborg, Denmark

Lasse Andersen
lman20@student.aau.dk
Department of Computer Science,
Aalborg University
Aalborg, Denmark

Victor Hansen
vdha20@student.aau.dk
Department of Computer Science,
Aalborg University
Aalborg, Denmark

Abstract

NL2SQL systems democratise access to databases by enabling non-expert users to retrieve information through natural language. Although state-of-the-art NL2SQL models show impressive SQL generation, they typically lack mechanisms to assess question answerability, limiting their practical reliability. Existing approaches that incorporate abstention often require extensive training for each database, which hinders generalisability.

We explore a schema-guided approach to abstention that enables NL2SQL systems to identify and abstain from infeasible questions before SQL generation. Our schema-guided abstention approach uses the pretrained features of existing decoder-only models to capture the connection between user question and schema, and from this information drive the abstention prediction. With our method, we create an abstention mechanism that transfer to unseen domains, reducing the amount of data required for training. Our experiments show that we achieve state-of-the-art performance in the abstention task, while maintaining a better balance between abstention and SQL generation than existing NL2SQL systems. This improves reliability and moves NL2SQL systems closer to real-world usability.

1 Introduction

Natural language to SQL (NL2SQL) systems enable users to query structured databases using natural language, lowering the barrier of entry to information stored in databases. These systems are becoming increasingly important as organisations seek to democratise data access, allowing non-technical users to retrieve insights efficiently [5, 22].

Recent advancements in large language models (LLMs) have significantly improved NL2SQL performance, achieving near-human accuracy on benchmarks such as Spider [29] and BIRD [20]. Despite these advancements, NL2SQL models still exhibit fundamental reliability issues. They are trained to generate SQL queries, but lack the ability to decide whether a query is answerable given a particular database [2, 5]. This behaviour results in two primary failure patterns:

(1) The model fails to correctly link a user's question to the database schema when the schema lacks the necessary information, resulting in incorrect schema mappings or hallucinations of non-existent schema elements [2, 5, 26].
(2) User questions can be underspecified or ambiguous, meaning that the intended relationship to the database schema is unclear. In such cases, the model generates SQL queries based on its interpretation, which may not align with the actual intent of the user [27].

In both cases, a reliable NL2SQL system should be able to detect when a suitable SQL query cannot be generated. Attempting to answer when no valid response exists degrades trustworthiness and usability, particularly for non-expert users who may not recognise incorrect outputs [2]. In this paper, we focus on the first type of failure, when a user's question is unanswerable given the database schema. We investigate how NL2SQL systems can be designed to abstain from generating SQL queries when the necessary information is not available.

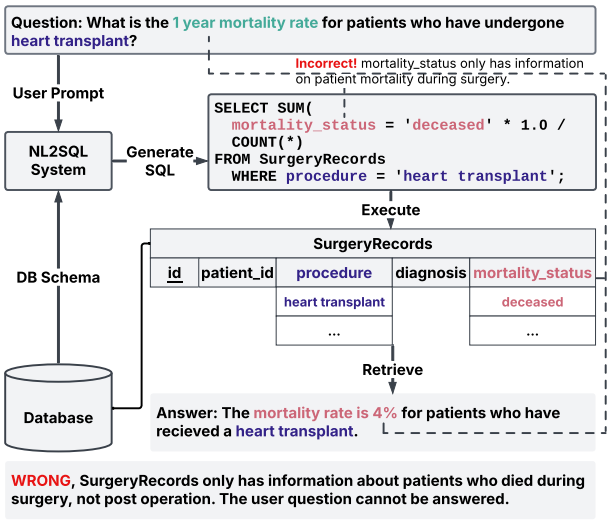

Figure 1: Illustration of an NL2SQL translation failure due to a mismatch between user intent and database semantics

Figure 1 shows an example of this, where a user asks: *"What is the one-year mortality rate for patients who have undergone heart transplant?"* The database only contains immediate mortality data, with no follow-up information, hence this question cannot be answered. Instead of recognising that the table lacks information on time, the system selects the most similar attribute name and generates a query that retrieves immediate mortality rate, misrepresenting them as one-year outcomes. This type of incorrect generation can lead users to make decisions based on false information [17].

Current NL2SQL abstention mechanisms rely on techniques such as fine-tuning [11, 13, 14, 16], schema linking [3], and token probabilities [3, 13, 16]. All three approaches currently assume large labelled training datasets for the databases they are deployed to, which are costly to create, making them impractical in real-world scenarios [30]. Furthermore, current solutions assume that

Table 1: Features of existing NL2SQL systems with abstention

Method	Fine-tuning	Schema Linking	Uncertainty Estimation	Extraction	Generation
TrustSQL_P[16]	Task & Schema				✓
TrustSQL_U[16]	Schema		✓		✓
PLUQ[13]	Schema		✓		✓
PromptMind[11]	Schema				✓
RTS[3]	Schema	✓	✓		✓
SGAM (ours)	Task	✓	✓	✓	

the context window of a model can accommodate an entire database schema, which becomes problematic when dealing with large databases [5]. Even when the schema fits, including irrelevant context has been shown to degrade performance compared to using a more focused schema [21].

An effective abstention mechanism must detect when a database schema cannot answer a query and clearly communicate this limitation [16, 17]. It should function autonomously, without human intervention or frequent retraining as schemas evolve. Additionally, the system must be able to scale across large enterprise databases and function effectively in a zero-shot setting, as fine-tuning for each new database is impractical in real-world applications [10, 28].

In this paper, we propose Schema Guided Abstention Mechanism (SGAM), a method that combines segmented schema linking with a decoder-only classifier to determine when an NL2SQL system should abstain from generating an SQL query. By incorporating segmented schema linking, SGAM scales to large database schemas beyond the scope of existing benchmarks, such as BIRD [20] and Spider [29]. Moreover, by decoupling abstention from the generative process, SGAM can be added to any system as a preprocess to the SQL generation. SGAM represents a significant step towards scalable and robust NL2SQL systems in zero-shot settings, where training data is limited and schema complexity high.

Our work presents the following contributions:

- We propose SGAM, an extractive abstention approach that leverages database schema information to determine whether a question is answerable. We show that this approach provides state-of-the-art performance in abstention for the NL2SQL task.
- We evaluate NL2SQL and abstention systems in a zero-shot setting on difficult real-world databases.

2 Related Work

While numerous solutions have been proposed for NL2SQL, most aim to maximise SQL generation accuracy and assume all input questions are answerable [7, 18, 27]. However, few address the critical issue of abstention. In this section, we review recent approaches to abstention in NL2SQL and identify two key limitations: reliance on large labelled datasets and limited generalisability across databases.

Table 1 summarises current approaches and the techniques they use. We identify two fine-tuning strategies, task-level and schema-specific, as well as four techniques for abstention: schema linking, uncertainty estimation, extraction, and generation.

TrustSQL [16] introduces two approaches for abstention: TrustSQL_P and TrustSQL_U, both based on either GPT-3.5-turbo-0125 or T5-3B [25] for SQL generation, trained on the TrustSQL dataset, which contains 13,958 training examples.

TrustSQL_P employs two separate LLMs: the first is fine-tuned on ~385,000 examples from TriageSQL [31] to classify infeasible questions prior to SQL generation. If a question passes this check, SQL is generated and passed to a second model, fine-tuned on 2,460 examples from both the TrustSQL validation and TriageSQL datasets, to detect and reject erroneous SQL.

TrustSQL_U uses uncertainty estimation directly from the SQL generation model. It calculates maximum entropy or maximum probability over token predictions to determine whether to abstain based on an uncertainty threshold. This approach is fine-tuned to the target schema and a validation set. Across most evaluations, TrustSQL_U outperforms the pipeline method. However, both approaches depend heavily on large labelled datasets, for classifier training, SQL generation, and threshold calibration, limiting their generalisability.

PLUQ [13] integrates fine-tuning and uncertainty estimation into a single approach leveraging their joint advantages. It uses a GPT-3.5-Turbo-0125 model fine-tuned on EHRSQL, containing 5,124 training examples, of which 8% are infeasible [18]. To address this class imbalance, PLUQ performs self-training by pseudo-labelling additional infeasible examples from the unlabelled test set, incorporating them into the training data. This augmentation aims to improve the model's ability to recognise and abstain from answering infeasible questions. In a second stage, PLUQ applies uncertainty estimation using a maximum entropy method to flag and abstain from questions with high uncertainty. The entropy threshold is calibrated based on the estimated proportion of infeasible questions. While this approach enhances PLUQ's robustness to infeasible questions, it introduces a risk of overfitting by incorporating test set data into training. Furthermore, its effectiveness depends on the calibration of the threshold, demanding a pre-known estimate of infeasible queries.

PromptMind [11] is an ensemble method that combines generated outputs from multiple LLMs, all fine-tuned on the EHRSQL dataset, including infeasible questions. PromptMind shows that using a two- or three-model ensemble increases the amount of infeasible questions detected compared to using a single model. There is a trade-off to this improvement, as the ensembles tend to identify more false positives as additional models are added. In critical domains, this might be preferable, but for the method to provide utility, this trade-off needs to be considered to have a functional NL2SQL system. While ensembles help validate outputs, they are also limited as they require large amounts of labelled training data from the target database. Furthermore, ensemble methods increase the computational resources needed to answer each question due to their use of multiple LLMs.

RTS [3] uses schema linking and token-level uncertainty estimation to abstain. RTS fine-tunes a Deepseek-7B model for schema linking on a labelled dataset, derived from the target database, enabling the model to select relevant schema elements and abstain if none exist. When the system finds relevant schema parts, it uses token-level probability estimates during SQL generation to detect schema uncertainty. If it detects uncertainty, it invokes a separate classification model to verify the relevance of the linked schema elements or to abstain from generating SQL. This classification model

utilises a Deepseek-7B fine-tuned on the BIRD [20] and Spider [29] training datasets, to classify if a given schema element is relevant to generate the query. As with other methods based on fine-tuning for a target database, supervised training of the schema linking model is not shown to be generalisable.

Despite growing interest in abstention for NL2SQL, existing strategies consistently depend on domain-specific fine-tuning, large-scale labelled data for the target database, or extensive model ensembles. These constraints hinder their practical deployment across diverse real-world databases. Specifically, current methods require a large collection of domain-specific labelled data for fine-tuning or threshold calibration, and show no evidence for generalisability in zero-shot or low-resource settings [2, 5, 30]. To address these limitations, we propose SGAM, a method that combines schema-linking and extraction utilising fine-tuning for the task of abstention, instead of relying on labelled dataset for the target database. By leveraging existing cross-domain datasets, such as BIRD [20] and Spider [29], SGAM mitigates the need for domain specific training, supporting zero-shot abstention, enabling new-domain NL2SQL deployment.

3 Problem Definition

Given a database D, we denote its schema by S. Each table $t \in S$ has its own set of columns $c \in C(t)$. Each column c is a tuple (l, dt, V), where l is the name of the column, dt the data type and V sample values from D.

Information from D can be retrieved by an SQL query $y \in Y$, where Y denotes the infinitely countable SQL queries that can be created given S and y is the representative of the equivalence class $[y]=\{y' \in Y : y' \sim y\}$ with $\sim$ denoting the equivalence relation meaning that y' and y produce the same results and share semantics. Similarly, we say that each SQL query $y \in Y$ maps to a natural language question $q \in Q$, describing the intent of the specific SQL query, where Q is the set of all possible questions and q is the representative of the equivalence class $[q]=\{q' \in Q : q' \sim q\}$ with $\sim$ denoting the equivalence relation meaning that q and q' have the same intent and request the same information. However, it does not apply that all $q \in Q$ have a corresponding SQL query $y \in Y$ for a given database D, we refer to these as infeasible questions, as they require information that is not contained in S.

NL2SQL is described as a function $NL2SQL : Q \rightarrow \mathcal{P}(Y)$, where $\mathcal{P}(Y)$ denotes the power set of Y. The function $NL2SQL$ maps $q \in Q$ to either the equivalence class $[y]$ for the corresponding SQL query $y \in Y$, that can be executed on D to retrieve the information required by q, or $\emptyset$ if the question is infeasible. The function must be able to determine whether q is feasible or not. In this work, we study the SQL abstention problem as follows:

Problem (Binary Classification for SQL Abstention). Given an NL2SQL system, a database D and a question $q \in Q$, the Binary Classification for SQL Abstention problem requires to design a binary classification function: $f : Q \rightarrow \{\perp, \top\}$, where $f(q)=\top$ if an SQL query y exists such that $NL2SQL(q) \subseteq [y]$, and $f(q)=\perp$ if the question is infeasible, hence $NL2SQL(q)=\emptyset$.

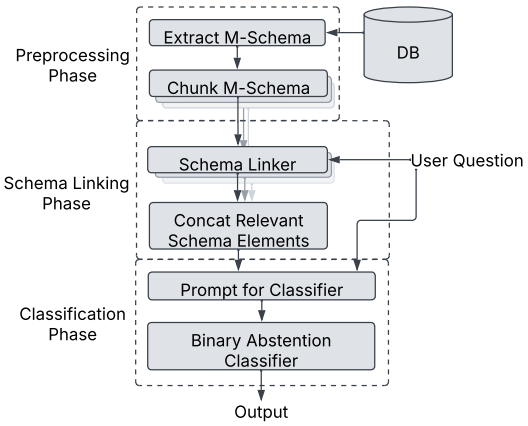

Figure 2: Model architecture overview of SGAM

4 Methodology

In this section, we outline the key components and design of SGAM. SGAM aims to identify unanswerable questions before SQL generation. SGAM utilises pre-generation abstention to decide whether or not to abstain entirely based on the database schema. This allows for seamless integration with existing NL2SQL systems as an abstention module prior to SQL generation.

Figure 2 presents an overview of SGAM, showing the different phases of the approach. In the preprocessing phase, the database schema is converted into the M-Schema [7] format and divided into smaller chunks. During the schema linking phase, SGAM evaluates the relevance of tables and columns to the user's question for each chunk, filtering out unrelated schema elements. The remaining subsets are then concatenated into a reduced schema. The last phase is the binary abstention classifier, which leverages the reduced schema, provided by the schema linker, to decide when a users question is unanswerable by SQL.

4.1 SGAM Input Data

SGAM's input is a tuple (q, D), where q is a natural language question, and D the input database. SGAM adopts the M-Schema representation for database schemas [7]. This representation extends the traditional DDL schema representation by incorporating a set of example values for each column, together with explicitly listing all foreign key relations. Figure 3 illustrates the difference between a given database schema in both DDL and M-Schema format. M-Schema provides additional semantic grounding beyond DDL, addressing the issue that LLMs misinterpret semantically similar column and table names when structural context is insufficient [2]. By incorporating content-level cues along with structural metadata, M-Schema reduces schema ambiguity and improves schema linking performance [7].

4.2 Preprocessing

As a preprocessing step, we propose schema chunking, to split the database schema into smaller chunks containing sub-parts of the schema. We choose this approach as recent studies show that

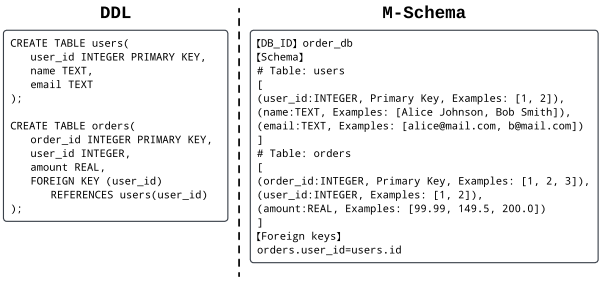

Figure 3: Example of a database schema presented in both M-Schema and DDL format, illustrating the difference

exploiting LLMs large context windows to provide more information does not necessarily result in better performance, and can in some cases have a negative impact [5, 21]. This is relevant as many databases in the real world are larger than what is seen in current state-of-the-art benchmarks [5] This also mitigates any limitations with LLMs context windows being too small to fit the entire database schema. Chunking takes the database D as input, extracts its schema and divides the schema into chunks such that each chunk $S_i \subseteq S$, and $\bigcup_{i=1}^{n} S_i = S$. The schema chunking procedure, shown in Algorithm 1, groups entire tables together and includes related tables using foreign keys for context. Schema chunking is considered a preprocessing step, as schema chunking only needs to be rerun when the database schema is updated.

Algorithm 1: Schema Chunking

Input : database D, max chunks k, context size C
Output: List of schema chunks $S_{\text{chunked}} = \{S_1, \ldots, S_n\}$ such that $S_i \subseteq S$ for all i

$S \leftarrow$ `extractSchema`(D);
$S_{\text{chunked}} \leftarrow [\,]$;
$S_i \leftarrow \emptyset$; `// Initialise empty set`
foreach $t \in S$ **do**
  $t_{\text{size}} \leftarrow$ `computeTokenSize`(S_i) ;
  if $(k > 0 \wedge |S_{chunked}| >= k) \vee (t_{size} > \frac{C}{2})$ **then**
    append S_i to S_{chunked};
    $S_i \leftarrow \emptyset$; `// Initialise empty set`
    $S_i \leftarrow S_i \cup T \cup$ `getRelatedTables`(t, S);
  else
    $S_i \leftarrow S_i \cup T \cup$ `getRelatedTables`(t, S);
if $S_i \neq \emptyset$ **then**
  append S_i to S_{chunked};
return `convertChunksToSchemaStrings`(S_{chunked});

4.3 Schema Linking

In the schema linking phase, we use a decoder-only model to identify the schema elements relevant to answer a user's question. This reduces the size of the schema and narrows the search space. As a result, SQL generation is improved [5, 9, 21].

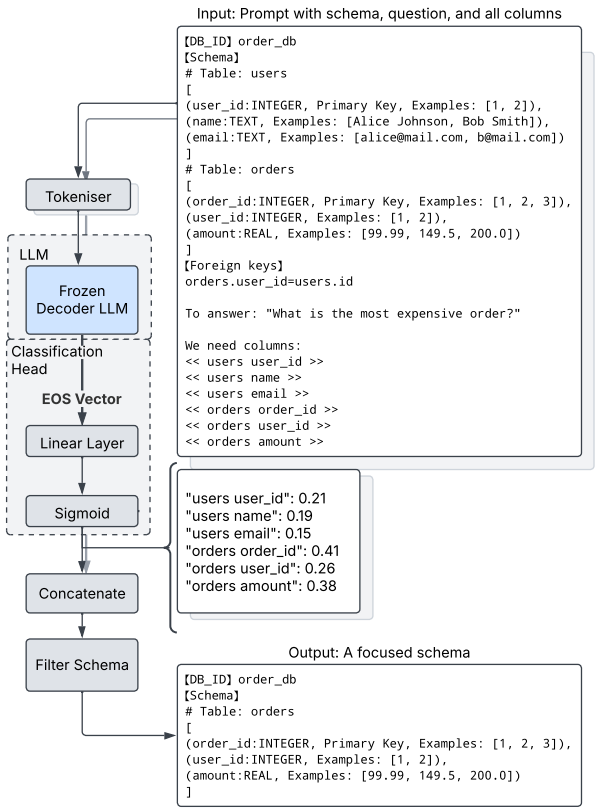

Figure 4: The schema linker takes the chunked M-Schema as an input and predicts the relevance of each column. The relevant columns are concatenated into a reduced M-Schema. The blue block in the decoder-only model visualise that it is frozen, such that we only train the classification head added on top of the model

We implement schema linking using an extractive approach [9]. Unlike generative methods, which are computationally expensive and prone to hallucinating non-existent schema elements, the extractive method is lightweight and directly tied to the provided input.

We implement schema linking by augmenting a decoder-only model with a lightweight classification head, which is implemented using a linear layer trained on the Spider dataset [29]. This layer replaces the model's standard language modelling head and operates over the final hidden states of schema tokens, capturing interactions between the question and schema. This approach is selected as we can then use decoder-only models trained specifically for NL2SQL and only adapt their outputs for the schema linking task, reducing the cost of training and the amount of data required for training [9].

The process of schema linking can be seen in Figure 4, the input to the schema linker is a tuple $(S_i, q, C(S_i))$, where S_i is a schema chunk, q a natural language question, and $C(S_i)$ is the set of columns

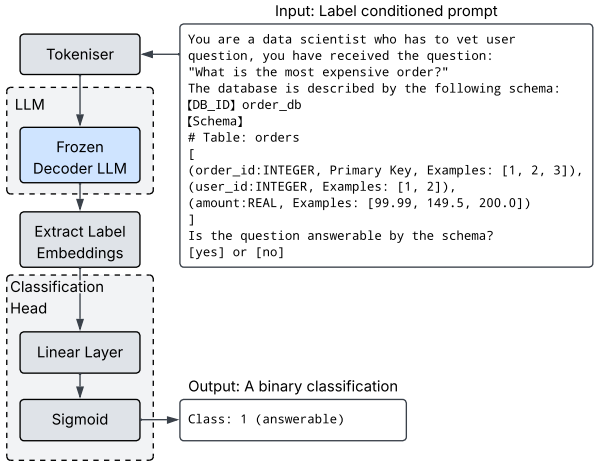

Figure 5: The abstention mechanism receives the reduced schema from the schema linker and a user question, and classifies whether the question is answerable. The blue block indicates that the decoder-only model is frozen, and only the classifier head is trained

in that chunk. These elements are concatenated into a single input sequence for the model.

The schema linker computes a relevance score for each column such that, for each column $c \in C(S_i)$, the linear layer predicts the probability $p_{rel}(c|q)$ that column c is relevant to the given natural language question q. This is repeated across all schema chunks, allowing the model to assess relevance for the full schema. After scoring, we apply a relevance threshold to filter columns. The selected elements across all chunks are then merged into a reduced schema $S_{reduced}$, which is the output of the schema linking phase.

4.4 Extractive Abstention Classifier

In the final phase of SGAM, a binary classifier determines whether a user question can be answered given the reduced schema. This component addresses the limitation that traditional NL2SQL systems will produce SQL output, even when no valid SQL query exists [2, 3, 16].

The abstention classifier takes as input the tuple $(q, S_{reduced})$, where q is the natural language question and $S_{reduced}$ is the subset of schema elements identified as relevant by the schema linker. Figure 5 shows how these elements are composed into a natural language prompt that primes the model to assess whether the user questions is answerable given the reduced schema.

The classifier is implemented using a decoder-only model, where the original generation head is replaced with a lightweight classification head. Rather than generating output, we extract hidden states corresponding to the label tokens "yes" and "no" from the final decoder layer. The embeddings of "yes" and "no" tokens are concatenated along with their difference vector, to form a representation of the prompt. This representation is passed to the classification head, which first uses a linear layer to produce a scalar logit, then converts it into a probability over "yes" and "no" via a sigmoid function.

As in the schema linking phase, we freeze the base decoder-only model's parameters and train only the classification head. This approach preserves the model's semantic understanding of instructions and natural language, while enabling efficient adaptation to the binary classification task.

We use a decoder-only model for its increased context size, and ability to follow instructions [6]. Works has shown that decoder models can be effectively adapted to classification via label conditioning [23].

A dedicated abstention classifier is necessary because neither schema linking nor generation confidence provides a reliable signal of feasibility [3, 16]. Schema linkers may incorrectly identify irrelevant or spurious schema elements as relevant, and generation models often exert high confidence in all its answers, even on unanswerable queries [13, 16]. By framing abstention as a separate classification task, we aim to enhance model reliability on infeasible questions, ensuring that NL2SQL systems are more reliable for real-world applications.

5 Experimental Design

We describe the datasets, training procedures, and methods used to evaluate our approach. We seek to answer the following research questions (RQ):

- RQ1 Is an extractive abstention approach a viable solution for classification of unanswerable user questions in NL2SQL systems?
- RQ2 Is the extractive abstention approach more accurate in classifying unanswerable user questions than state-of-the-art systems?
- RQ3 Is extractive schema linking accurate at identifying relevant schema elements on databases with varying sizes compared to state-of-the-art schema linkers?
- RQ4 What training hyperparameters optimise the accuracy of the schema-guided abstention mechanism?

5.1 Datasets

To comprehensively evaluate our approach to abstention in NL2SQL systems, we use four datasets that reflect different real-world challenges. As seen in Table 2, the datasets are both widely used public benchmark datasets and custom datasets specifically designed to test capabilities such as handling infeasibility and generalising to unseen domains.

We use the Spider dataset [29] to train and evaluate schema linking under standard conditions, as this has been the go-to for other schema linkers [24]. We introduce a modified BIRD dataset [20], which allows training and evaluation on a large dataset, that spans multiple domains. To evaluate abstention in sensitive or high-risk domains, we use EHRSQL [17], which simulates queries on medical records with 20% infeasible questions in the test set. Finally, to evaluate the abstention mechanism's ability to handle less structured databases, while in an unseen domain, we utilise TrialBench [2].

Spider is a widely used benchmark dataset for NL2SQL tasks, consisting of 7,000 training examples, 1,034 validation examples, and 2,148 test examples across 200 databases from 138 different

Table 2: Datasets used for experiments. Infeasibility refers to the percentage of questions that are infeasible in the split. * *not publicly available*

Dataset	Examples	Split (%)	Domain	Infeasibility (%)
Spider	10,182	68/10/22	cross-domain	0/0/0
BIRD_{mod}*	10,821	77/9/14	cross-domain	45/45/25
EHRSQL	7,454	68/15/16	Health	9/20/20
TrialBench*	206	-/-/100	Clinical Trials	-/-/13.6

domains [29]. We use Spider exclusively to train and evaluate our schema linker to ensure direct comparability with prior work [12, 24]. As most recent schema linking systems report performance on Spider, it also serves as the benchmarking for our schema linking model.

Modified BIRD is a dataset based on BIRD that serves for fine-tuning, validation, and testing. The dataset consists of a total of 10,821 examples, of which 8,379 are used in training, 931 for validation, and 1,511 are reserved for testing. The dataset is created by modifying BIRD to include questions that are unanswerable. BIRD is modified by removing columns from the target databases, and marking the questions whose resulting SQL utilises any of these removed columns as infeasible. The training and validation datasets have 45% infeasible questions. This split was chosen as it results in an almost balanced dataset, while not removing so many columns that questions became trivial. This is a concern, as we cannot remove columns that are foreign or primary keys. The test dataset consists of 25% infeasible question, to vary from the split seen in training and validation. We choose to use a modified version of BIRD for fine-tuning both SGAM and TrustSQL, as one of our key objectives is to demonstrate generalisability and zero-shot performance. BIRD, known for its diverse databases spanning 37 different domains, aligns well with this goal. Its domain diversity provide a robust foundation for assessing how well models can generalise to unseen data and perform without explicit task-specific training.

EHRSQL is based on electronic health records and serves as a benchmark for abstention systems [16–18]. We use EHRSQL's test set, consisting of 3,600 examples, of which 30% are infeasible. The dataset targets the healthcare domain, where abstention is important due to the high cost of incorrect responses. We use EHRSQL to evaluate the models' ability to transfer abstention to a new domain.

TrialBench is a benchmark dataset derived from clinical trial metadata provided by Novo Nordisk [2]. It comprises 206 challenging samples, of which 28 are classified as infeasible. Despite its small size, TrialBench is valuable for stress-testing models in real-world scenarios with sparse data, large databases, and irregular schemas. Its database features 91 tables and 998 columns, far more than BIRD or EHRSQL, and lacks explicit relations while using a highly specialised vocabulary. Using TrialBench, we assess the generalisation capabilities of SGAM, particularly its robustness to out-of-domain vocabulary, novel domains, and complex database schemas.

5.2 Training

In this section, we describe the details of how we train the schema linking model and the extractive abstention classifier.

5.2.1 Schema Linking. Our extractive schema linking model is trained using column-level annotations derived from the Spider dataset. For each question-query pair, columns appearing in the gold query are labelled as positives, and all others as negatives. We use OmniSQL-7B [19] as the base model and freeze its parameters, as this model is already trained for NL2SQL. We train using binary cross entropy loss against the extracted relevance labels, using an AdamW optimiser with a learning rate of $4.98e{-}5$ and weight decay of 0.0004 for two epochs yields the best results on the validation set. We use a filtering threshold of 0.15. With exception of experiment 4, we use the above described hyperparameters across all experiments.

5.2.2 Abstention Mechanism. The classification head for the abstention mechanism is trained using the modified BIRD dataset. We adopt XiYanSQL-7b [7] because of its strong SQL reasoning abilities. During training, we freeze its parameters and train only the classification head for abstention. The classification head is trained using binary cross entropy loss, supervised by the feasibility labels from the modified BIRD dataset. With the exception of experiment 4, we train using an AdamW optimiser with a learning rate of $7.25e-4$, weight decay of $6.99e-2$, and two epochs. We use a threshold of 0.70 for the abstention classifier.

5.3 Evaluation Methods and Metrics

In this section, we provide details on the design of our experiments, outlining the purpose, metrics, and baselines to compare against.

5.3.1 Experiment 1 - Pipeline w/ NL2SQL. This experiment seeks to answer research question 1. We evaluate how combining SGAM with existing NL2SQL systems affects the performance of the entire pipeline. We consider the baseline LLM XiYanSQL-32B [7], in addition to $\text{TrustSQL}_{\text{U}}$ [16]. For SGAM configurations, we consider:

- SGAM + XiYanSQL: SGAM as preprocessing step for XiYanSQL.
- SGAM_{NSL} + XiYanSQL: SGAM_{NSL} as preprocessing step for XiYanSQL.

To evaluate the utility of the entire pipeline, we consider the metrics execution accuracy (EX), precision, recall, and F1-score, and evaluate on EHRSQL and TrialBench. We define EX as:

$$EX = \begin{cases} 1, & \text{if } q_i \in Q \land f(q_i) = \top \land \mathbb{1}(V_i, \hat{V}_i) = 1, \\ 0, & \text{if } q_i \in Q \land f(q_i) = \bot, \\ 0, & \text{if } q_i \in Q \land f(q_i) = \top \land \mathbb{1}(V_i, \hat{V}_i) = 0, \\ 0, & \text{if } q_i \in \overline{Q} \land f(q_i) = \top, \\ 1, & \text{if } q_i \in \overline{Q} \land f(q_i) = \bot, \end{cases}$$

$$\mathbb{1}(V, \hat{V}) = \begin{cases} 1, & \text{if } V = \hat{V} \\ 0, & \text{if } V \neq \hat{V} \end{cases}$$

where V_i is the result of executing the goal query, $\hat{V}_i$ is the result from executing the generated SQL, Q the set of feasible questions and $\overline{Q}$ the set of infeasible questions.

5.3.2 Experiment 2 - Abstention Mechanism. This experiment seeks to answer research question 2. It serves as the primary evaluation of our abstentions mechanism's effectiveness in determining the feasibility of questions across different settings. We assess whether

SGAM can accurately identify infeasible queries, when the underlying database schema does not support a correct SQL translation.

We compare against TrustSQL_P and TrustSQL_U [16], which represent current state-of-the-art for abstention mechanisms in NL2SQL, details of our implementation can be found in subsection 8.3.

We conduct this evaluation on three datasets: Modified BIRD, EHRSQL and TrialBench. We evaluate using precision, recall, and F2-score on the abstention decision, focussing on the model's ability to identify infeasible questions. We consider SGAM and SGAM without schema linking (SGAM_{NSL}).

5.3.3 Experiment 3 - Schema Linking. With this experiment, we aim to answer research question 3. We evaluate the effectiveness of our extractive schema linker, in identifying relevant tables within a database schema. As a baseline, we include DTS-SQL [24], which represents the current state-of-the-art in schema linkers and employs a generative strategy for schema linking. We consider the precision, recall, and relative reduction on the size of the filtered schema compared to the full schema, on both table- and column-level. An effective schema linker should maximise recall, while also minimising the relative schema size, reducing irrelevant schema elements for downstream components.

We conduct our evaluation on Spider and TrialBench. These two datasets allow us to test the schema linker under varying database sizes.

We assess the extractive schema linker under three different input chunking strategies:

- No chunking (ExSL): the entire database schema is processed at once.
- Naive chunking (ExSL_n): tables are packed into chunks until the model's context window is full, attempting to maximise schema context per chunk.
- Extreme chunking (ExSL_e): each table is evaluated independently.

5.3.4 Experiment 4 - Hyperparameters. In this experiment, we aim to answer research question 4. We consider two components, the extractive schema linker and the abstention classifier. For each, we perform a hyperparameter search using Bayesian optimisation [1], treating each configuration as a study evaluated on the Spider and BIRD validation set respectively. The objectives differ by component:

- For the schema linker we aim to maximise precision, while keeping recall at 1.000, measuring recall efficiency.
- The abstention classifier aims to maximise the F2-score, prioritising high recall in detecting infeasible questions.

The F2-score is a variant of the F-measure that weighs recall twice as much as precision. This emphasis on recall is intentional, because it prioritises reducing false negatives, minimising the chance of answering infeasible questions. This is crucial in domains where incorrect answers can have serious repercussions, such as healthcare [16].

The search space for each objective is detailed in Table 3. After searching for the optimal training hyperparameters, we perform a second search for the optimal threshold for each component using a grid search.

Table 3: Hyperparameter search space for training the classification heads of the schema linker and abstention mechanism

Objective	Learning Rate	Weight Decay	Epochs
Schema Linking	$(5e-5)$ - $(5e-7)$	0.10 - 0.01	2 - 4
Abstention Mechanism	$(5e-3)$ - $(5e-6)$	0.10 - 0.00	2 - 5

For the schema linker, the threshold search optimises a balance between recall, precision, and relative schema reduction, while for the abstention mechanism, the search targets maximising the F2-score once again. We calculate the relative schema reduction as $\frac{|S|-|S_{reduced}|}{|S|}$, where $|S|$ denotes the amount of columns in the full schema and $|S_{reduced}|$ the amount of columns in the reduced schema.

6 Results

In this section, we evaluate SGAM to answer our RQs.

6.1 RQ1 - Pipeline w/ NL2SQL

Table 4 presents the EX, precision, recall, and F1-score for various abstention mechanisms integrated into NL2SQL pipelines, evaluated on the EHRSQL and TrialBench datasets. Comparing the baselines, we observe contrasting behaviours. Trust_U emphasises caution, abstaining from SQL generation on infeasible questions, at the cost of abstention on all feasible questions too. In contrast, XiYanSQL is overly permissive, generating SQL for all questions, including the infeasible ones. The combination of SGAM_NSL with XiYanSQL consistently achieves the best performance across all metrics, outperforming both the baseline Trust_U and XiYanSQL alone.

On EHRSQL, SGAM_NSL + XiYanSQL achieves an EX of 0.31 and F1-score of 0.32, compared to XiYanSQL alone with an EX of 0.21 and F1-score of 0.20. On TrialBench, it reaches an EX of 0.36 and F1-score of 0.52, substantially improving over XiYanSQL with an EX of 0.26 and F1-score of 0.42. This indicates that inserting SGAM_NSL prior to an NL2SQL system improves end-to-end performance, leveraging its strength in abstention while preserving the SQL generation capabilities of the underlying NL2SQL model. Interestingly, SGAM_NSL, which does not use schema linking, outperforms SGAM, which includes schema linking. The weaker performance of SGAM suggests that the current implementation of schema linking does not effectively support abstention decisions.

RQ1: The results show that extractive abstention is a more viable solution for classifying unanswerable user questions in NL2SQL systems. Integrating SGAM_NSL with an existing NL2SQL model improves performance across abstention benchmarks, achieving a better balance between abstaining on infeasible inputs and generating SQL for feasible ones.

6.2 RQ2 - Abstention Mechanism

The objective of SGAM is to abstain from answering user questions that cannot be reliably grounded in the schema of the target database.

Table 5 compares SGAM, SGAM_NSL, and TrustSQL baselines across three datasets using precision, recall, and F2-score.

Table 4: EX, Precision (P), recall (R), and F1-score (F1) of different abstention mechanisms on EHRSQL and TrialBench

	EHRSQL				TrialBench			
NL2SQL Pipeline	**EX**	**P**	**R**	**F1**	**EX**	**P**	**R**	**F1**
Trust_{U}	0.20	0.20	0.20	0.20	0.14	0.14	0.14	0.14
XiYanSQL	0.21	0.19	0.20	0.20	0.26	0.45	0.39	0.42
SGAM_{NSL} + XiYanSQL	**0.31**	**0.31**	**0.32**	**0.32**	**0.36**	**0.55**	**0.49**	**0.52**
SGAM + XiYanSQL	0.26	0.26	0.26	0.26	0.31	0.48	0.42	0.45

Table 5: Precision (P), recall (R), and F2-score (F2) of different abstention mechanisms on modified BIRD, EHRSQL, and TrialBench

Abstention	BIRD			EHRSQL			TrialBench		
Mechanism	**P**	**R**	**F2**	**P**	**R**	**F2**	**P**	**R**	**F2**
Trust_{P}	**0.406**	0.034	0.042	**0.333**	0.009	0.011	-	-	-
Trust_{U}	0.253	**1.000**	**0.628**	0.199	**1.000**	0.555	0.136	**1.000**	0.440
SGAM	0.270	0.859	0.598	0.229	0.979	0.591	0.378	0.892	0.702
SGAM_{NSL}	0.268	0.827	0.584	0.283	0.927	**0.637**	**0.615**	0.857	**0.795**

$\text{TrustSQL}_{\text{U}}$ achieves perfect recall at 1.000 by abstaining on every question, yielding the highest F2-scores on modified BIRD and EHRSQL. However, this blanket abstention is impractical, as it treats all queries as unanswerable and its precision merely reflects the fraction of infeasible questions. Real-world systems must instead discriminate between answerable and unanswerable queries.

Among the models that do not default to abstention, SGAM and SGAM_{NSL} outperform both TrustSQL variants. On modified BIRD, $\text{TrustSQL}_{\text{U}}$ attains the highest precision of 0.406 but with minimal recall at 0.034, limiting practical utility. Conversely, SGAM and SGAM_{NSL} bias towards detecting unanswerable questions. SGAM achieves much higher recall at 0.859 with a precision of 0.270, slightly above the infeasible-question baseline. This results in an F2-score of 0.598 for SGAM, compared to 0.042 for $\text{TrustSQL}_{\text{P}}$.

On EHRSQL, SGAM_{NSL} leads with an F2-score of 0.637, combining a precision of 0.283 with recall at 0.927. The similar performance of SGAM and SGAM_{NSL} suggests that schema linking does not significantly affect abstention effectiveness on this dataset.

TrialBench results reinforce this pattern. SGAM_{NSL} achieves the highest precision of 0.615 and F2-score at 0.795, outperforming $\text{TrustSQL}_{\text{U}}$ despite its perfect recall. This underscores SGAM_{NSL}'s capacity to handle complex schemas seen in real-world scenarios.

Overall, extractive decoder-based architectures like SGAM provide more balanced and practical abstention behaviour than purely generative or overly conservative baselines. While perfect recall is trivial through over-abstention, the key challenge lies in maintaining high precision without sacrificing recall. SGAM and SGAM_{NSL} demonstrate progress toward this goal.

We observe a notable gap between validation and test performance. SGAM attains an F2-score of 0.864 on the modified BIRD validation set, but generalises less effectively on the test set, indicating possible overfitting or dataset shift. Additionally, this could be the result of the train set containing a higher frequency of infeasible questions, resulting in the model overemphasising abstention.

To explore SGAM's failure modes, we analyse 50 stratified random samples from both SGAM and SGAM_{NSL} across the test sets. We identify recurring patterns in misclassification and assess whether these errors stem from architectural limitations, dataset artefacts, or represent broader challenges in the abstention task.

A significant proportion of errors occurs in the questions that require reasoning across multiple tables. For instance, the model frequently predicts questions to be infeasible when answering requires joining more than two tables. This suggests that SGAM struggles to perform compositional reasoning over relational structures, a capability that is necessary to differentiate between infeasible queries and those that are complex.

Errors are also common in cases where questions include indirect references to schema content, which require interpretive inference. For example, in a modified BIRD instance asking "Which sets have Italian translations?", the schema lacks an explicit "translations" column, but relevant information is encoded in the "name" column. SGAM often abstains in these cases, failing to resolve implicit semantic mappings to the schema.

A minority of failure cases on modified BIRD are attributable to mislabelled infeasible examples. Because infeasibility is induced by removing columns referenced in the original SQL, questions referencing those columns are automatically labelled as unanswerable, even when semantically equivalent alternatives exist. These edge cases may introduce noise into the training signal, causing the model to overfit to superficial patterns rather than true question discrimination based on the database schema.

This behaviour mirrors findings in prior work on NL2SQL systems, where models perform well in-domain but degrade on unseen schemas [14, 16, 18]. SGAM is designed to generalise without domain-specific retraining, but its current reliance on surface-level features is insufficient to meet this objective.

Improving reliability likely requires a combination of better training data, particularly with more diverse and realistic infeasible cases, and architectural enhancements that support structured reasoning.

RQ2: The extractive abstention models SGAM and SGAM_{NSL} outperform state-of-the-art baselines in accurately classifying unanswerable user questions. While challenges remain in compositional reasoning and handling implicit schema references, these results confirm that extractive abstention is a more accurate and balanced solution than existing methods.

6.3 RQ3 - Schema Linking

We present schema linking results on the Spider and TrialBench datasets, evaluating both table-level and column-level filtering, as seen in Table 6.

On the Spider test set, DTS-SQL achieves state-of-the-art performance in table-level filtering, showing the best schema reduction of 0.570 and precision of 0.979, while maintaining a high recall at 0.969. This indicates that DTS-SQL is effective at selecting a compact yet relevant subset of tables for Spider's relatively moderate schema size.

Among the methods that support column-level filtering, ExSL_{n} achieves the highest precision at the column-level of 0.425 alongside a schema reduction of 0.593. This reflects its ability to accurately isolate relevant columns while reducing the schema size. Notably, ExSL_{e} yields the highest recall at both levels, with a recall of 0.994

Table 6: precision (P), recall (R), and Relative schema size reduction (Red) for table- and column-level filtering on the Spider and TrialBench test sets

Method	Table			Column		
	P	R	Red	P	R	Red
Spider						
DTS-SQL	**0.979**	0.969	**0.570**	–	–	–
ExSL	0.580	0.946	0.297	0.405	0.892	0.586
$ExSL_n$	0.564	0.945	0.314	**0.425**	0.890	**0.593**
$ExSL_e$	0.458	**0.994**	0.202	0.407	**0.945**	0.555
TrialBench						
DTS-SQL	0	0	0	–	–	–
ExSL	0.021	0.284	**0.814**	0.005	0.162	0.950
$ExSL_n$	**0.025**	0.236	0.811	**0.007**	0.153	**0.964**
$ExSL_e$	0.011	**1.000**	0.017	0.003	**0.982**	0.414

Table 7: Approximation of optimal training hyperparameters found during Bayesian search for the schema linker and abstention mechanism

Objective	Learning Rate	Weight Decay	Epochs
Schema Linker	4.98e-5	4e-4	2
Abstention Mechanism	7.25e-4	6.99e-2	2 - 5

at table-level and 0.945 at column-level, though this comes at the cost of lower precision, indicating over-inclusion.

On TrialBench, which involves a much larger and more complex schema compared to Spider, we observe clear challenges for schema linking methods. DTS-SQL fails to return results, highlighting that it does not scale to real-world databases with large schemas.

In contrast, $ExSL_e$ includes nearly the entire schema with a table-level schema reduction of just 0.017 and recall of 1, ensuring that no relevant tables are omitted. However, this comes at the cost of extremely low precision at 0.011, offering limited filtering. This result confirms that table-level filtering alone becomes ineffective in large-scale settings like TrialBench, where including all or most tables is often necessary to preserve coverage.

More importantly, column-level filtering shows better adaptability to large schemas. Although precision remains low across methods, $ExSL_e$ achieves a column-level recall of 0.982 while still significantly reducing the schema size by 0.414. This is critical in NL2SQL system as precision and reduction are less informative if relevant schema elements are excluded. Therefore, maintaining high recall alongside significant schema reduction is the key objective. $ExSL_n$ and ExSL achieve similar reductions of 0.964 and 0.950 respectively, which comes at the cost of limited recall, indicating that most relevant schema elements have been removed.

RQ3: Extractive schema linking achieves substantial schema size reduction while maintaining high recall, especially at the column-level, even on large databases. However, this comes with a trade-off in precision, as extractive schema linking tends to overinclude irrelevant schema elements to preserve recall, whereas state-of-the-art methods achieve higher precision on smaller databases.

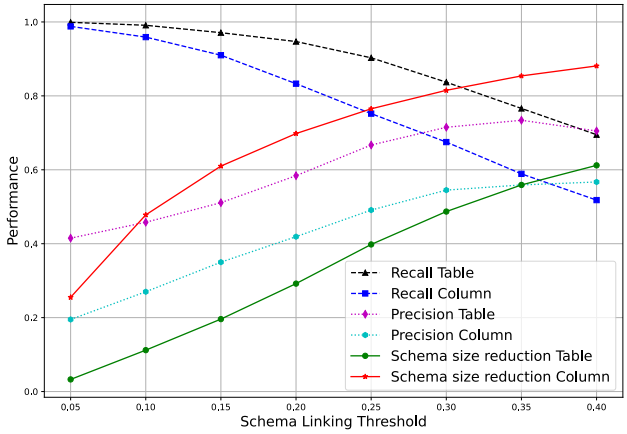

Figure 6: Shows the behaviour of recall, precision, and schema size reduction at different schema linking filtering thresholds. The metrics have been included for filtering at both table- and column-level

6.4 RQ4 - Hyperparameter Search

Table 7 summarises the optimal hyperparameters identified for the schema linker and abstention classifier using Bayesian optimisation. These configurations are used for final training in all other experiments.

Figure 6 shows threshold behaviour for the schema linker. Increasing the threshold improves precision and schema reduction by pruning more aggressively but reduces recall. A threshold of 0.15 provides a strong trade-off, column-level recall remains high at 0.910, with a precision of 0.350 and a schema size reduction of 0.610. At table-level, recall remains near perfect at 0.971, with a modest reduction of 0.196 and a precision of 0.511.

By increasing the threshold beyond 0.15, recall drops sharply, particularly at columns-level, indicating diminishing returns from further threshold increases. These results highlight the schema linker's sensitivity to threshold tuning, especially when precision must be improved without sacrificing recall.

For the abstention classifier, Figure 7 shows that performance improves significantly around a threshold of 0.7. This threshold yields the highest F2-score of 0.84, balancing recall of 0.89 and precision of 0.84. Below 0.4, the model fails to predict any infeasible questions, while higher thresholds cause performance to decline, reflecting a narrow confidence distribution.

This sensitivity suggests that small changes in thresholding can meaningfully alter abstention behaviour, showing that careful tuning is essential for the abstention mechanism.

RQ4: Performance for both SGAM components varies substantially with hyperparameter and threshold choice. Using Bayesian and grid search, we identify strong configurations for the validation sets of Spider and BIRD. However, the narrow range of effective thresholds indicates that transferring these values to unseen distributions may require recalibration.

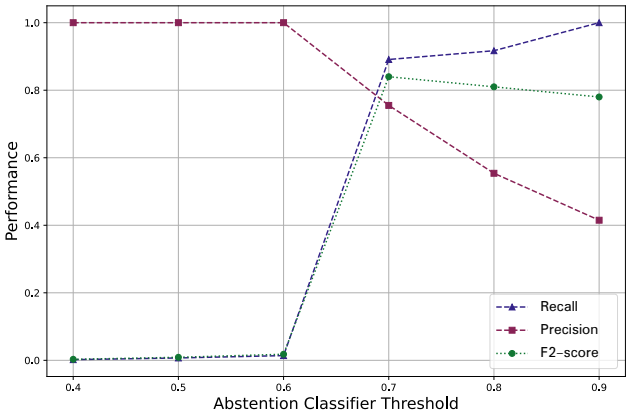

Figure 7: Recall, precision, and F2-score at different abstention classification thresholds. The thresholds starts at 0.4, as any threshold below resulted in no infeasible predictions

7 Conclusion

In this work, we introduce SGAM, a schema-guided abstention mechanism for NL2SQL systems, which identifies unanswerable questions before SQL generation. Unlike prior abstention approaches, SGAM decouples feasibility detection from SQL generation and avoids schema-specific fine-tuning, enabling more reliable performance in zero-shot settings.

SGAM shows state-of-the-art performance on abstention benchmarks, especially in realistic and high-risk settings such as Trial-Bench. SGAM without schema linking, consistently outperforms both baseline and SGAM in end-to-end pipeline evaluations, suggesting that current schema linking strategies do not yet support the abstention task.

Our analysis further reveals that failures often stem from limited reasoning and understanding of questions, highlighting the challenge of abstract schema-level understanding. We observe signs of overfitting as performance discrepancies emerged between validation and test splits.

In general, our results show that SGAM represents a practical step toward more reliable and deployable NL2SQL systems.

8 Future Work

In this section, we suggest directions for further development of SGAM. This includes improving the precision of abstention and expanding on how abstention is handled.

8.1 Improving Precision of Abstention Prediction

Recent work suggests that LLMs struggle with abstract reasoning due to limited generalisation beyond context-bound patterns [8]. This limitation impacts tasks such as abstention prediction in NL2SQL, where models must infer the feasibility of a question with minimal surface cues.

To improve abstention precision, SGAM could explore the integration of causal reasoning and program induction. Causal reasoning would enable models to identify the underlying factors that make a query infeasible, rather than relying on shallow heuristics. [8, 15] Program induction could support the construction of interpretable logic-based decision paths, improving consistency and generalisation across diverse questions [8, 32].

By moving beyond pattern recognition toward structured, abstract reasoning, these methods hold promise for reducing false positives and improving the precision of abstention decisions in NL2SQL systems. Additionally, they may provide a basis for generating informative justifications for abstentions, improving transparency and user trust.

8.2 Exposing Feasibility Score

SGAM treats abstention as a binary decision without offering insight into the reasoning behind that decision.

A straightforward approach would be to expose the feasibility score predicted by the classifier prior to filtering. This score could indicate whether a question was clearly classified as infeasible or abstention resulted from ambiguity or uncertainty. Presenting this information could help users refine their questions to better align with the database schema.

Exposing the feasibility score may also be beneficial for queries that proceed to SQL generation, as we observe that many feasible questions still result in incorrect SQL. In such cases, the feasibility score might serve as a proxy for confidence in the generated SQL, potentially flagging outputs for user verification or downstream filtering.

However, this approach raises several open questions. It remains to be seen if a meaningful correlation exists between feasibility scores and SQL generation quality, warranting further research. Additionally, the utility of a numeric score for end users is unclear, and would require user studies to assess its interpretability and effectiveness in practice.

Acknowledgement

We thank Novo Nordisk, and in particular Henning Pontoppidan Föh and Rasmus Stenholt, for their invaluable assistance with data provision, question generation, and support throughout this project. We also extend our gratitude to our advisors Daniele Dell'Aglio, Juan Manuel Rodriguez and Matteo Lissandrini for their guidance and insights, which have been instrumental in shaping this work.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2623–2631.

[2] Lasse Andersen, Victor Hansen, Claes Mortensen, Martin Mortensen, and Jacob Skadborg. 2025. Generating Robust and Reliable Datasets for Benchmarking NL2SQL in Real-World Applications. https://kbdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/a7me0f/alma9921966402605762

[3] Kaiwen Chen, Yueting Chen, Nick Koudas, and Xiaohui Yu. 2025. Reliable Text-to-SQL with Adaptive Abstention. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–30.

[4] defog. 2025. *sqlcoder-7b-2.* https://huggingface.co/defog/sqlcoder-7b-2

[5] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, et al. 2024. NL2SQL is a solved problem... Not!. In *CIDR*.

[6] Tucudean G, Bucos M, Dragulescu B, and Caleanu CD. 2024. Natural language processing with transformers: a review. *PeerJ Computer Science* 10:e2222 (2024). https://doi.org/10.7717/peerj-cs.2222

[7] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599 [cs.AI] https://arxiv.org/abs/2411.08599

[8] Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. 2024. Large Language Models Are Not Strong Abstract Reasoners. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, Kate Larson (Ed.). International Joint Conferences on Artificial Intelligence Organization, 6270–6278. https://doi.org/10.24963/ijcai.2024/693 Main Track.

[9] Michael Glass, Mustafa Eyceoz, Dharmashankar Subramanian, Gaetano Rossiello, Long Vu, and Alfio Gliozzo. 2025. Extractive Schema Linking for Text-to-SQL. *arXiv preprint arXiv:2501.17174* (2025).

[10] Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation. arXiv:2306.08891 [cs.CL] https://arxiv.org/abs/2306.08891

[11] Satya Gundabathula and Sriram Kolar. 2024. PromptMind Team at EHRSQL-2024: Improving Reliability of SQL Generation using Ensemble LLMs. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, Tristan Naumann, Asma Ben Abacha, Steven Bethard, Kirk Roberts, and Danielle Bitterman (Eds.). Association for Computational Linguistics, 360–366.

[12] IBM. [n. d.]. *IBM's text-to-SQL generator takes top place on a benchmark for handling complex database queries.* https://research.ibm.com/blog/granite-LLM-text-to-SQL

[13] Yongrae Jo, Seongyun Lee, Minju Seo, Sung Ju Hwang, and Moontae Lee. 2024. LG AI Research & KAIST at EHRSQL 2024: Self-Training Large Language Models with Pseudo-Labeled Unanswerable Questions for a Reliable Text-to-SQL System on EHRs. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, Tristan Naumann, Asma Ben Abacha, Steven Bethard, Kirk Roberts, and Danielle Bitterman (Eds.). Association for Computational Linguistics, 635–643.

[14] Sangryul Kim, Donghee Han, and Sehyun Kim. 2024. ProbGate at EHRSQL 2024: Enhancing SQL Query Generation Accuracy through Probabilistic Threshold Filtering and Error Handling. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, Tristan Naumann, Asma Ben Abacha, Steven Bethard, Kirk Roberts, and Danielle Bitterman (Eds.). Association for Computational Linguistics, 687–696.

[15] Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. 2024. Causal Reasoning and Large Language Models: Opening a New Frontier for Causality. arXiv:2305.00050 [cs.AI] https://arxiv.org/abs/2305.00050

[16] Gyubok Lee, Woosog Chay, Seonhee Cho, and Edward Choi. 2024. TrustSQL: Benchmarking Text-to-SQL Reliability with Penalty-Based Scoring. (2024).

[17] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. EHRSQL: A Practical Text-to-SQL Benchmark for Electronic Health Records. *Advances in Neural Information Processing Systems* 35 (2022), 15589–15601.

[18] Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. 2024. Overview of the EHRSQL 2024 Shared Task on Reliable Text-to-SQL Modeling on Electronic Health Records. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*. 644–654.

[19] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. arXiv:2503.02240 [cs.CL] https://arxiv.org/abs/2503.02240

[20] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.

[21] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/tacl_a_00638

[22] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109* (2024).

[23] Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Noisy Channel Language Model Prompting for Few-Shot Text Classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 5316–5330. https://doi.org/10.18653/v1/2022.acl-long.365

[24] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 8212–8220. https://doi.org/10.18653/v1/2024.findings-emnlp.481

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[26] Vipula Rawte, Swagata Chakraborty, Agnibh Pathak, Anubhav Sarkar, SM Towhidul Islam Tonmoy, Aman Chadha, Amit Sheth, and Amitava Das. 2023. The Troubling Emergence of Hallucination in Large Language Models-An Extensive Definition, Quantification, and Prescriptive Remediations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2541–2573.

[27] Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 2023. Know what I don't know: Handling ambiguous and unknown questions for text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*. 5701–5714.

[28] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2023. DBCopilot: Natural Language Querying over Massive Databases via Schema Routing. (2023).

[29] 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. In *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*. 3911–3921.

[30] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. *Proceedings of the VLDB Endowment* 17, 4 (2023), 685–698.

[31] Yusen Zhang, Xiangyu Dong, Shuaichen Chang, Tao Yu, Peng Shi, and Rui Zhang. 2020. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. *arXiv preprint arXiv:2010.12634* (2020).

[32] Ruilin Zhao, Feng Zhao, Long Wang, Xianzhi Wang, and Guandong Xu. 2024. KG-CoT: Chain-of-Thought Prompting of Large Language Models over Knowledge Graphs for Knowledge-Aware Question Answering. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, Kate Larson (Ed.). International Joint Conferences on Artificial Intelligence Organization, 6642–6650. https://doi.org/10.24963/ijcai.2024/734 Main Track.

8.3 Appendix A: Implementation on TrustSQL Baselines

To evaluate the performance of our method against current state-of-the-art, we implement both the pipeline-based and unified approaches introduced in TrustSQL [16]. Since the original TrustSQL dataset is not publicly available, we adapt their methods to work on the modified BIRD dataset, which serves as our benchmark for supervised fine-tuning. This appendix details how we reproduced their methodology.

8.3.1 A.1 Pipeline-based approach. TrustSQL proposes a three stage pipeline for SQL generation: (1) classification of infeasible questions, (2) SQL generation via a sequence to sequence model, and (3) detection of incorrect SQL outputs. We implement each of these stages using models and data aligned with TrustSQL's architecture, while applying them to our modified BIRD.

Table 8: Training configuration used to replicate TrustSQL's pipeline approach on the BIRD dataset

Model	T5-3B [25]	sqlcoder-7b-2 [4] (infeasibility)	sqlcoder-7b-2 [4] (sql errors)
Input	Question + serialised schema	Question + serialised schema (labelled infeasibility)	Question + serialised schema (labelled SQL errors)
Optimiser	Adam	Adam	Adam
Precision	BF16	FP16	FP16
Epochs	Until loss flattens	1	1
Learning Rate	1e-4	1e-4	1e-4
Trainer	Seq2SeqTrainer	AutoModelForCausalLM	AutoModelForCausalLM

Until loss flattens is not defined specifically in the paper. Here we assume the first two decimals as important, when they stabilise, we stop training.

8.3.2 A.2 Unified Approach. We only implement their unified approach using T5 and maximum entropy, as this approach showed the most promising results in the original paper. We use the same model for SQL generation here, as for the pipeline approach. To find the maximum entropy threshold we apply the heuristic described in TrustSQL appendix C.2:

(1) For each example in a validation set, compute the mean output entropy.
(2) Sort examples by entropy, and compute a cumulative score (+1 for correct outputs, -1 for incorrect ones).
(3) Threshold is selected at the point where the cumulative score stops increasing.