Calibrating LLMs for Text-to-SQL Parsing by Leveraging Sub-clause Frequencies

Anonymous ACL submission

Abstract

While large language models (LLMs) achieve strong performance on text-to-SQL parsing, they sometimes exhibit unexpected failures in which they are confidently incorrect. Building trustworthy text-to-SQL systems thus requires eliciting reliable uncertainty measures from the LLM. In this paper, we study the problem of providing a calibrated confidence score that conveys the likelihood of an output query being correct. Our work is the first to establish a benchmark for post-hoc calibration of LLM-based text-to-SQL parsing. In particular, we show that Platt scaling, a canonical method for calibration, provides substantial improvements over directly using raw model output probabilities as confidence scores. Furthermore, we propose a method for text-to-SQL calibration that leverages the structured nature of SQL queries to provide more granular signals of correctness, named "sub-clause frequency" (SCF) scores. Using multivariate Platt scaling (MPS), our extension of the canonical Platt scaling technique, we combine individual SCF scores into an overall accurate and calibrated score. Empirical evaluation on two popular text-to-SQL datasets shows that our approach of combining MPS and SCF yields further improvements in calibration and the related task of error correction over traditional Platt scaling. Our method could be easily adapted to improve calibration of LLMs on other semantic parsing or structured prediction tasks.

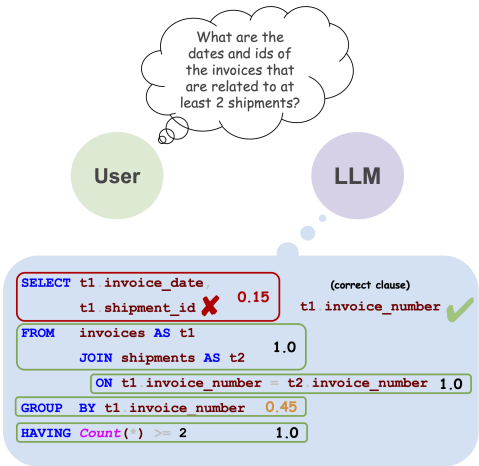


Figure 1: An example question from the SPIDER dataset and an output produced by T5 3B. Each box in the SQL output corresponds to a separate sub-clause, identified using a standard parser. The boundaries of the boxes in green and red denote sub-clauses that are correct and incorrect (respectively) with respect to a ground truth query. In each box, we have shown a sub-clause frequency (SCF) score computed by our method. The SCF corresponds to the proportion of outputs that contain each sub-clause among additional outputs sampled via nucleus sampling and beam search (Figure 2 illustrates this further). SCF scores contain a granular signal for correctness. Here, the incorrect `SELECT` clause (which contains `t1.shipment_id` instead of the correct column, `t1.invoice_number`) has a low SCF of 0.15.

1 Introduction

Text-to-SQL parsing is the problem of translating natural language queries into Structured Query Language (SQL) code. As the scale of structured data continues to grow, languages such as SQL provide a natural way to access the data of interest (Popescu et al., 2003). Text-to-SQL parsing therefore aims to provide a natural language interface for users who need not become familiar with the SQL language syntax and data schema (Giordani and Moschitti, 2012; Iyer et al., 2017; Zhong et al., 2017).

While these large language models (LLMs) show improved performance on standard benchmarks (Rai et al., 2023; Gao et al., 2023; Pourreza and Rafiei, 2024), they can produce erroneous outputs (Huang et al., 2023) while exhibiting high confidence (Borji, 2023; Zhou et al., 2024). Developing real-world systems requires more than just achieving high accuracy on a task—in order for users to have more control over how an output of the model is used, such systems must possess the ability to provide additional transparency in the sys-

tem's behavior (Yao et al., 2019; Wang et al., 2023). Consequently, applying uncertainty quantification to language modeling has become imperative for many NLP systems—including text-to-SQL—in order to communicate the risks associated with treating model outputs as correct (Dong et al., 2018).

In this paper, we study uncertainty quantification in the form of calibration, aiming to derive uncertainty (or "risk") estimates using post-hoc methods, which can be applied to any model output. These estimates take on the form of probabilities, and the goal is for the reported probabilities to match the true probability of the model being correct.

Deep learning-based models often produce probabilities that are not well-calibrated (Guo et al., 2017). Therefore, we consider a canonical *post-hoc* calibration approach, Platt scaling (Platt et al., 1999), and demonstrate that it significantly improves calibration of LLMs on text-to-SQL. Moreover, we then propose an extension of Platt scaling, which we call multivariate Platt scaling (MPS), to further improve performance by exploiting the fact that SQL syntax is highly structured and can be broken down into sub-clauses. Our intuition is that the frequency of each of the generated sub-clauses across multiple likely output samples from the model (named "sub-clause frequency" or SCF) is indicative of confidence (or lack thereof) and can therefore be used in MPS. In Figure 1, we present a graphical summary of our approach.

Our contributions are as follows:

- We benchmark post-hoc calibration of LLMs on text-to-SQL parsing using a variety of metrics for directly measuring calibration, as well as the related task of error detection—we find that LLMs are very miscalibrated out-of-the-box;
- We propose multivariate Platt scaling (MPS), which uses sub-clause frequencies (SCF) from multiple generated samples—our experiments show that MPS consistently outperforms baseline approaches;
- We perform analyses of calibration across models, query complexity, and shifts in probability magnitude to identify patterns related to calibration performance.

2 Related Work

Uncertainty Quantification of NLP models. Uncertainty quantification has been studied for a few NLP tasks such as machine translation (Blatz et al., 2004; Ueffing and Ney, 2005; Kumar and Sarawagi, 2019; Wang et al., 2020) and question-answering (Gondek et al., 2012; Jiang et al., 2021; Si et al., 2022; Kadavath et al., 2022). More recently, Lin et al. (2022) and Xiong et al. (2023) explore whether a model can express uncertainty through verbalized confidence. Detommaso et al. (2024) study a variation of calibration—*multi*calibration (Hébert-Johnson et al., 2018)—for question-answering. Framing long-form text generation as a *conformal prediction* (Shafer and Vovk, 2008) problem, Mohri and Hashimoto (2024) provide high probability guarantees for factuality. Liu and Wu (2024) extend both works, studying multicalibration and *multivalid* conformal prediction (Jung et al., 2022) for long-form text generation.

Calibration in Semantic Parsing. A couple works have studied calibration for semantic parsing tasks. Dong et al. (2018) train a gradient-boosted tree model to predict confidence scores by using features derived from knowledge of the training data or by applying perturbations to the model itself. Stengel-Eskin and Van Durme (2023) benchmark how well-calibrated token probabilities from the underlying language model are for text-to-SQL parsing, finding that these probabilities are very miscalibrated. In contrast, our work also establishes how calibrated LLMs are on text-to-SQL after correcting model probabilities using post-hoc calibration while then proposing an improved post-hoc approach that only requires access to additional generations from the model.

Text-to-SQL Parsing. We propose a method for calibration that partially relies on extracting additional information from sub-clauses in SQL outputs. Similarly, past works have also used the structure of SQL syntax for various purposes in text-to-SQL parsing. For example, Chen et al. (2023b) propose a SQL query correction method that considers edits at the clause level, and Tai et al. (2023) adapt chain-of-thought prompting for text-to-SQL, constructing reasoning paths based on the logical execution order of SQL clauses. For error detection, which we use as an auxiliary evaluation metric that supplements our calibration study, Chen et al. (2023a) build an error detection model trained on realistic parsing errors that they collect from model outputs. Finally, Wang et al. (2023) propose methods for handling ambiguous and unanswerable questions for text-to-SQL models.

3 Calibration

Calibration refers to an alignment between reported and observed probabilities. To define calibration formally, we have to first define the event of interest for which probabilities are being reported. In the context of text-to-SQL parsing using LLMs, we consider the problem of producing calibrated probabilities for the correctness of the query that has been output by the LLM. For our problem, a sample looks like $(X, Y) \sim \mathcal{D}$ where $X \in \mathcal{X}$ is the SQL query output by the LLM and $Y \in \{0, 1\}$ indicates the correctness of the SQL query corresponding to the natural language prompt:

$$Y = \mathbf{1}\{X \text{ is an accurate SQL query}\}.$$

Our goal is to produce an uncertainty score function $s : \mathcal{X} \to [0, 1]$ that measures the confidence of X being correct. While a basic requirement for s is that higher values should denote higher levels of confidence, we can also ask if s is calibrated, meaning that it satisfies,

$$P_{\mathcal{D}}(Y = 1 \mid s(X) = p) = p, \forall x \in \mathcal{X}. \quad (1)$$

In other words, when the reported probability or confidence score is p, the probability of the SQL query being correct is also p.

However, because p can take on an infinite number of values between 0 and 1, ensuring that Equation 1 holds for all p is intractable. A typical simplification is to consider calibration when the output of s is discretized into coarser probability bins (e.g., $[0.0.2), \ldots, [0.8.1.0]$) and ensuring calibration on those bins. This is the definition we will aim for.

Definition 1. *(Calibration) Suppose we have k probability bins S_i that partition $[0, 1]$. Then a scoring function s is calibrated w.r.t $\mathcal{D}$ if*

$$\Delta_i(s) = 0, \forall i \in 1 \ldots k, \quad (2)$$

where $\Delta_i(s)$ is the bias of s for the k-th bin S_i:

$$\Delta_i(s) = \mathbb{E}_{\mathcal{X} \sim \mathcal{D}}[Y - s(X) \mid s(X) \in S_i]. \quad (3)$$

Note that Equation 1 achieves calibration in the sense of Definition 1.

4 Methods

Our methodological contributions are (1) the method of multivariate Platt scaling and (2) its adaptation to semantic parsing using extracted sub-clause frequencies. We highlight that our approach to uncertainty quantification for text-to-SQL, which is motivated by high-level statistical insights, are novel and likely have implications for other tasks in NLP and machine learning in general.

4.1 Platt Scaling

Platt scaling (Platt et al., 1999) is a technique that calibrates the probabilistic outputs of a model using a logistic regression mapping. Given a score $s \in [0, 1]$ produced using some model, canonical Platt scaling corresponds to learning two weights, $w_0, w_1 \in \mathbb{R}$ that map s to a new value in $[0, 1]$:

$$s_{\text{PS}} \mapsto \text{sigmoid}(w_0 + w_1 \cdot \text{logit}(s)). \quad (4)$$

Here $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ and $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$. These are inverses, so setting $(w_0, w_1) = (0, 1)$ recovers the identity mapping.[1] These weights are typically learned on held out data such as the validation data. We discuss this further in the following subsection.

Platt scaling is simple, interpretable, and often works well in practice. Many variants of Platt scaling have recently been developed: for calibrating the top-label probabilities of deep nets (Guo et al., 2017; Gupta and Ramdas, 2022), calibrating full-vector probabilities in multiclass settings (Kull et al., 2019), and combining calibration with online learning style adversarial (worst-case) guarantees (Gupta and Ramdas, 2023).

4.2 Multivariate Platt Scaling

Our work provides a multivariate extension of Platt scaling (MPS) that is natural, yet understudied (to the best of our knowledge). Suppose a number of signals $s_1, s_2, \ldots, s_m \in \mathbb{R}$ are available for predicting a single probability in $[0, 1]$. Given these signals, we learn (in the spirit of Equation 4) $m + 1$ weights $w_0, w_1, \ldots, w_m \in \mathbb{R}$ that correspond to the mapping,

$$s_{\text{MPS}} \mapsto \text{sigmoid}(w_0 + w_1 s_1 + \ldots + w_m s_m), \quad (5)$$

so that $s_{\text{MPS}} \in [0, 1]$ is the final, combined signal.

Although Equation 5 resembles typical ML models, we emphasize a key difference in that while signals s_i are learned on the training data, **the weights $\mathbf{w_i}$ are learned on a held out validation set**—also called **calibration** data. Typically, the validation set is only used for model selection and not modeling itself so that we have some out-of-sample metric that checks for overfitting. In our case, since the secondary Platt scaling model is small ($m + 1$ parameters), we can reuse validation data *without significant overfitting*, as long as a reasonable number of validation points are available (say $\Omega(m^2)$).

[1] Other values of (w_0, w_1) correspond to natural monotonic mappings that can be used for calibration; see Niculescu-Mizil and Caruana (2005, Figure 1 and others) for examples.

Multivariate Platt scaling is one of many "post-hoc" calibration approaches that aim to improve calibration of a model without sacrificing accuracy.

Upon reviewing the literature, we found only a couple of instances where an approach similar to MPS is used. In work on object detection in images, Kuppers et al. (2020) utilize predicted bounding box locations as additional signals for calibration. In addition, Gopalan et al. (2022) apply Platt scaling for multicalibration, where the goal is to be calibrated while conditioning on predefined subgroups. In their formulation, signals are constrained to be binary features that denote whether some data point belongs to a group. In contrast, (1) high-level statistical insight discussed in the preceding paragraph and (2) its application to uncertainty quantification for text-to-SQL are novel, and likely have implications for other problems, especially in the context of semantic parsing.

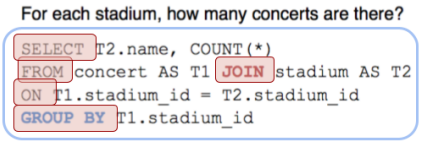
4.3 Extracting Sub-Clause Frequencies

We now describe how the signals s_i are computed for the task of text-to-SQL parsing. Generally, one can obtain many signals from LLMs aside from the probability of the full sequence output. In our work, we leverage the fact that the output is structured, comprising multiple sub-clauses for which we can compute output frequencies.

Our method draws inspiration from *self-consistency* (Wang et al., 2022) for a model that produces multiple outputs for the same prompt. Each candidate output can then be assigned a frequency score that counts the number of times the candidate appears among all outputs. The one with the highest frequency score (i.e., the most consistent candidate) is selected as the final output.

In our method, we too calculate frequency scores for each output. However, we exploit the fact that SQL queries are structured in a particular way. For example, all queries must contain `SELECT` and `FROM` clauses, and there exist only a finite number of additional clause types that a query may include (e.g., `WHERE`, `GROUP BY`, etc.). Moreover, while a theoretically infinite number of SQL queries can be composed together (via operators such as `JOIN`, `UNION`, etc.), the number of sub-queries found in most SQL queries (including those in prevalent text-to-SQL benchmarks) is small.

Thus, instead of calculating the number of times the entire SQL query output appears, as is typically done in *self-consistency*-based methods, our method counts the number of times each clause appears. As a result, we produce signals s_i for every sub-clause, to which we apply multivariate Platt scaling as described in the previous subsection. In Figure 2, we provide a simplified example of what this procedure looks like using nucleus sampling (Holtzman et al., 2020) and beam search (Freitag and Al-Onaizan, 2017).



Sub-clause		nucleus	beam
SELECT	name, COUNT(*)	9 / 10	5 / 10
FROM	concert, stadium	7 / 10	2 / 10
ON	stadium_id = stadium_id	4 / 10	4 / 10
WHERE	**[None]**	1 /10	2 / 10
GROUP BY	stadium_id	3 / 10	6 / 10

Figure 2: We show how to parse a query from the SPIDER dataset in order to derive sub-clause frequency scores s_i. Specifically, we count the number of times each sub-clause appears among outputs derived from (1) nucleus sampling and (2) beam search. Note that for illustrative purposes, we show a simplified example in which we assume in our data universe that we do not have multiple `SELECT` statements composed together and the only possible sub-clauses that can appear are `SELECT`, `FROM`, `ON`, `WHERE`, and `GROUP BY`. Thus in total, we have 10 signals s_i that can be provided to MPS.

5 Experimental Setup

We note that for all experiments, we use the calibration set (defined in Section 5.1) to construct our mappings s_{PS} and s_{MPS}. We then evaluate our mappings on the test sets for each dataset, reporting our findings over a single run of the calibration methods (without hyperparameter tuning).

5.1 Datasets

We evaluate our methods on two popular benchmarks for (English) text-to-SQL parsing: SPIDER (Yu et al., 2018) and BIRD (Li et al., 2024).

SPIDER consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases covering 138 domains. Serving as a complex, cross-domain dataset, SPIDER helped define a new semantic parsing benchmark in which databases at test time are not seen during training. As a result, models are tasked to generalize to new database schemas, which are provided alongside natural lan-

guage questions during evaluation. The **test** set has 2,148 examples, and the development set (which we use as our **calibration** set) has 1,034 examples.

BIRD consists of 12,751 unique question-SQL pairs on 95 databases, covering 37 domains. Focusing on the challenge of handling dirty and noisy database values, the queries in BIRD are often more complex than those in SPIDER. In addition, Li et al. (2024) introduce the notion of external knowledge, which contains information mapping natural language questions to database values. Each question is paired with some external knowledge to provide additional assistance to LLMs parsing the question. Because a test set has not yet been released publicly, we use the development set (of size 1,534) as our **test** set and randomly sample 1,000 examples from the train set to use as our **calibration** set.

5.2 Models

We use T5-3B-PICARD (which we will refer to as **T5 3B** going forward) and **LLAMA 3.1 8B INSTRUCT** on the SPIDER dataset. The T5-3B-PICARD model is a T5 model (available at this link) fine-tuned on SPIDER using constrained decoding for text-to-SQL parsing, achieving strong results on the task (Scholak et al., 2021). LLAMA 3.1 8B INSTRUCT (Dubey et al., 2024) is used with zero-shot prompting because the model is also able to obtain strong accuracy on this task. For the BIRD dataset, we use both **LLAMA 3.1 8B INSTRUCT** and **LLAMA 3.1 70B INSTRUCT** (Dubey et al., 2024) with zero-shot prompting as they also have been shown to perform relatively well. We did not use T5 variants for this dataset as their accuracies were low (23.34%) (Li et al., 2024). Prompts for our experiments are presented in Appendix C.

We note that our experiments use open-weight models because we have access to token probabilities and are able to control the resampling of outputs, which is not possible using closed-source models like GPT-4o. In addition, open-weight models facilitate reproducibility of our findings.

In addition, as mentioned in Section 4.3, our calibration approaches rely on generating multiple outputs for each question in our datasets. For each set of generations, we take the output with the highest model probability (sum over log token probabilities given by the model) as our primary prediction. The probability for the primary prediction is then relearned (calibrated) using the techniques of this paper. Table 2 in the Appendix shows the accuracy of the primary prediction on two datasets.

5.3 Computing Sub-Clause Frequencies

To run MPS using parsed sub-clause frequencies as signals, we sample outputs via nucleus sampling and beam search. In our experiments, we set the number of samples for each method to $k = 10$,[2] leading to a total of 20 samples. Beam search takes the top $k = 10$ outputs by probability (after length normalization) using a beam width of $2k$. Then, we compute sub-clause frequencies for each sample using all samples from nucleus sampling and beam search (as illustrated in Figure 2).

We note that we remove from calibration a small proportion of outputs that have syntactical errors since the correctness of such outputs can be determined quite easily with automated SQL parsers.

5.4 Metrics for Calibration

We now present the evaluation metrics we use to measure calibration. Specifically, we consider expected calibration error, adaptive calibration error, and Brier score together in order to provide a more holistic and robust evaluation of calibration.

First, given some set of probability bins S_i, we define ℓ_1-calibration error as the following:

$$\ell_1\text{-CE}(s) = \sum_{i=1}^{k} P_{\mathcal{X}\sim\mathcal{D}}(s(X) \in S_i)\,|\Delta_i(s)|\,. \quad (6)$$

(Recall the definition of Δ_i from Definition 1.) As Equation 6 suggests, calibration error is highly dependent on the choice of probability bins S_i. We consider two variations:

1. ***expected* calibration error (ECE)**: bins S_i are defined as k equal-width bins: $[0, \frac{1}{k}), [\frac{1}{k}, \frac{2}{k}), \ldots, [\frac{k-1}{k}, 1]$.
2. ***adaptive* calibration error (ACE)**: bins S_i are defined such that the all sizes of all bins are the same (i.e., $P_{\mathcal{X}\sim\mathcal{D}}(s(X) \in S_i) = \frac{1}{k}$).

In our results, we calculate ECE and ACE using $k = 10$ bins.

In addition to minimizing calibration error, it is often desirable to output a wide range of scores s that can help users better distinguish between high and low confidence outputs. The spread of scores is sometimes referred to as *resolution* (Bröcker, 2009) in the literature, where higher resolution corresponds to a larger spread. Calibration error, however, does not capture this spread. For example,

[2] These numbers were set after initial experiments on the development set. We note that we did not attempt to further tune these hyperparameters, which can only lead to better results for our proposed approach.

Dataset	Model	Method	Brier (↓)	ECE (↓)	ACE (↓)	AUC (↑)
SPIDER	T5 3B	Uncalibrated	0.1893	0.1632	0.1632	0.7200
		PS	0.1656	0.0623	0.0628	0.7200
		MPS (*ours*)	**0.1566**	**0.0264**	**0.0253**	**0.7785**
	Llama 3.1 8B Instruct	Uncalibrated	0.3576	0.4161	0.4147	0.7123
		PS	0.1726	0.0437	0.0449	0.7123
		MPS (*ours*)	**0.1626**	**0.0244**	**0.0260**	**0.7475**
BIRD	Llama 3.1 8B Instruct	Uncalibrated	0.2378	0.1722	0.1688	0.6913
		PS	0.2129	0.0556	0.0739	0.6913
		MPS (*ours*)	**0.1950**	**0.0418**	**0.0423**	**0.7353**
	Llama 3.1 70B Instruct	Uncalibrated	0.2866	0.2343	0.2324	0.6650
		PS	0.2316	0.0549	0.0631	0.6650
		MPS (*ours*)	**0.2141**	**0.0314**	**0.0323**	**0.7173**

Table 1: We report Brier score, expected calibration error (ECE), and adaptive calibration Error (ACE) on the SPIDER and BIRD datasets. In addition, we report AUC for the task of error detection. These metrics are calculated after applying Platt scaling (**PS**, *baseline*) to token probabilities and Multiplicative Platt scaling (**MPS**, *ours*) to token probabilities and parsed frequencies (derived from nucleus sampling and beam search). For reference, we also report (**Uncalibrated**) how well calibrated the model token probabilities are prior to applying Platt scaling.

as an extreme case, an algorithm could output the same probability $s(x) = P_D(Y = 1)$ for all examples in $x \in D$. Despite being perfectly calibrated (i.e., calibration error of 0), this scoring function serves no purpose in communicating risk for individual examples X.

Consequently, another metric that is often considered for evaluating calibration is **Brier score**, which is the mean squared error of $s(x)$

$$BS(s) = \mathbb{E}_{\mathcal{X} \sim \mathcal{D}}[(Y - s(X))^2] \quad (7)$$

Intuitively, a lower Brier score is desirable since correct outputs ($Y = 1$) should be accompanied by higher probability estimates ($s(X)$ closer to 1) and incorrect outputs ($Y = 0$) by lower probability estimates ($s(X)$ closer to 0). Moreover, unlike calibration error, Brier score penalizes algorithms with low resolution.[3]

6 Results

6.1 Calibration

Table 1 presents the results of our LLM-based text-to-SQL parsers on both datasets using the two approaches to calibration: Platt scaling (PS) and multivariate Platt scaling (MPS) with sub-clause frequencies (SCF). Here, MPS uses SCF scores derived from both nucleus sampling and beam search.[4] As reference, we provide the calibration metrics for (uncalibrated) token probabilities that are outputted directly from the model.

We observe that the probabilities produced by the LLMs are consistently and significantly miscalibrated across all experiments. However, Platt scaling (PS) improves calibration across both datasets and models. In some cases, PS improves calibration metrics by very wide margins (e.g. LLAMA 3.1 8B INSTRUCT on SPIDER shows a reduction of 0.17 in Brier score and 0.37 in ECE and ACE).

Our method (MPS) further improves performance of calibration, cutting calibration error by over 50% in some cases (e.g., T5 3B on SPIDER). We highlight that MPS maintains this advantage across all datasets and models, providing strong evidence for the utility of using the sub-clause structure of SQL syntax to derive additional signals s_i for producing calibrated probabilities.

6.2 Evaluating Error Detection

We also evaluate our methods for calibration on the related task of error detection to demonstrate their utility. In particular, we assess how the (calibrated) probabilities derived from multivariate Platt scaling can be used to detect errors in the model outputs. Using AUC as our evaluation metric, our

[3] In fact, as shown in Bröcker (2009, Equation 13), Brier score can be decomposed into calibration error and (negative) resolution, thereby capturing both properties that are valuable for uncertainty quantification.

[4] Section 7.4 discusses these choices for calculating SCF scores in more detail.

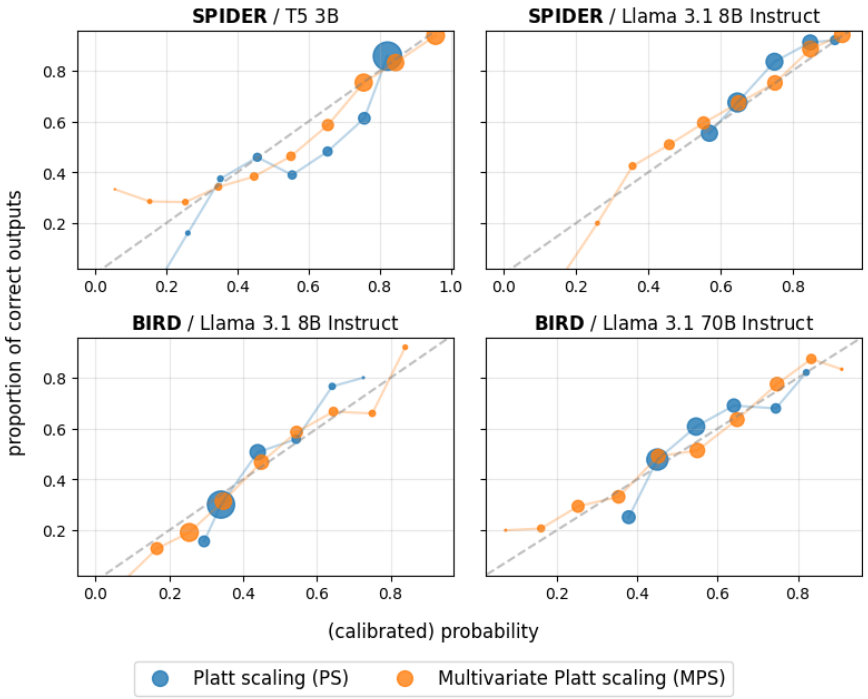


Figure 3: Calibration curves (using fixed-width ECE-style bins) comparing Platt scaling (blue) applied to final token probabilities, and (ours) multivariate Platt scaling (orange) applied to token probabilities and sub-clause frequencies. The radius of the markers in the scatter plot are proportional to the number of points in the corresponding probability bin. For instance, the blue marker near $(0.8, 0.8)$ of the top-left plot corresponds to 67.4% of the data with predicted probabilities in the range $[0.8, 0.9)$. Across all models (T5, Llama) and datasets (SPIDER, BIRD), our method is better calibrated (scatter plot close to the $x = y$ line), and has higher *resolution* (varied values on the x-axis).

results characterize the trade-offs between true and false positive rates when choosing different thresholds for determining if a SQL output is erroneous (e.g., one could predict that a SQL output is wrong whenever MPS produces a probability ≤ 0.3).

In Table 1, we again show that MPS using SCF performs the best, achieving higher AUC across all datasets and models. We note that because PS applies a monotonic transformation to the uncalibrated model probabilities, it—unlike MPS—can only affect calibration but not error detection (i.e., AUC remains unchanged after applying PS).

7 Analysis

7.1 Calibration Plots

Figure 3 shows the calibration plots using ECE, comparing probabilities calibrated by PS and MPS. To convey the number of examples in each bin, we scale the radius of each marker by the bin size. Generally, we observe that calibration errors are closer to 0 (i.e., points lie closer to $y = x$) for MPS compared to PS, reflecting the lower ECE of MPS.

In addition, we examine qualitatively the spread of the probabilities produced by PS and MPS. In most cases, a more spread out or higher *resolution* forecast is more desirable for uncertainty quantification and is correlated with smaller Brier scores (see Section 5.4). For the SPIDER dataset, we observe that probabilities produced by PS with T5 3B are heavily concentrated around 0.8. PS with LLAMA 3.1 8B INSTRUCT only outputs probabilities ≥ 0.5. Similarly, on the BIRD dataset, PS with LLAMA 3.1 8B INSTRUCT mostly outputs probabilities around a single value (0.3), while for PS with LLAMA 3.1 70B INSTRUCT, most probabilities produced by PS fall between 0.4 and 0.7. In contrast, we observe that MPS consistently produces probabilities that are more spread out (while still lying close to $y = x$).

Calibration plots for ACE can be found in the Appendix Figure 4. Qualitatively, these plots are similar to those in Figure 3 and thus lead to similar findings, which we discuss in Appendix A.

7.2 Comparing Calibration across Models

When examining (uncalibrated) probabilities outputted directly by the LLM on SPIDER, we observe that those produced by fine-tuned T5 3B are better

calibrated (i.e., Brier score, ECE, and ACE are almost 2x smaller) than those of the larger LLAMA 3.1 8B INSTRUCT model, despite the two models achieving similar test accuracies (Table 2). However, after applying (multivariate) Platt scaling, the differences become small, providing strong evidence that when evaluating calibration for LLMs, post-hoc calibration (e.g., Platt scaling) should be conducted before forming conclusions about how well-calibrated LLMs are relative to each other.

On BIRD, Table 1 reports that the smaller LLAMA 3.1 8B INSTRUCT MODEL is better calibrated w.r.t. Brier score than its larger 70B counterpart, both before and after applying (multivariate) Platt scaling. We stress, however, that Brier score cannot be compared across models that achieve such different test accuracies (Table 2) since Brier score is very sensitive to this value. For example, the base rates (i.e., the Brier score when simply outputting the proportion of correct outputs $P(Y = 1)$ for all data points) for LLAMA 3.1 8B INSTRUCT and LLAMA 3.1 70B INSTRUCT are 0.2300 and 0.2489 respectively, and so one might expect that the Brier score of the 8B variant to be lower, post-calibration.

7.3 Stratified Magnitude Analysis

Next, we study the magnitude of adding features derived from parsing additional samples and how this impacts calibration scores. We thus analyze outputs with the largest (absolute) difference in output probability between MPS and PS by aggregating the top and bottom 1%, 5%, 10%, and 20% of samples, ordered by the difference (Δ) in the calibrated probability produced by PS and MPS.

We see that across all strata, MPS produces probabilities that are more calibrated, suggesting that queries for which MPS and PS *disagree on the most* are those for which *MPS is better calibrated on*. For example, in the top 1%, PS on average produces a probability of 0.464 while MPS produces an average probability of 0.825, which is closer to the true proportion of correct samples of 0.714. Full results are in the Appendix Table 6, with examples of queries with the largest absolute difference in probability between MPS and PS in the Appendix Tables 7, 8, 9 and 10.

7.4 Ablation Study for Deriving SCF Scores

We run ablation studies to identify the impact of using different methods for computing the outputs used to calculate the SCF scores: nucleus sampling only, beam search only, or both (which corresponds to the results for MPS presented in Table 1).

The results, available in full in Table 3 in the Appendix, show that no single sampling method is better than the other, with different methods being best depending on the dataset and LLM used. We find that using both provides the most well-rounded performance across all experiments. However, we stress that data and model specific optimal performances could be obtained if one were to treat signal s_i selection as a hyperparameter that can be tuned.

Finally, we emphasize that regardless of this choice, MPS outperforms PS in all cases.

7.5 Analysis by Query Complexity

In the SPIDER dataset, each example is annotated with difficulty ratings of *easy, medium, hard,* and *extra hard* Yu et al. (2018), while in BIRD, examples are annotated with difficult ratings of *simple, moderate,* and *challenging* Li et al. (2024). We calculate metrics for each level of difficulty in order to see if there is any relationship between query difficulty and calibration, as identified in Stengel-Eskin and Van Durme (2023, Figure 6) for LLMs' uncalibrated probability outputs. Our experiments, however, do not show consistent patterns in ECE, ACE and AUC across models and datasets after applying PS and MPS.[5] Full results are in the Appendix Tables 4 and 5.

8 Conclusion

Uncertainty quantification remains an open, yet crucial problem for LLMs. In this paper, we study calibration for text-to-SQL parsing, with a focus on measuring the effectiveness of post-hoc calibration methods. We introduce a method, multivariate Platt scaling, that leverages the structured nature of outputs for this task and uses sub-clause frequency scores as inputs. Our experiments demonstrate that this method significantly gives better calibrated probabilities for the correctness of the query output by the LLM than the baseline approaches. We hope that this work will inspire future work on leveraging the structured outputs in semantic parsing tasks for uncertainty quantification.

[5] Note that while for completeness, we report Brier score in Tables 4 and 5, we cannot use Brier score to compare between levels of difficulty since the model accuracy for each difficulty varies substantially (See Section 7.2) for a more detailed explanation.

9 Limitations

First, the datasets used in our experiments are solely in English, although our methods are applicable to other languages directly as long as a large language model covering the target language is available. This choice allows for consistency and comparability across the datasets, but it does not test the generalizability of our findings to other languages. In future work, we plan to extend our research to a multilingual setting to address this limitation, for example using the multi-lingual SPIDER dataset (Jose and Cozman, 2023).

The proposed method for calibration requires generating multiple candidate output SQL queries given a natural language input. This generation introduces an increase in inference time, although sampling can be performed in parallel, limiting the potential latency of the response if calibration was included in a real-world application.

The experiments with the Llama 3.1 models are limited to zero-shot prompts. We did not experiment with additional fine-tuning or instruction-tuning with text-to-SQL task data, as the models already obtain good performance on the test set and were potentially already trained with data for this task. However, our approach can easily be adopted and used with such models, as shown in experiments with the T5-3B model trained using SPIDER.

10 Ethics Statement

We use publicly available datasets intended for the task of text-to-SQL semantic parsing. The datasets and large language models we used have permissive licenses allowing for research use. We do not envision any potential risks associated with the task discussed in this paper.

References

John Blatz, Erin Fitzgerald, George Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchis, and Nicola Ueffing. 2004. Confidence estimation for machine translation. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 315–321, Geneva, Switzerland. COLING.

Ali Borji. 2023. A categorical archive of chatgpt failures. *arXiv preprint arXiv:2302.03494*.

Jochen Bröcker. 2009. Reliability, sufficiency, and the decomposition of proper scores. *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography*, 135(643):1512–1519.

Shijie Chen, Ziru Chen, Huan Sun, and Yu Su. 2023a. Error detection for text-to-sql semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11730–11743.

Ziru Chen, Shijie Chen, Michael White, Raymond Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. 2023b. Text-to-sql error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1359–1372.

Gianluca Detommaso, Martin Bertran, Riccardo Fogliato, and Aaron Roth. 2024. Multicalibration for confidence scoring in LLMs. *arXiv preprint arXiv:2404.04689*.

Li Dong, Chris Quirk, and Mirella Lapata. 2018. Confidence modeling for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 743–753, Melbourne, Australia. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*.

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 preprint arXiv:2308.15363*.

Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of COLING 2012: Posters*, pages 401–410, Mumbai, India. The COLING 2012 Organizing Committee.

D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty. 2012. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3.4):14:1–14:12.

Parikshit Gopalan, Michael P Kim, Mihir A Singhal, and Shengjia Zhao. 2022. Low-degree multicalibration. In *Conference on Learning Theory*, pages 3193–3234. PMLR.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

Chirag Gupta and Aaditya Ramdas. 2022. Top-label Calibration and Multiclass-to-binary Reductions. In *International Conference on Learning Representations*.

Chirag Gupta and Aaditya Ramdas. 2023. Online Platt scaling with calibeating. In *International Conference on Machine Learning*, pages 12182–12204. PMLR.

Ursula Hébert-Johnson, Michael Kim, Omer Reingold, and Guy Rothblum. 2018. Multicalibration: Calibration for the (computationally-identifiable) masses. In *International Conference on Machine Learning*, pages 1939–1948. PMLR.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How Can We Know When Language Models Know? On the Calibration of Language Models for Question Answering. *Transactions of the Association for Computational Linguistics*, 9:962–977.

Marcelo Archanjo Jose and Fabio Gagliardi Cozman. 2023. A multilingual translator to sql with database schema pruning to improve self-attention. *International Journal of Information Technology*, 15(6):3015–3023.

Christopher Jung, Georgy Noarov, Ramya Ramalingam, and Aaron Roth. 2022. Batch multivalid conformal prediction. In *International Conference on Learning Representations*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. 2019. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. *Advances in neural information processing systems*, 32.

Aviral Kumar and Sunita Sarawagi. 2019. Calibration of encoder decoder models for neural machine translation. *arXiv preprint arXiv:1903.00802*.

Fabian Kuppers, Jan Kronenberger, Amirhossein Shantia, and Anselm Haselhoff. 2020. Multivariate confidence calibration for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 326–327.

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. *Advances in Neural Information Processing Systems*, 36.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*.

Terrance Liu and Zhiwei Steven Wu. 2024. Multi-group uncertainty quantification for long-form text generation. *arXiv preprint arXiv:2407.21057*.

Christopher Mohri and Tatsunori Hashimoto. 2024. Language models with conformal factuality guarantees. *arXiv preprint arXiv:2402.10978*.

Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632.

John Platt et al. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. 2023. Improving generalization in language model-based text-to-SQL semantic parsing: Two simple semantic boundary-based techniques. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 150–160, Toronto, Canada. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901.

Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3).

Chenglei Si, Chen Zhao, Sewon Min, and Jordan Boyd-Graber. 2022. Re-examining calibration: The case of question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2814–2829.

Elias Stengel-Eskin and Benjamin Van Durme. 2023. Calibrated interpretation: Confidence estimation in semantic parsing. *Transactions of the Association for Computational Linguistics*, 11:1213–1231.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393.

Nicola Ueffing and Hermann Ney. 2005. Word-level confidence estimation for machine translation using phrase-based translation models. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 763–770, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

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*, pages 5701–5714, Toronto, Canada. Association for Computational Linguistics.

Shuo Wang, Zhaopeng Tu, Shuming Shi, and Yang Liu. 2020. On the inference calibration of neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3070–3079.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. 2023. Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5447–5458, Hong Kong, China. Association for Computational Linguistics.

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

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Kaitlyn Zhou, Jena Hwang, Xiang Ren, and Maarten Sap. 2024. Relying on the unreliable: The impact of language models' reluctance to express uncertainty. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3623–3643, Bangkok, Thailand. Association for Computational Linguistics.

A Additional results

In Table 2 we show model performance on SPIDER and BIRD. For our experiments, we sample 10 outputs using nucleus sampling and 10 using beam search. Then for each query, we take the model output (among all outputs sampled) with the highest model probability (sum over log token probabilities) as our prediction. Table 2 shows the performance when choosing outputs in this way, picking the top output among the set of samples outputted by (1) nucleus sampling (**Nucleus**), (2) beam search (**Beam**), and (3) both (**N+B**). Generally speaking, we do not see significant different in model accuracy among these three choices and thus choose (3) for our experiments out of convenience.

Dataset	Model	Nucleus	Beam	N + B
SPIDER	T5 3B	73.52%	73.52%	73.52%
	Llama 3.1 8B Instr.	74.77%	73.77%	74.60%
BIRD	Llama 3.1 8B Instr.	34.60%	35.65%	35.85%
	Llama 3.1 70B Instr.	52.28%	53.18%	53.35%

Table 2: We report the accuracy of each model on the corresponding dataset's test set.

In Figure 4, we plot adaptive calibration error, where each point (or bin) represents roughly the same number of samples. Like in Figure 3, our method, MPS, is better calibrated (scatter plot close to the $x = y$ line) and has higher resolution (varied values on the x-axis) when compared to PS.

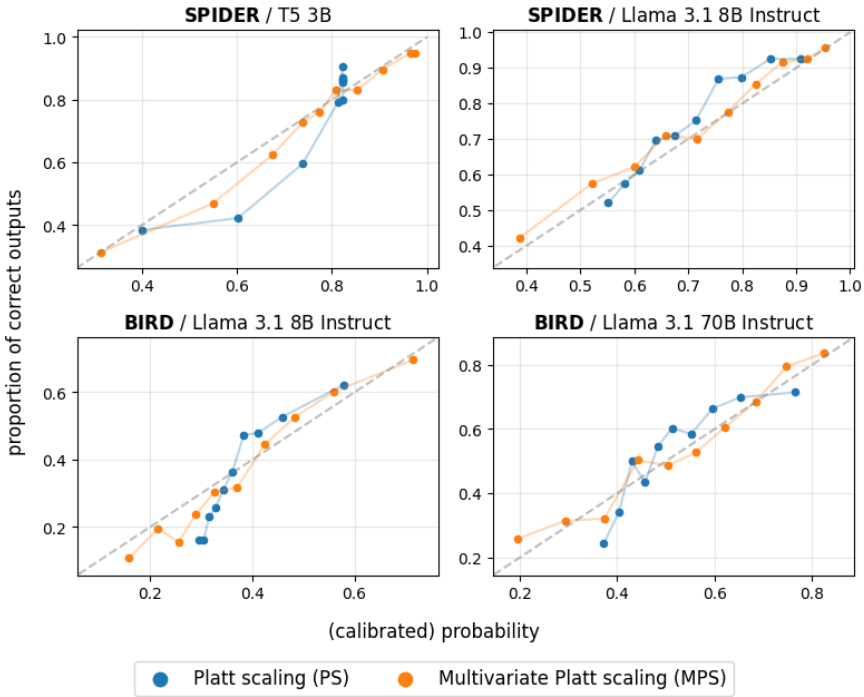


Figure 4: Calibration curves (using **ACE**) comparing (baseline) Platt scaling (PS, blue) applied to final token probabilities, and (ours) multivariate Platt scaling (MPS, orange) applied to token probabilities and sub-clause frequencies. Across all models (T5, Llama) and datasets (SPIDER, BIRD), our method is better calibrated (scatter plot close to the $x = y$ line), and has higher *resolution* (varied values on the x-axis).

In Table 3, we show how the performance of MPS varies based on the choice of deriving SCF scores (from nucleus sampling only, beam search only, or both).

We split SPIDER and BIRD datasets according to their difficulty annotations provided by the dataset curators. We find that after applying (multivariate) Platt scaling, there does not exist any strong correlation between the difficulty of a query and how well-calibrated these methods are on the example query (Tables

Dataset	Model		Method	Brier	ECE	ACE	AUC
SPIDER	T5 3B		Uncalibrated	0.1893	0.1632	0.1632	0.7200
		(PS)	Token Probs.	0.1656	0.0623	0.0628	0.7200
		(MPS)	+ Nucleus	0.1639	0.0460	0.0538	0.7229
			+ Beam	**0.1565**	**0.0241**	0.0308	**0.7804**
			+ N + B	0.1566	0.0264	**0.0253**	0.7785
	Llama 3.1 8B Instruct		Uncalibrated	0.3576	0.4161	0.4147	0.7123
		(PS)	Token Probs.	0.1726	0.0437	0.0449	0.7123
		(MPS)	+ Nucleus	**0.1625**	0.0280	0.0360	**0.7506**
			+ Beam	0.1687	**0.0244**	**0.0230**	0.7194
			+ N + B	0.1626	**0.0244**	0.0260	0.7475
BIRD	Llama 3.1 8B Instruct		Uncalibrated	0.2378	0.1722	0.1688	0.6913
		(PS)	Token Probs.	0.2129	0.0556	0.0739	0.6913
		(MPS)	+ Nucleus	0.1976	0.0574	0.0542	0.7307
			+ Beam	0.1989	0.0425	0.0442	0.7213
			+ N + B	**0.1951**	**0.0418**	**0.0423**	**0.7353**
	Llama 3.1 70B Instruct		Uncalibrated	0.2866	0.2343	0.2324	0.6650
		(PS)	Token Probs.	0.2316	0.0549	0.0631	0.6650
		(MPS)	+ Nucleus	**0.2130**	**0.0290**	**0.0267**	**0.7203**
			+ Beam	0.2215	0.0384	0.0416	0.6921
			+ N + B	0.2141	0.0314	0.0323	0.7173

Table 3: We report how the performance of Multiplicative Platt Scaling (**MPS**) is affected by changes in whether the parsed frequencies are derived from nucleus sampling only (**Nucleus**), beam search only (**Beam**), and both nucleus sampling and beam search (**N + B**). For reference, we also include the performance of only using token probabilities (i.e., **PS**). We report Brier score, expected calibration error (ECE), and adaptive calibration Error (ACE) on the SPIDER and BIRD datasets. ECE and ACE are calculated using 10 bins. In addition, we report AUC for the task of error detection.

4 and 5)

In Table 6, we show how and to what degree MPS shifts its calibrated probability output when compared to PS. In particular, we show the average probability for top 1%, 5%, 10%, and 20% samples, sorted by the change Δ in probability between MPS and PS. We see that MPS shifts the probabilities so that for each set of examples, the calibration error is smaller when compared to PS.

In Tables 7, 8, 9, 10, we show specific examples of queries from the SPIDER and BIRD datasets for which MPS and PS differ the most. In particular, we show examples of correct outputs where MPS shifts the calibrated probability upwards the most and incorrect outputs where MPS shifts probabilities downwards.

Model	Difficulty	Size	Accuracy	Method	Brier	ECE	ACE	AUC
T5 3B	easy	22%	0.917	PS	0.081	0.126	0.115	0.691
				MPS	0.076	0.075	0.069	0.714
	medium	40%	0.770	PS	0.161	0.052	0.057	0.678
				MPS	0.153	0.023	0.030	0.756
	hard	22%	0.641	PS	0.198	0.081	0.087	0.727
				MPS	0.188	0.060	0.060	0.764
	extra	16%	0.533	PS	0.246	0.155	0.155	0.674
				MPS	0.234	0.135	0.123	0.711
Llama 3.1 8B Instruct	easy	22%	0.893	PS	0.096	0.089	0.087	0.721
				MPS	0.085	0.044	0.043	0.778
	medium	40%	0.783	PS	0.164	0.082	0.083	0.697
				MPS	0.151	0.051	0.049	0.741
	hard	22%	0.646	PS	0.220	0.035	0.053	0.628
				MPS	0.221	0.071	0.068	0.628
	extra	16%	0.594	PS	0.233	0.063	0.103	0.587
				MPS	0.219	0.035	0.037	0.679

Table 4: Results on SPIDER, grouped by the difficulty of the SQL query. We observe that ECE, ACE, and AUC (for error detection) are not well-correlated with the difficulty of the query. Note that while for completeness, we report Brier score, it cannot be use to compare between levels of difficulty since the model accuracy for each difficulty varies substantially (See Section 7.2) for a more detailed explanation.

Model	Difficulty	Size	Accuracy	Method	Brier	ECE	ACE	AUC
Llama 3.1 8B Instruct	simple	62%	0.420	PS	0.226	0.044	0.076	0.681
				MPS	0.209	0.043	0.043	0.720
	moderate	29%	0.286	PS	0.201	0.066	0.070	0.621
				MPS	0.185	0.064	0.055	0.708
	challenging	9%	0.173	PS	0.161	0.163	0.230	0.778
				MPS	0.135	0.147	0.137	0.801
Llama 3.1 70B Instruct	simple	62%	0.591	PS	0.228	0.074	0.076	0.655
				MPS	0.210	0.034	0.041	0.714
	moderate	29%	0.479	PS	0.239	0.040	0.039	0.627
				MPS	0.222	0.045	0.036	0.688
	challenging	9%	0.324	PS	0.230	0.125	0.119	0.581
				MPS	0.213	0.104	0.120	0.632

Table 5: Results on BIRD, grouped by the difficulty of the SQL query. We observe that ECE, ACE, and AUC (for error detection) are not well-correlated with the difficulty of the query. Note that while for completeness, we report Brier score, it cannot be use to compare between levels of difficulty since the model accuracy for each difficulty varies substantially (See Section 7.2) for a more detailed explanation.

Dataset	Model			Prob. Δ	PS	MPS	Correct
SPIDER	T5 3B	1%	top	+ 0.361	0.464	**0.825**	0.714
			bottom	- 0.380	0.739	**0.359**	0.476
		5%	top	+ 0.223	0.635	**0.857**	0.764
			bottom	- 0.280	0.680	**0.400**	0.425
		10%	top	+ 0.187	0.729	**0.916**	0.859
			bottom	- 0.224	0.682	**0.458**	0.521
		20%	top	+ 0.169	0.775	**0.944**	0.904
			bottom	- 0.167	0.692	**0.525**	0.528
	Llama 3.1 8B Instruct	1%	top	+ 0.310	0.591	**0.902**	0.810
			bottom	- 0.351	0.638	**0.287**	0.429
		5%	top	+ 0.254	0.631	**0.884**	0.879
			bottom	- 0.276	0.654	**0.378**	0.477
		10%	top	+ 0.227	0.651	**0.878**	0.888
			bottom	- 0.234	0.659	**0.425**	0.500
		20%	top	+ 0.189	0.669	**0.858**	0.857
			bottom	- 0.180	0.671	**0.491**	0.577
BIRD	Llama 3.1 8B Instruct	1%	top	+ 0.371	0.386	**0.757**	0.929
			bottom	- 0.236	0.480	**0.244**	0.286
		5%	top	+ 0.305	0.400	**0.705**	0.676
			bottom	- 0.200	0.409	**0.210**	0.149
		10%	top	+ 0.267	0.410	**0.677**	0.642
			bottom	- 0.181	0.396	**0.215**	0.176
		20%	top	+ 0.211	0.402	**0.613**	0.598
			bottom	- 0.156	0.378	**0.222**	0.209
	Llama 3.1 70B Instruct	1%	top	+ 0.330	0.456	**0.786**	0.857
			bottom	- 0.376	0.658	**0.282**	0.429
		5%	top	+ 0.272	0.486	**0.758**	0.808
			bottom	- 0.297	0.556	**0.259**	0.397
		10%	top	+ 0.243	0.508	**0.751**	0.774
			bottom	- 0.261	0.519	**0.258**	0.356
		20%	top	+ 0.205	0.520	**0.724**	0.757
			bottom	- 0.214	0.507	**0.293**	0.353

Table 6: For each combination of dataset and model, we aggregate the top and bottom 1%, 5%, 10%, and 20% of data points, ordered by the difference (Δ) in the calibrated probability ouputted by Platt scaling (PS) and multiplicative Platt scaling (MPS) to summarize how adding features derived from parsing additional samples affects calibration. In addition, we report the proportion of outputs in each quantile that are actually correct. Between PS and MPS, we **bold** the method who has the smaller calibration error in each bucket. As show in these results, among the outputs for which MPS and PS differ the most (i.e., each quantile), MPS produces more calibrated probabilities in all case.

Question	Output	Label	PS	MPS	Δ

Return the names and ages of pilots who have flown Piper Cub and are older than 35, or have flown the F-14 Fighter and are younger than 30.	`SELECT pilot_name,`
	`  age`
	`FROM  pilotskills`
	`WHERE  plane_name='Piper Cub'`
	`  AND age > 35`
	`UNION`
	`SELECT pilot_name,`
	`  age`
	`FROM  pilotskills`
	`WHERE  plane_name='F-14 Fighter'`
	`  AND age < 30`
		Correct	0.275	0.928	**+ 0.652**

Find the receipt numbers where both Cake and Cookie were bought.	`SELECT t1.receiptnumber`
	`FROM  receipts AS t1`
	`  JOIN goods AS t2`
	`  ON t1.customerid=t2.id`
	`WHERE  t2.food="Cake"`
	`INTERSECT`
	`SELECT t1.receiptnumber`
	`FROM  receipts AS t1`
	`  JOIN goods AS t2`
	`  ON t1.customerid=t2.id`
	`WHERE  t2.food="Cookie"`
		Incorrect	0.611	0.145	**- 0.466**

Table 7: Example SQL outputs from **T5 3B** on the **BIRD** dataset, which were selected in the following way: We calculate the difference (Δ) in (calibrated) probability outputted by Platt scaling (PS, *baseline*) and multiplicative Platt scaling (MPS, *ours*). We then split outputs by their label (whether the output is correct or incorrect). For correct examples, we select the one with the largest positive change in Δ, and for incorrect examples, we select the one with the largest negative change in Δ.

Question	Output	Label	PS	MPS	Δ
What is the top speed and power of the vehicle manufactured in the year of 1996?	`SELECT top_speed, power FROM vehicle WHERE build_year=1996`	Correct	0.551	0.942	**+ 0.391**
Find the name of all the services which either have been used by customer "Hardy Kutch" or have been rated as "good" in one of the customer interactions.	`SELECT T1.service_details FROM services AS T1 INNER JOIN customers_and_services AS T2 ON T1.service_id=T2.service_id INNER JOIN customer_interactions AS T3 ON T2.customer_id=T3.customer_id WHERE T3.status_code='good' OR T2.customer_id=(SELECT customer_id FROM customers WHERE customer_details='Hardy Kutch')`	Incorrect	0.654	0.237	**- 0.417**

Table 8: Example SQL outputs from **LLAMA 3.1 8B INSTRUCT** on the **SPIDER** dataset, which were selected in the following way: We calculate the difference (Δ) in (calibrated) probability outputted by Platt scaling (PS, *baseline*) and multiplicative Platt scaling (MPS, *ours*). We then split outputs by their label (whether the output is correct or incorrect). For correct examples, we select the one with the largest positive change in Δ, and for incorrect examples, we select the one with the largest negative change in Δ.

Question	Output	Label	PS	MPS	Δ
How many majors are there in "College of Humanities and Social Sciences"?	`SELECT COUNT(*) FROM major WHERE college='college of humanities and social sciences'`	Correct	0.377	0.783	**+ 0.406**
Which different county has the most number of closed schools? Please provide the name of each school as well as the closure date.	`SELECT county, school, closeddate FROM schools WHERE statustype='Closed'`	Incorrect	0.461	0.186	**- 0.274**

Table 9: Example SQL outputs from **LLAMA 3.1 8B INSTRUCT** on the **BIRD** dataset, which were selected in the following way: We calculate the difference (Δ) in (calibrated) probability outputted by Platt scaling (PS, *baseline*) and multiplicative Platt scaling (MPS, *ours*). We then split outputs by their label (whether the output is correct or incorrect). For correct examples, we select the one with the largest positive change in Δ, and for incorrect examples, we select the one with the largest negative change in Δ.

Question	Output	Label	PS	MPS	Δ

What segment did the customer have at 2012/8/23 21:20:00?	`SELECT T1.segment`
	`FROM  customers AS T1`
	`INNER JOIN transactions_1k AS T2`
	`ON T1.customerid=T2.customerid`
	`WHERE  T2.date='2012-08-23'`
	`  AND T2.time='21:20:00'`
		Correct	0.443	0.804	**+ 0.361**

What is the title of the post that is owned by csgillespie and has the highest popularity?	`SELECT title`
	`FROM  posts`
	`WHERE`
	`  ownerdisplayname='csgillespie'`
	`ORDER BY viewcount DESC`
	`LIMIT 1`
		Incorrect	0.686	0.256	**- 0.429**

Table 10: Example SQL outputs from **LLAMA 3.1 70B INSTRUCT** on the **BIRD** dataset, which were selected in the following way: We calculate the difference (Δ) in (calibrated) probability outputted by Platt scaling (PS, *baseline*) and multiplicative Platt scaling (MPS, *ours*). We then split outputs by their label (whether the output is correct or incorrect). For correct examples, we select the one with the largest positive change in Δ, and for incorrect examples, we select the one with the largest negative change in Δ.

B Additional details for deriving parsed frequencies (SCF)

In Figure 2 of the main body, we provide an example of how we parse a SQL query. As noted in the figure, however, we make two simplifications in the example: (1) We assume the SQL query is composed of a single SELECT statement and (2) we only count sub-clauses for `SELECT`, `FROM`, `ON`, `WHERE`, and `GROUP BY`. We now provide the exact details of our parsing procedure.

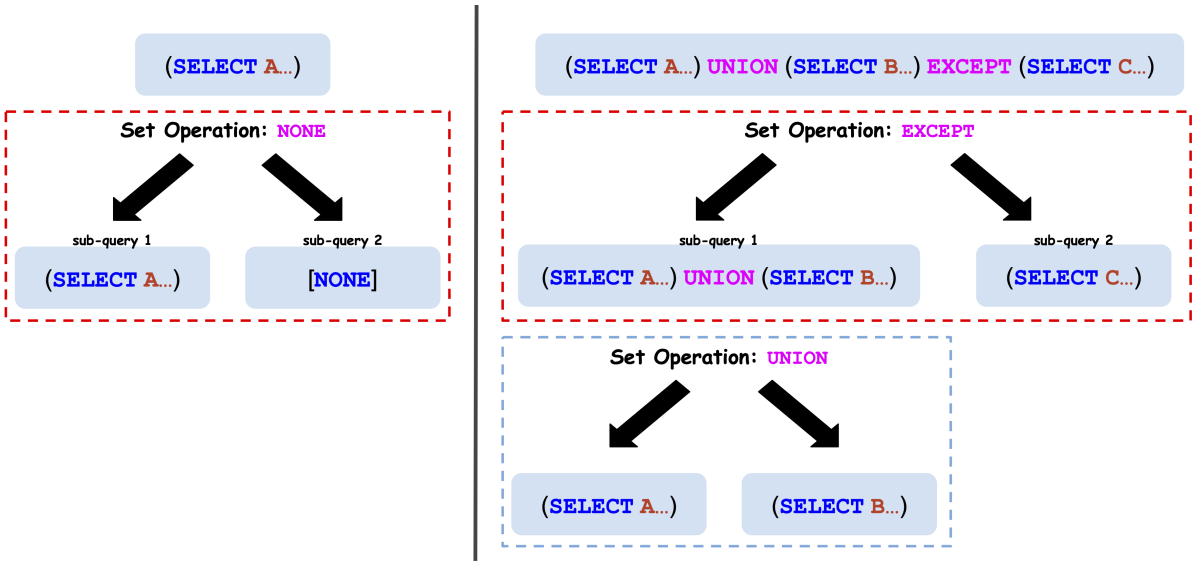


Figure 5: We illustrate two examples how a SQL query is parsed via its tree structure representation. The red box denotes the set operation and two sub-queries that are used for calculating our SCF signals. In the **left** example, we have a SQL statement that is **not** composed of multiple `SELECT` statements. Hence, the set operation is denoted as NONE, and we derive two sub-queries, one of which is the original statement itself (`SELECT A...`) and the other of which is simply [NONE]. In the **right** example, we parse it via the logicical execution order of the query (done automatically via standard SQL parsing libraries), giving us a set operation `EXCEPT` with two sub-queries. We note sub-query 1 can be further parsed into an additional tree (blue box) with the set operation `UNION`.

Queries composed of multiple `SELECT` statements. In our dataset, (two) `SELECT` statements can be composed together using the set operators `UNION`, `INTERSECT`, and `EXCEPT`. In these cases, we then individually parse each `SELECT` statement that is being composed together using the set operator (i.e., if we represent the SQL statement in its tree structure, we then traverse the tree). Note that theoretically, the sub-statements being composed together may be another query that is composed (via another set operator) of multiple sub-statements. Thus, we can recursively traverse the tree structure of the sub-statements until we reach a "leaf", which in our case, is a SQL statement that does not contain any set operators (i.e., a single SELECT statement). We illustrate tree traversal in Figure 5.

Additional sub-clauses not shown in 2. We parse each (single) `SELECT` statement (found at leaves of a SQL output's tree representation (Figure 5)) into the following nine sub-clauses / keywords:

- `DISTINCT`
- `SELECT`
- `FROM`
- `ON`
- `WHERE`
- `GROUP BY`
- `HAVING`
- `ORDERED BY`
- `LIMIT`

If a query does not contain one of the above, we tag it as **`NONE`**. Parsing was done automatically using the SQLGlot library (link).

B.1 Computing the final SCF scores.

Algorithm 1 SCF (sub-clause frequency scorer)

Input: SQL query Q and set of additional queries $\mathbf{Q}_{\text{additional}}$
return $\frac{1}{|\mathbf{Q}_{\text{additional}}|} \sum_{Q' \in \mathbf{Q}_{\text{additional}}} \text{Q-MATCH}(Q, Q')$

Algorithm 2 Q-MATCH (query match function)

Input: SQL queries Q_A and Q_B
Let Q_{A_1} and Q_{A_2} be the subqueries of Q_A {See Figure 5; red box}
Let Q_{B_1} and Q_{B_2} be the subqueries of Q_B
Set $s_{\text{set_op}} = \mathbb{1}\,[\text{SET_OP}(Q_A) = \text{SET_OP}(Q_B)]$
{SET_OP returns the set operation (e.g. `UNION`) of the input query}
Set $\mathbf{s}_1 = [\text{SQ-MATCH}(Q_{A_1}, Q_{B_1}), \text{SQ-MATCH}(Q_{A_2}, Q_{B_2})]$ {Algorithm 3}
Set $\mathbf{s}_2 = [\text{SQ-MATCH}(Q_{A_1}, Q_{B_2}), \text{SQ-MATCH}(Q_{A_2}, Q_{B_1})]$
if $\sum \mathbf{s}_1 \geq \sum \mathbf{s}_1$ **then**
  return $[s_{\text{set_op}}, \mathbf{s}_1]$
else
  return $[s_{\text{set_op}}, \mathbf{s}_2]$
end if

Algorithm 3 SQ-MATCH (sub-query match function)

Input: SQL (sub-)queries Q_1 and Q_2
Let the set of sub-clauses $\mathbf{C}$ = { `DISTINCT`, `SELECT`, `FROM`, `ON`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT` }
for sub-clause $c \in \mathbf{C}$ **do**
  if $Q_1 =$ `NONE` AND $Q_2 =$ `NONE` **then**
    Set $s_c = 1$
  else if $Q_1 =$ `NONE` AND $Q_2 \neq$ `NONE` **then**
    Set $s_c = 0$
  else if $Q_1 \neq$ `NONE` AND $Q_2 =$ `NONE` **then**
    Set $s_c = 0$
  else
    Set $s_c = \text{TRAVERSE_AND_MATCH}(c, Q_1, Q_2)$
    {$\text{TRAVERSE_AND_MATCH}(c, Q_1, Q_2)$ traverses the tree representation (Figure 5; blue box) of Q_1 and Q_2, returning 1 (TRUE) if sub-clause c matches at every corresponding node and 0 (FALSE) otherwise.}
  end if
  return $(s_c)_{c \in \mathbf{C}}$ {Return array of scores for each sub-clause}
end for

We summarize our procedure for calculating SCF scores in the pseudocode found in Algorithms 1, 2, and 3. We also provide more details below:

To compute our sub-clause frequency scores, we must match the sub-clauses of each output to the set of additional outputs generated by the model. To constrain the number of signals s_i in our framework, we only conduct matching (or frequency scoring) at the second level (starting from the root) of the tree (corresponding to the dotted red square in Figure 5), which we denote as sub-queries 1 and 2. This corresponds to checking for the proportion of exact matches on

- (1) the set operation (i.e., NONE, `UNION`, `INTERSECT`, `EXCEPT`) matches

- (9) sub-clauses for sub-query 1
- (9) sub-clauses for sub-query 2

In addition, we add a signal that is the product of the above 19 scores as an additional feature.[6] Thus in total, we have 20 SCF signals s_i, in addition to the log sum token probability outputted by the model directly (i.e., the one used for standard Platt scaling). Note that when using samples from nucleus sampling only or beam search only, we use these 20 (+1) signals. In the case where use both (N + B), we concatenate these signals, giving us 40 (+1) in total.

Matching sub-queries. We note some special cases when comparing two sub-queries, Q_1 and Q_2. These cases can also be found in Algorithm 3.

Cases:

- $Q_1 =$ `NONE` AND $Q_2 =$ `NONE`:
  - return TRUE for all sub-clauses
- $Q_1 =$ `NONE` AND $Q_2 \neq$ `NONE`:
  - return FALSE for all sub-clauses
- $Q_1 \neq$ `NONE` AND $Q_2 =$ `NONE`:
  - return FALSE for all sub-clauses
- $Q_1 \neq$ `NONE` AND $Q_2 \neq$ `NONE`:
  - traverse the tree representation of subqueries Q_1 and Q_2 (Figure 5; blue box), return whether sub-clauses match at every node for each (of the 9) sub-clause types.

Finally, note that as discussed above, each query has two sub-queries Q_{A_1}, Q_{A_2}, Q_{B_1}, and Q_{B_2}. To determine which sets of sub-queries from Q_A and Q_B should go together, we compute the number of matches (over the 9 sub-clauses) for all possible pairings

- (Q_{A_1}, Q_{B_1}) and (Q_{A_2}, Q_{B_2})
- (Q_{A_1}, Q_{B_2}) and (Q_{A_2}, Q_{B_1})

and choose the set of pairings that yield the highest number of exact sub-clause matches (out of $9+9=18$ total). This part of the procedure is described in Algorithm 2.

[6] We found in initial testing that adding this product of scores as an additional feature made results across experiments more consistent. However, we did not further investigate or tune this signal to optimize calibration performance.

C Additional experimental details

C.1 Prompts

In our experiments, we used the following prompts described below. Note that for both prompts, the **[schema]** is serialized in the format: **[table] : [column] , [column], ... | [table] : ... | ...**

C.1.1 SPIDER

T5 3B. The prompt format is as follows: **[question] | [db_id] | [schema]**

LLAMA 3.1 8B INSTRUCT. Using zero-shot prompting, the prompt format is as follows:

```
<|start_header_id|> system <|end_header_id|>
You are a helpful assistant who answers questions about database tables by responding
with SQL queries. Users will provide you with a database id [DB_ID], followed by the
schema of the database.
The schema given by the user will be formatted as the following: TABLE_1 : column_1,
column_2, ... | TABLE_2 : column_1, column_2, ... | ...
After, the user will ask a question. You should respond with a SQL query that can be
executed on the provided database schema to answer the user's question.
Your response should be formatted as: [DB_ID] | [SQL_QUERY]. Your sql query
[SQL_QUERY] should begin with SELECT and end with a semicolon.
<|start_header_id|> user <|end_header_id|>
[question] | [db_id] | [schema]
<|start_header_id|> assistant <|end_header_id|>
```

C.1.2 BIRD

LLAMA 3.1 8B INSTRUCT / 70B INSTRUCT. Using zero-shot prompting, the prompt format is as follows:

```
<|start_header_id|> system <|end_header_id|>
You are a helpful assistant who answers questions about database tables by responding
with SQL queries. Users will provide you with a database id [DB_ID], followed by the
schema of the database.
The schema given by the user will be formatted as the following: TABLE_1 : column_1,
column_2, ... | TABLE_2 : column_1, column_2, ... | ...
After, the user will ask a question. You should respond with a SQL query that can be
executed on the provided database schema to answer the user's question.
Finally, users will provide additional evidence relating the question to the schema.
You may use this information to help you answer the question
Your response should be formatted as: [DB_ID] | [SQL_QUERY]. Your sql query
[SQL_QUERY] should begin with SELECT and end with a semicolon.
<|start_header_id|> user <|end_header_id|>
[question] | [db_id] | [schema] | [evidence]
<|start_header_id|> assistant <|end_header_id|>
```

C.2 Hyperparameters.

For generating SQL outputs using the models we evaluate, we use the Hugging Face's transformers library. To generate outputs using nucleus sampling, we set `top_p=0.95`, `temperature=1.0`, and `num_return_sequences=10`. Using beam search, we set `num_return_sequences=10` and `num_beams=20`. We set `max_new_tokens=512`.

For implementing Platt scaling and multivariate Platt scaling, we train a logistic regression model using scikit-learn with default parameters. We do not conduct any hyperparameter tuning when running these

calibration methods.

C.3 GPU requirements.

For generating SQL outputs using T5 3B PICARD and LLAMA 3.1 8B INSTRUCT, we use a single NVIDIA A100 80GB GPU. For generating outputs using LLAMA 3.1 8B INSTRUCT, we use eight NVIDIA A100 80GB GPUs. Generations (for the calibration and test sets combined) took approximately the following number of GPU hours:

- SPIDER; T5 3B PICARD : 4 hours
- SPIDER; LLAMA 3.1 8B INSTRUCT : 4 hours
- BIRD; LLAMA 3.1 8B INSTRUCT : 4 hours
- BIRD; LLAMA 3.1 70B INSTRUCT 18 (x 8 GPUs) hours

C.4 Licenses

Wikidata and Wikipedia are licensed under the Creative Commons CC0 License. Llama 3.1 is licensed under Meta's Llama 3.1 license. The fine-tuned, T5 3B PICARD (link) checkpoint and Hugging Face's transformers library are licensed under Apache 2.0 license. To help parse SQL queries, we use SQLGlot (link), released under the MIT license. The SPIDER dataset (link) is released under Apache 2.0. The BIRD dataset (link) is released under CC BY-SA 4.0.