Coarse and Fine-grained Confidence Calibration of LLM-based Text-to-SQL Generation

Anonymous ACL submission

Abstract

Calibration plays a crucial role as LLMs are increasingly deployed to convert natural language questions into SQL over commercial databases. In this work, we study the calibration of the confidence attached to both the whole query, and for the first time, to sub-parts of the query. For whole queries, we demonstrate that the simple baseline of deriving confidence from model assigned whole sequence probability yields the best calibration surpassing recent self-check and verbalization methods. For fine-grained calibration, we propose a novel method of assigning confidence to nodes of a logical relational algebra tree representation of the SQL string. We present an extensive comparison spanning two popular Text-to-SQL benchmarks on multiple LLMs, and draw interesting insights about various calibration methods.

1 Introduction

As enterprises attempt to harness LLMs for converting natural language queries over their databases into SQL programs, it is critical for them to obtain well-calibrated probabilities (Steyvers et al., 2024; Baan et al., 2023) for when the generated SQL is incorrect. Recently, several techniques (Xiong et al., 2024; Tian et al., 2023) have been evaluated for calibrating LLMs, including pooling token probabilities (Stengel-Eskin and Van Durme, 2023), prompting LLM to verbally express confidence, self-reflection using True/False questions (Kadavath et al., 2022) and entropy over multiple generated responses (Kuhn et al., 2023). Most of these have been studied over tasks where the response is either a single label or a short string. For the Text-to-SQL task, the generated response is long and structured, and it is unclear if the conclusions of existing studies carry over to this task.

We evaluate several techniques for obtaining well-calibrated confidence for the correctness of the whole SQL. We evaluate on two prevalent Text-to-SQL benchmarks Spider and BIRD over predicted SQLs obtained using two models GPT-4, a proprietary model and CodeS, an open source state of the art model finetuned for the Text-to-SQL task. Our study brings out two conclusions different from prior calibration studies: (1) We show that the simple baseline of deriving confidence from the model assigned whole sequence probability yields the best calibration. In earlier work on calibration of QA tasks (Tian et al., 2023) verbalized scores from follow-up questions was shown to provide up to 50% better calibration. One reason could be that the SQL output is significantly more complicated than short answers in QA and classification tasks, and the model struggled to reason on correctness of a whole SQL. These conclusions are in alignment with the paradox highlighted in recent work on the gap between the generative and evaluation capacity of modern LLMs (West et al., 2024; Oh et al., 2024). Prior calibration studies on simpler tasks seemed to have not hit that boundary. Second, unlike previous work (Stengel-Eskin and Van Durme, 2023) which proposed the minimum of token probabilities for whole-sequence calibration, we found the product of probabilities, which is also the model assigned whole sequence probability, to perform significantly better.

For long responses, a user may find it useful to obtain fine-grained confidences over various parts of the generated outputs, instead of a single confidence score over the entire output. In Table 1 we show an example to motivate the usefulness of fine-grained calibration of an SQL. For fine-grained calibration of logical outputs like SQL, one important design decision is choosing the unit at which confidence is measured. Token-level confidence is not meaningful for SQL, even though the LLM generates the SQL as a string token-by-token. One reason is that there are many different equivalent ways of expressing the same logic as an SQL string. We propose an alternative design where we convert

Natural language question: Which gas station has the highest amount of revenue?
Gold SQL: select transactions_1k.gasstationid from transactions_1k group by transactions_1k.gasstationid order by sum(transactions_1k.price) desc limit 1
Predicted SQL: select transactions_1k.gasstationid from transactions_1k group by transactions_1k.gasstationid order by sum(transactions_1k.amount) desc limit 1

Table 1: An example query where the predicted SQL is wrong only in one column (marked in red). A calibration method that could assign low confidence to this wrong output could be more useful than assigning low confidence to the entire predicted SQL (Schema not shown).

the SQL into the implied relational algebra tree (RAT). We then measure confidence in units of a subtree of the RAT. We train a separate Confidence model that assigns to each node a probability of the subtree below it being correct. For training the model we collect predicted SQLs from diverse LLMs and also generate perturbations of the gold SQL to introduce synthetic errors. Our evaluation on the test set shows that the error model that we trained is significantly better calibrated than the calibration of whole SQL. Further, not surprisingly, node level calibration provides much better agreement with ground truth label compared to token level calibration.

Our main contributions are as follows: (1) We compare calibration of several methods of attaching confidence to SQL generated from state-of-the-art LLMs. (2) We introduce the problem of fine grained calibration for the Text-to-SQL task and propose a novel method of attaching fine grained confidence course in units of subtrees in the relational algebra tree (RAT) corresponding to the predicted SQL. (3) We design a Confidence model to attach fine-grained confidence scores to nodes of the RAT. (4) We present extensive comparison of both existing methods and our proposed methods for both whole SQL and fine-grained confidence on two popular benchmarks for Text-to-SQL generation and over predictions from two different LLMs.

2 Related Work

Calibration of classification models is a classical ML topic (Niculescu-Mizil and Caruana, 2005; Guo et al., 2017a), with much work in pre-LLM NLP literature (Kumar and Sarawagi, 2019; Desai and Durrett, 2020). We focus on recent work on calibration of tasks on LLMs .

Calibration of LLMs for short response generation Kadavath et al. (2022) study LLMs on a variety of tasks and propose to extract confidence by a self-probe using a follow up True/False question to the LLM on whether the generated response was correct. Probability of True in the follow up question is measured as confidence. Tian et al. (2023) further expand the set of prompts asking to verbalize confidence and show that a better strategy for calibration is getting the LLM to generate top-K responses with probabilities. Ren et al. (2023) also show that self-evaluation improves calibration. Zhou et al. (2023) study if language markers like: "I believe", "I am sure.."etc reflect confidence, and show that these language markers do not faithfully reflect uncertainty. Kuhn et al. (2023) proposes to generate multiple answers with LLM assigned confidence for each, cluster them based on semantic similarity, measures entropy over the total confidence across the clusters. Xiong et al. (2024) also studies these techniques and additionally introduces PairRank that scores based on the ranking of responses across multiple Top-K sets.

Uncertainty for Semantic Parsing and SQL. Stengel-Eskin and Van Durme (2023) reports lack of calibration of Text-to-SQL systems and measure confidence as the minimum token probability over tokens in the entire predicted SQL sequence. Another related topics is measuring how well semantic parsing models represent ambiguity in the input by, for example, outputting both ambiguous logical forms in the top-k output (Stengel-Eskin et al., 2024) and (Bhaskar et al., 2023).

Fine-grained Quality Estimation. In the pre-LLM era, one area of focus in the machine translation community was assigning word-level quality metrics (Lommel et al., 2014) to translations. The techniques deployed range from training special models to score words by synthetically inserting errors in a gold parallel dataset (Zhou et al., 2021; Tuan et al., 2021) and reasoning on likelihood obtained from the original model on various perturbations of the source or target based on the error type Vamvas and Sennrich (2022); Jain et al. (2022). They focus on insertion and omission errors of words in the source and target sentences, whereas for semantic parsing we propose a more logical definition of error in terms of mismatch of operators. Huang et al. (2024) extends above for long form generation, example as in summarization. They do not assign fine-grained confidence, and their main focus is obtaining a distribution over confidence over the entire long form generation.

3 Whole Query Calibration

Let $\mathbf{x}_i$ be an input natural language question on a database schema $\mathbf{s}$ for which a Text-to-SQL model $\mathcal{M}$ predicted an output SQL $\hat{\mathbf{y}}_i$. We explore a number of methods of attaching a score $r(\hat{\mathbf{y}})$ that indicates if $\hat{\mathbf{y}}_i$ is a correct SQL for $\mathbf{x}$. The LLM prompts used for each of these methods appear in Tables 6,7 and 8 of the Appendix.

Pooled Token-level Probabilities. The generative model $\mathcal{M}$ assigns a probability $P(\hat{\mathbf{y}}|\mathbf{x})$ composed out of auto-regressive token probabilities $\Pr(\hat{y}_t|\mathbf{x}, \hat{\mathbf{y}}_{<t})$. A natural method is to use these token probabilities for calibration. Let n denote the number of tokens in $\hat{\mathbf{y}}$. These can be converted into a confidence score r for the whole query $\hat{\mathbf{y}}$ by pooling the token probabilities in various ways:

1. product of probability $\prod_t^n \Pr(\hat{y}_t|\mathbf{x}, \hat{\mathbf{y}}_{<t})$ **[prod]**
2. geometric mean $\sqrt[n]{\prod_t^n \Pr(\hat{y}_t|\mathbf{x}, \hat{\mathbf{y}}_{<t})}$ **[geo]**
3. minimum $\min_{t\in[n]} \Pr(\hat{y}_t|\mathbf{x}, \hat{\mathbf{y}}_{<t})$ **[min]**
4. arithmetic mean $\frac{1}{n}\sum_{t=1}^n \Pr(\hat{y}_t|\mathbf{x}, \hat{\mathbf{y}}_{<t})$ **[avg]**

LLM Self-checks generated SQL. Another emerging trend is asking the LLM to self-reflect on the correctness of the generated SQL. This can be in the form of a True/False answer **[Bool]**, where confidence is measured as $r(\hat{\mathbf{y}}) = P(\text{True}|\hat{\mathbf{y}}, \mathbf{x})$ (Kadavath et al., 2022) or where the LLM is asked to directly output the probability of the SQL being correct **[Probs]**, measuring confidence as $r(\hat{\mathbf{y}}) = \mathcal{M}(\hat{\mathbf{y}}, \mathbf{x})$ (Tian et al., 2023).

Relative score with Variant output SQLs. Given the huge difference in the level of difficulty of SQL generation across different questions and schema, it may be difficult to obtain comparable scores across different instances. Relative scores across alternative SQLs may be more meaningful. Accordingly, we designed this method: First prompt the model $\mathcal{M}$ to generate multiple structurally diverse SQLs. Denote alternative plausible SQLs $\mathcal{Y}_\mathbf{x}$ for $\mathbf{x}$. Out of these we eliminate those SQLs that are semantically equivalent to $\hat{\mathbf{y}}$ based on whether $\text{Acc}(\hat{\mathbf{y}}, \mathbf{y}')$ is the same for each $\mathbf{y}' \in \mathcal{Y}_\mathbf{x}$. Then measure the difference in score of the predicted SQL and the best alternative SQL, that is $r_{\text{ALT}} = r(\hat{\mathbf{y}}) - \max_{(\hat{\mathbf{y}}')\in\mathcal{Y}_\mathbf{x}:\text{Acc}(\hat{\mathbf{y}},\hat{\mathbf{y}}')=0} r(\hat{\mathbf{y}}')$. Other measures could be entropy in scores of the alternatives as proposed here (Kuhn et al., 2023).

4 Fine-grained Confidence

Whole query calibration does not allow for fine-grained error-identification. A single score for a whole SQL is not useful to identify what part is likely incorrect. A fine-grained confidence calibrator could be useful to draw a user's attention to specific parts of the generated SQL (see example in Table 1) that the model is uncertain about.

4.1 Baseline Token-level method

A baseline for fine-grained calibration is to just assign token-level confidence derived from the forward probability assigned by the LLM during auto-regressive generation. We list the limitations of token-level confidence for the Text-to-SQL task, and then present our approach.

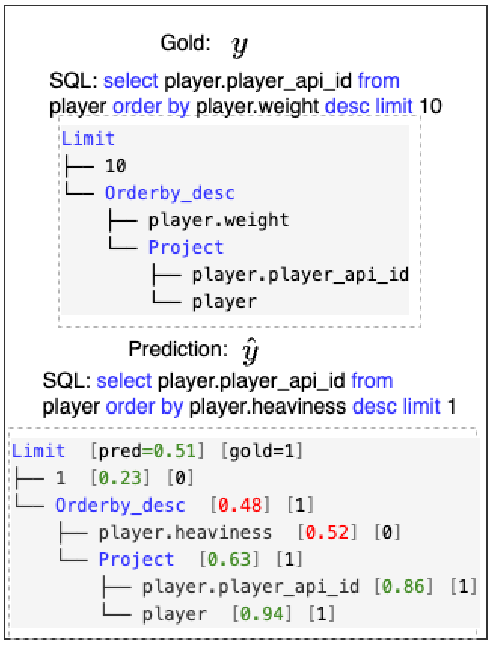
Limitation of Token-level confidence. SQL is a structured language, and token-level confidence may provide inconsistent scores. For example, if a generated SQL has chosen a wrong table, the LLM may assign arbitrarily different confidence values to different tokens of the table name. Further, the SQL language is declarative and the order in which token probabilities are assigned during auto-regressive generation, could fail to capture consistency errors across different parts of the SQL. For example, one common source of hallucination is using column names in the `select` clause that are not part of the tables mentioned in the `from` clause. A model with bidirectional attention has a better chance at reasoning about such inconsistencies. Another challenge with token-level calibration is that there are different ways of expressing the same underlying computation logic as an SQL string. For example, these two SQLs are equivalent.

```
SELECT T1.c1 AS col1 FROM tab1 T1 WHERE T1.c2 > 10
SELECT c1 FROM tab1 WHERE tab1.c2 > 10
```

In general, identifying isomorphisms of two SQLs is undecidable (Abiteboul et al., 1995). While heuristics exist for whole query equivalence (Zhao et al., 2024), fine-grained calibration entails a much harder task of assigning correctness labels to individual tokens.

4.2 Proposed: Confidence to Nodes of Relational Algebra Tree

We propose to reason about fine-grained calibration in terms of a logical Relational Algebra Tree (RAT) representation of the SQL, rather than the SQL


Gold: y

SQL: select player.player_api_id from player order by player.weight desc limit 10

```
Limit
├── 10
└── Orderby_desc
    ├── player.weight
    └── Project
        ├── player.player_api_id
        └── player
```

Prediction: $\hat{y}$

SQL: select player.player_api_id from player order by player.heaviness desc limit 1

```
Limit  [pred=0.51] [gold=1]
├── 1  [0.23] [0]
└── Orderby_desc  [0.48] [1]
    ├── player.heaviness  [0.52] [0]
    └── Project  [0.63] [1]
        ├── player.player_api_id [0.86] [1]
        └── player  [0.94] [1]
```

Figure 1: Relational algebra tree of Gold ($\mathbf{y}$) and Predicted SQL ($\hat{\mathbf{y}}$). With each node of $\hat{\mathbf{y}}$ are attached: (1) the predicted confidence and (2) the 0/1 correctness label of if the subtree underneath appears in Gold $\mathbf{y}$. The Green and Red denotes if confidence scores are accurate or not. **Postorder traversal** of $\hat{\mathbf{y}}$ is **((1)((player.heaviness)((player.player_api_id)(player) Project)Orderby_desc)Limit)**.

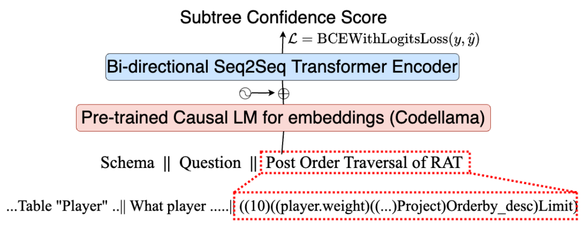

Subtree Confidence Score
$\mathcal{L}$ = BCEWithLogitsLoss($y, \hat{y}$)
Bi-directional Seq2Seq Transformer Encoder
Pre-trained Causal LM for embeddings (Codellama)
Schema || Question || Post Order Traversal of RAT
...Table "Player" ..|| What player .....|| ((10)((player.weight)((...)Project)Orderby_desc)Limit)

Figure 2: Architecture of the Confidence model for fine-grained calibration.

string. Figure 1 presents an example of a RAT for an SQL string. Each node is a either a full schema name or an relational operator or a string literal clearly marked as such in the tree. Our goal is to assign a score to each node of the RAT to denote the correctness of the subtree rooted at that node.

SQL to RAT. We pre-process the SQL string to standardize column names, convert SQL keywords and schema items except literals to lowercase, removing extra white spaces etc We then convert the canonicalized SQL to RAT using a relational algebra grammar as in (Rubin and Berant, 2021).

Node-level Confidence Model. We train a transformer-based confidence model $\mathcal{E}$ for fine grained calibration. The input to the confidence model $\mathcal{E}$ is a concatenation of the Schema $\mathbf{s}$, Question $\mathbf{x}$, and the predicted SQL $\hat{\mathbf{y}}$ converted into its RAT $\hat{\mathbf{t}}$. The RAT is converted into a token-sequence using a post-order traversal. An example of a post-order serialized RAT is shown in caption of Figure 1. We tokenize and encode the schema, question, and RAT with a pre-trained LLM like CodeLlama that employs causal attention. Since we traverse the RAT nodes in a post-order manner, each node gets contextualized with respect to all nodes under it in the subtree, along with the question $\mathbf{x}$ and schema $\mathbf{s}$. To these we add positional encodings, which are also learnable parameters. Then we apply multiple *bidirectional* attention layers on the encoded input. Finally, a linear layer is used at the *last token* of each RAT node n to get the output confidence $r(n, \hat{\mathbf{t}})$ that the sub-tree at the node is correct. Figure 2 presents an overview of our architecture.

Training of Confidence model. We train the model using SQL predicted from multiple LLMs. We also create perturbations of training data by replacing schema items with other schema items from the same database based on Cosine similarity of embeddings of schema items. To train the model to output correct confidence $r(n, \hat{\mathbf{t}})$ of a node n in a predicted RAT $\hat{\mathbf{t}}$ we use a cross-entropy loss on gold 0/1 correctness label $a(n, \hat{\mathbf{t}})$. We determine $a(n, \hat{\mathbf{t}})$ label based on whether subtree rooted at n appears in the gold RAT. The node matches have to be defined carefully because the same logical operation can be expressed in several isomorphic forms. First of, whenever a predicted SQL execution result matches those of the gold SQL, all nodes of the predicted RAT are labeled correct. Otherwise, we assign node-wise matching based scores as follows: Nodes are matched using hashing. Hashes are calculated for each node in a recursive manner where hash of a node is calculated based on hash of its children. For commutative operators children of a node are sorted before hashing . To handle permutation-invariance of table names in multi-way joins we perform the sorting across multiple levels of nodes in corresponding join nodes. In general a node should be labeled correct even if its parents are incorrect. For example consider subtrees `c > 10` and `c = 10` , here `c` and `10` should be marked as correct even if operator is wrong. But, to prevent incidental misaligned matches, we include certain parents e.g. `Project, Order-by` as part of the hash. We present an example of gold and predicted RAT along with assigned node correctness labels in Figure 1.

5 Experiments

We compare the whole query and fine-grained calibration methods discussed so far across different datasets and LLMs.

5.1 Datasets

We evaluate on 31 database schemas spanning two popular Text-to-SQL benchmarks Spider (Yu et al., 2019) and BIRD (Li et al., 2023) for natural language utterances $\mathbf{x}$ and their gold SQL $\mathbf{y}$. For each of these, we measure calibration of predictions obtained from two different LLMs GPT-4 (OpenAI et al., 2024)('gpt-3.5-turbo-16k') and CodeS (Li et al., 2024). The prompts used for SQL generation is provided in Table 6. Although these models do not guarantee syntactically valid SQL generation, we assume that a DB engine can be easily called to check for grammatical validity and eliminate invalid generations. For fine-grained experiments, we also filter away queries for which the library we used for generating relational algebra tree fails. The final statistics of our test data appear in Table 2.

	Spider		BIRD	
	GPT4	CodeS	GPT4	CodeS
Total Queries	1034		1534	
# databases	20		11	
% Correct	77.6 %	59.1 %	43.1 %	19.6 %

Table 2: Summary of datasets used for calibration.

5.2 Metrics

Each data sample is comprised of five parts: $(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}}_i, a_i, r_i)$ where $\mathbf{x}_i$ is natural language question, $\mathbf{y}_i$ is gold SQL, $\hat{\mathbf{y}}_i$ is predicted SQL, $a_i = \text{Acc}(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ is the 0/1 label indicating whether the predicted SQL $\hat{\mathbf{y}}$ produces identical execution results as the gold SQL $\mathbf{y}$, disregarding the order of columns or rows in the result set, and r_i is the confidence value returned by a method evaluated. The raw confidence scores returned by most methods often need to be monotonically transformed for recalibration. Many methods have proposed to use a small validation dataset to calibrate the raw scores (Niculescu-Mizil and Caruana, 2005; Guo et al., 2017a). We consider two options (1) Platt scaling **(P)** (Platt, 2000), where r_i is sigmoid scaled with two parameters temperate t and bias b, to maximize the likelihood of given a_i under a model $\sigma(tr_i + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$, and (2) Isotonic regression **(I)** (Zadrozny and Elkan, 2002) which adjusts r_i so that $\sum_i (a_i - T(r_i))$ is minimized, where T is a step-wise constant isotonic (non-decreasing) function. We denote the calibrated confidence score as r_i^T. The metrics are calculated using five-fold cross validation. For whole query experiments, we divide the dataset into five schema-disjoint splits. In each fold, one split is used for tuning parameters of calibration while the remaining 4 splits are used for evaluating the metrics.

We compare the different methods using their reliability plots and several calibration measures, as detailed below.

Reliability plots. The data samples are grouped into several bins based on their confidence scores. For each bin, the average confidence is plotted against the observed accuracy, which is the proportion of samples in the bin with $a_i = 1$. Generally, all bins have a fixed size (**Uniform Binning**). We also try the Constrained Pool Adjacent Violators Algorithm (Matsubara et al., 2023) **(Monotonic Binning)** which decides the binning such that the average difference between the average observed accuracy and confidence, weighted by the number of samples in each bin is minimized.

Calibration measures. We measure the Brier score (Brier, 1950) **(BS-P/BS-I)**, which is the mean square difference between calibrated confidence and correctness: $\frac{1}{n}\sum_i^n (a_i - r_i^T)^2$. We also assess the discriminative power of the methods using **AUC** (Geifman and El-Yaniv, 2017). Finally, we measure the expected calibration error (Guo et al., 2017b) for both raw **(ECE)** and calibrated **(ECE-P/ECE-I)** confidence scores. ECE is the mean absolute difference between the average observed accuracy and confidence, weighted by the number of samples in each bin: $\sum_{j=1}^{B} \frac{|B_j|}{n} \left| \frac{1}{|B_j|} \sum_{i \in B_j} (a_i - c_i) \right|$, where c_i is either r_i or r_i^T and the bins B are determined either using uniform binning or monotonic binning method. However, we find that ECE can be unreliable, as it may assign a better score to a classifier that always predicts a 50 % confidence level on a dataset where the labels evenly distributed.

5.3 Whole query methods

We present comparison of all the methods in Section 3 of obtaining whole query confidence.

Method		Spider				BIRD			
		BS-P↓	AUC↑	ECE↓	ECE-P↓	BS-P↓	AUC↑	ECE↓	ECE-P↓
Pooled token-level (CodeS)	min	0.221	0.636	0.669	0.120	0.205	0.651	0.293	0.070
	avg	0.233	0.541	0.097	0.135	0.213	0.618	0.465	0.065
	prod	0.194	0.746	0.685	0.112	0.193	0.730	0.314	0.082
	geo	0.234	0.539	0.216	0.130	0.213	0.631	0.226	0.069
Pooled token-level (Codestral)	min	0.215	0.663	0.662	0.114	0.202	0.670	0.266	0.076
	avg	0.223	0.606	0.133	0.126	0.198	0.705	0.526	0.067
	prod	0.172	0.788	0.678	0.098	0.188	0.757	0.305	0.075
	geo	0.228	0.598	0.228	0.133	0.202	0.694	0.253	0.079
Self-check Bool (GPT-4)		0.208	0.701	0.207	0.120	0.203	0.707	0.538	0.071
Self-check Bool (CodeLlama)		0.229	0.600	0.076	0.132	0.217	0.621	0.491	0.090
Self-check Probs (GPT-4)		0.223	0.598	0.269	0.130	0.216	0.584	0.627	0.063
Variant SQLs (Prod) (CodeS)		0.200	0.747	0.684	0.110	0.207	0.701	0.314	0.094

Table 3: The table presents evaluation metrics for different whole query methods on the Spider and BIRD datasets. The metrics include Platt-scaled Brier score **(BS-P)**, area under the ROC curve **(AUC)**, expected calibration error **(ECE)** and Platt-scaled ECE **(ECE-P)**. Uniform binning is used to calculate ECE and ECE-P. Highlighted numbers in green and yellow denote the best and second best methods, respectively.

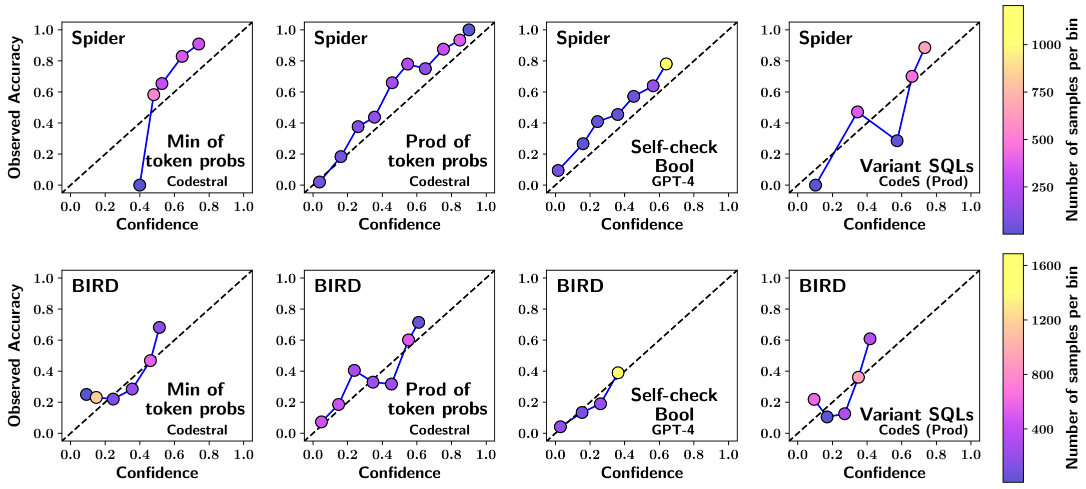


Figure 3: The reliability plots illustrate the calibration comparison between the different whole query methods. The four plots on top have been generated with predictions corresponding to the Spider dataset and four plots below, with the BIRD dataset. A well-calibrated plot aligns closely with the x=y line. Each point is color-coded based on the number of samples in the bin, as indicated by the colorbar on the right.

Evaluation Protocol. We choose two open-source models, CodeS and Codestral[1] for pooled token-level experiments since we need access to the token probabilities. CodeS is specifically trained for Text-to-SQL generation tasks, while Codestral has demonstrated superior performance in Text-to-SQL tasks compared to popular open-source models such as Llama-3[2] and CodeLlama (Rozière et al., 2024). Given the context and predicted SQL, we collect the probabilities assigned by the models to each token of the predicted SQL.

We utilize GPT-4 and CodeLlama for our self-check experiments due to their reasoning capabilities and ability in understanding self-check questions. For the self-check Bool method, given the context, the predicted SQL and two options (A: SQL is correct, B: SQL is incorrect), we collect and normalize the probabilities assigned to tokens 'A' and 'B'. The normalized probability of token 'A' is used for calibration. Tian et al. (2023) demonstrated that verbalizing confidences provides better calibration than the model's conditional probabilities in question answering tasks. To test if this holds in SQL generation, we conduct the self-check Probs experiment. Here, given the context and the predicted SQL, the model is asked to estimate the probability that the SQL is correct. This verbalized

[1]https://mistral.ai/news/codestral/
[2]https://ai.meta.com/blog/meta-llama-3/

probability is used for calibration.

To generate variant output SQLs, we prompt GPT-4 to produce 10 diverse SQLs given the context. For each generated SQL, we calculate the prod pooled token-level confidence score using CodeS.

The prompts used for the experiments are presented in Table 6, 7 and 8. Specific inference details are deferred to the Appendix.

Results. Table 3 shows the results for the whole query methods. Among pooled token-level approaches, we obtain the best calibration with the **prod** aggregation method in terms of the AUC and Brier scores, followed by **min**, with **avg** and **geo** providing poor calibration. These findings corroborate early findings (Stengel-Eskin and Van Durme, 2023) favoring **min** aggregation over **mean**, with our experiments highlighting **prod** as a significantly better alternative than **min**. Also note that **prod** is theoretically the most natural choice since it denotes the probability of generating the whole sequence in an auto-regressive model, and it is surprising that Stengel-Eskin and Van Durme (2023) only considered **min** and **avg**.

Another interesting conclusion is that self-check methods are not better than model's own sequence probability (**prod**). This aligns with recent research (West et al., 2024) highlighting of the gap between the generative and reasoning capabilities in large language models. The product of token probabilities, which is the likelihood of the whole-sequence, serves as a measure of its generative capability, contrasting with self-check which is an assessment of the model's understanding. However, recent calibration studies (Tian et al., 2023) have found self-check methods to be better, and that could be because they deal with short answers. Further, we observe that the calibration of Self-check-Prob approach is weaker than Self-check-Bool. These results are are contrary to those of Tian et al. (2023) evaluated on QA tasks.

The self-check Bool calibration of the proprietary model GPT-4 is stronger than the open source model CodeLlama. CodeS, a smaller 7 Billion parameter model but which is specifically trained for SQL generation has a weaker calibration than Codestral, a larger 220 billion parameter trained for generalized code completion. The calibration of variant SQLs approach falls between the pooled token-level and self-check approaches.

The reliability plots in Figure 3 illustrate the calibration comparison between the different methods of whole query evaluation. A well-calibrated plot aligns closely with the x=y line. Here again we observe that **Prod** of token probabilities is the best calibrated method with well-spread out confidence values. Reliability plots for experiments using other models and comparisons with isotonic regression and monotonic binning have been deferred to the Appendix E.1.

5.4 Fine-grained calibration

We compare our method of fine-grained calibration in units of nodes of the relational algebra tree (RAT) described in Section 4.2 with the baseline method where calibration is in units of tokens. For the baseline, we assigned gold 0/1 labels to tokens of predicted SQL as follows: We use Needleman–Wunsch algorithm, a global alignment technique between the Gold and predicted SQL after standardizing them both. The unit of comparison considered for alignment is tokens as obtained from the same LLM that we use to obtain token level probabilities for calibration. Post alignment at each position we check whether gold and predicted SQL token match or not to assign 0/1 labels. We use CodeS and Codestral to obtain token level probability of tokens in predicted SQL as used in the Whole query calibration model.

Details of Node-level Confidence Model. We implement the fine-grained Confidence model (Figure 2 as follows: We use CodeLlama to get the first layer token embeddings. These are input to the trainable bi-directional Transformer comprising of four encoder layers and eight attention heads. Output from this layer is passed through a linear layer and Sigmoid to get predicted confidence. Maximum sequence length is 2048 tokens. If an input goes beyond this, we prune the database schema schema to retain only items in the predicted SQL along with some random tables and columns. We train the model using predictions on the training split of the Spider and BIRD database using GPT-4 and CodeS-7B. The total number of training instances is 13,079. We augment t by perturbing the gold SQLs by replacing schema items with other schema items from the same database based on Cosine similarity of embeddings of schema items obtained from SentenceTransformer all-MiniLM-L6-v2.

Results. Table 4 shows the results. We find that though the baseline report comparable Platt-scaled Brier and ECE score, our method provides much

Model	Spider				BIRD			
	BS-P↓	AUC↑	ECE↓	ECE-P↓	BS-P↓	AUC↑	ECE↓	ECE-P↓
Baseline(CodeS)	0.14	0.59	0.17	0.03	0.18	0.56	0.21	0.04
Baseline(Codestral)	0.12	0.56	0.19	0.02	0.17	0.55	0.24	0.04
RAT Confidence	0.15	0.76	0.25	0.05	0.20	0.76	0.28	0.10

Table 4: Fine-grained calibration metrics for our RAT node-level Confidence model along with token-level baselines on the Spider and BIRD datasets. The metrics include Platt-scaled Brier score **(BS-P)**, area under the ROC curve **(AUC)**, expected calibration error **(ECE)** and Platt-scaled ECE **(ECE-P)**. Uniform binning is used to calculate ECE and ECE-P. Highlighted numbers in green and yellow denote the best and second best methods, respectively.

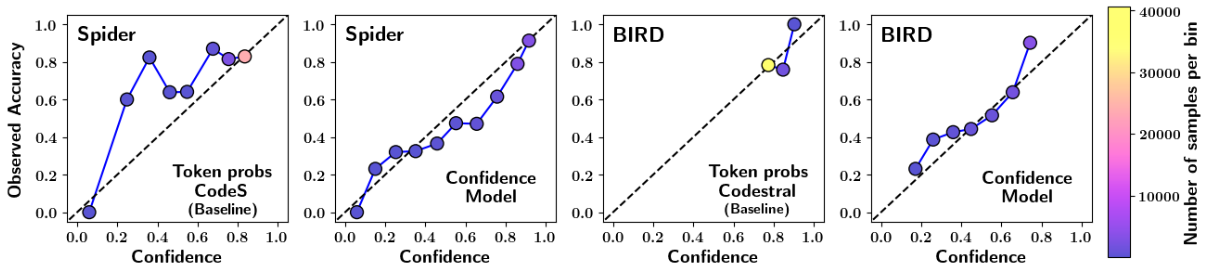


Figure 4: The reliability plots illustrate the fine-grained calibration comparison between the baseline and our Confidence model. Plots 1 and 3 correspond to baseline obtained on Spider and BIRD using CodeS and Codestral models respectively before alignment. Plots 2 and 4 demonstrate the confidence model's performance on Spider and BIRD data. Note the significantly better calibration of our RAT node based calibration than baseline token-level calibration. A well-calibrated plot aligns closely with the x=y line.

better AUC scores. Also, by taking into account the calibration plot as observed in Figure 4, both plots 1 and 3, show very poor calibration with irregular distribution of data points across bins compared to our method. Some anecdotes of fine-grained calibration from our model can be found in Table tab:example of the Appendix. We present ablations on our Confidence model in the Appendix.

6 Conclusion

We study calibration of whole SQL and parts of a generated SQL for LLM based Text-to-SQL generation. For Whole SQL, we compare with several recent calibration methods and draw interesting insights. We find that models show strong calibration when assigning probabilities to whole queries, outperforming verbalization methods. This confirms recent research highlighting differences in generative and reasoning capabilities of LLMs Additionally, using product aggregation for calibration-model assigned probability to the whole query-provides stronger calibration compared to other methods including minimum aggregation, which was proposed as better by earlier works. When verbalizing probabilities, prompting the model to output True/False is better than directly outputting the probability of the SQL correctness. This differs from previous findings in QA tasks.

To the best of our knowledge, no prior work has studied fine-grained calibration of generated SQLs. We propose a formulation where calibration is in units of nodes of a relational algebra tree rendition of the predicted tree. We present the design of a custom model for node-level confidence prediction.

This study's insights into model calibration for Text-to-SQL generation can be extended to broader applications such as Python or C++ code generation completion tasks. The study's analysis of the generative and reasoning capability of LLMs is also crucial for the design of better LLMs. Assigning confidence to text or code completions is an important and exciting area of research with potential to fasten the adaptation of LLMs and improve efficiency in various domains.

7 Limitations

Our results are based on the specific models employed in our experiments. Although, we have attempted to ensure the validity of our findings by utilizing different models for each method, we cannot guarantee these results will generalize to other models. This limitation is due to the lack of detailed technical information such as training methodologies for many of the models used.

Additionally, this limitation restricts our ability to fully explain why the calibration of self-check Bool method is weaker compared to the prod pooled token-level method. Furthermore, our study is restricted to identifying the best calibration methods for generated SQLs, particularly those whose complexity is similar to the SQLs found in the Spider and BIRD datasets.

For error model, we are constrained by the hashing mechanism and as such currently cannot handle isomorphic queries where related schema items may not be in close proximity. For example, a SQL with join across 5 tables should ideally match with any permutation of subtree with these 5 tables under join operation. Also, in subtree level calibration as we move to upper layers, hashes due to incorrect node in lower layers lead to poor calibration for nodes close to root.

One potential risk associated with our research is the imperfection of the calibration process. Due to this, the model cannot be applied directly in real world applications with absolute accuracy. The confidence scores predicted by the model should be taken as preliminary assessments. Hence, human evaluation is necessary after the models flag certain instances, ensuring a more reliable decision.

References

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. ddison-Wesley Reading.

Joris Baan, Nico Daheim, Evgenia Ilia, Dennis Ulmer, Haau-Sing Li, R. Fernández, Barbara Plank, Rico Sennrich, Chrysoula Zerva, and Wilker Aziz. 2023. Uncertainty in natural language generation: From theory to applications. *ArXiv*, abs/2307.15703.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-SQL generation under ambiguity. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Glenn W. Brier. 1950. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1):1.

Shrey Desai and Greg Durrett. 2020. Calibration of pre-trained transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 295–302, Online. Association for Computational Linguistics.

Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017a. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1321–1330.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017b. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

Yukun Huang, Yixin Liu, Raghuveer Thirukovalluru, Arman Cohan, and Bhuwan Dhingra. 2024. Calibrating long-form generations from large language models. *ArXiv*, abs/2402.06544.

Priyesh Jain, Sunita Sarawagi, and Tushar Tomar. 2022. Quality scoring of source words in neural translation models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10683–10691, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 2022. Language models (mostly) know what they know.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*.

Aviral Kumar and Sunita Sarawagi. 2019. Calibration of encoder decoder models for neural machine translation.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls.

Arle Lommel, Aljoscha Burchardt, and Hans Uszkoreit. 2014. Multidimensional quality metrics (mqm): A framework for declaring and describing translation quality metrics. *Tradumàtica: tecnologies de la traducció*, 0:455–463.

Takuo Matsubara, Niek Tax, Richard Mudd, and Ido Guy. 2023. Tce: A test-based approach to measuring calibration error.

Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *ICML*.

Juhyun Oh, Eunsu Kim, Inha Cha, and Alice Oh. 2024. The generative ai paradox on evaluation: What it can solve, it may not evaluate.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report.

John Platt. 2000. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10.

Jie Ren, Yao Zhao, Tu Vu, Peter J. Liu, and Balaji Lakshminarayanan. 2023. Self-evaluation improves selective generation in large language models.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Pro-*

ceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 311–324, Online. Association for Computational Linguistics.

Elias Stengel-Eskin, Kyle Rawlins, and Benjamin Van Durme. 2024. Zero and few-shot semantic parsing with ambiguous inputs. In *The Twelfth International Conference on Learning Representations*.

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

Mark Steyvers, Heliodoro Tejeda Lemus, Aakriti Kumar, Catarina Belem, Sheer Karny, Xinyue Hu, Lukas Mayer, and Padhraic Smyth. 2024. The calibration gap between model and human confidence in large language models. *ArXiv*, abs/2401.13835.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Yi-Lin Tuan, Ahmed El-Kishky, Adithya Renduchintala, Vishrav Chaudhary, Francisco Guzmán, and Lucia Specia. 2021. Quality estimation without human-labeled data. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 619–625, Online. Association for Computational Linguistics.

Jannis Vamvas and Rico Sennrich. 2022. As little as possible, as much as necessary: Detecting over- and undertranslations with contrastive conditioning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 490–500, Dublin, Ireland. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql.

Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D. Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2024. The generative AI paradox: "what it can create, it may not understand". In *The Twelfth International Conference on Learning Representations*.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. 2024. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

Bianca Zadrozny and Charles Elkan. 2002. Transforming classifier scores into accurate multiclass probability estimates. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Fuheng Zhao, Lawrence Lim, Ishtiyaque Ahmad, Divyakant Agrawal, and Amr El Abbadi. 2024. Llm-sql-solver: Can llms determine sql equivalence?

Chunting Zhou, Graham Neubig, Jiatao Gu, Mona Diab, Francisco Guzmán, Luke Zettlemoyer, and Marjan Ghazvininejad. 2021. Detecting hallucinated content in conditional neural sequence generation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1393–1404, Online. Association for Computational Linguistics.

Kaitlyn Zhou, Dan Jurafsky, and Tatsunori Hashimoto. 2023. Navigating the grey area: How expressions of uncertainty and overconfidence affect language models. In *Conference on Empirical Methods in Natural Language Processing*.

A License

The Spider and BIRD datasets are distributed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) license. We used code from the **codes** Github repository[3] released by (Li et al., 2024), which is distributed under the Apache-2.0 license. Additionally, we referred to the prompts and execution evaluation scripts from the MAC-SQL Github repository[4] released by (Wang et al., 2024); however it's license could not be found. The CodeS models are also distributed under the Apache-2.0 license. We used the CodeLlama model in accordance with the Llama 2 community license agreement[5]. The Codestral model was used in compliance with the Mistral AI Non-Production License[6]. For inference with GPT-4, we use the paid OpenAI API.

B Software and Hardware

All experiments were run with Python 3.11.5 and PyTorch 2.0.1. The whole query experiments did not require any training, but needed GPUs for inference. We used Nvidia A100 (80 GB) GPUs for this purpose. Each inference run took around 2-3 hours with a batch size of 4-6 depending on the model used in the experiment. For fine grained experiments we trained Error Model using Nvidia A100 (80 GB) GPUs. Models were trained for 5-10 epochs with a batch size 4 and took 2-3 hours per epoch. Our model has 813M learnable parameters along with frozen pre-trained embedding model like CodeLlama. We release the code for our experiments under Apache-2.0 license.

C Experiment Details

For whole query experiments involving Codestral, the model was used in 8-bit mode due to hardware constraints. The version of CodeLlama used are the 7b-instruct for whole query experiments and 7b (base) for fine grained experiments. The models-CodeS, Codestral and CodeLlama-were sourced from Hugging Face repositories. Specifically, the models were obtained from the following URLs:

- CodeS: https://huggingface.co/seeklhy/codes-7b
- Codestral: https://huggingface.co/bullerwins/Codestral-22B-v0.1-hf
- CodeLlama: https://huggingface.co/codellama/CodeLlama-7b-Instruct-hf https://huggingface.co/codellama/CodeLlama-7b-hf

[3] https://github.com/RUCKBReasoning/codes

[4] https://github.com/wbbeyourself/MAC-SQL/

[5] https://github.com/meta-llama/llama/blob/main/LICENSE

[6] https://mistral.ai/news/mistral-ai-non-production-license-mnpl/

Model	Parameters (in Billions)
CodeS	7
Codestral	22
CodeLlama	7
GPT-4	-

Table 5: Parameters in models used for experiments.

Parameter count for each of the models are presented in table 5. Perturbations data is grouped by question id for batch training with batch size 4. First off, all queries are grouped in batches of size 4 then followed by leftover queries.

D Prompts

We use the prompts from (Li et al., 2024), (Wang et al., 2024) and (Tian et al., 2023) as inspirations for prompts for pooled token-level, for self-check Bool and for self-check Probs experiments respectively. We present the prompts used in experiments in tables 6, 7 and 8. The prompts used to generated predictions are presented in 9.

E Additional Results

E.1 Whole query experiments

In table 10, we report the evaluation metrics along with standard deviation for all the whole query experiments. We note that the ECE and platt-scaled ECE are not very reliable metrics. A binary classifier can get a perfect ECE by guessing either label with 50% confidence on a data with equal distribution of labels. Aggregation using **prod** provides the best calibration among pooled token-level methods in terms of AUC and Brier-P. This is followed by the variant SQLs method which uses the **prod** pooled token-level confidence scores and then self-check Bool method.

Comparison with Isotonic scaling Table 11 and figure 6 shows the variation of the evaluation metrics, brier score and expected calibration error, with the two calibration methods, platt scaling and isotonic regression. Note that the AUC and ECE of the raw confidence scores, which are also reported in table 3 are indifferent to calibration.

Comparison with Monotonic binning Table 12 and figure 7 shows the variation of the evaluation metrics, expected calibration error of the raw and calibrated confidence scores, with the two different methods of binning, uniform and monotonic. Note that the AUC and Brier score, which are also reported in table 3 are indifferent to the binning method.

E.2 Fine grained experiments

We have identified several nodes which when considered during hash calculation of their children prevent incidental match in RAT. These nodes include: `Val_list` , `Orderby_desc`, `Orderby_asc`, `Project`, `Limit`, `Groupby`

We also consider sorting children nodes for hash calculation of some nodes since these operations are commutative and as such order of children for these nodes can be permuted and still provide correct SQL. These include `eq`, `add`, `neq`, `And`, `Or`, `union`, `intersect`, `Product`, `Val_list`

F Ablations on Confidence Model

In table 14 we train our model on different configurations of training data and positive weights to handle class imbalance in data. Also, while training for subtree level calibration model, we employed different techniques to handle node misalignment scenarios as discussed earlier and found that we get best results when we train model on predictions obtained from CodeS and Codestral and their column perturbations obtained using SentenceTransformer. We sampled our perturbations set such that our final training data had equal number of samples from both prediction set and perturbation set batched by question id to ensure model is trained on different variants of similarly structured trees in a batch. In table 4 we compare our model with baseline established using token level probabilities obtained from CodeS and Codestral and a global alignment algorithm to calculate ground truth.

We also compare our model with other variants trained with same perturbed dataset but without optimal weights to handle class imbalance or a variant trained only on prediction set without any perturbations both with and without handling for class imbalance. Choice of optimal weight is as provide in pytorch documentation. pos_weight parameter is a scalar that represents weight for positive class. Optimal choice of this parameter is defined as ratio of number of negative samples to number of positive samples. It is used to handle imbalanced class distribution during loss computation. Platt-scaled Brier score, AUC and other metrics are used to compare different training configurations along with calibration plots and we found that all configurations provide similar Platt-scaled Brier and ECE score but model trained on prediction set without perturbations without scaling for class imbalance reports best scores. But when we consider the calibration plots we find that out of all configurations the one trained with perturbations and handling for class imbalance achieves a well calibrated plot across both datasets as seen in Figure 4 in plots 2 and 4. Also, with this configuration we achieve minimum loss on dev set within 2 epochs whereas other variants did not report similar performance.

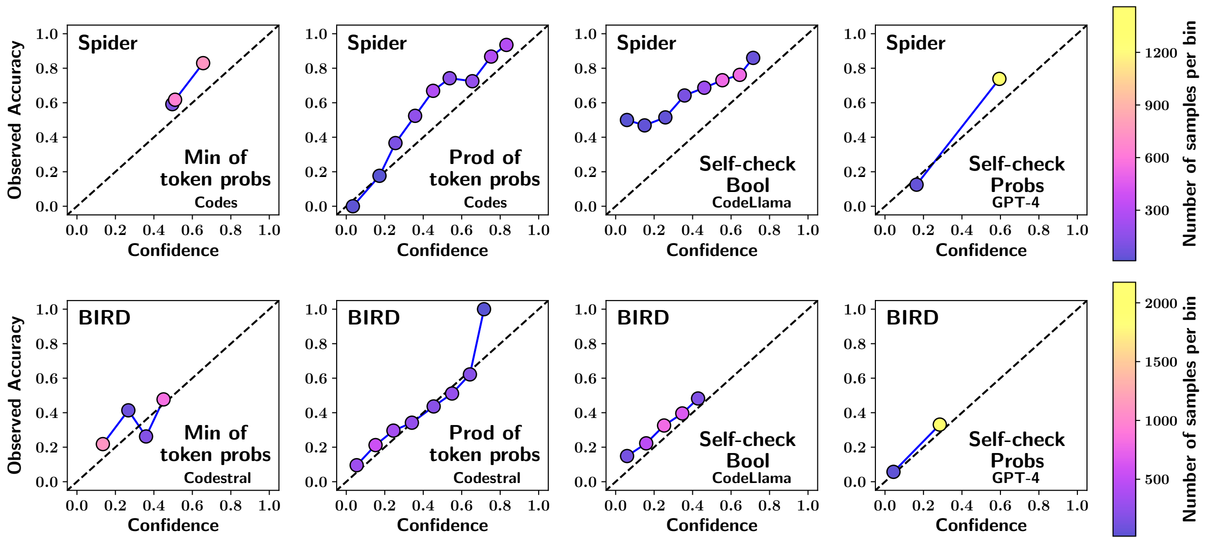


Figure 5: The reliability plots continued from 3 to illustrate the calibration comparison between the different whole query methods. The four plots on top have been generated with predictions corresponding to the Spider dataset and four plots below, with the BIRD dataset. A well-calibrated plot aligns closely with the x=y line. Each point is color-coded based on the number of samples in the bin, as indicated by the colorbar on the right.

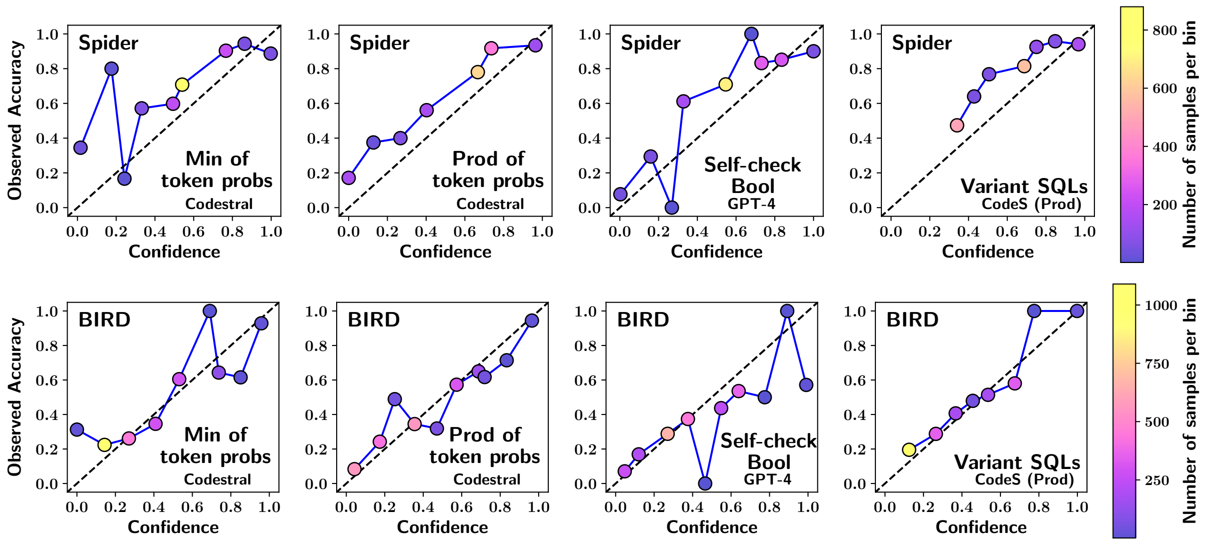


Figure 6: The plots have been generated using isotonic scaling in place of platt scaling used in 3. The four plots on top have been generated with predictions corresponding to the Spider dataset and four plots below, with the BIRD dataset. A well-calibrated plot aligns closely with the x=y line. Each point is color-coded based on the number of samples in the bin, as indicated by the colorbar on the right.

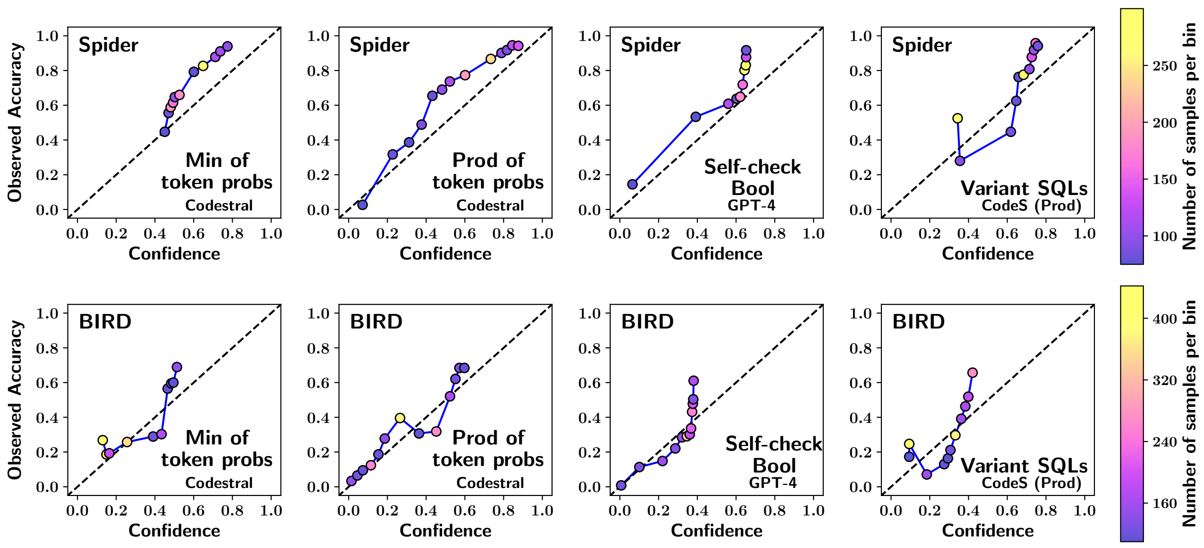


Figure 7: The plots have been generated using Monotonic binning in place of Uniform binning used in 3. The four plots on top have been generated with predictions corresponding to the Spider dataset and four plots below, with the BIRD dataset. A well-calibrated plot aligns closely with the x=y line. Each point is color-coded based on the number of samples in the bin, as indicated by the colorbar on the right.

Method	Prompt Template
Pooled token-level	You are provided with a sqlite database schema and a user question. Your task is to generate a sqlite query which can be executed on the sqlite database. database schema : table {table name}, columns = [{table name.column_name} ({data type} | {is primary key?} | values: {sample values}), ...] .. foreign keys : {foreign keys} matched contents : None {question} {SQL}
Self check Bool	[Instruction] Complete SQL query only and with no explanation. [Constraints] - In ‘SELECT <column>‘, just select needed columns in the [Question] without any unnecessary column or value - In ‘FROM <table>‘ or ‘JOIN <table>‘, do not include unnecessary table - If use max or min func, ‘JOIN <table>‘ FIRST, THEN use ‘SELECT MAX(<column>)‘ or ‘SELECT MIN(<column>)‘ - If [Value examples] of <column> has ’None’ or None, use ‘JOIN <table>‘ or ‘WHERE <column> is NOT NULL‘ is better - If using ‘ORDER BY <column> ASC|DESC‘, add ‘GROUP BY <column>‘ before to select distinct values [Query] – {question} [Evidence] {evidence} [Database info] # Table: {table name} [({column name}, {description of column}. Value examples: [{sample values}].), .. [Foreign keys] {foreign keys} The proposed SQL for the query is: [SQL] “‘sql {sql} “‘

Table 6: Prompt templates for the different whole query methods.

Method	Prompt Template
Self check Probs	[Instruction] Complete SQL query only and with no explanation. [Constraints] - In ‘SELECT <column>‘, just select needed columns in the [Question] without any unnecessary column or value - In ‘FROM <table>‘ or ‘JOIN <table>‘, do not include unnecessary table - If use max or min func, ‘JOIN <table>‘ FIRST, THEN use ‘SELECT MAX(<column>)‘ or ‘SELECT MIN(<column>)‘ - If [Value examples] of <column> has ’None’ or None, use ‘JOIN <table>‘ or ‘WHERE <column> is NOT NULL‘ is better - If using ‘ORDER BY <column> ASC|DESC‘, add ‘GROUP BY <column>‘ before to select distinct values [Query] – {question} [Evidence] {evidence} [Database info] # Table: {table name} [({column name}, {description of column}. Value examples: [{sample values}].), .. [Foreign keys] {foreign keys} The proposed SQL for the query is: [SQL] “‘sql {sql} “‘ Provide your best guess and the probability that it is correct (0.0 to 1.0). Give ONLY the probability, no other words or explanation. For example: Probability: <the probability between 0.0 and 1.0 that your guess is correct, without any extra commentary whatsoever; just the probability!>

Table 7: Prompt templates for the different whole query methods.

Method	Prompt Template
Generation of variant SQLs	When executing SQL below, some errors occurred, please fix up SQL based on query and database info. Solve the task step by step if you need to. Use SQL format in the code block, and indicate script type in the code block. When you find an answer, verify the answer carefully. Include verifiable evidence in your response if possible. [Constraints] - In ‘SELECT <column>‘, just select needed columns in the [Question] without any unnecessary column or value - In ‘FROM <table>‘ or ‘JOIN <table>‘, do not include unnecessary table - If use max or min func, ‘JOIN <table>‘ FIRST, THEN use ‘SELECT MAX(<column>)‘ or ‘SELECT MIN(<column>)‘ - If [Value examples] of <column> has ’None’ or None, use ‘JOIN <table>‘ or ‘WHERE <column> is NOT NULL‘ is better - If using ‘ORDER BY <column> ASC|DESC‘, add ‘GROUP BY <column>‘ before to select distinct values [Query] – {question} [Evidence] {evidence} [Database info] # Table: {table name} [({column name}, {description of column}. Value examples: [{sample values}].), .. [Foreign keys] {foreign keys} Generate ten structurally diverse SQLs for the above query

Table 8: Prompt templates for the different whole query methods.

LLM	Dataset	Prompt Template
CodeS	BIRD	You are provided with a sqlite database schema and a user question along with a hint to help create an SQL query. Your task is to generate a sqlite query which can be executed on the sqlite database. **Input:** Schema: {schema} CREATE TABLE table_name (column1 datatype CONSTRAINT constraint_name1, ... CONSTRAINT constraint_name3 PRIMARY KEY (column_name), CONSTRAINT constraint_name6 FOREIGN KEY (column_name) REFERENCES other_table ... ...); Question: {question} Hint: {evidence} **Output:** SQL:
CodeS	Spider	You are provided with a sqlite database schema and a user question to help create an SQL query. Your task is to generate a sqlite query which can be executed on the sqlite database. **Input:** Schema: {schema} CREATE TABLE table_name (column1 datatype CONSTRAINT constraint_name1, column2 datatype CONSTRAINT constraint_name2, ... CONSTRAINT constraint_name3 PRIMARY KEY (column_name), ...); Question: {question} **Output:** SQL:
GPT-4	Spider	### Complete sqlite SQL query only and with no explanation ### Sqlite SQL tables, with their properties: # # {table name}({column names}) .. # ### {question} SELECT
GPT-4	BIRD	### Complete sqlite SQL query only and with no explanation ### Sqlite SQL tables, with their properties: # # {table name}({column names}) .. # # Evidence: {evidence} ### {question} SELECT

Table 9: Prompt templates for the generating SQL predictions for Spider and BIRD data using CodeS and GPT4.

Method		Spider			
		BS-P↓	AUC↑	ECE↓	ECE-P↓
Pooled token-level (CodeS)	min	0.221 ± 0.0128	0.636 ± 0.0126	0.669 ± 0.0244	0.120 ± 0.0308
	avg	0.233 ± 0.0163	0.541 ± 0.0026	0.097 ± 0.0153	0.135 ± 0.0400
	prod	0.194 ± 0.0128	0.746 ± 0.0066	0.685 ± 0.0247	0.112 ± 0.0358
	geo	0.234 ± 0.0166	0.539 ± 0.0076	0.216 ± 0.0277	0.130 ± 0.0379
Pooled token-level (Codestral)	min	0.215 ± 0.0137	0.663 ± 0.0145	0.662 ± 0.0239	0.114 ± 0.0349
	avg	0.223 ± 0.0147	0.606 ± 0.0065	0.133 ± 0.0250	0.126 ± 0.0355
	prod	0.172 ± 0.0104	0.788 ± 0.0087	0.678 ± 0.0244	0.098 ± 0.0318
	geo	0.228 ± 0.0157	0.598 ± 0.0104	0.228 ± 0.0251	0.133 ± 0.0341
Self-check Bool (GPT-4)		0.208 ± 0.0131	0.701 ± 0.0040	0.207 ± 0.0216	0.120 ± 0.0231
Self-check Bool (CodeLlama)		0.229 ± 0.0131	0.600 ± 0.0071	0.076 ± 0.0246	0.132 ± 0.0388
Self-check Probs (GPT-4)		0.223 ± 0.0162	0.598 ± 0.0034	0.269 ± 0.0248	0.130 ± 0.0304
Variant SQLs (Prod) (CodeS)		0.200 ± 0.0131	0.747 ± 0.0051	0.684 ± 0.0244	0.110 ± 0.0252

Method		BIRD			
		BS-P↓	AUC↑	ECE↓	ECE-P↓
Pooled token-level (CodeS)	min	0.205 ± 0.0052	0.651 ± 0.0072	0.293 ± 0.0129	0.070 ± 0.0065
	avg	0.213 ± 0.0054	0.618 ± 0.0075	0.465 ± 0.0137	0.065 ± 0.0292
	prod	0.193 ± 0.0037	0.730 ± 0.0111	0.314 ± 0.0128	0.082 ± 0.0225
	geo	0.213 ± 0.0054	0.631 ± 0.0068	0.226 ± 0.0102	0.069 ± 0.0188
Pooled token-level (Codestral)	min	0.202 ± 0.0045	0.670 ± 0.0085	0.266 ± 0.0120	0.076 ± 0.0072
	avg	0.198 ± 0.0040	0.705 ± 0.0061	0.526 ± 0.0121	0.067 ± 0.0165
	prod	0.188 ± 0.0034	0.757 ± 0.0103	0.305 ± 0.0125	0.083 ± 0.0141
	geo	0.202 ± 0.0043	0.694 ± 0.0079	0.254 ± 0.0113	0.075 ± 0.0084
Self-check Bool (GPT-4)		0.203 ± 0.0069	0.707 ± 0.0125	0.538 ± 0.0169	0.071 ± 0.0306
Self-check Bool (CodeLlama)		0.217 ± 0.0059	0.621 ± 0.0107	0.491 ± 0.0174	0.090 ± 0.0372
Self-check Probs (GPT-4)		0.216 ± 0.0054	0.584 ± 0.0091	0.627 ± 0.0152	0.063 ± 0.0267
Variant SQLs (Prod) (CodeS)		0.207 ± 0.0040	0.701 ± 0.0127	0.314 ± 0.0128	0.094 ± 0.0377

Table 10: Replication of Table 3 with standard deviation. The first table present the metrics for the Spider dataset, and the second table for the BIRD dataset. The metrics include Platt-scaled Brier score **(BS-P)**, area under the ROC curve **(AUC)**, expected calibration error **(ECE)** and Platt-scaled ECE **(ECE-P)**. Uniform binning is used to calculate ECE and ECE-P. Highlighted numbers in blue, green, and yellow denote the best, second best, and third best methods, respectively.

	Method		**Spider**		**BIRD**	
			BS-(P/I)↓	**ECE-(P/I)↓**	**BS-(P/I)↓**	**ECE-(P/I)↓**
Platt	**Pooled token-level** (CodeS)	min	0.221 ± 0.0128	0.120 ± 0.0308	0.205 ± 0.0052	0.070 ± 0.0065
		avg	0.233 ± 0.0163	0.135 ± 0.0400	0.213 ± 0.0054	0.065 ± 0.0292
		prod	0.194 ± 0.0128	0.112 ± 0.0358	0.193 ± 0.0037	0.082 ± 0.0225
		geo	0.234 ± 0.0166	0.130 ± 0.0379	0.213 ± 0.0054	0.069 ± 0.0188
	Pooled token-level (Codestral)	min	0.215 ± 0.0137	0.114 ± 0.0349	0.202 ± 0.0045	0.076 ± 0.0072
		avg	0.223 ± 0.0147	0.126 ± 0.0355	0.198 ± 0.0040	0.067 ± 0.0165
		prod	0.172 ± 0.0104	0.098 ± 0.0318	0.188 ± 0.0034	0.083 ± 0.0141
		geo	0.228 ± 0.0157	0.133 ± 0.0341	0.202 ± 0.0043	0.075 ± 0.0084
	Self-check Bool (GPT-4)		0.208 ± 0.0131	0.120 ± 0.0231	0.203 ± 0.0069	0.071 ± 0.0306
	Self-check Bool (CodeLlama)		0.229 ± 0.0131	0.132 ± 0.0388	0.217 ± 0.0059	0.090 ± 0.0372
	Self-check Probs (GPT-4)		0.223 ± 0.0162	0.130 ± 0.0304	0.216 ± 0.0054	0.063 ± 0.0267
	Variant SQLs (Prod) (CodeS)		0.200 ± 0.0131	0.110 ± 0.0252	0.207 ± 0.0040	0.094 ± 0.0377
Isotonic	**Pooled token-level** (CodeS)	min	0.223 ± 0.0118	0.124 ± 0.0318	0.206 ± 0.0056	0.066 ± 0.0194
		avg	0.235 ± 0.0161	0.142 ± 0.0362	0.215 ± 0.0027	0.072 ± 0.0253
		prod	0.196 ± 0.0106	0.112 ± 0.0402	0.193 ± 0.0040	0.080 ± 0.0230
		geo	0.235 ± 0.0189	0.140 ± 0.0304	0.212 ± 0.0053	0.063 ± 0.0289
	Pooled token-level (Codestral)	min	0.221 ± 0.0140	0.132 ± 0.0327	0.200 ± 0.0045	0.064 ± 0.0229
		avg	0.224 ± 0.0144	0.131 ± 0.0306	0.199 ± 0.0029	0.062 ± 0.0238
		prod	0.174 ± 0.0109	0.096 ± 0.0360	0.184 ± 0.0042	0.077 ± 0.0241
		geo	0.225 ± 0.0157	0.134 ± 0.0324	0.202 ± 0.0030	0.069 ± 0.0148
	Self-check Bool (GPT-4)		0.206 ± 0.0096	0.119 ± 0.0232	0.197 ± 0.0061	0.062 ± 0.0288
	Self-check Bool (CodeLlama)		0.238 ± 0.0174	0.148 ± 0.0497	0.221 ± 0.0068	0.101 ± 0.0357
	Self-check Probs (GPT-4)		0.220 ± 0.0163	0.130 ± 0.0301	0.214 ± 0.0050	0.065 ± 0.0332
	Variant SQLs (Prod) (CodeS)		0.195 ± 0.0102	0.111 ± 0.0250	0.194 ± 0.0040	0.064 ± 0.0163

Table 11: The table compares evaluation metrics across the two calibration methods, Platt scaling and isotonic regression, for various whole query methods on the Spider and BIRD datasets. The first six rows present Platt-scaled Brier score **(BS-P)** and Platt-scaled ECE **(ECE-P)** and the last six rows present isotonic-regression Brier score **(BS-I)** and isotonic-regression ECE **(ECE-I)**. Uniform binning is used to calculate ECE-P and ECE-I. Highlighted numbers in green and yellow denote the best and second best methods, respectively.

Binning	Method		Spider ECE↓	Spider ECE-P↓	BIRD ECE↓	BIRD ECE-P↓
Uniform Binning	**Pooled token-level** (CodeS)	min	0.669 ± 0.0244	0.120 ± 0.0308	0.293 ± 0.0129	0.070 ± 0.0065
		avg	0.097 ± 0.0153	0.135 ± 0.0400	0.465 ± 0.0137	0.065 ± 0.0292
		prod	0.685 ± 0.0247	0.112 ± 0.0358	0.314 ± 0.0128	0.082 ± 0.0225
		geo	0.216 ± 0.0277	0.130 ± 0.0379	0.226 ± 0.0102	0.069 ± 0.0188
	Pooled token-level (Codestral)	min	0.662 ± 0.0239	0.114 ± 0.0349	0.266 ± 0.0120	0.076 ± 0.0072
		avg	0.133 ± 0.0250	0.126 ± 0.0355	0.526 ± 0.0121	0.067 ± 0.0165
		prod	0.678 ± 0.0244	0.098 ± 0.0318	0.305 ± 0.0125	0.083 ± 0.0141
		geo	0.228 ± 0.0251	0.133 ± 0.0341	0.254 ± 0.0113	0.075 ± 0.0084
	Self-check Bool (GPT-4)		0.207 ± 0.0216	0.120 ± 0.0231	0.538 ± 0.0169	0.071 ± 0.0306
	Self-check Bool (CodeLlama)		0.076 ± 0.0246	0.132 ± 0.0388	0.491 ± 0.0174	0.090 ± 0.0372
	Self-check Probs (GPT-4)		0.269 ± 0.0248	0.130 ± 0.0304	0.627 ± 0.0152	0.063 ± 0.0267
	Variant SQLs (Prod) (CodeS)		0.684 ± 0.0244	0.110 ± 0.0252	0.314 ± 0.0128	0.094 ± 0.0377
Monotonic Binning	**Pooled token-level** (CodeS)	min	0.669 ± 0.0244	0.120 ± 0.0318	0.293 ± 0.0129	0.069 ± 0.0075
		avg	0.089 ± 0.0151	0.131 ± 0.0386	0.464 ± 0.0137	0.065 ± 0.0273
		prod	0.685 ± 0.0247	0.111 ± 0.0368	0.314 ± 0.0128	0.079 ± 0.0232
		geo	0.214 ± 0.0289	0.132 ± 0.0332	0.213 ± 0.0126	0.078 ± 0.0141
	Pooled token-level (Codestral)	min	0.662 ± 0.0239	0.114 ± 0.0340	0.266 ± 0.0119	0.083 ± 0.0064
		avg	0.133 ± 0.0252	0.127 ± 0.0347	0.526 ± 0.0121	0.068 ± 0.0169
		prod	0.678 ± 0.0244	0.096 ± 0.0354	0.305 ± 0.0126	0.088 ± 0.0133
		geo	0.228 ± 0.0254	0.132 ± 0.0339	0.253 ± 0.0117	0.079 ± 0.0114
	Self-check Bool (GPT-4)		0.204 ± 0.0232	0.128 ± 0.0209	0.538 ± 0.0169	0.089 ± 0.0212
	Self-check Bool (CodeLlama)		0.073 ± 0.0250	0.129 ± 0.0406	0.491 ± 0.0173	0.087 ± 0.0385
	Self-check Probs (GPT-4)		0.261 ± 0.0237	0.138 ± 0.0234	0.625 ± 0.0155	0.084 ± 0.0156
	Variant SQLs (Prod) (CodeS)		0.684 ± 0.0244	0.123 ± 0.0254	0.314 ± 0.0128	0.112 ± 0.0147

Table 12: The table compares evaluation metrics across the two binning method, Uniform Binning and Monotonic Binning, for various whole query methods on the Spider and BIRD datasets. The first six rows present **ECE** and Platt-scaled ECE **(ECE-P)** obtained using Uniform binning and the last six rows present **ECE** and Platt-scaled ECE **(ECE-P)** obtained using Monotonic binning. Highlighted numbers in green and yellow denote the best and second best methods, respectively.

Index	SQL
1	**Ques:** Find the last name of the student who has a cat that is age 3. **Gold:** select student.lname from student join has_pet on student.stuid = has_pet.stuid join pets on pets.petid = has_pet.petid where pets.pet_age = 3 and pets.pettype = 'cat' **Predicted:** select student.lname from has_pet join pets on has_pet.petid = pets.petid join student on has_pet.stuid = student.stuid where pets.pettype = 'cat' and student.age = 3
2	**Ques:** How many male patients have their glutamic oxaloacetic transaminase in the normal range? **Gold:** select count(patient.id) from patient join laboratory on patient.id = laboratory.id where laboratory.got < 60 and patient.sex = 'M' **Predicted:** select count(distinct patient.id) from patient join laboratory on patient.id = laboratory.id where patient.sex = 'M' and laboratory.got < 60
3	**Ques:** What is the average height of a non-human superhero in Dark Horse Comics? **Gold:** select avg(superhero.height_cm) from superhero join publisher on superhero.publisher_id = publisher.id join race on superhero.race_id = race.id where publisher.publisher_name = 'Dark horse Comics' and race.race <> 'Human' **Predicted:** select avg(superhero.height_cm) from superhero join race on superhero.race_id = race.id join publisher on superhero.publisher_id = publisher.id where race.race <> 'Human' and publisher.publisher_name = 'Dark horse Comics'
4	**Ques:** Show me the season page of year when the race No. 901 took place. **Gold:** select seasons.url from races join seasons on seasons.year = races.year where races.raceid = 901 **Predicted:** select seasons.url from seasons where seasons.year = (select races.year as year from races where races.raceid = 901)
5	**Ques:** How many heroes have stealth power? **Gold:** select count(hero_power.hero_id) from hero_power join superpower on hero_power.power_id = superpower.id where superpower.power_name = 'Stealth' **Predicted:** select count(distinct hero_power.hero_id) from hero_power join superpower on hero_power.power_id = superpower.id where superpower.power_name = 'stealth'

Table 13: This table demonstrates some anecdotes for confidence score by our Error Model on predicted SQL's. Subtree with gold label 0 is marked in red font and confidence score provided by model highlighted by range of score. Score in range of 0-0.2 in purple, 0.2-0.4 in orange and 0.4-0.6 in yellow. For subtree above level 2 we highlight only root node for better readability. One can observe that higher level nodes in RAT are often marked wrong due to accumulation of any incorrect hash in children nodes

Training Methodology	**Spider**				**BIRD**			
	BS-P↓	**AUC↑**	**ECE↓**	**ECE-P↓**	**BS-P↓**	**AUC↑**	**ECE↓**	**ECE-P↓**
Confidence Model	0.15	0.76	0.25	0.05	0.20	0.76	0.28	0.10
w/o optimal pos_weight	0.15	0.75	0.05	0.05	0.19	0.78	0.18	0.08
w/o perturbed data	0.12	0.56	0.19	0.02	0.17	0.55	0.24	0.04
w/o pos_weight	0.14	0.78	0.03	0.05	0.20	0.78	0.08	0.13

Table 14: Abalation Study of Confidence Model demonstrates the impact of including column perturbed data to training set and use of optimal positive weights in BCEWithLogitLoss to adjust for class imbalance. Highlighted numbers in green and yellow denote the best and second best methods, respectively