Disambiguate First, Parse Later: Generating Interpretations for Ambiguity Resolution in Semantic Parsing

Irina Saparina and **Mirella Lapata**
Institute for Language, Cognition and Computation,
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB
i.saparina@sms.ed.ac.uk mlap@inf.ed.ac.uk

Abstract

Handling ambiguity and underspecification is an important challenge in natural language interfaces, particularly for tasks like text-to-SQL semantic parsing. We propose a modular approach[1] that resolves ambiguity using natural language interpretations before mapping these to logical forms (e.g., SQL queries). Although LLMs excel at parsing *unambiguous* utterances, they show strong biases for *ambiguous* ones, typically predicting only preferred interpretations. We constructively exploit this bias to generate an initial set of preferred disambiguations and then apply a specialized infilling model to identify and generate missing interpretations. To train the infilling model, we introduce an annotation method that uses SQL execution to validate different meanings. Our approach improves interpretation coverage and generalizes across datasets with different annotation styles, database structures, and ambiguity types.

1 Introduction

Natural language utterances are often ambiguous, vague, or underspecified, giving rise to multiple valid interpretations. Figure 1 shows an ambiguous request ("return the rating of each hotel") in the context of natural language interfaces. In the example, "rating" could refer to the number of stars hotels receive as an indication of their quality (e.g., 4 stars) or guest reviews on booking websites (e.g., 8.5 out of 10), or both. Ignoring such ambiguity can lead to incomplete or incorrect results, undermining user trust and limiting the practical usefulness of any conversational system.

While state-of-the-art large language models (LLMs) demonstrate remarkable performance on tasks like question answering, text-to-SQL parsing, and natural language inference, recent studies (Liu et al., 2023; Floratou et al., 2024) have shown they

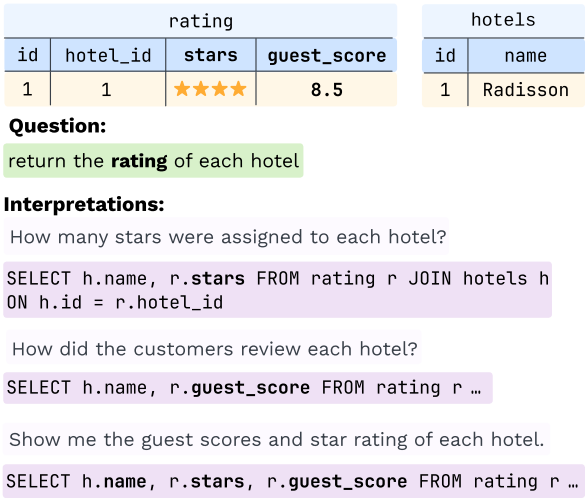

Figure 1: Example of ambiguous question and its interpretations (in natural language and SQL).

are not adept at handling ambiguity. They display systematic biases in their choice of interpretation (Kamath et al., 2024; Stengel-Eskin et al., 2024; Saparina and Lapata, 2024a), typically defaulting to a single interpretation when multiple ones exist.

What are the response strategies LLMs should adopt to address ambiguity? An approach might be to respond with a clarification question, which directly engages users and ensures accurate resolution of the ambiguity but introduces additional interaction turns. Alternatively, presenting multiple possible interpretations (an "overton response") would allow users to select the most relevant answer themselves. This approach minimizes interruptions, caters to users with different levels of expertise, and provides interpretability by making the system's reasoning explicit. For tasks like semantic parsing (see Figure 1), a hypothetical response should not only include interpretations in their final form (e.g., SQL queries) but also their different readings in natural language. From a modeling perspective, interpretations serve as explanations (of the system's output) and as *intermediate repre-*

[1]Our code and data can be downloaded from github.com/saparina/disambiguate-then-parse.

Findings of the Association for Computational Linguistics: ACL 2025, pages 16825–16839

sentations, providing a way to decompose complex semantic parsing problems into simpler steps.

In this work, we focus on text-to-SQL parsing of ambiguous questions, and propose a two-stage approach that first disambiguates by generating all possible meanings in natural language, and then parses each unambiguous interpretation. Separating the disambiguation and parsing tasks allows us to use existing semantic parsers that perform generally well on *unambiguous* inputs. We obtain interpretations, by prompting an LLM to generate all possible meanings for an utterance. These initial interpretations are often incomplete due to inherent biases stemming from statistical patterns found in LLM training data, lack of intrinsic knowledge, and different alignment preferences. Rather than attempting to correct these biases, we introduce an infilling model that reviews the ambiguous question and initial interpretations, and then generates any missing ones.

The infilling model is trained on pairs of default interpretations and missing readings. Problematically, existing datasets provide (multiple) SQL parses for ambiguous questions without explicitly verbalising the interpretations they correspond to. We create synthetic reference interpretations for AmbiQT (Bhaskar et al., 2023), a recently proposed benchmark for parsing ambiguous questions into SQL. We exploit the fact that generated interpretations can be converted to SQL queries which we execute to verify whether they are correct *and* to establish which interpretations are missing, i.e., whether there exist gold SQL parses for which no interpretation was found. We evaluate our approach on Ambrosia (Saparina and Lapata, 2024a), a dataset different from AmbiQT in terms of database content and the types of ambiguity it represents. Our contributions are summarized as follows:

- We propose a modular approach that uses natural language to spell out ambiguity before mapping individual interpretations to logical forms (e.g., SQL queries).
- We use LLMs to generate an initial set of preferred disambiguations and then apply a specialized infilling model to identify and generate missing interpretations.
- Experiments show that our "disambiguate first parse later" strategy improves the coverage of interpretations for ambiguous questions and generalizes across annotation styles, database structures, and ambiguity types.

2 Disambiguate First Parse Later

2.1 Problem Formulation

Semantic parsing is the task of mapping a natural language utterance u to formal expression e in grammar G, where e captures the intended meaning of u. The expression e can be then executed in an environment $\mathcal{E}$ to produce a denotation $[\![e]\!]$. In the unambiguous case, there is a single valid expression e that corresponds to the user's intent. However, when u is ambiguous, there are multiple valid expressions $\{e_1, ..., e_n\}$ where $[\![e_i]\!] \neq [\![e_j]\!]$ for some i, j, with each e_i representing a valid interpretation of the user's intent.

In our text-to-SQL task, grammar G defines valid SQL queries for a given database schema S. The database schema and table descriptions provide context $\mathcal{C}$ that can help disambiguate some queries. We focus on questions that remain ambiguous even when $\mathcal{C}$ is known, i.e., there exist multiple valid SQL queries $\{e_1, ..., e_n\}$ that respect S and produce different result sets $[\![e_i]\!]$ when executed on the database. In the example in Figure 1, the question u = "return the rating of each hotel" remains ambiguous for a given schema S (see top table), as "rating" could refer to star ratings and guest reviews even when the database content is known.

2.2 Natural Language Interpretations for Explicit Disambiguation

We propose to resolve ambiguity prior to generating SQL expressions with natural language interpretations. More formally, for ambiguous utterance u, we first produce a set of unambiguous natural language interpretations $\{\hat{u}_1, ..., \hat{u}_n\}$, where each $\hat{u}_i$ captures one meaning of u. We then map each $\hat{u}_i$ to formal expression e_i. For example, given u = "return the rating of each hotel", our goal is to produce $\hat{u}_1$ = "How many stars were assigned to each hotel?", $\hat{u}_2$ = "How did the customers review each hotel?" and $\hat{u}_3$ = "Show me the guest scores and star rating of each hotel" and map them to corresponding SQL queries e_1, e_2 and e_3 (see Figure 1). This approach has several advantages:

- Unambiguous interpretations $\hat{u}_i$ are easier to translate into formal expressions, as they can be handled by standard semantic parsers (e.g., existing text-to-SQL models);

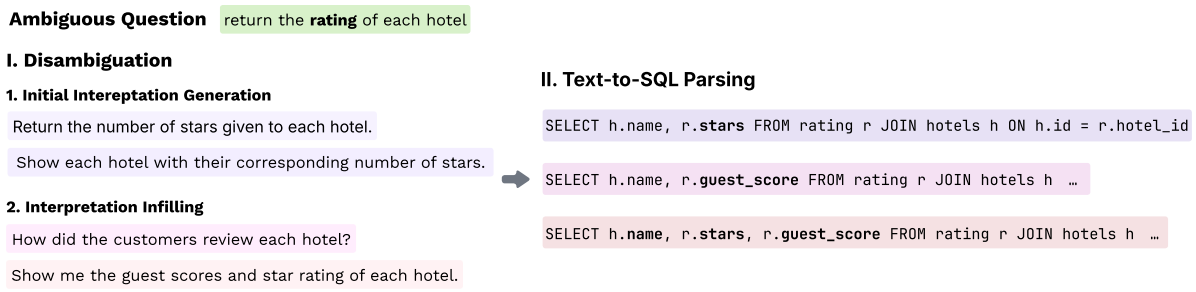

Figure 2: Disambiguate first parse later: we disambiguate database questions into natural language interpretations (by generating an initial set and filling in missing ones), and then a parser translates each interpretation into SQL.

- Natural language interpretations provide transparency, making the system's internal working explicit to users;
- The modular design allows us to optimize the disambiguation and formal expression mapping components independently.

Building on this modular approach, we propose to further decompose explicit disambiguation into two steps: initial interpretation generation followed by infilling of missing interpretations. Our core idea is based on the observation that **modern LLMs, like humans, resolve ambiguous questions by gravitating toward preferred, default interpretations** (Kamath et al., 2024; Stengel-Eskin et al., 2024; Saparina and Lapata, 2024b). We leverage this tendency by using LLMs to generate preferred interpretations. We then train a specialized model that, given an ambiguous question and its default interpretations, identifies and generates missing readings to ensure all valid meanings are covered. Figure 2 illustrates our approach: generating default interpretations (using LLMs), infilling missing ones, and text-to-SQL parsing.

2.3 Default Interpretation Generation

LLMs exhibit strong biases in interpretation generation, consistently favouring certain types while missing others. We take advantage of these biases by using LLMs to generate an initial set of interpretations for ambiguous utterances.

The first component of our approach is a prompt-based interpretation generator that produces candidate interpretations given utterance u and database context $\mathcal{C}$. We prompt an LLM to identify and list all *semantically different* interpretations of u. In practice, LLMs generate only a subset of possible interpretations, typically the most straightforward ones. We refer to them as **preferred** or **default interpretations**. The prompt is designed to handle *both* ambiguous and unambiguous cases, allowing the model to return a single interpretation when appropriate. As we rely on the LLM's ability to disambiguate to its default interpretations, no additional training is required. The output of this module is a set of natural language interpretations $\{\hat{u}_1, ..., \hat{u}_k\}$, which are mapped to formal expressions $\{e_1, ..., e_k\}$ using a standard semantic parser. For text-to-SQL tasks, executing these queries allows us to identify paraphrases (interpretations that lead to identical execution results) and filter redundancy.

2.4 Interpretation Infilling

While initial interpretations from LLMs capture a common understanding of queries, they often miss valid alternative meanings. We propose to address this issue by introducing a specialised model that identifies and generates missing interpretations.

Given utterance u, database context $\mathcal{C}$, and default interpretations $\{\hat{u}_1, ..., \hat{u}_k\}$, our model determines whether additional interpretations exist and generates these accordingly. The output is a set of interpretations $\{\hat{u}_l, ..., \hat{u}_m\}$ which complement the initial set. These interpretations are then also mapped to formal expressions $\{e_l, ..., e_m\}$ (SQL queries in our case) using a standard semantic parser. We train the infilling model in a supervised manner. For now, let us assume we have access to reference interpretations and corresponding gold SQL queries. By comparing the set of default interpretations with reference interpretations, we identify which meanings are missing from the initial set. Reference interpretations absent from the default set serve as target outputs for the infilling model.

Determining whether two sentences have the *same* meaning, given some (database) context, is an extremely challenging task. However, instead of comparing natural language expressions directly, we compare their corresponding SQL queries. In

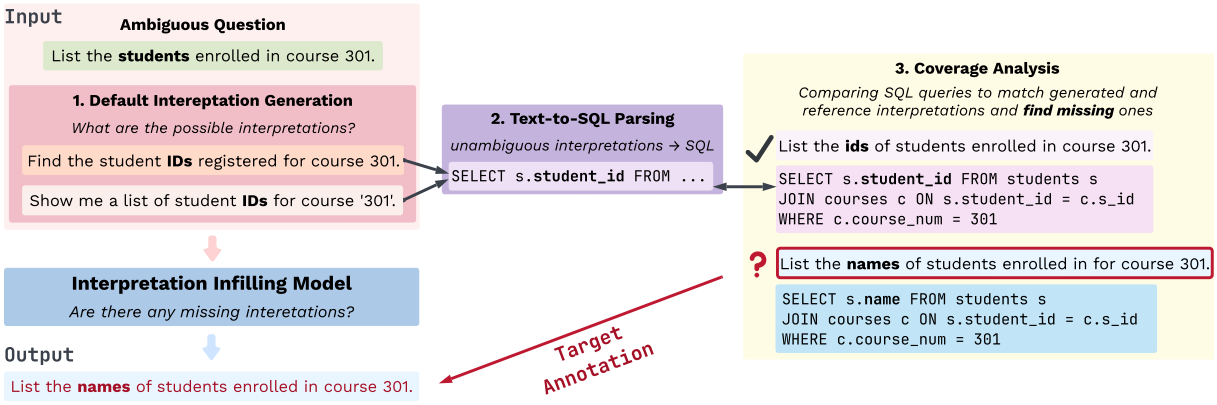

Figure 3: The Infilling Model receives an ambiguous question and initial default interpretations as input and generates missing interpretations as output. Supervision comes from comparing SQL queries for default interpretations with gold SQL queries to identify which disambiguations are not captured.

particular, we know which gold SQL queries are associated with each reference interpretation. Similarly, we can obtain SQL queries predicted by a text-to-SQL parser for the initial set of default interpretations. By executing these queries and comparing their results, we determine which interpretations match and are thus covered by the initial set. Reference interpretations absent from this set serve as training targets for the infilling model which also explicitly indicates when all interpretations are covered by the initial set, and there is nothing to add (the target output in this case is the sentence: "All interpretations are covered.") Figure 3 illustrates the annotation process just described.

2.5 Discussion

The strength of our proposal lies in its modular design, consisting of three separate components: default interpretation generation, infilling, and text-to-SQL parsing. Only infilling requires training, since we resort to existing pre-trained models for generating interpretations and parsing these into SQL. This plug-and-play functionality allows us to experiment with different LLMs and text-to-SQL parsers, without retraining. Moreover, any improvements in the infilling module should in theory translate into improved coverage.

A potential limitation is that infilling requires reference interpretations and SQL queries, which may be difficult to come by. In the next section, we demonstrate how reference interpretations can be synthetically generated. In addition, the text-to-SQL parser may occasionally introduce errors, leading to duplicate interpretations in the final set, coming from the initial generator and infilling model. However, this redundancy can improve robustness by providing multiple formulations of the same meaning. Additionally, duplicates can be filtered in post-processing by comparing execution results. It is also important to note that the infilling does not filter any interpretations from the initial set; it can only add new ones or leave the set unchanged. This means that, in some cases, incorrect interpretations may persist, potentially reducing precision. However, we believe this is not a major concern in practice, as users can simply ignore irrelevant interpretations.

When questions are unambiguous, the initial set contains a single interpretation (or paraphrases with the same meaning, which can be filtered as duplicates), and the infilling model simply outputs "All interpretations are covered." This means that we do not need to explicitly determine whether a question is ambiguous or not, our approach naturally handles both cases.

3 Experimental Setup

3.1 Evaluation Datasets

We evaluate our approach on two recent benchmarks aiming to assess the capabilities of text-to-SQL parsers when faced with ambiguous questions: AmbiQT (Bhaskar et al., 2023) and Ambrosia (Saparina and Lapata, 2024a). Examples can be found in Appendix A.

AmbiQT builds upon the widely used Spider dataset (Yu et al., 2018). Specifically, ambiguity is injected by generating synonyms for column and table names (via ChatGPT and heuristically), by having tables with overlapping column names (which leads to `join` ambiguities), and by introducing columns which are aggregates of certain

values in addition to aggregating values with the `group-by` clause of the `SELECT` statement. AmbiQT inherits from Spider diverse SQL queries and real-world databases, but also introduces redundancy as the databases can contain duplicates (e.g., identical columns "singer" and "performer"). It features two types of ambiguity, namely lexical ambiguity (based on ambiguous column and table names) and structural ambiguity in SQL queries (due to `join` and pre-computed aggregates). While the AmbiQT test set contains 3K examples, we filter out those that yield empty execution results, leaving a final set of 1.8K non-trivial examples.

Ambrosia showcases three different types of ambiguity, namely scope ambiguity, attachment ambiguity, and vagueness. It contains human-written questions, SQL queries either manually written or based on templates, and synthetically generated databases. In addition to ambiguous questions and corresponding SQL queries, Ambrosia has explicit interpretations in natural language. The official test set has 1K ambiguous questions and 2K interpretations. We use the ambiguous subset in all experiments unless otherwise stated. Notably, AmbiQT and Ambrosia handle vagueness differently: AmbiQT consistently provides two valid interpretations, whereas Ambrosia may provide up to three interpretations for vague questions.

3.2 Evaluation Metrics

Our primary metric is **Full Interpretation Coverage**, which measures the proportion of examples where *all* valid SQL interpretations are present in the output. This metric aligns with the "All Found" metric from Ambrosia and "BothInTopK" from AmbiQT. Full Coverage is the most informative and relevant metric for our task, as it directly measures the system's ability to recover the full set of available interpretations. We also report **Single Interpretation Coverage**, i.e., the proportion of examples where at *least one* valid interpretation is found ("EitherInTopK" from AmbiQT), as well as **Precision** and **Recall**. We compare SQL queries based on their execution results.

3.3 Training Data

To train the infilling model, we need three components: (1) generated default interpretations, (2) reference SQL queries, and (3) reference interpretations to identify missing interpretations from the default set. It is straightforward to elicit default interpretations from an LLM and SQL queries are available in many text-to-SQL datasets. To collect reference interpretations, we propose a novel approach that extends the data generation process in AmbiQT (Bhaskar et al., 2023).

AmbiQT relies on ChatGPT to generate two synonyms for a selected column or table in an SQL query which naturally renders the corresponding question vague. The original SQL mentions are replaced with their synonyms, resulting in two gold SQL queries. Building upon this approach, we use these synonyms and prompt an LLM (with three in-context examples) to replace the original mentions in the *questions* with their synonyms. Our experiments use the instruction tuned Llama 3.1 8B (Dubey et al., 2024). The full prompt is provided in Appendix B. We verify the quality of the synthetically generated interpretations by attempting to generate correct SQL queries using a specialized code generation LLM (instruction-tuned Qwen2.5-Coder 32B; Hui et al. 2024). We only accept examples where both interpretations succeed within five attempts.

This approach is particularly effective for AmbiQT and other datasets based on Spider (Yu et al., 2018) as it contains many questions which directly mention table and column names (Deng et al., 2021; Suhr et al., 2020; Gan et al., 2021), making synonym substitution natural and fail-safe. We generate interpretations for approximately 5K examples from a subset of the Spider training data. Although our experiments focus primarily on AmbiQT, the proposed interpretation generation method can be applied to other domains and datasets (together with AmbiQT's approach of generating ambiguous examples using column and table synonyms).

3.4 Implementation Details

The first component of our method is to generate default interpretations for an ambiguous question. We design a zero-shot prompt that uses the provided database and question to generate interpretations (see Appendix C). Here and throughout, we represent the database as an SQL dump. For our experiments, we use instruction-tuned Llama-3.1 8B, as it produced the most coherent interpretations among similarly sized models.

The second component is the infilling model which takes the database, question, and default interpretations as input, and outputs missing interpretations. As our infilling model, we train a LoRA adapter (Hu et al., 2022) on top of the instruction-

End-to-End Text-to-SQL	AmbiQT				Ambrosia			
	Single	Full	Recall	Precision	Single	Full	Recall	Precision
0-shot Prompt	62.3	12.3	37.3	58.1	29.4	0.9	15.0	21.9
3-shot Prompt	44.3	10.9	27.6	33.0	35.7	1.3	17.5	21.3
SFT	82.1	**63.2**	72.7	74.3	38.0	0.4	20.0	29.4

Disambiguate-and-Parse	AmbiQT				Ambrosia			
	Single	Full	Recall	Precision	Single	Full	Recall	Precision
Interp. Prompt	81.8	26.0	53.4	31.8	81.9	16.9	49.0	29.1
w. Self-Correction	77.4	13.9	45.7	46.0	65.7	5.9	34.5	29.4
Gold Interp. SFT	87.4	61.2	**74.1**	**79.6**	62.6	0.3	32.1	**49.5**
Ours	**92.3**	53.2	72.8	38.2	**84.4**	**18.8**	**51.9**	24.2

Table 1: **Single** and **Full** Interpretation Coverage, **Recall** and **Precision** (%) on AmbiQT and Ambrosia datasets.

tuned Llama-3.1 8B (see Appendix D).

The final component is a text-to-SQL model for which we select the instruction-tuned Qwen2.5-Coder-32B, a specialized model for code generation. We also use this model to match and identify missing interpretations (Section 2.4). A zero-shot prompt for unambiguous text-to-SQL parsing is provided in Appendix E. Thanks to our modular structure, any component of our system can be easily replaced with a more powerful or more efficient model if needed.

4 Experimental Results

It is better to disambiguate first and parse later both in in-domain and out-of-domain settings. Table 1 presents our main experimental results on AmbiQT and Ambrosia. We compare our approach with both prompt-based and fine-tuning baselines. Note that for methods requiring fine-tuning, AmbiQT represents in-domain evaluation, whereas evaluation on Ambrosia is out of domain. All methods use the same model, instruction-tuned Llama-3.1 8B, and those that require training are fine-tuned using a LoRA adapter on the same AmbiQT subset, augmented with interpretations.

Table 1 is split into two sections. The first one presents end-to-end approaches, which attempt to *directly* predict multiple SQL queries for ambiguous questions. We report results for zero-shot prompting, few-shot prompting, and end-to-end fine-tuning (SFT). For prompting, we follow Saparina and Lapata (2024a) and explicitly instruct the model to generate multiple SQL queries if the question is ambiguous. For few-shot prompting, we sample 3 random examples from the corresponding dataset. The second section in Table 1 lists methods which use natural language interpretations to disambiguate first and then rely on text-to-SQL parsing of unambiguous questions. We report results for generating all possible interpretations through LLM prompting (which corresponds to our method without infilling), applying self-correction to this approach (details are in Appendix F), a fine-tuning method which is trained to generate reference interpretations instead of SQL queries, and our proposal which uses infilling to augment the set of default interpretations. We use an instruction-tuned Qwen2.5-Coder 32B for text-to-SQL generation.

As can be seen in Table 1, prompting (0-shot, 3-shot) performs poorly on ambiguous questions, which is consistent with the findings of Saparina and Lapata (2024a). Fine-tuning achieves excellent results on in-domain evaluation (AmbiQT), but fails to generalize to Ambrosia, which suggests that the model overfits specific patterns in AmbiQT. Interpretation generation (via prompting) shows promising results on both datasets compared to end-to-end methods. Self-correction increases precision but also eliminates some valid interpretations, suggesting that the filtering task is non-trivial and requires further research. Fine-tuning on interpretations behaves very similarly to fine-tuning on SQL queries, but provides higher precision and single-interpretation coverage, which suggests it is more accurate in predicting *at least one* correct interpretation.

Interpretation generation with infilling further improves single and full interpretation coverage

	AmbiQT		Ambrosia	
Interpretations	Single	Full	Single	Full
Llama 3.1 8B	81.8	26.0	81.9	16.9
w. infilling	92.3	53.2	84.4	18.8
Qwen 2.5 7B	65.0	25.9	77.4	12.4
w. infilling	88.7	48.6	80.0	13.6
Gemma 2 9B	77.3	17.3	69.3	2.9
w. infilling	91.1	55.9	76.6	5.3
Gold Interp.	—	—	75.2	49.0

Table 2: **Single** and **Full** Interpretation Coverage (%) on AmbiQT and Ambrosia when comparing different models for default interpretation generation, both with and without infilling. We also include an upper bound obtained using gold interpretations from Ambrosia.

	AmbiQT		Ambrosia	
Text-to-SQL Model	Single	Full	Single	Full
Qwen2.5-Coder 32B	92.3	53.2	84.4	18.8
Qwen2.5-Coder 7B	84.7	40.2	72.2	10.9

Table 3: **Single** and **Full** Interpretation Coverage (%) on AmbiQT and Ambrosia datasets when different text-to-SQL models are used.

Error Type	AmbiQT	Ambrosia
Redundant interpr.	23.3	20.0
Overly general interpr.	16.7	20.0
Interpr. not matching intent	26.7	30.0
SQL generation errors	33.3	30.0

Table 4: Error types (%) on AmbiQT and Ambrosia.

on both datasets. While it does not achieve the highest full coverage on AmbiQA, it substantially improves over end-to-end prompting reaching 53%, while delivering the best single interpretation coverage of 92%. It effectively generalises to new domains and ambiguity types, showing the best results on Ambrosia among all methods across three out of four evaluation metrics. However, the full coverage on Ambrosia is still relatively low (at 19%). Ambrosia is a challenging benchmark on its own but it is also possible that some interpretations are missed due to annotation differences between the two datasets.

The results in Table 1 show that recall closely mirrors full coverage, offering little additional insight. Precision, on the other hand, can be misleadingly inflated when systems rarely predict all valid interpretations at once. For this reason, in subsequent experiments, we only report full and single interpretation coverage as the most informative metrics. See Appendix G for additional results.

Infilling boosts performance across interpretation generation models. We next analyze key components of our approach through ablation studies. Table 2 focuses on the first stage of our method, namely disambiguation. We compare default interpretations from three different instruction-tuned models of similar size: Qwen-2.5 7B (Yang et al., 2024), Llama-3.1 8B, and Gemma-2 9B (Riviere et al., 2024). We observe that Llama-3.1 8B provides the best default interpretations and our infilling model improves upon all interpretations, irrespective of the generation model.

Text-to-SQL parsing is hard even with gold interpretations! To provide an upper bound for our approach, we use gold interpretations from Ambrosia and evaluate how well the text-to-SQL parser performs when all interpretations are correct. The results are, as expected, significantly higher, indicating room for improvement in interpretation generation. However, since the full coverage reaches only 49%, a substantial number of errors may come from the unambiguous text-to-SQL parsing alone.

Table 3 compares our text-to-SQL model, the instruction-tuned Qwen-2.5 Coder 32B, with its smaller 7B variant. Results decrease substantially when the weaker text-to-SQL model is used. As our approach is modular, we anticipate our results would improve with a better text-to-SQL parser.

Generalisation to new ambiguity types remains an open challenge. To better understand our system's limitations, we performed a manual error analysis on 50 randomly sampled examples with errors (30 from AmbiQT and 20 from Ambrosia). Table 4 summarizes the distribution of these errors. Redundant interpretations refer to near-paraphrases that differ slightly in structure. If their SQL queries yield the same result, we typically filter them out; however, minor variations (e.g., extra columns) can lead to execution mismatches, which are considered errors. Additionally, some interpretations remain overly vague, failing to fully disambiguate the input. These two error categories mostly originate from default interpretations (75%). We also observed SQL errors for valid interpretations,

End-to-End		*Disambiguate-and-Parse*	
0-shot Prompt	35.9	Interp. Prompt	75.1
3-shot Prompt	43.0	Gold Interp. SFT	56.9
SFT	44.5	Ours	**77.9**

Table 5: Coverage (%) on **unambiguous** subset of Ambrosia test set (single coverage is the same as full coverage since each question has only one interpretation).

with 57% stemming from infilled interpretations, including incorrect joins and column mismatches. Syntax errors occurred in up to 20% of cases.

Most AmbiQT failures are due to join and aggregate ambiguities, which are not represented in the training data. Ambrosia presents even greater challenges with scope and attachment ambiguities, which remain particularly difficult.

Disambiguation also improves coverage for unambiguous questions. While our approach does not specifically target unambiguous examples, in Table 5 we examine how different methods perform when the input question is unambiguous, i.e., when it has one interpretation and one SQL query. In this case, single and full coverage are the same and we simply show whether the gold SQL query was found. Note that we do not penalize additional queries in the answer.

As Table 5 shows, methods which disambiguate first, show better results by a wide margin (compared to end-to-end systems), with our proposed method achieving the best score of 77.9%. Explicit disambiguation serves as an intermediate representation of the question, thereby clarifying its meaning. Our results further demonstrate that generating a single interpretation is less challenging than handling multiple valid interpretations simultaneously (compare Table 1).

The infilling module is robust to different interpretation types. We now evaluate our approach when the infilling model is trained on Ambrosia. Specifically, we re-split Ambrosia to use 80% for training (1K ambiguous and 2K unambiguous examples), 10% for validation, and 10% for testing (131 ambiguous examples). All databases in the test set are unseen during training, similar to the original split. Note that in this setting the infilling model is trained on human-written interpretations which Ambrosia provides. As a result, Ambrosia becomes the in-domain test set, whereas AmbiQT represents out-of-domain evaluation.

	AmbiQT		Ambrosia*	
Method	Single	Full	Single	Full
Text-to-SQL FT	56.1	10.2	77.1	**66.4**
Interp. Prompt	81.8	26.0	81.9	16.9
Gold Interp. SFT	81.9	4.1	71.0	38.9
Ours	**88.0**	**30.0**	**87.8**	30.5

Table 6: **Single** and **Full** Interpretation Coverage (%) on AmbiQT and Ambrosia* test sets. Symbol * denotes our split of the Ambrosia test set into training, development, and testing. Models are fine-tuned on Ambrosia* train.

Table 6 shows results for an end-to-end fine-tuned text-to-SQL model, and three variants of the disambiguate first parse later framework: a model fine-tuned on gold (Ambrosia) interpretations, a prompt-based model that generates default interpretations without infilling, and the full model with infilling. The latter achieves the best single and full interpretation coverage on the out-of-domain AmbiQT test set and the highest single interpretation coverage on Ambrosia. However, the full interpretation coverage on Ambrosia is much lower than the end-to-end fine-tuned baseline. Manual examination of the predicted interpretations revealed that some correct interpretations are parsed incorrectly during the text-to-SQL stage, which relies on zero-shot prompting and may not capture dataset-specific conventions. This finding is supported by the results of the model fine-tuned on interpretations, which also performs significantly worse than end-to-end fine-tuning. Finally, we found that training on both datasets leads to results similar to the in-domain setting, see Appendix G for details.

5 Related Work

Ambiguity Resolution in NLP Numerous studies have focused on ambiguity in natural language tasks using strategies like generating multiple answers (Min et al., 2020), asking clarification questions (Lee et al., 2023; Zhang et al., 2024), and estimating uncertainty (Cole et al., 2023). Similar to our work, Sun et al. (2023) use iterative prompting to refine and generate alternative interpretations in the context of question answering, while Kim et al. (2024) first detect ambiguous questions and then resolve them through clarification requests.

Ambiguity in Semantic Parsing Ambiguity has been studied across semantic parsing tasks, from code generation (Li et al., 2023; Mu et al., 2024)

to λ-calculus translation (Rasmussen and Schuler, 2020), and logical form prediction (Stengel-Eskin et al., 2024). In the domain of text-to-SQL parsing, recent work has emphasized the fact that benchmarks often overlook ambiguity by providing single interpretations (Floratou et al., 2024; Pourreza and Rafiei, 2023). Existing approaches focus on detecting column ambiguity through counterfactual examples (Wang et al., 2023), special-purpose decoding (Bhaskar et al., 2023), and resolving ambiguity through clarification questions (Dong et al., 2024). Our work uses explicit disambiguation before parsing and thus extends to different types of ambiguities, question styles, and database formats.

Intermediate Representations in Text-to-SQL Intermediate representations are commonly used to bridge the gap between natural language and database queries. Several approaches decompose complex questions into a sequence of simpler operations expressed in natural language (Wolfson et al., 2022; Saparina and Osokin, 2021), modular execution plans (Eyal et al., 2023), or simplify the parsing task by augmenting questions with SQL keywords that mention necessary computation steps (Liu and Tan, 2024; Caferoglu and Ulusoy, 2024). Building on this work, we also use intermediate representations to make implicit information explicit but focus on resolving ambiguity.

Learning to Correct LLM Outputs More recently, various approaches have been proposed to correct systematic biases in LLM outputs. For example, Ji et al. (2024) propose a model-agnostic module that learns correctional residuals between preferred and dispreferred outputs. Welleck et al. (2023) use a corrector model to iteratively review imperfect generations from a base model. Similarly, critique generators can be developed using reinforcement learning (Wadhwa et al., 2024a) or through fine-grained feedback (Wadhwa et al., 2024b). Such correction approaches are most effective when guided by external tools (Kamoi et al., 2024). We follow this paradigm using a specialized infilling model to correct systematic LLM biases towards certain interpretations and validate our output through SQL execution.

6 Conclusion

In this work, we present a novel approach for handling ambiguity in text-to-SQL semantic parsing. We first disambiguate questions by explicitly verbalising their interpretations and then leverage LLM capabilities to predict all valid SQL queries. A generator (LLM) provides an initial set of default interpretations, which are then augmented with a specialized infilling model. We propose a method for training this model based on automatic annotations obtained by comparing SQL query execution results rather than natural language interpretations.

Our experimental results on AmbiQT and Ambrosia demonstrate the effectiveness of our approach. Our method achieves the highest single interpretation coverage on both datasets and maintains consistent full interpretation coverage in both in-domain and out-of-domain evaluation. However, generating all valid interpretations remains challenging. Future work could explore ways to improve ambiguity handling through better processing of database structure or using query execution as a signal for both training and test-time search.

7 Limitations

Our approach relies on reference interpretations and SQL queries for training the infilling model. While we show how to generate synthetic interpretations, scaling to new domains might be challenging. The sequential pipeline can propagate errors, with mistakes in disambiguation affecting text-to-SQL parsing and parser errors leading to incorrect SQL predictions. Duplicates in generated interpretations can help capture valid meanings through different wording but at the cost of longer output. The generator can produce incorrect interpretations that are currently not filtered. The infilling model can also miss valid interpretations that are underrepresented in the training data.

Additionally, generating and parsing multiple interpretations requires more computation than single-stage approaches. Although our approach handles both ambiguous and unambiguous questions, it occasionally provides multiple interpretations for unambiguous requests. Future work should focus on improving precision through better filtering and interpretation validation.

8 Acknowledgements

We thank the meta-reviewer and anonymous reviewers for their constructive feedback. We gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council (grant EP/W002876/1).

References

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-SQL generation under ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074, Singapore. Association for Computational Linguistics.

Hasan Alp Caferoglu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *ArXiv*, abs/2409.16751.

Jeremy Cole, Michael Zhang, Daniel Gillick, Julian Eisenschlos, Bhuwan Dhingra, and Jacob Eisenstein. 2023. Selectively answering ambiguous questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 530–543, Singapore. Association for Computational Linguistics.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, Patrick Ng, and Zhiguo Wang. 2024. Practiq: A practical conversational text-to-sql dataset with ambiguous and unanswerable queries. *ArXiv*, abs/2410.11076.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony S. Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 510 others. 2024. The llama 3 herd of models. *ArXiv*, abs/2407.21783.

Ben Eyal, Moran Mahabi, Ophir Haroche, Amir Bachar, and Michael Elhadad. 2023. Semantic decomposition of question and SQL for text-to-SQL parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13629–13645, Singapore. Association for Computational Linguistics.

Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... not! In *14th Conference on Innovative Data Systems Research, CIDR*.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515, Online. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Neel Jain, Ping yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. NEFTune: Noisy embeddings improve instruction finetuning. In *The Twelfth International Conference on Learning Representations*.

Jiaming Ji, Boyuan Chen, Hantao Lou, Donghai Hong, Borong Zhang, Xuehai Pan, Juntao Dai, Tianyi Qiu, and Yaodong Yang. 2024. Aligner: Efficient alignment by learning to correct. In *Proceedings of the 38st Annual Conference on Neural Information Processing Systems*.

Gaurav Kamath, Sebastian Schuster, Sowmya Vajjala, and Siva Reddy. 2024. Scope ambiguities in large language models. *Preprint*, arXiv:2404.04332.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sanggoo Lee, and Taeuk Kim. 2024. Aligning language models to explicitly handle ambiguity. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1989–2007, Miami, Florida, USA. Association for Computational Linguistics.

Dongryeol Lee, Segwang Kim, Minwoo Lee, Hwanhee Lee, Joonsuk Park, Sang-Woo Lee, and Kyomin Jung. 2023. Asking clarification questions to handle ambiguity in open-domain QA. In *Findings of the Association for Computational Linguistics: EMNLP*

2023, pages 11526–11544, Singapore. Association for Computational Linguistics.

Haau-Sing (Xiaocheng) Li, Mohsen Mesgar, André Martins, and Iryna Gurevych. 2023. Python code generation by asking clarification questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14287–14306, Toronto, Canada. Association for Computational Linguistics.

Alisa Liu, Zhaofeng Wu, Julian Michael, Alane Suhr, Peter West, Alexander Koller, Swabha Swayamdipta, Noah Smith, and Yejin Choi. 2023. We're afraid language models aren't modeling ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 790–807, Singapore. Association for Computational Linguistics.

Xiping Liu and Zhao Tan. 2024. Keyinst: Keyword instruction for improving sql formulation in text-to-sql. *Preprint*, arXiv:2411.00788.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5783–5797, Online. Association for Computational Linguistics.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, Chenxue Wang, Shichao Liu, and Qing Wang. 2024. Clarifygpt: Empowering llm-based code generation with intention clarification.

Mohammadreza Pourreza and Davood Rafiei. 2023. Evaluating cross-domain text-to-SQL models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1601–1611, Singapore. Association for Computational Linguistics.

Nathan Rasmussen and William Schuler. 2020. A corpus of encyclopedia articles with logical forms. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1051–1060, Marseille, France. European Language Resources Association.

Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan Ferret, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *ArXiv*, abs/2408.00118.

Irina Saparina and Mirella Lapata. 2024a. Ambrosia: A benchmark for parsing ambiguous questions into database queries. *Proceedings of the 38th Annual Conference on Neural Information Processing Systems*.

Irina Saparina and Mirella Lapata. 2024b. Improving generalization in semantic parsing by increasing natural language variation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1178–1193, St. Julian's, Malta. Association for Computational Linguistics.

Irina Saparina and Anton Osokin. 2021. SPARQLing database queries from intermediate question decompositions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8984–8998, Online and Punta Cana, Dominican Republic. 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 12th International Conference on Learning Representations*.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

Weiwei Sun, Hengyi Cai, Hongshen Chen, Pengjie Ren, Zhumin Chen, Maarten de Rijke, and Zhaochun Ren. 2023. Answering ambiguous questions via iterative prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7669–7683, Toronto, Canada. Association for Computational Linguistics.

Manya Wadhwa, Xinyu Zhao, Junyi Jessy Li, and Greg Durrett. 2024a. Learning to refine with fine-grained natural language feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12281–12308, Miami, Florida, USA. Association for Computational Linguistics.

Somin Wadhwa, Adit Krishnan, Runhui Wang, Byron C Wallace, and Luyang Kong. 2024b. Learning from natural language explanations for generalizable entity matching. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 6114–6129, Miami, Florida, USA. 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.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*.

Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly supervised text-to-SQL parsing through question decomposition. In *Findings of the Association for Computational Linguistics: NAACL*

2022, pages 2528–2542, Seattle, United States. Association for Computational Linguistics.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 24 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

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. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Michael J.Q. Zhang, W. Bradley Knox, and Eunsol Choi. 2024. Modeling future conversation turns to teach llms to ask clarifying questions. *ArXiv*, abs/2410.13788.

A AmbiQT and Ambrosia Examples

We provide samples from the AmbiQT (Bhaskar et al., 2023) and Ambrosia (Saparina and Lapata, 2024a) evaluation sets.

AmbiQT Examples

Column Ambiguity	
Database	`singer: singer_id, artist_name, performer_name, song_name, age, country ...`
Question	Show **name**, country, age for all singers ordered by age from the oldest to the youngest.
SQL Query 1	`SELECT artist_name, country, age FROM singer ORDER BY age DESC`
SQL Query 2	`SELECT performer_name, country, age FROM singer ORDER BY age DESC`
Table Ambiguity	

Database	`Canine_Breeds: breed_name, breed_code`
	`Dog_Types: breed_name, breed_code`
	`dogs : owner_id, breed_code ...`

Question	What is the name of the **breed** with the most dogs?
SQL Query 1	`SELECT T1.breed_name FROM Canine_Breeds T1 JOIN dogs T2 ... ORDER BY COUNT(*) DESC LIMIT 1`
SQL Query 2	`SELECT T1.breed_name FROM Dog_Types T1 JOIN dogs T2 ... ORDER BY COUNT(*) DESC LIMIT 1`
Join Ambiguity	
Database	`country: surfacearea, indepyear, name, population, code ...` `country_surfacearea: surfacearea , code ...`
Question	What are the name, independence year, and **surface area of the country** with the smallest population?
SQL Query 1	`SELECT T1.name, T2.surfacearea, T1.indepyear FROM country T1 JOIN country_surfacearea T2 ON T1.code = T2.code ORDER BY population LIMIT 1`
SQL Query 2	`SELECT name, surfacearea, indepyear FROM country ORDER BY population LIMIT 1`
Precomputed Aggregates	
Database	`show: result, attendance, show_id ...` `show_attendance: avg_attendance, sum_attendance ...`
Question	What is the **average attendance** of shows?
SQL Query 1	`SELECT AVG(attendance) FROM show`
SQL Query 2	`SELECT avg_attendance FROM show_attendance`

Ambrosia Examples	
Scope Ambiguity	
Database	**Spa**: spa_id, name, address **Treatments**: treatment_id, name ... Spa_Treatments: spa_treatment_id, spa_id, treatment_id
Question	What treatment options do we have **for each spa?**
SQL Query 1	SELECT T.name FROM Spa_Treatments ST JOIN Treatments T ... JOIN Spa S ... **GROUP BY ST.treatment_id HAVING COUNT(DISTINCT ST.spa_id) = (SELECT COUNT(*) FROM Spa)**
SQL Query 2	SELECT **S.name, T.name** FROM Spa_Treatments ST JOIN Treatments T ... JOIN Spa S ...
Attachment Ambiguity	
Database	EventSpaces: id, Name, **Event_Space**, Capacity, Address, ContactInfo ...
Question	List all banquet halls and conference rooms **with a 200 person capacity**.
SQL Query 1	SELECT ES.Name FROM EventSpaces ES WHERE **(**ES.Event_Space = "Banquet Hall" OR ES.Event_Space = "Conference Room"**)** AND ES.Capacity = 200
SQL Query 2	SELECT ES.Name FROM EventSpaces ES WHERE ES.Event_Space = "Banquet Hall" **OR** ES.Event_Space = "Conference Room" **AND** ES.Capacity = 200
Vagueness	
Database	hospitals: id, name, **city**, **neighborhood**, phone_number ...
Question	**Where** are the clinics located?
SQL Query 1	SELECT **neighborhood** FROM hospitals WHERE name LIKE '%Clinic%'
SQL Query 2	SELECT **city** FROM hospitals WHERE name LIKE '%Clinic%'
SQL Query 3	SELECT **neighborhood**, **city** FROM hospitals WHERE name LIKE '%Clinic%'

AmbiQT contains column and table ambiguities, join ambiguity, and ambiguity due to precomputed aggregates. Ambrosia covers scope and attachment ambiguities, and vagueness. Note that column and table ambiguities from AmbiQT correspond to vagueness in Ambrosia, although they differ in the number of gold queries provided (2 in AmbiQT versus 3 in Ambrosia).

AmbiQT is publicly available under the MIT license and Ambrosia is under the CC BY 4.0 license.

B Prompt for Annotating AmbiQT with Interpretations

We use the following prompt with three in-context examples to generate natural language interpretations for ambiguous questions in AmbiQT:

```
Your task is to rewrite the question using a
given word or phrase.

Examples:
Question: Show titles of songs and names of
singers.
Please rewrite using "stage name":
Give me titles of songs and stage names of
singers.

Question: Show the name of the conductor that
has conducted the most number of orchestras.
Please rewrite using "director":
List the name of the director who has
conducted the most number of orchestras.

Question: Return the id of the document with
the fewest paragraphs.
Please rewrite using "passages":
What is the id of the document with the
fewest passages?

Please provide rewritten question for the
following instance. Do not add any
explanation or description, output only the
rewritten question.

Question:  ...
Please rewrite using ...
```

C Prompt for Default Interpretation Generation

The following prompt is used to generate default interpretations:

```
You are tasked with analyzing questions and
providing their possible interpretations.
The questions are related to database
queries and may be ambiguous or unambiguous.

Your task:
- List every distinct way the question could
be understood
- Be thorough and consider all possible
meanings
- Explore different ways the question could
be interpreted
- Don't limit yourself to obvious
interpretations

Important:
- List each interpretation on a separate line
- Do not include explanations or reasoning
- Focus on semantically different
interpretations
- Be specific and precise in wording
```

```
Given the following database context:
...

Provide interpretations for this question:
...
```

D Interpretation Infilling Details

We use the following instructions for the model:

```
The task is to review the provided context,
question, and existing interpretations, and
determine if any additional interpretations
are missing. If there are missing
interpretations, list them on separate lines
without explanations. If all interpretations
have already been covered, simply state:
"All possible interpretations are covered."

Given the following context: ...

Question: ...

Existing interpretations: ...

Provide any missing interpretations or
confirm that all possible interpretations
are covered.
```

We fine-tune the instruction-tuned Llama 3.1 8B model using LoRA (Hu et al., 2022) with rank 16 ($\alpha = 16$) and NEFTune (Jain et al., 2024) (noise $\alpha = 5$). The model is trained for 15 epochs using a cosine learning rate schedule with an initial learning rate of 5e-5, weight decay of 0.01, and a warmup ratio of 0.01. Training is performed on a single NVIDIA A100 GPU with a batch size of 8 and gradient clipping at 0.3. The total time for training and evaluation of one run is under 10 hours. We sample 10% of the training data as development set and select the best-performing model from a single run for final evaluation.

We apply the same fine-tuning procedure to all comparison methods, i.e., end-to-end text-to-SQL fine-tuning and fine-tuning to predict interpretations. We observe that these methods tend to overfit with more epochs and thus reduce the number of training epochs to 5.

E Text-to-SQL Parsing

To generate SQL queries for unambiguous questions (or interpretations), we use the following prompt across all text-to-SQL tasks, including AmbiQT annotation validation, evaluation of baseline models with interpretations, and our approach:

```
The task is to write SQL queries based on the
provided questions in English. Questions can
take the form of an instruction or command.
Do not include any explanations, and do not
select extra columns beyond those requested
in the question.

Given the following SQLite database schema:
...

Answer the following: ...
```

F Self-Correction Prompt

We compare our approach against a self-correction method where the LLM is prompted to review the generated interpretations (Appendix C), choose the valid ones, and add any missing interpretations. The prompt is shown below:

```
The task is to review the provided context,
question, and candidate interpretations, and
based on this information provide the
interpretations that accurately reflect the
meaning (or one of the possible meanings) of
the question. If any of the candidate
interpretations are correct, provide them as
a list of interpretations. If there are
missing interpretations, provide them as
well. Avoid providing interpretations that
are incorrect or duplicates. Do not provide
any explanations.

Given the following context: ...

Question: ...

Candidate interpretations: ...

Provide the interpretations that accurately
reflect the meaning (or one of the possible
meanings) of the question.
```

G Additional Results

Comparison with a larger-sized model Table 7 extends Table 1 by including a larger baseline: the instruction-tuned Llama 3.1 70B model. While the 70B variant outperforms the 8B version overall, our system (based on the 8B model) still achieves higher coverage and recall. We expect that larger models could further improve results, however, this comes at a significantly higher computational cost.

Training on Ambrosia* Table 8 extends Table 6 with recall and precision. Consistent with earlier findings, our approach achieves the highest coverage and recall on the AmbiQT test set, demonstrating strong out-of-domain generalization.

Llama 3.1 70B	AmbiQT				Ambrosia			
	Single	Full	Recall	Precision	Single	Full	Recall	Precision
0-shot Prompt	76.6	26.5	51.5	65.8	32.3	0.0	14.1	32.0
3-shot Prompt	75.7	28.0	51.9	**69.2**	57.5	3.9	28.1	**50.9**

Llama 3.1 8B	AmbiQT				Ambrosia			
	Single	Full	Recall	Precision	Single	Full	Recall	Precision
0-shot Prompt	62.3	12.3	37.3	58.1	29.4	0.9	15.0	21.9
3-shot Prompt	44.3	10.9	27.6	33.0	35.7	1.3	17.5	21.3
Ours	**92.3**	**53.2**	**72.8**	38.2	**84.4**	**18.8**	**51.9**	24.2

Table 7: Comparison of Llama 3.1 70B, Llama 3.1 8B and our approach on AmbiQT and Ambrosia datasets. Our approach is based on Llama 3.1 8B and **fine-tuned on AmbiQT train**.

	AmbiQT				Ambrosia*			
Method	Single	Full	Recall	Precision	Single	Full	Recall	Precision
Text-to-SQL FT	56.1	10.2	33.2	52.7	77.1	**66.4**	**71.6**	**73.3**
Interp. Prompt	81.8	26.0	53.4	31.8	81.9	16.9	49.0	29.1
Gold Interp. SFT	81.9	4.1	43.0	**79.5**	71.0	38.9	53.7	61.5
Ours	**88.0**	**30.0**	**58.5**	35.3	**87.8**	30.5	59.7	26.3

Table 8: **Single** and **Full** Interpretation Coverage, **Recall** and **Precision** (%) on AmbiQT and Ambrosia* test sets. Models are **fine-tuned on Ambrosia* train**. Symbol * denotes our split of the Ambrosia test set into training, development, and testing.

	AmbiQT				Ambrosia*			
Method	Single	Full	Recall	Precision	Single	Full	Recall	Precision
Text-to-SQL FT	81.0	**63.1**	72.1	73.3	83.2	**69.5**	**76.2**	**78.1**
Gold Interp. SFT	86.2	60.4	**73.3**	**79.8**	75.6	37.4	55.0	65.1
Ours	**92.5**	54.0	73.2	38.8	**84.7**	29.8	57.1	28.8

Table 9: **Single** and **Full** Interpretation Coverage, **Recall** and **Precision** (%) on AmbiQT and Ambrosia* test sets. All models are **fine-tuned on both AmbiQT and Ambrosia* train**. Symbol * denotes our split of the Ambrosia test set into training, development, and test.

Training on AmbiQT and Ambrosia* Table 9 provides results when the models are trained on the maximum available data: the AmbiQT train set and re-split Ambrosia* training set. As both test sets are in-domain, we would not expect to see any advantages from our method in terms of full coverage. Nevertheless, it still provides the best single-interpretation coverage. Overall, the results in Table 9 are very similar to those obtained in the in-domain setting.