`SQLSpace`: A Representation Space for Text-to-SQL to Discover and Mitigate Robustness Gaps

Neha Srikanth[♠*] **Victor Bursztyn**[♣] **Puneet Mathur**[♣] **Ani Nenkova**[♣]
[♠]University of Maryland [♣]Adobe Research
nehasrik@umd.edu

Abstract

We introduce `SQLSpace`, a human-interpretable, generalizable, compact representation for text-to-SQL examples derived with minimal human intervention. We demonstrate the utility of these representations in evaluation with three use cases: *(i)* closely comparing and contrasting the composition of popular text-to-SQL benchmarks to identify unique dimensions of examples they evaluate, *(ii)* understanding model performance at a granular level beyond overall accuracy scores, and *(iii)* improving model performance through targeted query rewriting based on learned correctness estimation. We show that `SQLSpace` enables analysis that would be difficult with raw examples alone: it reveals compositional differences between benchmarks, exposes performance patterns obscured by accuracy alone, and supports modeling of query success.

1 Introduction

Systems tasked with translating a natural language utterance to an executable SQL query (text-to-SQL, or NL2SQL for short) are typically evaluated for their accuracy on one or more benchmarks such as BIRD (Li et al., 2024) or SPIDER (Lei et al., 2024). While these accuracies can be used to rank models on a leaderboard in a coarse-grained way, this evaluation paradigm obscures parts of evaluation that are useful for researchers and practitioners to know. It cannot answer naturally arising questions around dataset composition (*Why are the accuracies for the same model so different on the two benchmarks?*), open challenges for the field that future work must address (*What subsets of data are easy or difficult for all models?*), and explorations into performance and cost trade-offs (*Are there subsets of data on which cheaper models perform as well as more expensive models?*).

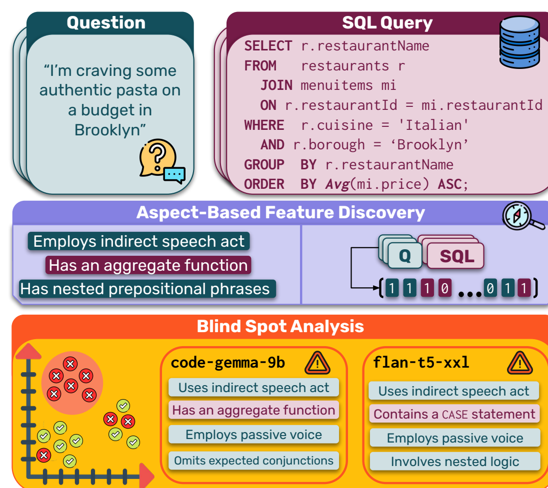


Figure 1: Our framework generates compact representations of NL2SQL examples by ingesting a dataset, discovering shared properties of dataset items in natural language, and evaluating these properties on examples to produce binary feature vectors. Clustering these feature vectors and examining a model's cluster-level accuracy reveals classes of examples that it systematically struggles with, called *blind spots*.

A deeper, finer-grained understanding of text-to-SQL examples could help us better understand model performance and benchmark composition, making it easy for practitioners to compare their own data to existing benchmarks in order to select the top-performing model on a benchmark that best reflects their own data. Similarly, such understanding would help researchers looking to build new robustness benchmarks better analyze existing gaps in datasets and systems in order to inform the design of new challenge evaluation sets.

To facilitate informed decision-making in scenarios like these, we introduce `SQLSpace`, a framework to increase visibility into benchmark composition and model behavior with minimal human interven-

[*] Work on the `SQLSpace` representation was completed during an internship at Adobe Research. Analysis of model performance for open-source models was done at the University of Maryland after the completion of the internship.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 1533–1559
November 4-9, 2025 ©2025 Association for Computational Linguistics

tion.[1] SQLSpace ingests one or more NL2SQL benchmarks and extracts human-interpretable features of examples (e.g. *"has an aggregate function"*) in order to construct generalized vector representations of any NL2SQL example (§3) in a largely automated manner.

We demonstrate the utility of these vector representations with three use cases that leverage the representations in different ways. First, we compare the distribution of examples in popular NL2SQL benchmarks (§4), allowing us to understand which particular dimensions each dataset uniquely evaluates. Then, we compare the strengths and weaknesses of 13 models with greater detail than in leaderboards by clustering examples across benchmarks that share similar features, and identifying "blind spots", or systematic classes of examples that a model struggles with (§5). We identify two clusters of universal blind spots and multiple clusters for which cheaper models can outperform expensive ones. Finally, show that learning a correctness estimator for a given model to inform rewrites of NL queries that are predicted to fail can improve accuracy (§6).

2 Background

Task and Evaluation. Text-to-SQL involves translating natural language question X to a valid SQL query Y that is executable over a database with schema $\mathcal{S}$. Along with knowledge of $\mathcal{S}$ and SQL, this task may require other domain knowledge or forms of linguistic and inferential reasoning (Gan et al., 2021b). Generated queries Y are typically evaluated by executing them and comparing the resulting output with that produced by executing the gold query (Li et al., 2024), and computing execution accuracy (EX), or the proportion of examples for which the results of the predicted and gold SQL queries match. While other metrics, such as PCM-F1 (Hazoom et al., 2021), have been proposed to award partial credit, we report EX as our primary evaluation metric due to its wider adoption in well-established benchmarks (Li et al., 2024; Lei et al., 2024).

Datasets. Datasets such as SPIDER (Yu et al., 2018; Lei et al., 2024) and BIRD-BENCH (Li et al., 2024) are general-purpose benchmarks designed to reflect realistic queries and use-cases, spanning several database schemas and domains. These benchmarks maintain leaderboards to facilitate efficient and standardized evaluation and are helpful in gauging model performance. While some benchmarks release accompanying analyses on the composition of examples (e.g. Yu et al. (2018) include SQL pattern coverage, or "difficulty" metadata), these statistics are not standardized across datasets (Appendix C). In turn, it is challenging to identify qualitative differences between benchmarks and, more importantly, understand the specific strengths and weaknesses of the models evaluated on them.

Robustness-oriented benchmarks help address this by targeting specific model weaknesses. Gan et al. (2021a) create SPIDER-SYN to measure paraphrastic robustness (Srikanth et al., 2024) after finding models vulnerable to synonym substitution (Utama et al., 2018; Ma and Wang, 2021). Other SPIDER variants incorporate domain knowledge (Gan et al., 2021b). Pi et al. (2022) study model robustness to table perturbations, and Chang et al. (2023) explore 17 perturbation types to both questions and SQL queries. These types of robustness benchmarks typically rely on hand-designed perturbations informed by human priors, requiring significant manual effort. Experts must identify systematic model failures and craft challenge sets targeting those errors, as in Chang et al. (2023). As models improve, this process resembles an iterative cycle where researchers (1) identify remaining model weaknesses, (2) design challenge sets to target those weaknesses, after which (3) models are optimized to saturate those challenge benchmarks. Not only is this process costly, but it risks overlooking robustness gaps for particular *combinations* of properties of NL2SQL examples, or more broadly, along other dimensions not explicitly included in these benchmarks. SQLSpace addresses these gaps in NL2SQL model robustness evaluation with minimal human intervention (Appendix A).

3 Representation Discovery

Understanding the composition of NL2SQL datasets offers both theoretical and practical benefits. It facilitates a deeper scientific understanding of the NL2SQL task by identifying requisite reasoning abilities as well as informed decision-making about model and benchmark selection for different use cases. We introduce a method, SQLSpace, to discover interpretable feature-based representations of NL2SQL examples which we use for downstream analysis on benchmarks (§4) and model performance (§5).

[1] https://github.com/nehasrikn/robust-sql.

3.1 Motivation

We motivate `SQLSpace` with a simplified thought experiment designed to illustrate the opacity of accuracy-based leaderboard evaluation (Figure 2). Consider a model M that scores 80% accuracy on benchmark B. Conventional NL2SQL leaderboards simply rank models by this accuracy, obscuring deeper performance characteristics of M. However, grouping examples according to shared properties (e.g., queries containing nested `SELECT` clauses or ambiguous entity references) and analyzing model predictions within and across these groups could reveal more about the strengths and weaknesses of M. For this thought experiment, we assume exclusive class membership, though we revisit this in §5.

Fig. 2 illustrates three scenarios that all maintain the 80% accuracy of M while exhibiting fundamentally different error distributions: (1) perfect accuracy on certain classes and complete failure on others, (2) uniform performance across all classes, and (3) mixed performance within individual classes. This breakdown helps us better understand M. Furthermore, the lower panel of Fig. 2 illustrates that example class distributions may vary significantly across benchmarks (§4). If benchmark B contains disproportionately many examples with nested clauses relative to other evaluation sets, this compositional bias could account for M's degraded performance on B, since the upper panel shows a scenario in which M systematically fails on those types of examples. The `SQLSpace` representations we build in this section help us automatically identify such shared properties, facilitating scalable benchmark and model analysis.

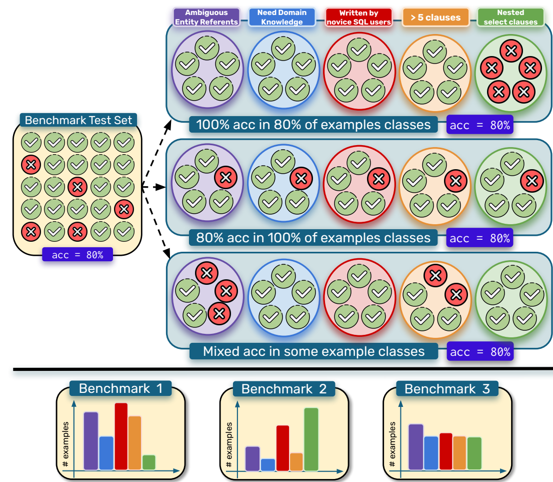


Figure 2: Aggregate accuracy may obscure important performance characteristics of models. Three models with identical 80% accuracy on benchmark B exhibit different error patterns across example classes, while varying class distributions across benchmarks can explain performance differences.

3.2 Representation Construction

`SQLSpace` ingests a set of examples and involves four steps to semi-automatically produce general-purpose vector representations of examples: (1) aspect-based example **description generation**, (2) **feature discovery from descriptions**, (3) **feature deduplication**, and lastly, (4) **representation construction (inference) on examples**.

Development Example Set. We use a portion of the development set from the UNITE corpus (Lan et al., 2023) (UNITE-DEV) for automatic feature discovery since it collates several public NL2SQL datasets. The portion we use includes a total of 10,697 examples from SPIDER (Yu et al., 2018), SQUALL (Shi et al., 2020), SPIDER-SYN (Gan et al., 2021a), CRITERIA2SQL (Yu et al., 2020), SPARC (Yu et al., 2019b), COSQL (Yu et al., 2019a), SPIDER-DK (Gan et al., 2021b), PARAPHRASEBENCH (Utama et al., 2018), KAGGLEDBQA (Lee et al., 2021), ACL-SQL (Kaoshik et al., 2021), SEOSS-QUERIES (Tomova et al., 2022), and FIBEN (Sen et al., 2020). Examples in these benchmarks span various natural language constructions, SQL patterns, and database schemas, making the compilation well-suited for discovering generalizable features. Though we select UNITE-DEV, our method is dataset-agnostic, and in practice can be applied to any example collection.

Step 1: Description Generation. Building a human-interpretable example representation requires identifying *diverse* and *meaningful* properties of NL2SQL examples that may not necessarily be captured by traditional embedding methods.[2]

For an example e with natural language question X and gold SQL query Y, a **`Describer`** model generates five descriptions of e that focus on different "aspects" related to the NL2SQL task:

1. **Syntax** (d_{syn}): Commentary on the linguistic syntax of X, including word order, grammatical relations, sentence structure, etc.
2. **SQL Syntax** ($d_{\text{sql-syn}}$): Covers elements of Y, including the structure and complexity of

[2] We experimented with instruction-tuned embedding models for generating representations, but found they focused on shallow features, even after masking schema-specific entities.

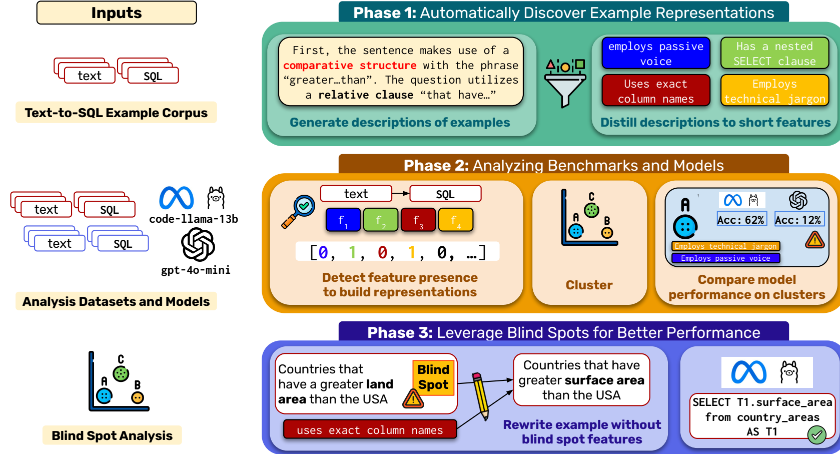


Figure 3: We discover representations of NL2SQL examples, and then use these representations in two applications: fine-grained analysis of models and benchmarks, and improving performance of models by rewriting NL questions to eliminate features associated with incorrect predictions.

Schema ($\mathcal{S}$): `employee: id, name, salary, dept_id | department: dept_id, name`

Question (X): *How many employees earn more than their department's average salary?*

SQL (Y): `SELECT COUNT(*) FROM employee e WHERE salary > (SELECT AVG(salary) FROM employee WHERE dept_id = e.dept_id)`

Aspect	#	Example Predicates (p)	Evaluated
Syntax	74 (40%)	omits expected conjunctions contains subordinate clauses contains nested conditionals	0 1 0
SQL Syntax	41 (22%)	includes a subquery has an aggregate function contains a CASE statement	1 1 0
Example Semantics	27 (14%)	involves nested logic has direct relationship requires domain knowledge	1 0 0
Pragmatics	33 (18%)	employs direct speech acts relies on conversational impl. exhibits minimal ambiguity	1 0 1
Database Reasoning	17 (9%)	uses exact column names requires commonsense reasoning mirrors schema structure	1 0 1
Total ($\mathcal{P}$)	**187** (100%)		

Table 1: Our final set of natural language predicates, $\mathcal{P}$, spans multiple aspects of NL2SQL examples. Building a representation of an example e involves evaluating each predicate p on e to produce binary feature vector.

the query as well as SQL keywords or other syntactic elements.

3. **Example Semantics** (d_{sem}): Specific commentary on the relationship between X and Y, such as parallel characteristics, types of reasoning required to map between X and Y, and other similarities and differences.
4. **Pragmatics** (d_{prag}): Discussion of pragmatic elements of X such as speech acts (Searle, 1975), adherence to Gricean maxims (Grice, 1975), uses of presupposition (Beaver, 1997) and implicature (Grice, 1975), and relevance. May also include commentary on any ambiguity (Wang et al., 2023a) or vagueness (Saparina and Lapata, 2024).
5. **Database Reasoning** (d_{db}): Commentary on the relationship between X and the database schema $\mathcal{S}$, including necessary reasoning required to map between entities in X and columns in $\mathcal{S}$, etc.

Inspired by aspect-based summarization (Angelidis and Lapata, 2018), these descriptions are designed to capture fine-grained details about examples in the input dataset. The choice of aspects above is informed by previous studies on the types of reasoning required to solve NL2SQL problems. We select `gpt-4o-2024-05-13` as our `Describer`[3] and generate these five aspect-based descriptions for all 10,697 examples in UNITE-DEV using Prompts D.1–D.5 (see Table 11 for example descriptions).

Step 2: Feature Discovery. Long-form descriptions generated by the `Describer` in Step 1 serve as a repository of aspect-related facts about an example. Examples similar in certain aspects may share phrases or sentences in their descriptions. To discover shared properties across descriptions, we use a module from the goal-driven explainable clustering pipeline in Wang et al. (2023b) designed to propose binary *natural language predicates* from descriptions. These predicates then serve as candidate features of an NL2SQL example (henceforth, we use *predicate* and *feature* interchangeably). Given a natural language predicate p and a NL2SQL example e, $p(e) = 1$ when e expresses p,

[3] `SQLSpace` can be run using any `Describer`, open-source or proprietary. Future work may explore the effects of different `Describers` on generated features.

and $p(e) = 0$ when e does not express p.

We run a predicate proposal module from Wang et al. (2023b) which ingests a collection of texts and a natural language goal, and has a **Proposer** LLM generate a list of candidate predicates, essentially performing "in-context clustering." Concretely, this process iteratively (1) samples a random subset of documents from the input collection and (2) prompts the `Proposer` to generate a structured list of n predicates, repeating the process for j iterations.

For each collection of aspect-based descriptions $\mathcal{C}_{\text{aspect}}$ (e.g. $\mathcal{C}_{syn} = \{d_{syn_1}, d_{syn_2}, ...\}$), the `Proposer` generates $n = 40$ predicates per iteration for $j = 5$ iterations (Appendix E). We use `gpt-3.5-turbo-0125` as our `Proposer` (see Table 1 for example predicates). **Note that describing examples in Step 1 before predicate proposal disentangles reasoning and understanding examples from the task of finding commonalities across examples in this step.** We also use `gpt-4o-2024-08-06` as an additional `Proposer` to diversify candidate predicates, as different models prioritize different parts of descriptions when proposing predicates (see Table 5 for counts of proposed candidate predicates).

Step 3: Feature Deduplication. The pool of candidate predicates for each aspect generated by the `Proposers` in the previous step contains duplicates (*"contains a nested* `JOIN`*"* and *"uses nested* `JOINS`*"*). We remove them by filtering out those with high token similarity as measured by Levenshtein distance,[4] and then manually removing any remaining paraphrases to ensure a clean set.[5] This process yields 187 general-purpose features across five aspects (Table 1), henceforth denoted as $\mathcal{P}$, which we exhaustively list in Table 10 as an artifact of our work that other researchers may leverage.

Step 4: Example Representation Construction. We now have a set of predicates that can serve as descriptive features of NL2SQL examples and that span all components of the example: the natural language question X, its relationship to the schema $\mathcal{S}$, and the corresponding SQL query Y. We create a vector representation of an example e by eliciting binary judgments from a predicate **Evaluator** with Prompts F.2–F.5 on $\{p(e) \mid p \in \mathcal{P}\}$ (Wang et al., 2023b), yielding a $|\mathcal{P}|$-dimensional binary vector. We use `gpt-4o-2024-08-05` as our `Evaluator`. For each aspect, we only include the relevant components of the example in the predicate evaluation prompt (e.g., syntax-based predicates are only evaluated over the natural language question, or database reasoning-predicates are only evaluated over the natural language question and the schema).

We estimate the accuracy of our chosen `Evaluator` by randomly sampling 50 example-predicate pairs for each aspect (250 examples total) and have two authors independently evaluate the predicate on the example. The average accuracy of the `Evaluator` based on the two authors was 73%. We compute the agreement between the two annotators, obtaining Cohen's kappa values of $\kappa = 60.2$, indicating substantial agreement (Artstein and Poesio, 2008) (see Table 6 for statistics per aspect). While we use a larger closed-source model to evaluate predicates for proof-of-concept, future work could explore using fine-tuning lighter-weight models on the task of predicate evaluation.

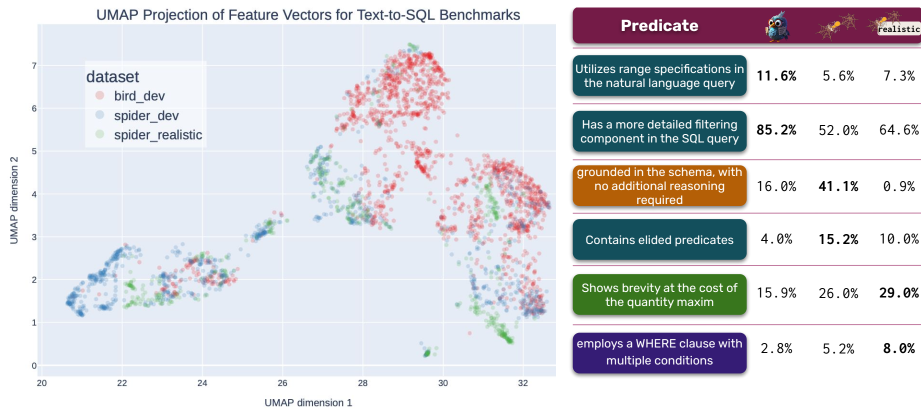
What do these example representations help achieve? Our four-step pipeline (see Appendix B for ablation discussion and further intuitions) produces $|\mathcal{P}|$-dimensional binary vectors that serve as a **general-purpose unified representation for any NL2SQL example**. These representations enable various analyses, including comparing dimensions along which benchmarks significantly differ (§4) as well as understanding fine-grained classes of examples across benchmarks on which NL2SQL models struggle (§5). They also allow us to build a correctness classifier to estimate the likelihood that a model will produce correct SQL for a natural language example, and in cases where an example exhibits features associated with blind spots, intervene to remove them (§6).

4 Comparing Text-to-SQL Benchmarks

Analyzing the composition of datasets can reveal distributions of example properties *within* datasets, and highlight similarities and differences of examples *across* benchmarks. These analyses enhance our understanding of the reasoning skills or knowledge needed to perform well on NL2SQL, and in turn, can help inform decisions about model design or new benchmark construction. For instance,

[4]We use `thefuzz` with a token set similarity threshold of $\epsilon = 70$. This method, based on the Levenshtein edit distance, compares sets of tokens, ignoring order and redundancies. It effectively removes predicates with similar wording (e.g., *contains a* `JOIN`" and *uses a* `JOIN`").

[5]We also experimented with automatic methods such as instruction-tuned embedding models, but ultimately relied on manual deduplication to ensure a clean final predicate set.



Predicate			realistic
Utilizes range specifications in the natural language query	**11.6%**	5.6%	7.3%
Has a more detailed filtering component in the SQL query	**85.2%**	52.0%	64.6%
grounded in the schema, with no additional reasoning required	16.0%	**41.1%**	0.9%
Contains elided predicates	4.0%	**15.2%**	10.0%
Shows brevity at the cost of the quantity maxim	15.9%	26.0%	**29.0%**
employs a WHERE clause with multiple conditions	2.8%	5.2%	**8.0%**

Figure 4: Visualizing a UMAP projection (left) of our example representations for three NL2SQL datasets reveals classes of examples across datasets that share certain properties. Computing the proportions of examples exhibiting certain features (right) reveals dimensions along which the composition of datasets statistically significantly differs.

discovering the prevalent syntactic complexity of questions may inspire new linguistically-informed approaches to modeling or data creation.

Dataset comparison also offers practical benefits. When a practitioner has a proprietary NL2SQL dataset, $\mathcal{D}_{\text{target}}$, understanding prevalent properties of examples in $\mathcal{D}_{\text{target}}$ as well as the dimensions along which it resembles or differs from existing benchmarks $\mathcal{D}_A$ or $\mathcal{D}_B$ helps them select the cheapest model that performs well on their data, and opens up other avenues such as data augmentation with benchmarks that closely resemble $\mathcal{D}_{\text{target}}$.

Setup. We compute $|\mathcal{P}|$-dimensional vector representations for examples (Step 4 in §3) in three different datasets: (1) the development set (1,034 examples) of SPIDER (Yu et al., 2018), one of the prevailing cross-domain benchmarks in the NL2SQL community (SPIDER-DEV), (2) the development set (1,534 examples) of BIRD-BENCH (Li et al., 2024), another popular large-scale cross-domain benchmark released as an updated, more challenging alternative to SPIDER-DEV(BIRD-DEV), and (3) SPIDER-REALISTIC (Deng et al., 2021), a dataset of 508 examples based off of SPIDER-DEV designed to reflect more realistic natural language questions. We pose a scenario in which SPIDER-REALISTIC is a target dataset whose composition we want to better understand as compared to established benchmarks SPIDER-DEV ($\mathcal{D}_A$) and BIRD-DEV ($\mathcal{D}_B$) to illustrate the utility of our vector representations.

Comparing SPIDER-DEV and BIRD-DEV. Figure 4 visualizes a UMAP projection (McInnes et al., 2018) of feature vectors of all examples in $\mathcal{D}_A$, $\mathcal{D}_B$, and $\mathcal{D}_{\text{target}}$ into two dimensions.[6] While dataset-specific clusters do emerge for SPIDER-DEV and BIRD-DEV, Figure 4 reveals classes of examples that may share similar properties in $\mathcal{P}$. Referencing hand-annotated difficulty metadata released for each example in BIRD-DEV reveals that these high regions of overlap between BIRD-DEV and SPIDER-DEV mainly occur with BIRD-DEV examples annotated as "simple" (Figure 5). This aligns with our priors on the construction of both datasets, as BIRD-DEV was introduced as a more difficult benchmark due to its complex schema (Li et al., 2024), and features that are often associated with *simpler* examples (use of exact column names in the natural language question, or clear mappings between the question and the SQL query) are more prevalent in SPIDER-DEV examples (Deng et al., 2021).

$\mathcal{D}_{\text{target}}$. We observe regions with substantial overlap between $\mathcal{D}_{\text{target}}$ and our two benchmarks, along with a couple of outlier clusters of $\mathcal{D}_{\text{target}}$-specific examples. To interpret these patterns, we compute the proportion of examples in each dataset exhibiting each feature and run a chi-square test to identify features with statistically significant differing proportions. This allows us to understand the dimensions along which each dataset is distinct.

The table on the right in Figure 4 shows six illustrative features and the proportion of examples in each dataset that express them. The third row shows a question-schema alignment feature rarely present in examples from SPIDER-

[6] To project feature vectors using UMAP, we use a neighbor count (`n_neighbors`) of 50 and a minimum distance (`min_dist`) of 0.01.

REALISTIC ($\mathcal{D}_{\text{target}}$). This validates our approach, since it aligns with the creation of SPIDER-REALISTIC, which was constructed by Deng et al. (2021) by manually modifying natural language questions in SPIDER-DEV to remove or paraphrase mentions of column names. A likely consequence of this process appears in Row 5, where SPIDER-REALISTIC contains the highest proportion of examples that trade satisfying the Gricean maxim of quantity (Grice, 1975) for brevity.

These analyses allow us to better understand the skills required to perform well on each dataset (e.g., BIRD-DEV contains more syntactically complex examples than SPIDER-REALISTIC: 20% of examples include mixed use of symbols and words as compared to 5% in SPIDER-DEV and 6% in SPIDER-REALISTIC). They illustrate how our framework can equip a user with detailed knowledge about their dataset when they are trying to make sense of new $\mathcal{D}_{\text{target}}$ data, and can be repeated with any other dataset.

5 Identifying Model Blind Spots

We now use the feature space constructed in §3 to analyze the performance of LLMs when solving NL2SQL problems. Models must not only be accurate (i.e, predict SQL queries correctly), but also robust (consistently correct in the face of diverse, potentially flawed, or difficult inputs).

While many robustness studies create datasets targeting specific weaknesses (Deng et al., 2021; Gan et al., 2021a), they may overlook model robustness gaps on examples with unique combinations of properties. Building on the ideas in §4, we cluster examples across datasets and analyze LLM correctness by *cluster* to identify challenging example classes for each model. Importantly, this analysis supplements the metric of execution accuracy on an entire dataset, which may obfuscate nuanced model behaviors (§3.1).

Clustering Example Representations. We run K-means clustering over the set union of examples in SPIDER-DEV and BIRD-DEV, setting $k = 14$ using the elbow method (Thorndike, 1953).[7] Table 2 visualizes the distribution of examples in each cluster along with the raw example counts.

Model Inference. We experiment with 13 instruction-tuned LLMs whose behavior we would like to better understand, making sure to include a mixture of closed and open-source models, code-based and general purpose language models, as well as models of different sizes: 7B code and chat-based gemma models (Team et al., 2024), 7B and 13B code and chat-based `llama-2` models (Touvron et al., 2023), 3B and 8B code `granite` models (Mishra et al., 2024), 1.3B and 7B code `deepseek` models (Guo et al., 2024), as well as `gpt-4o`, `gpt-4o-mini`, and `gpt-3.5-turbo` (Hurst et al., 2024).[8] We generate predictions for all examples in SPIDER-DEV and BIRD-DEV with all models using Prompt G.1 in a zero-shot manner (Gao et al., 2023; Yang et al., 2024), which includes the database schema and three example rows of values. We report execution accuracy (EX) over the full datasets (Table 2, Column 1) as well as per cluster (Columns 2–15).

Blind Spots. All models have varying performance across clusters that, in some cases, significantly deviate from their overall EX (Table 2). For example, all models perform well on examples in Clusters 1 ($\mathbf{C_1}$) and 12 ($\mathbf{C_{12}}$), averaging well above their overall accuracy on SPIDER-DEV. Conversely, all models struggled with $\mathbf{C_8}$ and $\mathbf{C_{10}}$, including the strongest model overall, `gpt-4o`.

We train a random forest classifier with 100 estimators on all (feature vector, cluster label $\mathbf{C}$) pairs to identify important features for each cluster using mean decrease in impurity. For example, $\mathbf{C_8}$ contains examples with complex conditional expressions and technical jargon (Table 9). We call this a *blind spot* of models, or a class of examples on which a model systematically achieves low performance as compared to its overall accuracy. In contrast, $\mathbf{C_1}$ contains examples that filter data from a single column of the table and uses aliases for table or column names, requiring minimal additional reasoning to map effectively (Table 9).

Cluster-based analysis also allows us to identify classes of examples for which cheaper, smaller models perform competitively with larger ones or where open-source models may perform at par with proprietary ones. For example, `granite-code-3b` performs on par with `deepseek-coder-7b` on examples in $\mathbf{C_4}$, and even exceeds its performance on examples in $\mathbf{C_1}$. `deepseek-coder-1.3b`, our smallest model, outperforms `granite-code-3b` on $\mathbf{C_{13}}$, while performing on par with it on $\mathbf{C_{14}}$.

[7] We use the implementation from `https://github.com/DistrictDataLabs/yellowbrick`.

[8] The analyses are model and dataset-agnostic. Others may select any model they want to analyze.

Cluster Number	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
# total examples	148	303	194	203	87	146	234	230	275	160	147	130	161	150
Distribution														
BIRD-DEV (1534)	52	254	167	92	13	101	14	223	258	133	41	4	116	66
SPIDER-DEV (1034)	96	49	27	111	74	45	220	7	17	27	106	126	45	84
%SPIDER-REALISTIC (508)	6.7	8.9	4.1	9.1	4.9	16.5	11.6	3.7	3.1	2.8	12.4	2.6	3.7	9.8
`gpt-4o-mini` (37.1/76.4)	84.5 $^{(13)}$	56.1 $^{(25)}$	39.7 $^{(24)}$	42.4 $^{(24)}$	63.2 $^{(23)}$	47.9 $^{(25)}$	68.4 $^{(22)}$	16.5 $^{(14)}$	46.5 $^{(25)}$	28.1 $^{(20)}$	69.4 $^{(21)}$	89.2 $^{(10)}$	65.2 $^{(23)}$	54.7 $^{(25)}$
`gpt-4o` (43.7/76.1)	87.8 $^{(11)}$	63.0 $^{(23)}$	40.7 $^{(24)}$	47.8 $^{(25)}$	59.8 $^{(24)}$	56.2 $^{(25)}$	69.2 $^{(21)}$	27.4 $^{(20)}$	52.0 $^{(25)}$	33.1 $^{(22)}$	68.0 $^{(22)}$	88.5 $^{(10)}$	65.8 $^{(22)}$	56.7 $^{(25)}$
`gpt-3.5-turbo` (33.9/62.1)	74.3 $^{(19)}$	49.5 $^{(25)}$	31.4 $^{(22)}$	34.5 $^{(23)}$	66.7 $^{(22)}$	43.8 $^{(25)}$	50.4 $^{(25)}$	13.9 $^{(12)}$	41.8 $^{(24)}$	26.2 $^{(19)}$	55.8 $^{(25)}$	70.8 $^{(21)}$	59.6 $^{(24)}$	48.0 $^{(25)}$
`gemma-7b` (19.9/60.9)	70.3 $^{(21)}$	38.6 $^{(24)}$	14.4 $^{(12)}$	39.9 $^{(24)}$	64.4 $^{(23)}$	24.0 $^{(18)}$	50.0 $^{(25)}$	5.7 $^{(5)}$	24.4 $^{(18)}$	13.1 $^{(11)}$	51.0 $^{(25)}$	76.2 $^{(18)}$	46.6 $^{(25)}$	31.3 $^{(22)}$
`code-gemma-7b` (31.2/66.4)	87.8 $^{(11)}$	48.2 $^{(25)}$	30.4 $^{(21)}$	30.5 $^{(21)}$	51.7 $^{(25)}$	47.3 $^{(25)}$	55.6 $^{(25)}$	10.0 $^{(9)}$	40.7 $^{(24)}$	21.9 $^{(17)}$	58.5 $^{(24)}$	85.4 $^{(12)}$	57.1 $^{(24)}$	44.0 $^{(25)}$
`deepseek-coder-1.3b` (14.8/54.4)	70.9 $^{(21)}$	27.1 $^{(20)}$	14.4 $^{(12)}$	19.7 $^{(16)}$	40.2 $^{(24)}$	30.1 $^{(21)}$	48.3 $^{(25)}$	9.1 $^{(8)}$	18.9 $^{(15)}$	12.5 $^{(11)}$	42.2 $^{(24)}$	66.9 $^{(22)}$	37.9 $^{(24)}$	26.0 $^{(19)}$
`deepseek-coder-7b` (23.5/68.2)	73.6 $^{(19)}$	38.3 $^{(24)}$	22.2 $^{(17)}$	27.6 $^{(20)}$	59.8 $^{(24)}$	41.1 $^{(24)}$	57.3 $^{(24)}$	14.3 $^{(12)}$	31.3 $^{(21)}$	21.2 $^{(17)}$	61.9 $^{(24)}$	84.6 $^{(13)}$	50.3 $^{(25)}$	40.7 $^{(24)}$
`granite-code-3b` (11.3/59.9)	78.4 $^{(17)}$	17.5 $^{(14)}$	12.9 $^{(11)}$	27.1 $^{(20)}$	51.7 $^{(25)}$	31.5 $^{(22)}$	54.3 $^{(25)}$	5.7 $^{(5)}$	15.3 $^{(13)}$	10.0 $^{(9)}$	44.2 $^{(25)}$	76.9 $^{(18)}$	31.1 $^{(21)}$	26.0 $^{(19)}$
`granite-code-8b` (16.6/68.8)	84.5 $^{(13)}$	27.7 $^{(20)}$	17.0 $^{(14)}$	28.1 $^{(20)}$	44.8 $^{(25)}$	43.8 $^{(25)}$	68.4 $^{(22)}$	5.2 $^{(5)}$	19.6 $^{(16)}$	13.8 $^{(12)}$	54.4 $^{(25)}$	87.7 $^{(11)}$	38.5 $^{(24)}$	40.0 $^{(24)}$
`code-llama-7b` (18.6/59.6)	76.4 $^{(18)}$	33.7 $^{(22)}$	20.1 $^{(16)}$	25.1 $^{(19)}$	52.9 $^{(25)}$	35.6 $^{(23)}$	47.0 $^{(25)}$	7.8 $^{(7)}$	21.1 $^{(17)}$	15.6 $^{(13)}$	49.7 $^{(25)}$	70.0 $^{(21)}$	45.3 $^{(25)}$	34.0 $^{(22)}$
`code-llama-13b` (21.2/66.1)	77.7 $^{(17)}$	39.6 $^{(24)}$	23.2 $^{(18)}$	26.6 $^{(20)}$	56.3 $^{(25)}$	41.1 $^{(24)}$	55.6 $^{(25)}$	7.0 $^{(6)}$	25.1 $^{(19)}$	16.9 $^{(14)}$	52.4 $^{(25)}$	78.5 $^{(17)}$	46.6 $^{(25)}$	46.0 $^{(25)}$
`llama-2-7b` (3.1/21.5)	46.6 $^{(25)}$	6.3 $^{(6)}$	4.6 $^{(4)}$	5.9 $^{(6)}$	11.5 $^{(10)}$	7.5 $^{(7)}$	9.0 $^{(8)}$	1.7 $^{(2)}$	3.6 $^{(4)}$	3.1 $^{(3)}$	16.3 $^{(14)}$	36.2 $^{(23)}$	10.6 $^{(9)}$	7.3 $^{(7)}$
`llama-2-13b` (6.5/45.6)	58.8 $^{(24)}$	15.5 $^{(13)}$	10.3 $^{(9)}$	9.4 $^{(8)}$	25.3 $^{(19)}$	24.7 $^{(19)}$	34.6 $^{(23)}$	3.5 $^{(3)}$	7.3 $^{(7)}$	5.0 $^{(5)}$	32.0 $^{(22)}$	75.4 $^{(19)}$	26.7 $^{(20)}$	22.7 $^{(18)}$

Table 2: We cluster feature vectors for all examples in SPIDER-DEV and BIRD-DEV and compute the mean and variance of each model's correctness per cluster. This yields clusters on which models perform well above their dataset-level execution accuracy (`bird`/`spider`) such as in $\mathbf{C_1}$ and $\mathbf{C_{12}}$, while revealing blind spots, or clusters with systematically low performance ($\mathbf{C_8}$ and $\mathbf{C_{10}}$). Mapping examples in our $\mathcal{D}_{\text{target}}$ (SPIDER-REALISTIC) to clusters with K-means inference reveals that the dataset most resembles clusters with weaker performance (Row 3).

Open-source models like `code-gemma-7b` perform on par with proprietary models like `gpt-4o` on examples in $\mathbf{C_1}$.

Interestingly, we also observe some general-purpose language models exceeding the performance of their code counterparts on certain clusters (e.g. `gemma-7b` exceeds the performance of `code-gemma-7b` on $\mathbf{C_4}$ and $\mathbf{C_5}$, while performing lower on all other clusters). This could indicate that examples in these clusters require linguistic or reasoning skills that general-purpose LLMs may be better at. While clusters are dataset-dependent and can be produced with any inference dataset(s), we explicitly propose and will release these example clusters of BIRD-DEV and SPIDER-DEV since they are popular community benchmarks.

Discussion. This analysis can inform a variety of downstream applications. Research into the robustness of NL2SQL models may leverage low-performing clusters to build challenge splits of the data, similar in spirit to existing hard splits of natural language inference data (Gururangan et al., 2018). Moreover, blind spots could assist in the *generation* of adversarial test sets of new challenge examples expressing combinations of properties that models systematically struggle with.

Practically, this analysis could help users make informed model selections, since the majority of their own data ($\mathcal{D}_{\text{target}}$) may closely resemble only a handful of clusters. We illustrate this by running K-means inference on SPIDER-REALISTIC ($\mathcal{D}_{\text{target}}$) to understand which clusters it most closely resembles. Table 2 includes the proportion of SPIDER-REALISTIC that mapped to each cluster. The highest performing cluster, $\mathbf{C_{11}}$, includes the smallest amount of SPIDER-REALISTIC data, with only 2.6% of SPIDER-REALISTIC examples mapped to it, while the majority of data maps to $\mathbf{C_6}$, where many models seem to struggle. Smaller models performing on par with larger, more expensive models in particular clusters that resemble $\mathcal{D}_{\text{target}}$ allows practitioners to perform cost-efficient inference.

6 Question Rewriting

While `SQLSpace` is primarily intended to enable offline analyses or applications (§4 and §5), we illustrate a query rewriting application that removes features associated with a model's blind spots as a proof-of-concept for an online application. We pose a scenario where a lightweight correctness estimator alerts a user when their NL2SQL system is likely to produce incorrect SQL for a natural language question. In this case, a user may be informed that their question can be rewritten *without* a particular feature to increase the chances that the system produces correct SQL.

Correctness Estimation. For a model M whose performance we seek to improve, we first build a lightweight correctness classifier C to predict whether M will produce correct SQL for an example given its feature vector. We obtain binary correctness labels by evaluating predictions from M on all 10,697 examples in UNITE-DEV and train a random forest classifier on a subset of data consisting of feature vectors and correctness la-

	No Rewriting	Rewriting	
		Acc@1	Acc@3
code-gemma-7b	31.2	31.9	**37.7**
deepseek-coder-7b	23.5	24.3	**30.8**

Table 3: Rewriting NL questions in BIRD-DEV that express features most likely to contribute to an incorrect prediction by a model improves execution accuracy. We report accuracy with the top 1 and 3 negative features (acc@1 and acc@3).

	code-gemma-7b-it		deepseek-coder-7b-it	
	Rewrite ✓	**Rewrite ✗**	**Rewrite ✓**	**Rewrite ✗**
Estimator ✓	22.8	43.7	18.5	51.4
Estimator ✗	16.0	17.5	15.4	14.7

Table 4: Percentage breakdown of BIRD-DEV examples by correctness estimator prediction accuracy ("estimator right or wrong") and rewrite success Acc@3 (whether rewriting yielded correct SQL) for code-gemma-7b-it and deepseek-coder-7b-it.

bels.[9] Using the remaining set of examples, we compute feature importance with permutation importance (Breiman, 2001) on negative examples, giving us an ordering of features in $\mathcal{P}$ most contributing to C's prediction that M will produce incorrect SQL for an example (see Table 7 for top predicates for different models).

Rewriting with Correctness Estimation. We select BIRD-DEV as the dataset on which we aim to improve model M's performance. We simulate an online inference scenario with BIRD-DEV examples: first, the correctness estimator C predicts whether M will generate correct SQL for example e (calibration plots in Figures 6 and 7). For predicted failures, we identify top negative features in e using the feature importances we previously computed and prompt a **Rewriter** (here, gpt-4o-2024-08-06) to rewrite the natural language question to address the negative feature it exhibits using Prompt H.1. Here, the Rewriter simulates a human user who may themselves rewrite their question when informed that it contains a feature that may yield incorrect SQL from M.[10] Both the correctness estimator C and the rewriting features are restricted to the 121 features derived from the natural language question and schema, excluding gold SQL-derived features to maintain a realistic inference setting (see Table 8 for performance metrics of C).

We then evaluate the generated SQL conditioned on the rewrite and report EX in two settings: Acc@3, which rewrites the question with the top three negative features and awards M credit if any of the three rewrites produce correct SQL, as well as Acc@1, where we consider only the rewrite conditioned on the top negative feature.

Results. We evaluate rewriting on two models: code-gemma-7b and deepseek-coder-7b. Rewriting questions to remove features that hurt model performance can help improve accuracy (Table 3), especially when rewrites address multiple negative features (Acc@3). This indicates that multiple negative features compound to hinder model performance, and rewriting to eliminate just a single negative feature (Acc@1) is often insufficient. Rather, comprehensive rewrites addressing multiple negative features that collectively degrade model accuracy yield more substantial improvements (Acc@3). A realistic setting that leverages this finding may ask a user to eliminate multiple negative features in a single rewrite of their query.

Table 4 reveals that in most cases, the correctness estimator produced a correct decision ("Estimator ✓"), but the generated rewrite did not produce correct SQL ("Rewrite ✗"), illustrating that the representation space produced by SQLSpace is meaningful for modeling correctness, but that a more nuanced procedure to produce better rewrites would likely further improve accuracy, which we leave to future work.

7 Conclusion

We present SQLSpace, a framework to construct unified representations of text-to-SQL examples semi-automatically. SQLSpace enables detailed and interpretable examination of NL2SQL benchmarks to better understand example properties as well as fine-grained analysis of LLM behavior across benchmarks, and we argue that having such analyses is essential for future benchmark construction and model development in NL2SQL. We lay the groundwork for several future research directions, such as data augmentation or adversarial example generation informed by model blind spots, the development of an online "router" to route difficult examples to larger models, or the design of a more robust, generalizable correctness estimator.

[9] We use 200 estimators, training them on 90% of UNITE-DEV examples and computing feature importance using the remaining 10%. Table 8 contains performance metrics.

[10] See Appendix H for a discussion on feature modulation as well as the cost of rewriting.

Limitations

Human Intervention. Our goal is to construct a meaningful but compact representation space for text-to-SQL examples with as little manual intervention as possible. While the human effort necessary to construct these representations is significantly less than other robustness studies that collect manual annotations from database engineers or SQL experts, our semi-automated pipeline still contains two points of manual intervention: determining the five aspects on which to condition description generation, and further predicate filtering.

Correctness Estimator Efficacy. The performance of the correctness estimators we train during our rewriting experiments varies depending on the dataset, potentially limiting its utility in certain settings. While the estimators do learn meaningful patterns for the two NL2SQL models we experiment with, it is possible that their ability to estimate the correctness of other NL2SQL models may vary based on the dataset. Future work could explore more complex models beyond random forests to improve the generalization ability of the correctness estimators, in turn enabling several downstream applications.

Cost. Several parts of our pipeline utilize closed-source LLMs. While we do this to establish a proof-of-concept, as well as an upper bound of performance, we acknowledge the cost associated with them. Future work could experiment with replacing the predicator evaluator in §3 with a lighter-weight, open-source model to enable inference time prediction of features in a new text-to-SQL example.

References

Stefanos Angelidis and Mirella Lapata. 2018. Summarizing opinions: Aspect extraction meets sentiment prediction and they are both weakly supervised. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3675–3686, Brussels, Belgium. Association for Computational Linguistics.

Ron Artstein and Massimo Poesio. 2008. Inter-coder agreement for computational linguistics. *Computational linguistics*, 34(4):555–596.

David Ian Beaver. 1997. Presupposition. In *Handbook of logic and language*, pages 939–1008. Elsevier.

Leo Breiman. 2001. Random forests. *Machine learning*, 45:5–32.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. 2023. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint arXiv:2301.08881*.

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.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. 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.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Herbert P Grice. 1975. Logic and conversation. In *Speech acts*, pages 41–58. Brill.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing*

for Programming (NLP4Prog 2021), pages 77–87, Online. Association for Computational Linguistics.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ronak Kaoshik, Rohit Patil, Shaurya Agarawal, Naman Jain, and Mayank Singh. 2021. Acl-sql: Generating sql queries from natural language. In *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD)*, pages 423–423.

Wuwei Lan, Zhiguo Wang, Anuj Chauhan, Henghui Zhu, Alexander Li, Jiang Guo, Sheng Zhang, Chung-Wei Hang, Joseph Lilien, Yiqun Hu, et al. 2023. Unite: A unified benchmark for text-to-sql evaluation. *arXiv preprint arXiv:2305.16265*.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. 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 2261–2273, Online. Association for Computational Linguistics.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Pingchuan Ma and Shuai Wang. 2021. Mt-teql: evaluating and augmenting neural nlidb on real-world linguistic and schema variations. *Proceedings of the VLDB Endowment*, 15(3):569–582.

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards robustness of text-to-SQL models against natural and realistic adversarial table perturbation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2007–2022, Dublin, Ireland. Association for Computational Linguistics.

Irina Saparina and Mirella Lapata. 2024. Ambrosia: A benchmark for parsing ambiguous questions into database queries. *arXiv preprint arXiv:2406.19073*.

John R Searle. 1975. A taxonomy of illocutionary acts.

Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment*, 13(12):2747–2759.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864, Online. Association for Computational Linguistics.

Neha Srikanth, Marine Carpuat, and Rachel Rudinger. 2024. How often are errors in natural language reasoning due to paraphrastic variability? *Transactions of the Association for Computational Linguistics*, 12:1143–1162.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Robert L Thorndike. 1953. Who belongs in the family? *Psychometrika*, 18(4):267–276.

Laura Toloşi and Thomas Lengauer. 2011. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*, 27(14):1986–1994.

Mihaela Todorova Tomova, Martin Hofmann, and Patrick Mäder. 2022. Seoss-queries-a software engineering dataset for text-to-sql and question answering tasks. *Data in Brief*, 42:108211.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Cetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. 2018. An end-to-end neural natural language interface for databases. *arXiv preprint arXiv:1804.00401*.

Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. 2023a. 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.

Zihan Wang, Jingbo Shang, and Ruiqi Zhong. 2023b. Goal-driven explainable clustering via language descriptions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10626–10649, Singapore. Association for Computational Linguistics.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, 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.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Xiaojing Yu, Tianlong Chen, Zhengjie Yu, Huiyu Li, Yang Yang, Xiaoqian Jiang, and Anxiao Jiang. 2020. Dataset and enhanced model for eligibility criteria-to-sql semantic parsing. In *12th International Conference on Language Resources and Evaluation (LREC)*.

A Human Intervention in SQLSpace

SQLSpace discovers features for representation construction in a semi-automatic manner, avoiding a scenario in which database engineers manually come up with perturbations, as in other robustness NL2SQL benchmarks.

Human Intervention in NL2SQL Robustness Studies. Many robustness studies rely on significant manual effort. Pi et al. (2022) study whether or not text-to-SQL systems are robust to adversarial perturbations to the tables and schemas involved in text-to-SQL examples. Their benchmark took 253 human hours to annotate. Chang et al. (2023) introduce DR. SPIDER, a dataset that introduces 17 human-created perturbations from database experts and crowdworkers. Here, human intervention involved (1) crowdsourcing annotators from Mechanical Turk to paraphrase questions from SPIDER, (2) task experts (3 SQL experts) that review the paraphrases and categorize them from the text-to-SQL task perspectives, and (3) more task expert hours (3 SQL experts) review the paraphrased questions that are generated from models. Task experts create ontologies of paraphrases. Lastly, Gan et al. (2021a) study the robustness of text-to-SQL models to synonym substitution. They introduce SPIDER-SYN, a human-curated dataset. Four graduates students in Computer Science annotate their dataset manually. The graduate students are trained with a detailed annotation guide, and must to annotate samples in a trial phrase before annotating the whole dataset. The annotators manually choose synonyms for substitution in the natural language questions. They have two rounds of annotation, and in total, annotators look at 5,672 questions.

Human Intervention in SQLSpace. In contrast to the related work above, we have no SQL experts or crowdworkers creating any data or ontologies for our framework. The only manual intervention in SQLSpace involves 1-2 hours of an author removing semantic duplicates from the generated predicate list in Step 3 of Section 3. This required no task expertise, only proficiency in English.

B Ablation Studies

The primary goal of our pipeline is to better characterize and understand features of text-to-SQL datasets in a human-interpretable fashion and how these features help users better understand model performance. Consider the four steps in the system:

> **[Step 1: Describe Examples based on Aspects]** Here, given an *aspect* such as "natural language syntax" or "linguistic pragmatics", an LLM generates an aspect-conditioned description of a text-to-SQL example.
>
> **[Step 2: Propose Features (Predicates) from Descriptions]** Here, given a collection of generated descriptions, an LLM can extract short binary natural language predicates that serve as features in SQLSpace.
>
> **[Step 3: Remove Duplicate Features]** Using a combination of automatic and manual methods, we clean the list of generated predicates to produce a deduplicated list of features.
>
> **[Step 4: Compute Feature Vectors for any Inference Example]** For a given text-to-SQL example e, we run through our list of deduplicated features, and use an LLM to predict whether or not the feature is present in e to produce a binary feature vector v that serves as the example's representation for downstream use.

What are the intutions behind each step in the pipeline? The overall goal of SQLSpace is to discover human-interpretable, generalizable features of text-to-SQL examples to use in downstream analysis. The overarching intuition is to create a space in which to represent queries, without committing to a prior researcher intuitions. **In Step 1**, we ask an LLM to produce a prose description of an example containing commentary related to a particular aspect. These descriptions are an intermediate checkpoint before predicate generation. One could imagine suggesting features without descriptions just by looking at examples, but this is significantly more challenging for language models since meta-reasoning about all aspects at once is challenging. Describing an example before coming up with features is similar to how chain-of-thought serves to improve problem solving in LLMs.

Step 2 simply suggests features extracted from descriptions. Natural language predicates as features lend themselves well to the main goal of SQLSpace: human interpretability. Features are short, human-readable, and binary in order to maximize the flexibility and interpretability of our framework. This predicate generation step is taken from an explainable clustering pipeline (Wang et al., 2023b) and is positioned as a module that performs "in-context clustering" based on a random subset of the corpus with explainable cluster properties. We use the predicate generation module from their pipeline to produce a list of "explanations" of shared properties in our corpus of examples. This automatic process mirrors a manual process of a database engineer sampling random slices of the example corpus and coming up with properties shared by the majority of examples in the slice.

We employ pruning in **Step 3** mainly to reduce the dimensionality of the feature vectors and duplicate features, removing paraphrases of features to reduce bloating of the feature set. A refined version of this step may include the practitioner determining what types of features they are looking to study (assuming a prior).

Step 4 is feature vector construction. Step 3 produces a feature set, and Step 4 builds the feature vector representation for any example by checking whether a feature is "on" or "off" for an example, as in any feature vector construction in classical machine learning with the exception that `SQLSpace` employs an LLM to evaluate whether a feature is "hot".

What steps in our pipeline *can* be ablated? We discuss three ablative settings that help illustrate the utility of the decisions made during our pipeline design: (1) predicate proposal from text-to-SQL examples directly *instead* of descriptions which eliminates Step 1 altogether (B.1), (2) aspect-agnostic description generation (B.2) which eliminates conditioning on aspects, (3) non-deduplicated predicates (B.3) which eliminates Step 3 altogether. We compare the predicates produced in these settings to those produced by our final pipeline in `SQLSpace`. These ablations revolve around Step 1 (description generation) and Step 3 (predicate deduplication). We note that Step 2 (actual generation of predicates) and Step 4 (inference) cannot be materially ablated (turned on or off).

B.1 Ablation Analysis 1: Predicate Proposal Directly from Text-to-SQL Examples

`SQLSpace` leverages a predicate proposal module from Wang et al. (2023b), designed to generate a list of explanations, or predicates by prompting an LLM to perform "in-context clustering". This proposal module ingests a subset of examples from the overall corpus, some natural language goal g, and an instruction to produce a set of explanations for the candidate clusters. Formally, it ingests a sample set $S = \{x_1 .... x_T\}$, a goal g, and an instruction (for example, "*Generate a list of n explanations for candidate clusters based on the sample set*"). See Section 3 of Wang et al. (2023b) for more details.

`SQLSpace` uses aspect-based descriptions of examples instead of NL2SQL examples themselves as input into the predicate proposal module in order to disentangle *individual* example understanding and reasoning from reasoning required for example *clustering*. When generating aspect-based descriptions, an LLM is forced to reason about an example from a particular perspective, and the LLM in the predicate proposal step (Step 2 in `SQLSpace`) can simply find commonalities among these descriptions.

Goal of Ablation. To illustrate the utility of this design decision, **we consider an ablative setting in which we turn off example description generation altogether (turn off Step 1 of `SQLSpace`)**, and input raw NL2SQL examples directly into the predicate proposal module, forcing the LLM to not only reason about individual examples, but also commonalities across examples. We analyze the predicates proposed from this setup and compare them to $\mathcal{P}$, the predicate set resulting from first running aspect-based example description generation for each example (Step 1) and inputting these descriptions into the proposal module.

Setup. We use all NL2SQL examples from UNITE-DEV as in `SQLSpace` with the same setup as described in Step 2 of Section 3. Each sample passed to the proposal LLM look like so:

```
Natural Language Question: subject states that he / she has current hepatic disease.

SQL: select id from records where hepatic_disease = 1
```

We set the goal as following (in contrast to aspect-specific goals in `SQLSpace`):

Prompt B.1: Control Goal

```
Prompt: Here are some examples of natural language questions and their corresponding SQL queries. I want to cluster these examples based on similarities.
```

Results. This process produced 167 predicates, and after automatically deduplicating with `thefuzz`, we obtain 102 predicates, listed in the first row of Table 12. Manually inspecting these predicates reveals a few different types of unhelpful properties:

- **Database Content-Based Predicates:** Many predicates revolve around database entities (e.g "`asks about flights and airlines`", "`queries about car details and specifications`", "`asks about music albums and songs`"). These are *not* generalizable and are database-specific. This is likely due to the proposal LLM is simply

finding shallow commonalities in entities in SQL queries. Our aspect-based predicates (Table 10) are designed to focus on meta-properties of examples instead of database content.

- **Generic Properties of SQL Queries:** Proposing predicates from examples themselves without any prior reasoning or example understanding step also yielded predicates that were vague phrases that generally apply to most SQL queries, and are hence not discriminative (e.g. "`requests specific information based on database criteria`", "`seeks details from a database`", or "`seeks to retrieve data based on specific criteria`"). These types of predicates do not convey meaningful information about NL2SQL examples, and would be treated as noise if we chose this setting in `SQLSpace`.

- **Operation Focused:** When predicates were not database-specific or generic/vague, they were mostly focused on the *operation* that the SQL query was trying to perform (e.g. "`requests for maximum or minimum values`", "`requests information about counts or totals`", "`asks about comparisons`"). While some of our predicates in `SQLSpace` are indeed concerned with these types of SQL operations, `SQLSpace`'s final set of predicates contain much richer information about *how* the natural language question and SQL query *achieve* the operation.

All in all, this ablation experiment highlights the need for a step which explicitly performs the necessary example understanding for high-quality predicates. While `SQLSpace` is designed for general-purpose use across NL2SQL applications, it is possible that practitioners may indeed want predicates related to topics as a part of their example representation vector. In these cases, `SQLSpace` offers *flexibility* to describe examples conditioned on whichever aspects practitioners find useful.

B.2 Ablation Analysis 2: Aspect-Agnostic Description Generation

In Appendix B.1, we demonstrated that the predicates produced by eliminating an example *understanding* step before predicate proposal yields low-quality, generic, and non-generalizable features. This helped to illustrate the utility of description generation (Step 1) of the `SQLSpace` pipeline described in Section 3.

Here, we explore another ablative experiment setting of Step 1 that removes *aspect-based* conditioning of description generation for each example (i.e the generation of d_{syn}, $d_{\text{sql-syn}}$, d_{sem}, d_{prag}, and d_{db} from Step 1 in Section 3), and instead produces general descriptions of each example.

Goal of Ablation. This ablation helps to demonstrate the utility of conditioning descriptions on a particular aspect. We analyze the predicates proposed from this setup and again compare them to $\mathcal{P}$ to understand the impact and role of focusing on particular aspects of NL2SQL examples.

Setup. For each example in UNITE-DEV, we generate a general description agnostic of any aspect using the prompt below (compare this prompt to Prompts D.1–D.5):

Prompt B.2: Generating Descriptions: Control

```
Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... I am a database engineer trying to learn about examples in my dataset. Output a paragraph describing the example.
```

We do not instruct the description generator LLM to focus on any aspect of the example, instead letting *it* choose what details and information to include in a summary description of each example. This process yields high-level descriptions of examples. Compare two descriptions of the same example below: the first description is the aspect-agnostic description, while the second description is d_{syn}.

```
In this example, we are working with a database named '204_335' that contains several tables related to tennis tournaments and match outcomes. The tables include 'tournament_address', 'opponents_in_the_final_list', 'score_in_the_final_list', 'score_in_the_final_list_number', 'score_in_the_final_list_number1', 'score_in_the_final_list_number2', and 'w'. Each table has columns that store various details about the matches, such as match IDs, addresses, opponents, scores, and other relevant information. The natural language question asks us to compare the years 1971 and 1972 to determine which year had the most matches won by a player named Osborne. The corresponding SQL query is designed to find this information by selecting the 'date_number' (which represents the year) from the 'w' table where the 'date_number' is either 1971 or 1972 and the 'outcome' is 'winner'. The query groups the results by 'date_number' and orders them by the count of matches won in descending order, limiting the result to the top entry. The SQL query effectively counts the number of matches won by Osborne in each of the specified years and returns the year with the highest count. This approach ensures that we can identify the year in which Osborne had the most victories, providing a clear comparison between the two years in question.
```

```
The natural language question "comparing 1971 and 1972, which year has the most won matches by osborne?" presents several interesting syntactic features. Firstly, the question starts with a non-canonical structure, known as "fronting," where the comparative element "comparing 1971 and 1972" is placed at the beginning of the sentence. This fronting not only functions to set the stage for the query but also highlights the focus of the comparison—namely, the two given years. Moreover, the question employs a Wh-type clause with "which year," invoking a selection process among a defined set of options (1971 and 1972). This clause acts as the main clause's subject, making the syntactic structure of the sentence an embedded question within a comparison framework. The verb phrase "has the most won matches" maintains subject-verb agreement with the singular subject "year," and showcases a superlative construction ("most") typical in comparative questions. Additionally, the indirect object "by osborne" follows a prepositional phrase, where "by" denotes the agent performing an action. This prepositional phrase functions syntactically as an adjunct that specifies the performer of the winning matches. The inclusion of the agent within a comparison emphasizes the relationship between the syntactic elements and endows the question with specificity regarding whose matches are being evaluated. The linguistically intricate layering of a subject-complement (which year), combined with a comparative framework (more), and the specification of the agent (by osborne), all contribute to the syntactic complexity of the natural language question. This structure efficiently narrows down the information scope while maintaining syntactic coherence in a query format.
```

It is worth noting that while the aspect-agnostic description does produce details that are scattered across different aspect-based descriptions, it focuses more on the database content and less on the meta-properties of the examples.

We take these aspect-agnostic descriptions and feed them into the same predicate proposal module from Step 2 in Section 3, but again change the natural language goal to be aspect-agnostic, encouraging the model to cluster examples and propose predicates solely based on similarities of generated descriptions:

Prompt B.3: Control Goal

`Prompt:` `Here are some descriptions of natural language questions and their corresponding SQL queries. I want to cluster these examples based on similarities.`

Results. This process produced 171 predicates, and after automatically deduplicating with `thefuzz` with a threshold of 85, we obtain 124 predicates, listed in the second row of Table 12. Inspecting these predicates yields similar results to Ablation B.1. Many predicates are yet again focused on *database content* but to **an even higher degree**, ignoring *meta-properties* of the example (e.g. “`concerns retrieving birth dates of tennis players`”, “`seeks the number of singles released in a specific year from a music database`” or “`seeks data on poker players' achievements`”). These predicates are not generalizable to other databases and therefore do not make for strong features in a unified NL2SQL example representation. Some predicates are again vague and high-level, applying to all or a majority of SQL examples (“`utilizes SQL functions and operators`”, “`asks for specific information retrieval`”, “`focuses on specific data extraction`”). We do however observe that introducing descriptions, and hence a step that focuses on reasoning about examples, does *start* to introduce meaningful predicates (e.g. “`demonstrates the use of INTERSECT operator in SQL queries`” or “`requires joining multiple tables in SQL queries`”), something that was not present at all in Ablation B.1.

It is exactly this behavior that we sought to encourage with the introduction of aspects in `SQLSpace`. However, we still see a focus on predicates related to the SQL query, and very little representation of the linguistic properties of the natural langugage question, an important part of the NL2SQL task. **This ablation, taken with the results from the previous ablation setting, help to underscore the importance of explicitly producing aspect-specific conditions, since they allow the model to focus on fine-grained meta-properties of NL2SQL examples that yield high-quality, generalizable, and useful predicates.**

B.3 Ablation Analysis 3: No Deduplication

Lastly, we discuss a setting in which we ablate Step 3 of `SQLSpace`, predicate deduplication. This step was included primarily for cleanliness and avoid noisy or useless features. We run the predicate proposal step in `SQLSpace`(Step 2) with two models: `gpt-4o-2024-08-06` as well as `gpt-3.5-turbo-0125`. Since both LLMs were provided the same descriptions, many predicates were duplicated. Without deduplication, Step 2 produced a total of 1,210 predicates (see breakdown in Table 5). This would have yielded 1,210-dimensional vectors, a prohibitively high dimension that would decrease interpretability and ease of usage for users of `SQLSpace`. Furthermore, duplicate and highly correlated features can affect downstream usage (Toloşi and Lengauer, 2011). This helps illustrate the importance of Step 3, which reduces 1,212 predicates down to a set of just 187, improving interpretability, utility, and removing noise. Furthermore, Step 3 is essential in reducing the overall latency of `SQLSpace`, since computing each individual feature for an NL2SQL example requires a call to an LLM.

C Comparing SQLSpace Features to Metadata in SPIDER and BIRD

Spider. The authors of Yu et al. (2018) conduct an analysis of the SQL hardness of examples in their dataset to understand dataset composition from a SQL component perspective. They divide SQL queries into *easy*, *medium*, *hard*, and *extra hard* solely based on the the number of SQL components, selections and conditions. This results in queries with more keywords considered as harder. These labels are **static**.

In contrast, SQLSpace does not prescriptively pre-define what makes an example difficult, but rather finds examples across datasets that share properties, and evaluates models on these groupings. These groupings can have features that not only span the gold SQL, but also involve properties of the natural language question and its relationship to the underlying schema, allowing us to understand what makes examples difficult **for a particular model** (clusters where a model performs lower than others).

Bird. The authors of Li et al. (2024) annotate examples as **simple**, **moderate**, and **challenging**, augmenting criteria beyond SQL complexity to include schema linking, reasoning required, and question intent understanding. Here, they ask annotators to provide a judgment on a scale from 1–3 across four dimensions and subsequently rank all examples. While this setup captures example properties better than in Yu et al. (2018), it is neither sufficiently fine-grained nor easily interpretable. A static label of "simple" does not offer much insight into example dynamics enough to carefully understand the specific properties that *make* the example simple.

Again, our framework does not pre-define example difficulty, simply identifying features that describe text-to-SQL examples in general, and evaluating models on groups of similar examples. Our 187-dimensional vector representations of examples allow practitioners to thoroughly understand properties of examples that are simple or challenging for models.

D Generating Example Descriptions

Prompt D.1: Generating Descriptions: Syntax

Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... I am a linguist trying to learn about examples in my dataset from a **linguistic syntax perspective**. This includes anything about word order, grammatical relations, hierarchical sentence structure (constituency), agreement, the nature of cross linguistic variation, and the relationship between form and meaning. Output a highly detailed paragraph describing **ONLY** fine-grained **linguistic syntactic** observations about the natural language question.

Prompt D.2: Generating Descriptions: SQL Syntax

Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... I am a database engineer trying to learn about examples in my dataset from a **SQL syntax perspective**. Specifically, I would like to learn about the **structure** of the SQL query, the **complexity** of the query, the **relationship between the query and the provided underlying database schema**, and the nature of cross-database variation. Output a highly detailed paragraph describing **ONLY** fine-grained observations about the SQL query.

Prompt D.3: Generating Descriptions: Example Semantics

Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... I am a database engineer trying to learn about examples in my dataset of SQL queries. Specifically, I would like to learn about the **relationship** between the provided natural language question and the SQL query. For example, how does the natural language question relate to the SQL query? Do they exhibit parallel characteristics, or is there some reasoning required to map between the two? What kind of reasoning is required? What are the similarities and differences between the two? Output a highly detailed paragraph describing **ONLY** fine-grained **comparison-based** observations about the natural language question versus the SQL query.

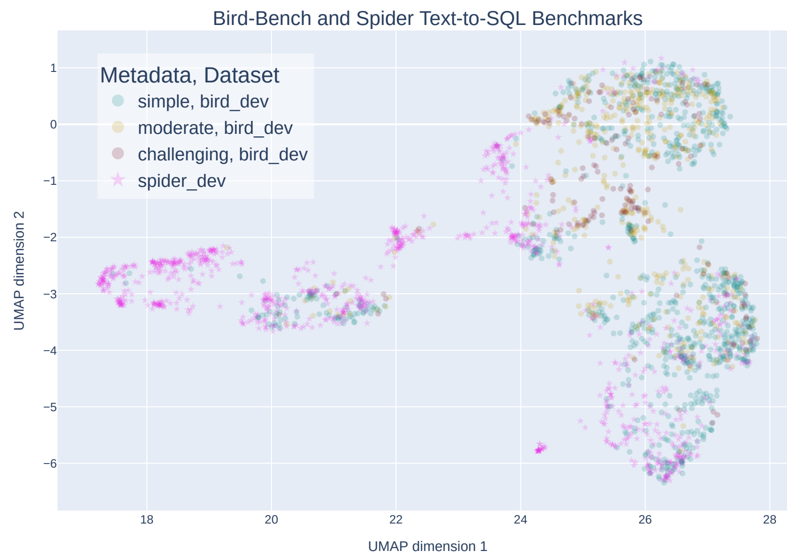


Figure 5: UMAP projection of feature vectors from BIRD-DEV and SPIDER-DEV. We color each point in BIRD-DEV using the hand-annotated metadata released with the dataset. We observe that areas of overlap between BIRD-DEV and SPIDER-DEV typically occur on examples annotated as "simple" in BIRD-DEV.

Prompt D.4: Generating Descriptions: Pragmatics

```
Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... I am a linguist trying to learn about examples in my dataset from a linguistic pragmatics perspective. Specifically, I would like to learn about the pragmatics of the natural language question. For example, what speech acts are used in the question? Include commentary on Gricean theory, implicature, relevance, and any other information about how word choice and context contribute to the meaning. Does the question exhibit vagueness, underspecification, or ambiguity that make it difficult to understand the author's intent? Output a highly detailed paragraph describing ONLY fine-grained linguistic pragmatic observations about the natural language question.
```

Prompt D.5: Generating Descriptions: Database Reasoning

```
Prompt: You are an expert SQL programmer and linguist. I will give you a natural language question and the corresponding SQL query. I will also provide the underlying database schema in the form [database_name] | [table] : [column], [column] ...| [table] : ... | ..... The provided natural language question is attempting to access information from the provided database schema. I am a database engineer and I want to learn about the relationship between the natural language question and the provided database schema. To what degree is the question grounded in the schema? Does the question use exact column names from the schema? Do the concepts and need expressed in the question have clear counterparts in the database schema? If not, what types of reasoning are required to map between the two? Explain what kind of reasoning is required. For example, is linguistic reasoning required (e.g. analogical reasoning, syntactic reasoning or paraphrastic reasoning)? Is commonsense reasoning required? Is logical reasoning required (e.g. deductive reasoning or causal reasoning). How does the structure of the question (syntactic or semantic) relate to the structure of the database schema? Output a highly detailed paragraph describing ONLY these sorts of fine-grained observations about the relationship between the natural language question and the provided database schema.
```

Aspect	# of Proposed Predicates gpt-4o	gpt-3.5-turbo	Total
Syntax	100	130	230
SQL Syntax	102	127	229
Example Semantics	160	171	331
Pragmatics	98	117	215
Database Reasoning	171	134	305

Table 5: We propose candidate predicates for each aspect using both gpt-4o and gpt-3.5-turbo in the predicate proposal step. Each model proposes n predicates in an iterative fashion. The number of predicates proposed by both models are shown above, stratified by aspect.

E Predicate Discovery

Prompt E.1: Predicate Discovery: Syntax

Prompt: Here are some detailed descriptions of natural language questions and their corresponding SQL queries. I want to cluster these descriptions based on observations about **linguistic syntax**.

Prompt E.2: Predicate Discovery: SQL Syntax

Prompt: Here are some detailed descriptions of natural language questions and their corresponding SQL queries. I want to cluster these descriptions based on observations about the **syntax of the SQL query**.

Prompt E.3: Predicate Discovery: Example Semantics

Prompt: Here are some detailed descriptions of natural language questions and their corresponding SQL queries. I want to cluster these descriptions based on **comparisons between the provided natural language question and the SQL query**.

Prompt E.4: Predicate Discovery: Pragmatics

Prompt: Here are some detailed descriptions of natural language questions and their corresponding SQL queries. I want to cluster these descriptions based on observations about linguistic pragmatics.

Prompt E.5: Predicate Discovery: Database Reasoning

Prompt: Here are some detailed descriptions of natural language questions written to query an underlying database schema. I want to cluster these descriptions based on the relationship and reasoning between the provided natural language question and the underlying database schema.

F Predicate Evaluation

Prompt F.1: Predicate Evaluation: Syntax

Prompt: You will be given some text. Determine whether the TEXT satisfies a PROPERTY. Respond with Yes or No. When uncertain, output No.

Now complete the following example:

PROPERTY: **property**
TEXT: **question**

Does the **text** exhibit the **PROPERTY**?:

Prompt F.2: Predicate Evaluation: SQL Syntax

Prompt: You will be given a SQL query and a PROPERTY. Determine whether the SQL query satisfies the PROPERTY. Respond with Yes or No. When uncertain, output No.

Now complete the following example:

PROPERTY: **property**
SQL Query: **query**

Does the **query** exhibit the **PROPERTY**?:

Prompt F.3: Predicate Evaluation: Semantics

Prompt: You will be given a natural language question and its SQL translation. You will also be given a PROPERTY. Determine whether the natural language question and its SQL translation satisfy the PROPERTY. Respond with Yes or No. When uncertain, output No.

Now complete the following example:

PROPERTY: **property**
NATURAL LANGUAGE QUESTION: **question**
SQL Query: **query**

Does the question and its SQL translation exhibit the **PROPERTY**?:

Prompt F.4: Predicate Evaluation: Pragmatics

Prompt: You will be given a natural language question and its SQL translation. You will also be given a PROPERTY. Determine whether the natural language question and its SQL translation satisfy the PROPERTY. Respond with Yes or No. When uncertain, output No.

Now complete the following example:

PROPERTY: **property**
NATURAL LANGUAGE QUESTION: **question**
SQL Query: **query**

Does the question and its SQL translation exhibit the **PROPERTY**?:

Aspect	Mean Accuracy	Annotator Agreement (κ)
Syntax	67.0	41.6
SQL Syntax	86.0	81.6
Example Semantics	77.0	61.4
Pragmatics	63.0	49.0
Database Reasoning	72.0	54.5

Table 6: The average accuracy of `gpt-4o` on predication evaluation for a sample of 250 example-predicate pairs, stratified by aspect. We also compute the Cohen's Kappa between the two annotator judgments.

Prompt F.5: Predicate Evaluation: Database Reasoning

```
Prompt: You will be given a natural language question that is trying to query a database as well as the database schema. You will also be given a PROPERTY. Determine whether the natural language question satisfies the PROPERTY. Respond with Yes or No. When uncertain, output No.

Now complete the following example:

PROPERTY: property
DATABASE SCHEMA: question
QUESTION: question

Does the natural language question exhibit the PROPERTY with respect to the database schema?
```

G NL2SQL Inference

Prompt G.1: NL2SQL Inference

````
Prompt: Write an SQLite query to answer the following question given the database schema and example rows. Please wrap your code answer using ```:
Schema: {schema}
Question: {question}
Write a SQLite query wrapped in ```to answer the question and output nothing else:
````

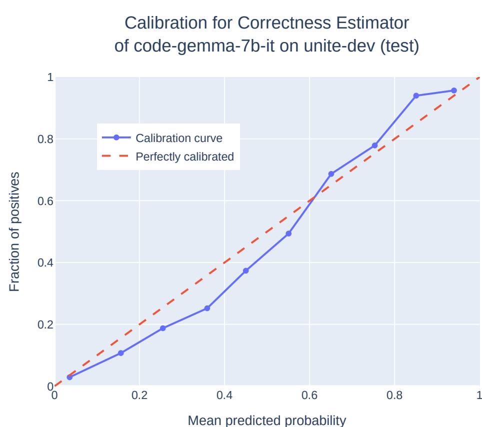
H Question Rewriting

Feature Modulation. Features can be helpful ("question is grounded in the database schema") or harmful ("has vagueness in meaning") to the NL2SQL task. Negative examples may express bad features, in which case we want to *remove* them during the rewrite, or be absent of positive features, in which case we want to *add* them during the rewrite. We modulate this mode during prompting based on manual assignments to features. In our proposed setting, a human rewriting would naturally modulate this mode themselves.

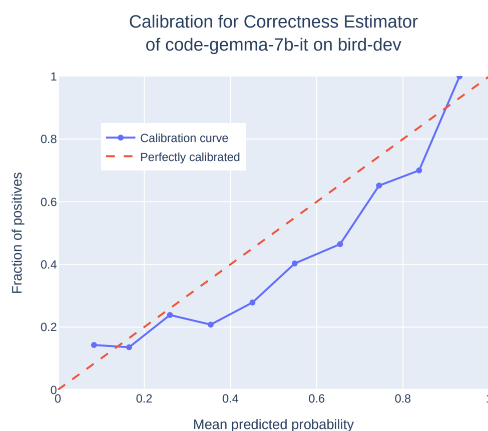
Cost of Rewriting. Section 6 poses an "online" scenario in which for an inference example consisting of only a natural language question and the underlying database schema, (1) a feature vector is computed for the example using only the set of features extractable from the question and schema alone (121 features), (2) a lightweight correctness estimator ingests this feature vector and outputs a binary prediction on whether or not the text-to-SQL model is likely to produce correct SQL for the question, and (3) if not, alerts the user to rewrite their query based on the blind-spot features present in the question. In our simulation of this scenario, we use an LLM to rewrite the question.

Constructing the feature vector from the natural language question and database schema is the step requiring the most compute resources, since an LLM evaluates whether or not each of the 121 features is present in the example or not. Our experiments leverage `gpt-4o` as the feature evaluator. While we do not know the number of parameters in `gpt-4o`, we send concurrent API requests for each of the 121 features to the model, reducing the overhead of feature vector computation down to an average of 10-15 seconds per example. Future work may explore the possibility of using a smaller LLM (e.g 1B—4B parameters) and sending batched inference calls for feature evaluation, which would likely dramatically decrease the overhead in an online setting.

We consider two settings in which we feel use of rewriting can be justified. First, consider a scenario in which we are trying to boost the performance of **a particularly *weak* model** on text-to-SQL (e.g. a model *not* trained on code). As demonstrated by our experiments in Section H, rewriting natural language questions to remove negative features can help boost performance. Rewriting questions to optimize the performance of a weak text-to-SQL model can improve accuracy without the need to retrain or finetune. Second, consider a scenario in which we are trying to boost the performance of **a particular *large model*** on text-to-SQL (e.g. a model with $>$ 100B parameters). Here, the cost of each call can greatly outweigh the cost of feature vector construction. Targeted natural language question rewrites can help optimize the performance of the model on text-to-SQL without excessive calls to the large model.

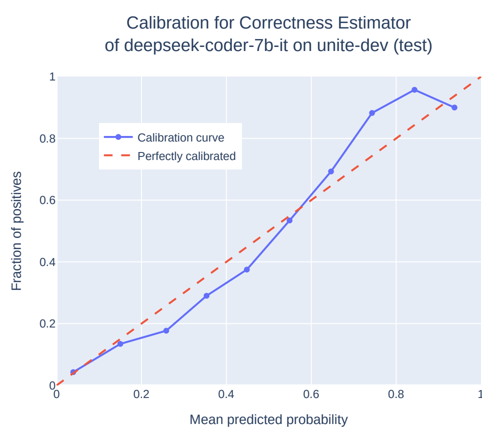


(a) Confidence Calibration Plot for `code-gemma-7b` on the held-out 10% test split of UNITE-DEV.

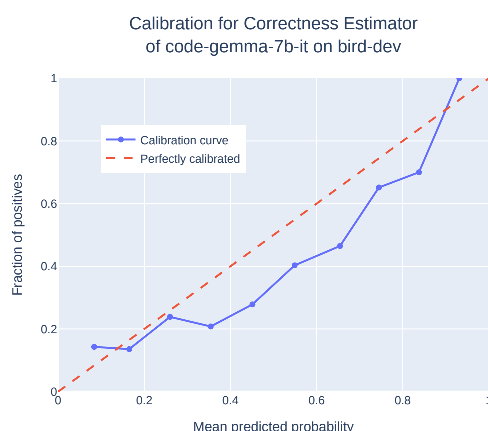


(b) Confidence Calibration Plot for `code-gemma-7b` on BIRD-DEV.

Figure 6: Visualizing calibration of the random forest classifiers we train to predict whether or not `code-gemma-7b` will produce correct SQL given a compact representation of the example.



(a) Confidence Calibration Plot for `deepseek-coder-7b` on the held-out 10% test split of UNITE-DEV.



(b) Confidence Calibration Plot for `deepseek-coder-7b` on BIRD-DEV.

Figure 7: Visualizing calibration of the random forest classifiers we train to predict whether or not `deepseek-coder-7b` will produce correct SQL given a compact representation of the example.

code-gemma-7b	deepseek-coder-7b
requires domain-specific knowledge	involves temporal constraints in the natural language question
involves semantic mapping from NL to DB	employs punctuation for clarity
adheres to Gricean conversational maxims	contains vague or ambiguous language
requires commonsense reasoning to understand the question	question is clearly grounded in the provided database schema
has a slight vagueness in meaning,	uses quantifiers for specificity

Table 7: Top five most discriminative features associated with a model M producing incorrect SQL. We use permutation importance to derive an ordered list of all predicates in $\mathcal{P}$. During rewriting, we remove (or add in) the most important feature that the example expresses.

		UNITE-DEV				BIRD-DEV			
		P	R	F1	Acc	P	R	F1	Acc
offline	gemma	79.0	82.1	80.5	81.1	52.4	73.7	61.2	70.9
offline	deepseek	79.7	74.4	77.0	80.4	40.1	63.2	49.0	70.0
online	gemma	71.0	76.3	73.5	73.6	46.5	50.3	48.3	66.4
online	deepseek	70.4	66.5	68.4	73.1	37.1	40.2	38.6	69.9

Table 8: Precision, recall, F1, and accuracy of the random forest correctness estimator for both code-gemma-7b-it and deepseek-coder-7b-it. We include results from both the offline version of the correctness estimator that has access to the full set of $\mathcal{P}$ features, as well as the online version that has access to only the 121 features related to the natural language question and schema. UNITE-DEV Test is the 10% holdout set used for feature importances.

Prompt H.1: Rewriting with Natural Language-based Features

```
Prompt: Given the definition of {feature}, I want to rewrite the following natural language question to mode {feature}. I want the meaning and intent of the question to be preserved. The question I want to rewrite is {question}

It is trying to query a database with this schema: {schema}. Only output your rewritten question and wrap it in “‘. Your question must be as detailed as possible. DO NOT drop information from the original question. If the question cannot be rewritten with the property, output “INVALID”.

Rewritten semantically equivalent natural language question that expresses {feature}:
```

I Computational Resources and Hyperparameters

We run inference for all open-source models on two NVIDIA A6000 GPUs. We set a temperature of 0.7 for description generation, and a temperature of 0.1 during NL2SQL generation.

Table 9: Top 10 most important features computed with permutation importance for all clusters, consisting of examples from both SPIDER-DEV and BIRD-DEV.

	Permutation Importance-Based Top Features
$\mathbf{C_1}$	involves logical inference beyond direct keyword matching (0.06) filters data from a single column of the table (0.06) contains a single condition in the WHERE clause (0.05) directly aligns with a specific column in the schema, requiring minimal additional reasoning to map effectively (0.04) requires additional contextual understanding (0.03) involves reasoning about data representation (0.03) contains a WHERE clause (0.03) uses aliases for table or column names (0.03) requires commonsense reasoning to understand the question (0.03)
$\mathbf{C_2}$	contains a single condition in the WHERE clause (0.07) filters data based on a single criterion (0.06) retrieves data from multiple tables (0.06) uses a JOIN operation (0.05) uses aliases for table or column names (0.03) utilizes prepositional phrases for modification (0.03) uses prepositional phrases to modify noun phrases (0.03) utilizes nominal phrases with modifying prepositional phrases (0.03) uses modifiers to specify conditions (0.03) employs a WHERE clause with multiple conditions (0.02)
$\mathbf{C_3}$	uses subordinate clauses for postmodification (0.06) contains subordinate clauses modifying main clauses (0.05) contains subordinate clauses (0.05) contains nested clauses (0.04) contains relative clauses (0.04) contains subordinate clauses modifying noun phrases (0.04) employs relative clauses to modify noun phrases (0.03) uses complex noun phrases (0.02) involves the use of logical operators to define selection criteria (0.02) contains a WHERE clause (0.01)
$\mathbf{C_4}$	uses a LIMIT clause (0.11) includes an ORDER BY clause (0.07) limits the selection based on a specific column condition (0.07) contains a WHERE clause (0.06) involves comparison operations (0.03) employs a GROUP BY clause (0.03) retrieves data from multiple tables (0.02) uses a JOIN operation (0.02) uses aliases for table or column names (0.02) filters data based on a single criterion (0.02)
$\mathbf{C_5}$	includes a HAVING clause (0.2) employs a GROUP BY clause (0.07) contains a WHERE clause (0.05) utilizes comparative numerals and inequality symbols (0.04) exhibits conditional logic (0.03) utilizes range specification in nominal phrases (0.02) directly maps threshold values to their respective columns in the SQL query (0.02) includes an ORDER BY clause (0.02) has a complex conditional expression (0.02) employs comparison operations for numerical values (0.02)
$\mathbf{C_6}$	filters data based on a single criterion (0.05) retrieves data from multiple tables (0.05) filters data from a single column of the table (0.05) uses a JOIN operation (0.04) contains a single condition in the WHERE clause (0.04) uses aliases for table or column names (0.03) involves missing or incomplete information in either natural language question or SQL query (0.02) requires commonsense reasoning to understand the question (0.02) clearly grounded in the provided database schema, with no additional reasoning required (0.02) contains vague or ambiguous language (0.01)
$\mathbf{C_7}$	limits the selection based on a specific column condition (0.08) contains a WHERE clause (0.07) has a more detailed filtering component in the SQL query (0.05) requires domain-specific knowledge (0.04) follows a basic SELECT-FROM-WHERE structure (0.04) involves reasoning about data representation (0.03) involves logical inference beyond direct keyword matching (0.02) involves interpretation and understanding of context (0.02) requires understanding of logical operations and condition grouping (0.02) contains a single condition in the WHERE clause (0.02)
$\mathbf{C_8}$	contains a CASE statement (0.06) has a complex conditional expression (0.04) exhibits minimal vagueness or ambiguity (0.03) adheres to Gricean conversational maxims (0.03) involves stricter conditions than the natural language question (0.02) uses aggregate functions in the SELECT clause (0.02) contains technical jargon (0.02) requires understanding of logical operations and condition grouping (0.02) primarily based on direct mappings with some synonymy and domain-specific knowledge required (0.02) closely grounded in the database schema, although it does not use exact column names (0.02)

Continued on next page...

Table 9 (continued)

	Permutation Importance-Based Top Features
$\mathbf{C_9}$	employs a WHERE clause with multiple conditions (0.1) involves the use of logical AND operators (0.09) contains a single condition in the WHERE clause (0.05) uses comparison operators to connect conditions in the WHERE clause (0.03) filters data based on a single criterion (0.03) has missing constraints in the sql query (0.02) contains vague or ambiguous language (0.02) contains relative clauses (0.02) underspecifies information (0.02) exhibits conditional logic (0.02)
$\mathbf{C_{10}}$	includes nested sub-conditions (0.05) involves the use of logical operators to define selection criteria (0.04) utilizes range specification in nominal phrases (0.04) employs logical quantifiers to set multiple criteria (0.04) involves the use of logical AND operators (0.04) includes numerical comparison with inequality symbols (0.04) utilizes range specifications (0.04) integrates quantified expressions within a conditional framework (0.03) employs a WHERE clause with multiple conditions (0.03) employs comparison operations for numerical values (0.03)
$\mathbf{C_{11}}$	employs logical connectors and operators (0.08) uses logical operators to connect clauses (0.07) uses coordinating conjunctions (0.05) employs coordinated noun phrases to express relationships (0.04) uses conjunctions to coordinate noun phrases (0.03) involves the use of logical operators to define selection criteria (0.03) uses coordinating conjunctions for enumeration (0.03) employs coordinated noun phrases (0.03) employs logical quantifiers to set multiple criteria (0.02) employs alternative conditions (0.02)
$\mathbf{C_{12}}$	requires recognition of implicit relationships between natural language concepts (0.08) limits the selection based on a specific column condition (0.06) involves logical inference beyond direct keyword matching (0.06) involves interpretation and understanding of context (0.05) breaks down the problem into specific conditions that need to be met (0.05) contains a WHERE clause (0.05) demonstrates specificity in language use (0.05) involves reasoning about data representation (0.04) directly aligns with a specific column in the schema, requiring minimal additional reasoning to map effectively (0.03) involves understanding of specific technical terms and concepts (0.03)
$\mathbf{C_{13}}$	contains a single condition in the WHERE clause (0.06) uses prepositional phrases to modify noun phrases (0.04) utilizes nominal phrases with modifying prepositional phrases (0.04) filters data based on a single criterion (0.04) utilizes prepositional phrases for modification (0.04) uses a JOIN operation (0.02) utilizes prepositional phrases to modify nouns (0.02) retrieves data from multiple tables (0.02) employs a WHERE clause with multiple conditions (0.02) uses aliases for table or column names (0.02)
$\mathbf{C_{14}}$	contains subqueries (0.15) has a nested subquery (0.15) has a nested SELECT statement (0.13) involves nested logic (0.06) uses a nested WHERE clause (0.06) uses a single select statement (0.04) uses aliases for table or column names (0.02) retrieves data from multiple tables (0.01) uses a JOIN operation (0.01) uses multiple nested parentheses (0.01)

Table 10: Final set of 187 predicates used as features in SQLSpace stratified by aspect.

	Predicates
Syntax	uses complex sentence structures, involves conditional statements, uses prepositional phrases to modify noun phrases, contains subordinate clauses modifying main clauses, has a compound sentence structure, utilizes prepositional phrases for modification, has a conditional clause, employs passive voice constructions, contains elided predicates, contains subordinate clauses, contains subordinate clauses modifying noun phrases, employs hierarchically structured sentence elements, contains multiple coordinate phrases joined by conjunctions, contains relative clauses, uses prepositional phrases to indicate hierarchy, contains a passive voice construction with a predicate adjective, contains subordinate clauses with multiple conditions, uses subordinate clauses for postmodification, uses coordinating conjunctions for enumeration, includes nested sub-conditions, contains nested clauses, employs ellipsis, uses mathematical symbols and units within the syntactic structure, integrates quantified expressions within a conditional framework, uses complex prepositional phrases, employs non-restrictive modifiers, incorporates post-modifier adjective phrases, contains nested prepositional phrases, employs technical lexicon, employs parenthetical expressions, employs punctuation for clarity, demonstrates parallel syntactic constructions, uses shorthand notation, exhibits truncated, telegraphic style, utilizes comparative numerals and inequality symbols, employs mathematical symbols and units, uses punctuation to parse complex syntactic units, uses non-standard shorthand notation for numerical range, includes mixed use of symbols and words, uses shorthand notation to indicate range, employs non-standard linguistic constructions, employs nominal phrases for specification, employs coordinated noun phrases, uses complex noun phrases, employs relative clauses to modify noun phrases, uses conjunctions to coordinate noun phrases, utilizes prepositional phrases to modify nouns, employs coordinated noun phrases to express relationships, includes modifiers for noun phrases, employs nominal phrases for subject and object, utilizes nominal phrases with modifying prepositional phrases, employs nominal phrases as subjects, utilizes range specification in nominal phrases, employs nominal phrases with elided predicates, uses typographic symbols to modify nouns, involves the use of disjunctions, employs logical connectors and operators, uses logical operators to connect clauses, uses comparison operations for selection criteria, uses coordinating conjunctions, uses conjunctions to connect clauses, has a clear logical flow, involves comparison operations, involves the use of logical operators to define selection criteria, employs logical quantifiers to set multiple criteria, utilizes range specifications, employs logical quantifiers like 'and' and 'or', employs comparison operations for numerical values, employs Boolean expressions conjoined with logical operators, uses conjunctions to express disjunction, employs inclusive disjunction with 'or' coordinating conjunction, uses modifiers to specify conditions, includes numerical comparison with inequality symbols
SQL Syntax	has a nested subquery, has a nested SELECT statement, uses a nested WHERE clause, contains subqueries, employs a LEFT JOIN, employs a LEFT OUTER JOIN, uses a JOIN operation, contains multiple joins, uses a self join, contains a cross join operation, utilizes a common table expression (CTE), uses logical OR operators, uses multiple nested parentheses, employs a GROUP BY clause, uses a union operator, includes aggregate functions, contains a WHERE clause, employs a WHERE clause with multiple conditions, includes a HAVING clause, follows a basic SELECT-FROM-WHERE structure, includes a window function, uses aggregate functions in the SELECT clause, limits the selection based on a specific column condition, contains a CASE statement, filters data based on a single criterion, contains a single condition in the WHERE clause, uses a LIMIT clause, involves the use of logical AND operators, employs a recursive common table expression, contains a pivot or unpivot operation, uses a correlated subquery, uses comparison operators to connect conditions in the WHERE clause, uses a single select statement, includes an ORDER BY clause, contains a UNION ALL operator, uses aliases for table or column names, retrieves data from multiple tables, filters data from a single column of the table, uses a distinct keyword, has a complex conditional expression
Example Semantics	requires understanding of logical operations and condition grouping, involves understanding of specific technical terms and concepts, involves semantic mapping, requires domain-specific knowledge, involves parallel characteristics between natural language and sql query, involves logical inference beyond direct keyword matching, requires mapping of natural language concepts to database schema, involves stricter conditions than the natural language question, has a direct relationship with no reasoning required, has a more detailed filtering component in the SQL query, directly maps threshold values to their respective columns in the SQL query, involves interpretation and understanding of context, requires recognition of implicit relationships between natural language concepts, involves reasoning about unique identifiers, involves reasoning about data representation, requires identifying table and column names, has missing constraints in the sql query, requires technical familiarity with the database schema, requires additional contextual understanding, involves complex logical reasoning, involves missing or incomplete information in either natural language question or SQL query, requires mapping of high-level concepts to database schema, breaks down the problem into specific conditions that need to be met, involves nested logic, requires recognizing present state vs. past action, involves temporal constraints in the natural language question
Pragmatics	uses quantifiers for specificity, relies on context for relevance, has a clear and specific request, employs direct speech acts, has a slight vagueness in meaning, adheres to Gricean conversational maxims, relies on conversational implicature, demonstrates assertive speech act, invokes epistemic modality, contains technical jargon, has a cooperative intention, uses declarative structure for assertive statement, demonstrates specificity in language use, employs declarative statements as questions, underspecifies information, exhibits conditional logic, employs reported speech, seeks specific and measurable information, contains vague or ambiguous language, performs an indirect speech act, uses disjunction to allow for multiple criteria, employs alternative conditions, shows brevity at the cost of quantity maxim, assumes prior knowledge, relies on presuppositions, presumes mutual understanding of context, exhibits minimal vagueness or ambiguity, uses rhetorical questions, demonstrates careful word choice, demonstrates a delicate balance of precision and interpretive flexibility, underspecified in terms of database schema, implies a causal relationship
Database Reasoning	requires commonsense reasoning to understand the question, requires semantic mapping between natural language and schema, requires both syntactic and semantic reasoning to bridge the question's semantics to the schema's structure, involves both synonymy and analogical mapping to align natural language terms with schema column names, requires commonsense reasoning to understand the implicit subject of the table entries, primarily based on direct mappings with some synonymy and domain-specific knowledge required, requires basic deductive logic to map between the natural language question and the schema, clearly grounded in the provided database schema, with no additional reasoning required, mirrors the structure of the database schema, with little room for ambiguity or interpretation, directly aligns with a specific column in the schema, requiring minimal additional reasoning to map effectively, uses exact column names from the schema to query for specific values, involves paraphrastic reasoning to map natural language expressions to schema-specific terminology, is already expressed in a highly structured and database-oriented way, requiring no additional reasoning to map to the schema, uses syntactic variation to describe a condition, closely grounded in the database schema, although it does not use exact column names, semantically aligns with the database schema, but requires some degree of linguistic and commonsense reasoning to accurately map the concepts

Table 11: Example descriptions used to mine predicates for SQLSpace.

	Predicates
Syntax	The natural language question Find the number of orchestras whose record format is 'CD' or 'DVD" displays a variety of syntactic features relevant to linguistic analysis. Firstly, the sentence employs a relative clause ('whose record format is 'CD' or 'DVD") that modifies the noun 'orchestras.' This relative clause specifies a condition related to 'orchestras,' showcasing typical English syntactic behavior where the relative pronoun 'whose' introduces additional information and establishes a relationship between 'orchestras' and 'record format.' The main clause 'Find the number of orchestras' contains an imperative verb 'Find,' which signals a direct command. This imperative construction lacks an explicit subject, a common feature in English commands where the subject 'you' is understood implicitly. The noun phrase 'the number of orchestras' consists of a definite article 'the,' a noun 'number,' and a prepositional phrase 'of orchestras,' which together form a well-defined noun phrase indicating a specific quantity. The prepositional phrase functions as a postmodifier to specify what kind of 'number' is being requested. Additionally, the relative clause 'whose record format is 'CD' or 'DVD" features a coordination structure 'CD or DVD,' highlighting a binary choice within the linguistic structure. The use of 'is' within the relative clause indicates singular agreement with 'record format,' demonstrating subject-predicate agreement. The relative clause further relies on the copular verb 'is' followed by the disjunctive conjunction 'or,' linking two alternatives in a linear syntactic arrangement, which presents a straightforward binary condition. Overall, the sentence structure exemplifies several critical syntactic phenomena: the use of imperative mood, relative clause formation, noun phrase construction, subject-predicate agreement, and coordination within a relative clause, all contributing to an intricate yet coherent inquiry.
SQL Syntax	The given SQL query is relatively straightforward and has a simple structure, which includes a SELECT statement aimed at retrieving a single column value from a specific table. The query's complexity is minimal because it involves only one table and a straightforward WHERE clause without any JOIN operations, nested queries, or aggregate functions. In terms of the relationship between the query and the provided underlying database schema from '204_206', the SQL operation specifically targets the 'w' table. The 'SELECT' clause chooses the 'time_greatest_utc' column, indicating that the objective is to retrieve the time in UTC format for a specific date. The 'WHERE' clause filters the rows of the table where the 'date' column matches 'october 3, 2415'. This implies a one-to-one mapping between the question's focus on a specific date and the database's date column, showing a direct translation of the natural language input to SQL command. Moreover, no type casting or formatting functions are employed, hinting that the provided date format in the query is directly comparable to the stored data. In terms of cross-database variation, this example uses a fairly common SQL syntax that should be universally understood across different SQL-compliant databases like MySQL, PostgreSQL, SQLite, and others without requiring modifications. This universality is due to its use of basic SQL keywords (SELECT, FROM, WHERE) and standard date format handling. Overall, the SQL statement is simple and efficient for this specific query because it directly addresses the requirement using the basic querying capabilities of SQL and is highly readable to those familiar with standard SQL operations.
Example Semantics	In the provided example, the natural language question directly translates to the corresponding SQL query with a high degree of parallelism. The question specifies 'how many automobiles,' which maps to the SQL function count(*); indicating a need to count rows. The term 'were produced' implies a filter condition, which in SQL is represented by the 'WHERE' clause. The specific year '1980' in the question is directly used in the SQL query's condition 'YEAR = 1980'. Thus, there is a clear one-to-one correspondence between the elements in the natural language and SQL query. The natural language inherently asks for a count based on production year, and the SQL query represents this request accurately without requiring any complex reasoning or transformation. Both the question and the query focus on the same entity (automobiles) and reference the same attribute (year), showing a straightforward and direct mapping with parallel structure and semantics. The differences lie mainly in syntax and format rather than in conceptual understanding or logical structure.
Pragmatics	The natural language question 'uncontrolled hypertension' displays several interesting facets from a pragmatics perspective. Firstly, it is not presented in the form of a typical question, exhibiting an elliptical style which omits the interrogative structure commonly expected in querying databases. This ellipsis can signal a form of implicit performative speech act, where the speaker's intent is to request information or specify a condition without using a direct question. According to Grice's Maxims, the utterance violates the Maxim of Manner which prefers clarity and the avoidance of ambiguity since the phrase is underspecified and lacks a clear syntactic structure. The Maxim of Quantity is also in question here, as the statement assumes a shared understanding of what is meant by 'uncontrolled' despite not explicitly mentioning it in the context, thereby creating some potential for ambiguity. This could implicature that the speaker had previously established a context where the control level of hypertension was discussed, although it is unexpressed here. Furthermore, the Maxim of Relevance suggests that within the context of the database schema provided, 'uncontrolled hypertension' is pertinent to 'hypertension' being coded as '1', implicating a shorthand reference within a well-understood domain-specific language. The choice of the term 'uncontrolled' implies a subjective, clinical threshold which is not necessarily informationally equivalent to the binary schema indicated in the database but relies on an implied understanding. Therefore, the phrase exhibits a form of semantic underspecification, leaving it open to interpretation whether 'uncontrolled' is directly equatable to 'hypertension = 1'. This concise expression's interpretive relies heavily on the presumed shared knowledge and context between the issuer of the query and the interpreter.
Database Reasoning	The natural language question, 'What are the country codes of countries where people use languages other than English?' is well-aligned with the provided database schema for the most part. The question and schema share direct counterparts: 'country codes' corresponds to the 'CountryCode' column in the 'countrylanguage' table, and 'languages other than English' is explicitly related to the 'Language' column in the same table. The phrasing 'languages other than English' requires a form of paraphrastic reasoning to translate into the SQL syntax 'LANGUAGE != 'English'', which involves a negation operation in the WHERE clause. Additionally, the word 'distinct' in SQL query ('SELECT DISTINCT CountryCode') is derived from understanding that multiple countries might have the same code listed under various languages, necessitating uniqueness. Therefore, both syntactic/semantic matching and an element of logical reasoning are required here to achieve the correct SQL query. The structure of the question semantically corresponds to the schema as it clearly asks about the diverse attribute values within specific columns, effectively tapping into the database's relational structure.

Table 12: Predicates from ablation studies.

	Predicates
NL2SQL Examples (Ablation B.1)	requests information about entities and their attributes, inquires about top results or rankings, requests data related to issues and resolutions, asks for information related to specific entities or categories, asks for comparisons between different data points, inquires about specific data based on criteria, inquires about specific details or attributes in a dataset, requests for maximum or minimum values, inquires about the maximum or minimum values within a dataset, asks for counts or frequencies, seeks information about the highest or lowest values, inquires about database queries related to entities, requests counts or sums based on specific conditions, requests information about counts or totals, seeks specific details based on conditions, asks about the popularity or rankings of certain items, asks about flights and airlines, seeks to compare different data points, asks for comparisons between different data entries, inquires about database statistics, queries about car details and specifications, inquires about the number of occurrences or counts, requests specific information based on numerical values, requests information about rankings or top results, requests specific data filtering criteria, seeks specific information with numerical results, asks about music albums and songs, asks about comparisons, seeks details on issues and resolutions, requests specific information based on database criteria, requests specific details about data based on certain criteria, requests comparisons between different entities, seeks aggregate data based on certain criteria, seeks details from a database, requests statistical analysis, requests comparison between different data points, inquires about the highest or lowest values, seeks to identify the top or bottom entries based on a specific attribute, seeks to identify extremes or limits, requires filtering based on multiple conditions, inquires about numerical values, inquires about numerical comparisons, asks for statistical calculations, requests information based on conditional criteria, "requests specific details about a single entity, involves comparison between different entities, seeks to find the maximum/minimum value, asks about singer details, involves querying for specific information, inquires about rankings or superlatives, inquires about comparative data, requests information about rankings or top values, inquires about relationships between entities, inquires about TV series and ratings, requests for the count of occurrences based on certain conditions, requests information about rankings or extremes, asks for counts or statistics from a database, inquires about specific data filtering criteria, asks for details on winners and rankings, asks for comparisons between different entities, seeks specific information about car details, inquires about flight details, seeks specific information based on comparisons, seeks details on issue tracking and resolution, requests counts or totals of data, requests specific information about data based on conditions, seeks details about particular entities, involves querying for statistical information, inquires about specific data based on conditions, requests counts or sums based on certain criteria, asks for comparisons or rankings, inquires about winners and rankings, seeks details about maximum or minimum values, requests information based on aggregate functions, asks about the count or sum of certain database entries, seeks details on data related to winners and rankings, inquires about specific data points, "seeks aggregated statistical information, seeks information about relationships between different entities, requests information on document templates and ids, seeks data related to multiple categories, asks for counts or totals, inquires about rankings or ordering, seeks aggregated results from multiple entities, inquires about data related to specific categories, requests information on issues and resolutions, requests statistical summaries, requests information on car-related data, requests information on flights and airlines, seeks to filter and display distinct data entries, seeks aggregate information, asks for details on specific attributes or characteristics, asks for statistical analysis, seeks details about document templates, seeks comparisons between different data entries, asks about flight details and statistics, asks for information related to TV series, requests information about the maximum or minimum value in a dataset, seeks to retrieve data based on specific criteria, asks about the number of occurrences based on certain conditions, seeks information on contestants and votes, inquires about rankings or extremes
Aspect-Agnostic Descriptions (Ablation B.2)	involves correcting errors in SQL queries, retrieves specific information about poker players from a database, concerns retrieving birth dates of tennis players, addresses a specific issue ID in a database, involves querying for specific information from databases, involves querying for data based on specific conditions, requests data related to poker player performance, inquires about states with both guardians and veterinarians, seeks the number of singles released in a specific year from a music database, asks for the birth date of a player with a specific name, pertains to identifying authors affiliated with a specific organization and working in a particular field, seeks a list of affiliates with the number of authors working in a specific field, pertains to querying player details from a tennis database, involves listing affiliates and the number of authors working in a specific field, seeks information on authors in a particular field and their affiliates, involves counting entries based on a condition in a database, asks for specific information about a player, asks for a count based on a specific condition, involves querying for specific player details in a tennis database, pertains to counting singles released in a specific year, involves querying multiple tables with joins, inquires about the number of authors in a specific field per affiliate, targets a specific issue and retrieves related information, focuses on counting authors in a specific field per affiliate, concentrates on listing affiliates and authors in a field, seeks information about authors affiliated with a particular organization and field, targets specific data retrieval based on provided criteria, seeks data on poker players' achievements, queries for fix version of a particular software issue, asks for details about final tables made and best finishes of poker players, involves retrieving author names based on affiliations and fields of study, demonstrates the importance of accurate SQL query formulation, utilizes SQL functions for data retrieval, asks for the fix version of an issue in a software engineering database, queries for specific details related to software development, requests information about a specific issue in software development, queries for the birth date of a specific tennis player in a sports database, inquires about authors affiliated with 'PRESTO Japan Science and Technology Corporation' working in a specific field, inquires about the presence of specific groups in certain states, utilizes multiple table joins for data retrieval, focuses on joining multiple tables in a database, retrieves data based on certain conditions, targets a specific issue's fix version in a software engineering database, utilizes filtering based on conditions, asks for details about authors and their affiliations, requests data on singles released in a specific year, focuses on database schema and tables, involves querying for birth date based on player name, inquires about authors affiliated with a specific organization and field of study, requires joining multiple tables in SQL queries, seeks fix version information for a software issue in an SEOSS database, queries for states with specific types of residents, requests a list of affiliates in a particular field, asks for details about poker players' performance and achievements, asks for the birth date of a player named Justine, involves counting entries based on specific criteria, emphasizes the importance of accurate SQL query translations, focuses on identifying common attributes between tables, inquires about poker player performance metrics, joins multiple tables to retrieve data, seeks information about authors affiliated with specific organizations and fields of study, inquires about the number of singles released in a particular year, addresses fixing versions of specific software issues, utilizes SQL functions and operators, requests specific data from a database related to academic research, pertains to final tables made and best finishes for poker players, concerns finding states with specific groups of residents, seeks information on singles released in a specific year, focuses on joining multiple tables in sql queries, aims to retrieve authors affiliated with a specific organization and field from an academic database, inquires about the fix version of a specific issue, requests the fix version of a specific issue in a software development database, filters data based on specific conditions, requires correcting errors in SQL queries, seeks data related to tennis players and their rankings, focuses on querying for specific information from databases, queries for final tables made and best finishes of poker players in a poker database, demonstrates the importance of accurate SQL query representation, addresses errors in SQL queries for accurate data retrieval, requests counts based on specific criteria, focuses on retrieving information about authors and their affiliations, requests data on final tables made and best finishes for poker players, involves finding common data points between two tables in a dog kennel database, requests the count of entries based on a specific criterion, asks for specific information about tennis players, requests information about poker player performance, inquires about fix versions for software development issues, involves working with database tables and columns, queries for specific information from a database, requests the count of authors working in a specific field per affiliate, focuses on querying for player data in a database, asks for specific information retrieval, involves joining multiple tables in SQL queries, seeks names of authors affiliated with a specific organization in a particular field, requests information on fix versions for software development issues, seeks information about music releases in a specific year, emphasizes the importance of accurate conditions in the WHERE clause, retrieves fix version of a specific issue from a database, addresses states with both dog owners and veterinary professionals residing, requests poker player performance metrics, asks for details about authors affiliated with specific organizations and working in particular fields, inquires about states with specific types of residents, requests birth date of a specific player, focuses on specific data extraction, requests specific player details from a tennis database, involves joining multiple tables to extract information, focuses on querying player details in a tennis database, asks for author information based on affiliations and fields of study, aims to extract fix version for a particular issue, inquires about authors affiliated with a particular organization in a specific field, provides counts and lists based on specified conditions, focuses on retrieving specific information from a database, demonstrates the use of INTERSECT operator in SQL queries, requests specific details about a player in a sports database, identifies states with specific types of residents in a dog kennel database, inquires about the number of singles released in a specific year in a music database, asks for the count of singles released in a specific year from a music database, seeks information on states with specific types of residents in a dog kennel database, emphasizes the importance of accurate query reflection, focuses on retrieving specific data from a database, illustrates the importance of correct conditions in SQL queries, focuses on filtering data based on conditions, involves using SQL to retrieve data based on conditions, relates to analyzing data from databases