Improving Retrieval-augmented Text-to-SQL with AST-based Ranking and Schema Pruning

Anonymous ACL submission

Abstract

We focus on Text-to-SQL semantic parsing from the perspective of Large Language Models. Motivated by challenges related to the size of commercial database schemata and the deployability of business intelligence solutions, we propose an approach that dynamically retrieves input database information and uses abstract syntax trees to select few-shot examples for in-context learning.

Furthermore, we investigate the extent to which an in-parallel semantic parser can be leveraged for generating *approximated* versions of the expected SQL queries, to support our retrieval. We take this approach to the extreme—we adapt a model consisting of less than 500M parameters, to act as an extremely efficient approximator, enhancing it with the ability to process schemata in a parallelised manner. We apply our approach to monolingual and cross-lingual benchmarks for semantic parsing, showing improvements over state-of-the-art baselines. Comprehensive experiments highlight the contribution of modules involved in this retrieval-augmented generation setting, revealing interesting directions for future work.

1 Introduction

Text-to-SQL semantic parsing aims at translating natural language questions into SQL, to facilitate querying relational databases by non-experts (Zelle and Mooney, 1996). Given their accessibility benefits, Text-to-SQL applications have become popular recently, with many corporations developing Business Intelligence platforms.

The success of Large Language Models (LLMs) in generalising across diverse Natural Language Processing tasks (Ye et al., 2023; OpenAI et al., 2024) has fuelled works that looked at how these multi-billion parameter models can be best employed for Text-to-SQL (Liu et al., 2023; Pourreza and Rafiei, 2023). Recent works in this space have focused on the in-context learning ability of LLMs, demonstrating that significant improvements can be achieved by selecting suitable (question, SQL) example pairs (Nan et al., 2023; Gao et al., 2023; Guo et al., 2024; Sun et al., 2024). In spite of its underlying benefits, conventional solutions for example selection are usually limited to retrieving examples based solely on the similarity of questions (Nan et al., 2023; An et al., 2023; Guo et al., 2024). Other approaches resort to a preliminary round of parsing which *approximates* expected SQL queries, and directly use these approximations in few-shot prompting (Sun et al., 2024), or to subsequently select (question, SQL) pairs by comparing the approximated query to queries within candidate examples (Gao et al., 2023). The approach proposed by Gao et al. transforms SQL queries into SQL skeletons (Li et al., 2023a) and then filters examples by considering overlap token ratio as the similarity between two skeletons. While incorporating SQL skeleton similarity improves over conventional example selection for Text-to-SQL (Gao et al., 2023), it can result in structural information loss as exemplified in Table 1, where two dissimilar SQL queries are treated as identical. In this paper, we propose a novel approach that selects examples using similarity of normalised SQL Abstract Syntax Trees (ASTs). We argue that considering the similarity of such hierarchical structures can significantly enhance LLMs' performance for Text-to-SQL parsing.

Apart from example selection, we refine database context input to LLMs by dynamically pruning schemata and selecting values. From the perspective of LLMs, existing studies achieve improvements by including the full database schema in the prompt and additionally *hinting* the importance of particular schema elements or values (Pourreza and Rafiei, 2023; Sun et al., 2024). In this paper, we show that the performance can be boosted with schemata of reduced size.

Inspired by Gao et al. that compute an approximated query for a given input question, we further explore how combinations of a sparse retriever with such an in-parallel semantic parser (we would refer to it as *approximator*) can be used to retrieve relevant database context input to LLMs. For efficiency, we adapt the semantic parser (a decoder-free model with $<$ 500M parameters) proposed by Vougiouklis et al. to process schemata in a parallelised manner. Using this efficient approximator, our schema pruning strategy selects a relevant sub-schema in order to simplify the task for LLMs and reduce the relevant computational workload. Furthermore, it enables LLM-based Text-to-SQL solutions to handle longer schemata (usually associated with commercial use-cases) exceeding their context window size.

We apply our approach on monolingual (SPIDER, SPIDER-DK, SPIDER-REAL and SPIDER-SYN) and cross-lingual (CSPIDER) benchmarks of different generalisation challenges. We evaluate the applicability of our framework across both closed- and open-source LLMs. Our framework, comprising only a single round of prompting, achieves state-of-the-art performance, outperforming other baselines which may comprise complex prompting and multiple iterations, when LLMs of equal capacity are involved. Through comprehensive experiments, we highlight strengths and limitations. Our contributions can be summarised as follows:

- We propose a novel approach for selecting (question, SQL) examples using similarity of normalised SQL ASTs.
- We take efficient approximation to the extreme, presenting a schema-parallelisable adaptation of the fastest semantic parser to date.
- We introduce a framework for dynamically selecting schema elements and database values, offering substantial execution accuracy improvements over prior works while significantly reducing the computational workload of LLMs.
- We shed light on the benefits of database value selection and its symbiotic relation to schema pruning for Text-to-SQL LLM prompting.

2 Preliminaries

Let $\mathbf{q}$ be the sequence of tokens of a natural language question for database $\mathbf{D}$ with tables $\mathbf{t} = t_1, t_2, \ldots, t_T$ and columns $\mathbf{c} = c_1^1, c_2^1, \ldots, c_j^i, \ldots, c_{C_T}^T$, where c_j^i is the j-th column of table t_i and $C_i \in \mathbb{N}$ is the total number of columns in table t_i. Furthermore, let $\mathbf{v_D} = \left\{ v_{c_1^1}, v_{c_2^1}, \ldots, v_{c_{C_T}^T} \right\}$ be the set of all values associated with the database $\mathbf{D}$ s.t. $v_{c_1^1}, \ldots, v_{c_{C_T}^T}$ are the DB value sets associated with respective columns $c_1^1, \ldots, c_{C_T}^T \in \mathbf{c}$. The goal of Text-to-SQL semantic parsing is to predict the SQL query $\mathbf{s}$ given the $(\mathbf{q}, \mathbf{D})$ combination, as follows:

$$\mathbf{s} = \arg\max_{\mathbf{s}} p\left(\mathbf{s} \mid \mathbf{q}, \mathbf{D}\right) \qquad (1)$$

For in-context learning, we seek to select pertinent input context including few-shot examples, schema, and database values to simplify the task for LLMs.

SQL$_1$	`SELECT T2.name, T2.capacity FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id WHERE T1.year >= 2014`
	Skeleton: `select _ from _ where _`
SQL$_2$	`SELECT name FROM highschooler WHERE grade = 10`
	Skeleton: `select _ from _ where _`

Table 1: Two SQL queries with identical SQL skeletons.

3 Example Selection using Abstract Syntax Trees

Our goal is to identify the most suitable set of $\mathbb{X}^\star = \{(\mathbf{q}_1^\star, \mathbf{s}_1^\star), \ldots, (\mathbf{q}_e^\star, \mathbf{s}_e^\star)\}$ question-SQL pairs from an index of examples, $\mathbb{X}$, s.t. $\mathbb{X}^\star \subseteq \mathbb{X}$, for maximising the probability of an LLM to predict the correct SQL given $(\mathbf{q}, \mathbf{D})$:

$$\mathbb{X}^\star = \arg\max_{\mathbb{X}} p\left(\mathbf{s} | \mathbf{q}, \mathbf{D}, \mathbb{X}\right) \qquad (2)$$

From the perspective of ranking, we consider the relevance score between a candidate example $(\mathbf{q}_j, \mathbf{s}_j) \in \mathbb{X}$ and the input $(\mathbf{q}, \mathbf{D})$. *Vanilla* semantic search is usually based solely on question embeddings, whereas the structure of SQL queries for similar questions is subject to target databases and can thus differ significantly.

To incorporate database context for selecting examples, we propose to re-rank examples retrieved by question embeddings based on normalised SQL ASTs. Inspired by Gao et al., our framework utilises a preliminary model to compute an approximated SQL query $\mathbf{s}'$, structurally similar to the ground truth, given $(\mathbf{q}, \mathbf{D})$ s.t. $\mathbf{s}' \sim \mathbf{s}$. Examples are then re-ranked by $\text{score}_{\text{AST}}(\mathbf{s}', \mathbf{s}_j)$ for each candidate $\mathbf{s}_j$.

AST represents the hierarchical structure of code in a tree form and can be applied to evaluation metrics for code generation (Tran et al., 2019; Ren et al., 2020). The fact that SQL queries sharing identical abstract meanings may not align with the same syntactic structure poses a challenge for measuring similarity through AST differencing.

AST Normalisation Although it is infeasible to exhaustively transform a SQL to another equivalent form, we can normalise ASTs to reduce undesired mismatch. Firstly, nodes of identifiers are lowercased and unnecessary references are removed (e.g. `<table>.<column>` is substituted with `<column>` if possible). We then delete nodes that create aliases and map each alias to a copy of the subtree to which it references. For cross-domain settings wherein databases at inference time are unseen in the train set, we mask out nodes of values and identifiers after resolving aliases. Otherwise for in-domain settings we further sort nodes associated with JOIN operations(s) to ensure the ordering of tables and keys is consistent.

AST Similarity Given two normalised ASTs, we adopt the Change Distilling algorithm (Fluri et al., 2007) that computes a list of tree edit operations to transform the source AST to the target AST. Types of tree edit operations include: `insert`, `delete`, `alignment`, `move` and `update`. It is essential to note that `move` operation relocates a node to a different parent while moving a node within the same parent is an `alignment`. Therefore, we calculate the similarity between ASTs simply as the ratio of alignments to the total number of operations within the list. Examples of our normalisation and AST similarity are provided in Appendix A.

4 Database Context Selection

Apart from relevant question-SQL pairs, prompting for Text-to-SQL parsing requires the context of database schema and values.

4.1 Schema Selection

We present a hybrid search strategy that selects a sub-schema given a test question to minimise lengthy and potentially irrelevant schema elements input to LLMs, while maintaining high recall.

Let r_j^i be a semantic representation of column c_j^i. We aggregate the semantic names[1] of c_j^i and the table it belongs to, t_i, and its corresponding value set in $\mathbf{D}$, $v_{c_j^i}$, as follows:

$$r_j^i = \left\{ t_i \cup c_j^i \cup v_{c_j^i} \mid i \in [1, T] \text{ and } j \in [1, C_i] \right\}.$$

Given question $\mathbf{q}$, we retrieve the most relevant columns using $\text{score}_{\text{BM25}}(\mathbf{q}, r_j^i)\ \forall i \in [1, T]$ and $j \in [1, C_i]$ (Robertson et al., 1994). A table is retrieved if any of its columns are retrieved.

4.1.1 Incorporating for Approximated Query

The semantic search for schema selection requires a comprehension of the relevance between heterogeneous database information and natural language questions, in addition to interactions across schema elements. To this end, a trained parser can inherently be a semantic search model for retrieving a sub-schema, where columns and tables are extracted from the approximated query $\mathbf{s}'$. We argue that a semantic parser which is performing reasonably on the task, can provide us with an $\mathbf{s}'$, whose structure would assimilate the structure of the expected final query, $\mathbf{s}$. Consequently, we opt to dynamically determine the number of columns to be retrieved by $\text{score}_{\text{BM25}}$ as proportional to the number of unique columns in $\mathbf{s}'$, returned by the approximator. A sub-schema is then obtained by merging schema elements selected by $\text{score}_{\text{BM25}}$ with elements from the approximated query.

4.1.2 Approximating for Longer Schemata

To further reduce the computational workload, we opt for using a *smaller* model for computing $\mathbf{s}'$. However, smaller models usually have shorter context windows (i.e. $<$ 2k tokens), and, as such, they cannot be easily scaled to the requirements of larger schemata. To this end, we propose an approach that enables transformer-based encoders to process longer schemata, in a parallelised manner.

We start with FastRAT (Vougiouklis et al., 2023), which exploits a decoder-free architecture for efficient text-to-SQL parsing. Given a concatenation of the input natural language question $\mathbf{q}$ with the column and table names of a database schema, FastRAT computes the SQL operation in which each element of the input schema would participate in the expected SQL query. We refer to these SQL operations as SQL Semantic Prediction (SSP) labels (Yu et al., 2021). SQL queries are then deterministically constructed from the predicted SSP

[1] Semantic name can refer to simply to the name of a particular or to a concatenation of its name and description.

labels. We introduce a schema splitting strategy to scale the model up to the requirements of schemata comprising several columns.

We augment the input embedding matrix of the model, with two special schema-completion tokens, `[full_schema]` and `[part_schema]`, which are used for signalling cases in which a full and a partial schema are provided as input respectively. Our goal is to split a schema consisting of $\sum_{j=1}^{T} C_j$ columns into r_m splits s.t. each split includes a maximum, pre-defined number of columns r. Each split consists of the question tokens, a single schema-completion token, the table names of the input $\mathbf{D}$ and up to a maximum r number of columns allocated to this particular split (see Algorithm 1 for further details).

Algorithm 1: Algorithm for splitting a schema into smaller splits.

Input: `# col. name tokens concat.`
  $\mathbf{c}^{\text{tok}} \leftarrow [c_1^{\text{tok}_1}, \ldots, c_{C_T}^{\text{tok}_T}]$
  `# flatten concat. of tab. name tokens`
  $\mathbf{t}^{\text{tok}} \leftarrow [t_1^{\text{tok}}, t_2^{\text{tok}}, \ldots, t_T^{\text{tok}}]$
  $\mathbf{q} \leftarrow [q_1, \ldots, q_Q]$ `# q tokens`
  r `: int`

`splits` $\leftarrow [\,]$;
if `len`($\mathbf{c}$) $> r$ **then**
  `prefix` $\leftarrow \mathbf{q} +$ [“`[part_schema]`”];
else
  `prefix` $\leftarrow \mathbf{q} +$ [“`[full_schema]`”];
end
`sp` $\leftarrow$ `prefix`; `# one sp per split`
for $j \leftarrow 1$ *to* $\sum_{j=1}^{T} C_j$ **do**
  `sp` $\leftarrow$ `sp` $+ \mathbf{c}^{\text{tok}}[j]$;
  if $j \mod r = 0$ **or** $j = \sum_{j=1}^{T} C_j$ **then**
    `sp` $\leftarrow$ `sp` $+ \mathbf{t}^{\text{tok}}$;
    `splits.append(sp)`;
    `sp` $\leftarrow$ `prefix`;
  end
end
Return `splits`

The returned schema splits along with the SSP labels corresponding to the schema elements of each split are treated as independent instances during training. At test time, an input schema is split according to Algorithm 1, and the model is input with a batch of the resulting splits. After aggregating the results from all splits, we obtain the SSP label for each column $\in \mathbf{c}$. Inconsistencies across the SSP labels of tables are resolved using majority voting. We refer to this model as FastRAT$_{\text{ext}}$.

4.2 Value Selection

The inference of LLMs for text-to-SQL parsing can be augmented with column values (Sun et al., 2024). We select values for columns in a schema (or a sub-schema) by simply matching keywords in questions and values. This is based on the assumption that LLMs can generalise to unseen values given a set of representative values; thus, the recall and precision of value selection are less critical. We consider value selection providing additional information for LLMs to discern covert differences among columns. An example of our resulting prompt is shown in Appendix B.

5 Experiments

We run experiments using two approximators: FastRAT$_{\text{ext}}$ and Graphix-T5 (Li et al., 2023b). Graphix-T5 is is the approximator used by DAIL-SQL (Gao et al., 2023), and is included to facilitate a fair comparison against the closest work to ours. FastRAT$_{\text{ext}}$ is trained and tested using $r = 64$, unless otherwise stated (cf. Section 5.4). For all experiments, we use 5 (question, SQL) examples.

We test our approach against both closed- and open-source LLMs: (i) `gpt-3.5-turbo` (`gpt-3.5-turbo-0613`), (ii) `gpt-4` (`gpt-4-0613`) and (iii) `deepseek-coder-33b-instruct`. Results using additional models from the DeepSeek family are provided in Appendix C.4.

5.1 Datasets

We experiment with several SQL datasets, seeking to explore the effectiveness of our approach on both monolingual and cross-lingual setups. Specifically, we report experiments on CSPIDER (Min et al., 2019) and SPIDER (Yu et al., 2018). Since CSPIDER is a translated version of SPIDER in Chinese, when it comes to the natural language questions, the characteristics of the two with respect to structure and number of examples are identical. We focus our evaluation on the development sets[2], which are used as test sets in our experiments. These splits consists of $1,034$ examples of questions on 20 unique databases that are not met at training time.

We rely on the training splits to maintain an index of (question, SQL) examples, one for each

[2] Appendix D includes results on the test sets.

dataset. Using these splits, we train a monolingual and a cross-lingual version of FastRAT$_{\text{ext}}$.

Furthermore, we use popular SPIDER variants: (i) SPIDER-DK (Gan et al., 2021b), (ii) SPIDER-REAL (Deng et al., 2021) and (iii) SPIDER-SYN (Gan et al., 2021a) to evaluate zero-shot domain generalisation in English (leveraging the SPIDER (question, SQL) examples index).

Consistently with the relevant leaderboards[3], we report results using execution (EX) and exact match (EM) accuracy.[4] Since CSPIDER comes without relevant DB content, we follow previous works, and we focus our evaluation on EM scores (Vougiouklis et al., 2023; Cao et al., 2023).

Model	EX	EM
GraPPa (Yu et al., 2021)	–	73.6
FastRAT (Vougiouklis et al., 2023)	73.2	69.1
FastRAT$_{\text{ext}}$	71.5	64.2
Graphix-T5 (Li et al., 2023b)	81.0	77.1
RESDSQL (Li et al., 2023a)	84.1	**80.5**
`deepseek-coder-33b-instruct`		
Ours (w/ FastRAT$_{\text{ext}}$)	81.5	62.1
Ours (w/ Graphix-T5)	83.4	64.7
`PaLM2`		
Few-shot SQL-PaLM (Sun et al., 2024)	82.7	–
`text-davinci-003`		
Zero-shot (Guo et al., 2024)	73.1	–
RAG w/ Rev. Chain (Guo et al., 2024)	85.0	–
`gpt-3.5-turbo`		
Zero-shot (Liu et al., 2023)	70.1	–
C3 (Dong et al., 2023)	81.8	–
DAIL-SQL (Gao et al., 2023)	79.0	–
Ours (w/ FastRAT$_{\text{ext}}$)	82.0	65.7
Ours (w/ Graphix-T5)	83.0	68.8
`gpt-4`		
Zero-shot (Pourreza and Rafiei, 2023)	72.9	40.4
DIN-SQL (Pourreza and Rafiei, 2023)	82.8	60.1
DAIL-SQL (Gao et al., 2023)	83.6	68.7
Ours (w/ FastRAT$_{\text{ext}}$)	84.3	73.8
Ours (w/ Graphix-T5)	**86.6**	77.3

Table 2: EX and EM accuracies on the development split of SPIDER. Fine-tuning-based baselines are listed at the top part of the table. Results of our approach are shown with both FastRAT$_{\text{ext}}$ and Graphix-T5 as approximators. The **best** model is in bold, the second best is underlined, and the best prompt-based setup is in blue.

5.2 Baselines

We dichotomize the landscape of baselines in fine-tuning- and prompting-based baselines. Further details are provided in Appendix E.

Fine-tuning-based (i) **GraPPa**, (ii) **DG-MAML**, (iii) **FastRAT**, (iv) **Graphix-T5**, (v) **RESDSQL** and (vi) **HG2AST**.

Prompting-based Zero-shot LLM prompting has been explored by Guo et al.; Liu et al.; Pourreza and Rafiei; (i) **C3** introduces calibration bias for LLM prompting; (ii) **DIN-SQL** uses chain-of-thought prompting with pre-defined prompting templates tailored for the assessed question hardness; (iii) **DAIL-SQL** uses query approximation and SQL skeleton-based similarities for example selection; (iv) **SQL-PaLM** proposes a framework for *soft* column selection and execution-based refinement; (v) **RAG w/ Rev. Chain** augments the input prompt with question skeleton-based example retrieval and an execution-based revision chain.

5.3 Text-to-SQL Evaluation

Table 2 and 3 summarise the results of our approach with `deepseek-coder-33b-instruct`, `gpt-3.5-turbo` and `gpt-4` against the baselines. Our approach, comprising a single-prompting round, surpasses other LLM-based solutions, that incorporate several prompting iterations, for LLMs of the same capacity. We note consistent improvements over DAIL-SQL, the closest work to ours, even when FastRAT$_{\text{ext}}$ is used as approximator (i.e. a model consisting of < 500M vs the ≥ 3B parameters that DAIL-SQL's approximator is using). For the same approximator, our framework is able to meet, performance standards of DAIL-SQL (equipped with `gpt-4` and an additional self-consistency prompting step) using an open-source model as backbone LLM, by achieving shorter prompts in a single prompting step.

SPIDER results are consistent with the results across the various Spider variants and CSPIDER[5] (Table 3). Our approach levering FastRAT$_{\text{ext}}$ and AST-based re-ranking for example selection outperforms other prompting-based solution, and is inline with the scores of state-of-the-art fine-tuning-based baselines. While `gpt-4` is the most capable model within our framework (with this being more noticeable in the case of SPIDER-SYN), we observe surprising findings with DeepSeek with which in many cases our approach can surpass much more computationally expensive alternatives

[3] https://taolusi.github.io/CSpider-explorer/ and https://yale-lily.github.io/spider

[4] EX and EM scores are computed using: https://github.com/taoyds/test-suite-sql-eval.

[5] In CSPIDER, questions are fully translated in Chinese while the DB content remains in English. Due to this limitation, DB schema and content selection are disabled.

Model	SPIDER-DK		SPIDER-REAL		SPIDER-SYN		CSPIDER
	EX	EM	EX	EM	EX	EM	EM
RAT-SQL + BERT (Wang et al., 2020)	–	40.9	62.1	58.1	–	48.2	–
DG-MAML (Wang et al., 2021)	–	–	–	–	–	–	51.0
FastRAT (Vougiouklis et al., 2023)	–	–	–	–	–	–	61.3
$\text{FastRAT}_{\text{ext}}$	–	44.1	–	47.8	–	48.5	53.2
HG2AST (Cao et al., 2023)	–	–	–	–	–	–	61.0[a]
RESDSQL (Li et al., 2023a)	66.0	55.3	**81.9**	**77.4**	**76.9**	**69.1**	–
`deepseek-coder-33b-instruct`							
Ours (w/ $\text{FastRAT}_{\text{ext}}$)	70.5	46.4	77.4	59.3	68.7	49.5	55.9
`gpt-3.5-turbo`							
Zero-shot (Liu et al., 2023)	62.6	–	63.4	–	58.6	–	32.6
DAIL-SQL (Gao et al., 2023)	–	–	67.9	–	–	–	–
Ours (w/ $\text{FastRAT}_{\text{ext}}$)	68.8	49.3	78.0	60.8	66.9	51.2	54.0
`PaLM2`							
Few-shot SQL-PaLM (Sun et al., 2024)	66.5	–	77.6	–	74.6	–	–
`gpt-4`							
DAIL-SQL (Gao et al., 2023)	–	–	76.0	–	–	–	–
Ours (w/ $\text{FastRAT}_{\text{ext}}$)	**72.3**	**59.1**	80.9	66.1	74.4	61.3	**64.4**

Table 3: Results on SPIDER-DK, SPIDER-REAL, SPIDER-SYN and CSPIDER. Fine-tuning-based baselines are listed at the top. The **best** model is in bold, second best is underlined, and the best prompt-based setup is in blue.

[a]Without using question translation; 64.0 EM when question translation is used.

based on larger closed-source LLMs. Our findings remain consistent with (Liu et al., 2023) since the EM scores of prompting-based methods fall behind those of their fine-tuning based counterparts.

5.3.1 Schema Selection Evaluation

We evaluate our proposed schema selection strategy in a two-fold manner, given that value selection is applied for selected columns. Firstly, we use recall and schema shortening (rate) to compute averaged metrics across all samples showcasing the extent to which (i) the most relevant schema elements are successfully retrieved, and (ii) the size of the resulting schema, after selection, with respect to its original size. Secondly, we explore how the performance of the end-system changes across different schema pruning settings by reporting EX and EM scores. Recall is the percentage of samples for which all ground-truth schema elements are selected. Schema shortening is the number of schema elements that are excluded divided by the total number of schema elements. Results are summarised in Table 4.

The benefits of schema selection are apparent in the oracle setup, in which only schema elements from the gold query are included in the prompt (cf. Table 8). In this setup, the highest execution accuracy is achieved while filtering out $> 70\%$ of the original schema on average. We note that our approach of coupling the schema elements returned in the approximated query with the ones returned by BM25 navigates a healthy trade-off between maximising recall and reducing processing of unnecessary schema elements. We also notice that our strategy of dynamically determining the number of retained schema elements per input (cf. Section 4.1.1) results in improvements compared to static top-k determination. For roughly the same extent of schema shortening (i.e. by comparing scores with dynamic top-k against top-7), the results with the former are higher across all metrics.

5.3.2 Ablation Study

Table 5 shows a comprehensive ablation study for the efficacy of our database context selection, and example selection methods including DAIL (Gao et al., 2023) and AST. We consistently notice improvement when selecting examples using AST, for the same approximator. Interestingly, the performance gap is increasing the better the approximator becomes, leading to an improvement $> 2.4\%$ in the case of an oracle approximator. This finding is in agreement with our hypothesis that AST re-ranking can preserve structural information for more precise example selection when $\mathrm{s}' \sim \mathrm{s}$. The inclusion of combined schema and value selection (SVS) leads to further improvements when coupled with example selection based on AST or DAIL.

Approximator	Schema Selection Setup	Recall	Schema Shorten.	EX	EM
Oracle	Gold Query	100.0	71.3	86.3	73.9
$\text{FastRAT}_{\text{ext}}$	N/A	100.0	0.0	79.3	63.6
$\text{FastRAT}_{\text{ext}}$	BM25 (top-10)	92.0	36.5	78.9	64.1
$\text{FastRAT}_{\text{ext}}$	BM25 (top-20)	98.3	14.1	80.7	64.9
$\text{FastRAT}_{\text{ext}}$	Approx. Query	86.8	71.3	78.4	63.8
$\text{FastRAT}_{\text{ext}}$	Approx. Query + BM25 (top-7)	93.3	50.4	81.1	65.4
$\text{FastRAT}_{\text{ext}}$	Approx. Query + BM25 (top-10)	97.0	37.3	81.2	65.6
$\text{FastRAT}_{\text{ext}}$	Approx. Query + BM25 (dynamic top-k)	97.2	49.0	**82.0**	**65.7**
Graphix-T5	N/A	100.0	0.0	79.8	65.6
Graphix-T5	Approx. Query	92.3	71.8	81.8	68.8
Graphix-T5	Approx. Query + BM25 (dynamic top-k)	97.9	49.4	**83.0**	**68.8**

Table 4: Recall, Schema Shortening, EX and EM scores (using `gpt-3.5-turbo`) across different schema selection setups, on the development split of SPIDER. Value selection is enabled for the selected columns across all setups. For the oracle setup, we report performance upper-bounds using *only* the schema elements from the gold query.

Approximator	Selection	EX	EM
N/A	Question Similarity	74.7	52.3
$\text{FastRAT}_{\text{ext}}$	DAIL	78.6	61.4
	DAIL + SVS	81.3	62.3
	AST	79.3	63.6
	AST + VS	80.4	63.8
	AST + SS	78.9	63.8
	AST + SVS	82.0	65.7
Graphix-T5	DAIL	77.8	61.9
	DAIL + SVS	81.0	63.7
	AST	79.8	65.6
	AST + VS	81.4	66.4
	AST + SS	80.2	66.2
	AST + SVS	83.0	68.8
Oracle	DAIL	79.1	63.2
	DAIL + SVS	82.5	66.0
	AST	81.0	67.6
	AST + VS	82.6	68.1
	AST + SS	82.5	69.6
	AST + SVS	84.6	71.3

Table 5: EX and EM scores on the development set of SPIDER, with `gpt-3.5-turbo`, across different approximators, and selection setups: example selection (with DAIL or AST), schema selection (SS), value selection (VS), and schema & value selection (SVS). We report results from oracle approximator using gold queries.

5.4 Schema Splitting

We evaluate the effect of splitting a schema into r_m splits, using $\text{FastRAT}_{\text{ext}}$ for schema selection. Figure 1 shows EX scores across different maximum number of columns per schema split (r), on the development set of SPIDER. We see that the EX scores of our approach remain consistent across different r. The performance in the case where particular schemata from the development set are split into $r_m = 3$ or $r_m = 4$ splits (i.e. for $r = 24$ or $r = 16$ respectively) is identical to the scores where schemata are split using the default r with which $\text{FastRAT}_{\text{ext}}$ has been trained.

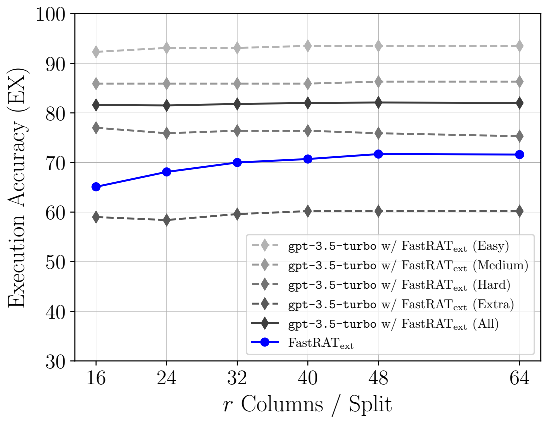

Figure 1: Execution accuracy scores on on the development set of SPIDER across different maximum numbers of columns per schema split, r. The results of our approach, using `gpt-3.5-turbo`, are presented across different SPIDER-query difficulty levels.

6 Discussion

Are there any theoretical performance upper limits for example selection using AST? For each data instance in the development set of Spider, we compute the average AST similarity between the approximated query and each SQL query that is included (after example selection) in the corresponding prompt. In Table 6, we measure EX scores on the development set of SPIDER, across different AST-similarity intervals. We see an obvious correlation between execution accuracy and AST scores–execution accuracy is higher for higher AST scores. Besides highlighting an empirical, execution accuracy upper-bound, in the case of test questions whose SQL structure is well-covered (i.e. with high AST score) in the examples space, our approach can hint data instances that might be challenging for the current configuration, without even requiring to prompt the target LLM or executing the resulting SQL against a database instance. Challenging data instances can be taken into consideration with respect to the existence of insufficient examples in $\mathbb{X}$ to support the expected SQL structure or a potentially harmful approximator.

AST Interval	Approximator		
	Oracle	Graphix	FastRAT$_{\text{ext}}$
$[0.0, 1.0]$	84.6	83.0	82.0
$[0.95, 1.0]$	90.8	88.4	88.7
$[0.9, 0.95)$	80.7	74.1	76.8
$[0.85, 0.9)$	73.0	68.3	59.4
$[0.8, 0.85)$	62.4	66.8	63.5
$[0.0, 0.8)$	50.0	54.1	58.7

Table 6: EX scores on the SPIDER development set using gpt-3.5-turbo, across different average AST-similarity intervals.

Is the choice of approximator critical? Although our AST re-ranking and schema selection benefit from more accurate SQL predicted by a stronger approximator, the choice of approximator depends on the desired trade-off between effectiveness and efficiency in practice. FastRAT$_{\text{ext}}$ is over 600 times faster than Graphix-T5 (Li et al., 2023b) on an A100 80G GPU, while the resulting difference, within our framework, in EX on the SPIDER development set is $\leq 1\%$ (Table 5).

Does schema selection improve the performance? In Table 5, we noted that performing schema selection (i.e. SS) without DB value selection does not necessarily lead to performance improvements. This is in partial agreement with Sun et al. that hard column selection can be harmful for the end-to-end performance, and can be attributed to the drop of recall, when less capable approximators are involved. Nonetheless, as we note in Section 5.3.2, the combination of schema and value selection (SVS) can consistently improve EX and EM, while significantly reducing the LLM token-processing cost due to shortened schema.

7 Related Work

Significant number of recent works have looked at how LLMs can be employed in Text-to-SQL scenarios (Rajkumar et al., 2022; Chang and Fosler-Lussier, 2023; Liu et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2023; Guo et al., 2024). More recent works have looked at how incorporating examples in the prompt could benefit the performance of LLMs in the end task (Pourreza and Rafiei, 2023; Gao et al., 2023; Guo et al., 2024; Sun et al., 2024). In spite of its underlying benefits, conventional solutions for example selection have focused on retrieving pairs using question similarity (Nan et al., 2023). Other approaches have sought to *approximate* expected SQL queries, and either directly use these approximations in the prompt, in a few-shot setting (Sun et al., 2024) or to filter candidate (question, SQL) pairs by taking into consideration the similarity of their corresponding SQL query skeletons (Li et al., 2023a) against the skeleton of the approximated SQL (Gao et al., 2023). We argue that such example selection strategies can result in information loss, and we propose an approach for re-ranking examples using similarity of normalised SQL ASTs.

The benefits of schema selection for Text-to-SQL have been highlighted across the relevant bibliography(Wang et al., 2020; Li et al., 2023b; Pourreza and Rafiei, 2023). From the LLMs perspective, pruning schema elements from the prompts has been usually leading to performance degradation(Sun et al., 2024). Inspired by Gao et al., we compute a preliminary query for a given $(\mathbf{q}, \mathbf{D})$ by we adapting FastRAT (Vougiouklis et al., 2023), to the requirements of processing longer schemata, in a parallelised manner. We couple the resulting *approximator* with a sparse retriever, and we propose a dynamic strategy for reducing the computational cost of the task while achieving performance improvements.

8 Conclusion

In this paper, we augment LLMs for Text-to-SQL semantic parsing by selecting suitable examples and database information. We present a novel AST-based metric to rank examples by similarity of SQL queries. Our hybrid search strategy for schema selection reuses a preliminary query to reduce irrelevant schema elements while maintaining high recall. Extensive experiments demonstrate that our AST-based ranking outperforms previous approaches of example selection and that a symbiotic combination of schema and value selection can further enhance the end-to-end performance of both closed- and open-source LLM solutions.

Limitations

There are limitations with regards to both of our example selection and schema selection. Our AST-based ranking can be biased when an approximated SQL deviates significantly from structurally correct answers. To address the failure of approximators, a future direction is to sensibly diversify selected examples such that LLMs can generalise compositionally. As for schema selection, our semantic search relies on an approximator which is essentially a parser with high precision in schema linking but lack mechanisms to control recall as a standalone model. Therefore, it is worth extending cross-encoder architecture such as FastRAT to support ranking schema elements while being a SQL approximator in the meantime.

We demonstrate that schema splitting strategies within our framework can be applied across various numbers of splits without noticeable performance degradation. Nonetheless, given the lack of available datasets that incorporate longer commercial schemata, we focus our experiments on the cross-database setting provided by CSPIDER and SPIDER variants.

Ethics Statement

We do not make use of any private, proprietary, or sensitive data. $\text{FastRAT}_{\text{ext}}$ is trained on publicly available Text-to-SQL datasets, using publicly available encoder-models as base. Our framework for retrieval-augmented generation builds on-top of large, pre-trained language models, which may have been trained using proprietary data (e.g. in the case of the OpenAI models). Given the nature of pre-training schemes, it is possible that our system could carry forward biases present in the datasets and/or the involved LLMs.

References

Shengnan An, Bo Zhou, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Weizhu Chen, and Jian-Guang Lou. 2023. Skill-based few-shot selection for in-context learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13472–13492, Singapore. Association for Computational Linguistics.

Ruisheng Cao, Lu Chen, Jieyu Li, Hanchong Zhang, Hongshen Xu, Wangyou Zhang, and Kai Yu. 2023. A heterogeneous graph to abstract syntax tree framework for text-to-SQL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13796–13813.

Shuaichen Chang and Eric Fosler-Lussier. 2023. How to prompt llms for text-to-SQL: A study in zero-shot, single-domain, and cross-domain settings. *Preprint*, arXiv:2305.11853.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

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.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-SQL with ChatGPT. *Preprint*, arXiv:2307.07306.

Beat Fluri, Michael Wursch, Martin PInzger, and Harald Gall. 2007. Change distilling:tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743.

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. *CoRR*, abs/2308.15363.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2024. Retrieval-augmented GPT-3.5-based text-to-SQL framework with sample-aware prompting and dynamic revision chain. In *Neural Information Processing*, pages 341–356, Singapore. Springer Nature Singapore.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-sql. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):13067–13075.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of ChatGPT's zero-shot text-to-SQL capability. *Preprint*, arXiv:2303.13547.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. A pilot study for Chinese SQL semantic parsing. 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 3652–3658, Hong Kong, China. Association for Computational Linguistics.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing text-to-SQL capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956, Singapore. Association for Computational Linguistics.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *Preprint*, arXiv:2304.11015.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine

Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-SQL capabilities of large language models. *Preprint*, arXiv:2204.00498.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a method for automatic evaluation of code synthesis. *Preprint*, arXiv:2009.10297.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST).

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems*, volume 33, pages 16857–16867. Curran Associates, Inc.

Ruoxi Sun, Sercan Ö. Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. SQL-PaLM: Improved large language model adaptation for text-to-sql (extended). *Preprint*, arXiv:2306.00739.

Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. 2019. Does bleu score work for code migration? In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 165–176.

Pavlos Vougiouklis, Nikos Papasarantopoulos, Danna Zheng, David Tuckey, Chenxin Diao, Zhili Shen, and Jeff Pan. 2023. FastRAT: Fast and efficient cross-lingual text-to-SQL semantic parsing. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 564–576, Nusa Dua, Bali. Association for Computational Linguistics.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021. Meta-learning for domain generalization in semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–379, Online. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of GPT-3 and GPT-3.5 series models. *Preprint*, arXiv:2303.10420.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

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

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055, Portland, Oregon. AAAI Press.

A SQL Similarity using Normalised Abstract Syntax Trees

Table 7 shows the corresponding SQL queries after each step of our AST normalisation as explained in Section 3. An example of the similarity between normalised ASTs is provided in Figure 2, where tables, columns and values are masked out for cross-domain settings.

B Prompt Formulation

Table 8 shows an example of our prompt, after example and DB context selection (i.e. schema and value selection). This prompt is provided as input

SQL	`SELECT T1.Category, COUNT(*) AS Num FROM Products AS T1 JOIN Orders AS T2 ON T1.id = T2.pid GROUP BY T1.Category ORDER BY Num ASC`
1. Unify Identifiers	`SELECT t1.category, COUNT(*) AS num FROM products AS t1 JOIN orders AS t2 ON t1.id = t2.pid GROUP BY t1.category ORDER BY num ASC`
2. Resolve Aliases	`SELECT products.category, COUNT(*) FROM products JOIN orders ON products.id = orders.pid GROUP BY products.category ORDER BY COUNT(*) ASC`
3. Reorder JOIN	`SELECT products.category, COUNT(*) FROM orders JOIN products ON orders.id = products.pid GROUP BY products.category ORDER BY COUNT(*) ASC`
4. Mask Identifiers & Values (cross-domain only)	`SELECT _, COUNT(*) FROM _ JOIN _ ON _ = _ GROUP BY _ ORDER BY COUNT(*) ASC`

Table 7: An example of the effects to corresponding SQL after each step of our AST normalisation.

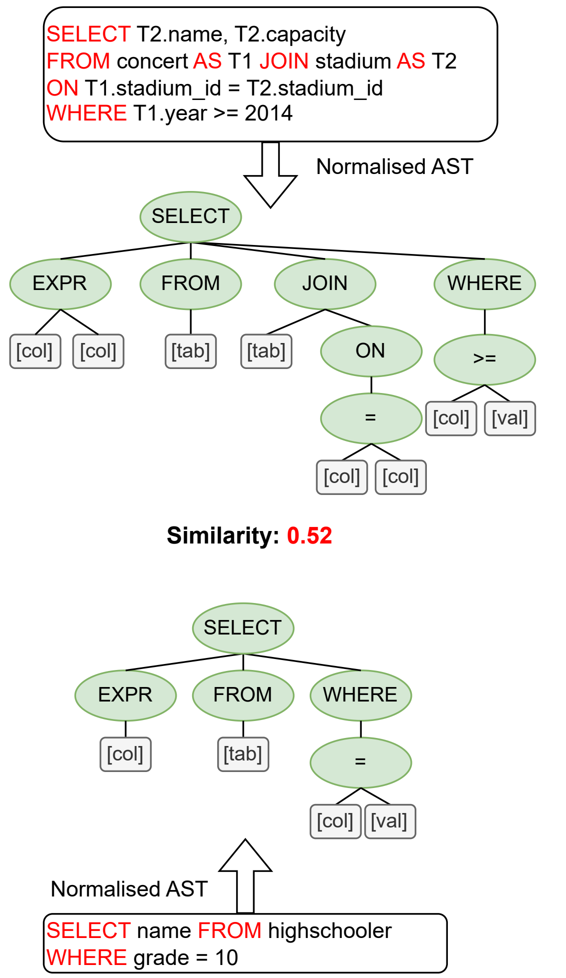

Figure 2: Example of how the similarity between two different SQL queries is computed using normalised ASTs.

to LLMs. Following the latest OpenAI example[6] for Text-to-SQL parsing, we represent a schema with `CREATE TABLE` statements in SQL. Semantic names or descriptions of tables and columns are included as `COMMENT` along with the corresponding columns or tables. Note that we filter out comments that can be obtained by simply lowercasing original names and/or removing underscores. To maintain a compact representation of database information, we append selected values of columns into their `COMMENT` rather than introducing additional lines as in the work by Chen et al. (2024). Example (question, SQL) pairs are provided in a similar manner to DAIL-SQL (Gao et al., 2023), followed by an instruction to prompt LLMs to generate SQL for the test question.

C Implementation Details

We use this section to provide further details about the implementation of our approach.

C.1 Example Selection

Following (Gao et al., 2023), we employ the pre-trained `all-mpnet-base-v2` model (Song et al., 2020) from Sentence Transformer (Reimers and Gurevych, 2019) to compute dense question embeddings for English datasets including SPIDER, SPIDER-DK, SPIDER-REAL, and SPIDER-SYN. For CSPIDER, `paraphrase-multilingual-MiniLM-L12-v2` is used instead. SQL queries are parsed into AST by using SQLGlot[7] and are then normalised as explained in Section 3. SQLGlot provides an implementation of the Change Distilling algorithm for AST differencing. We refer readers to SQLGlot's documentation[8] for more details. For selecting example question-SQL pairs, we first retrieve top 500 examples by question similarity and rerank them in terms of the similarity of normalised SQL ASTs. For relevant experiments in Table 5, we reproduced the implementation of

[6] https://platform.openai.com/examples/default-sql-translate

[7] https://github.com/tobymao/sqlglot

[8] https://github.com/tobymao/sqlglot/blob/main/posts/sql_diff.md

	# Given SQLite database schema student_transcripts:
Selected Schema w/ Selected Values	CREATE TABLE Departments(department_id number, department_name text COMMENT 'department name (e.g. engineer, statistics, medical) ', ...); CREATE TABLE Degree_Programs(degree_program_id number, degree_summary_name text COMMENT 'degree summary name (e.g. PHD, Master, Bachelor) ', ... PRIMARY KEY (degree_program_id), FOREIGN KEY (department_id) REFERENCES Departments(department_id));
	# Your task is to translate Question into SQL. # Some examples are provided based on similar problems:
Selected Examples	Question: How many courses does the department of Computer Information Systems offer? SQL: SELECT count(*) FROM department AS T1 JOIN course AS T2 ON T1.dept_code = T2.dept_code WHERE dept_name = "Computer Info.Systems" Question: ... SQL: ...
	# Complete the following SQL for schema student_transcripts:
Test Question	Question: How many degrees does the engineering department have? SQL:

Table 8: An example of the resulting prompt, after example and schema and DB content selection.

DAIL selection from the original paper (Gao et al., 2023). The number of few-shot examples is set to 5 across all experiments.

C.2 Schema & Value Selection

Each database schema is treated as an independent collection of columns that are analogous to documents to be retrieved by using BM25. As mentioned in Section 4.1, we represent a column by concatenating semantic names of both the column and its table, and the column values in the database. Semantic names and values are tokenized using spaCy[9] and preprocessed by lowercasing and stemming[10]. At inference time, the same processing is applied to questions. We adopt the implementation of Okapi BM25 (Robertson et al., 1994) from Rank-BM25[11]. The number of columns to retrieve is dynamically set to $\lfloor 1.5 \times \gamma \rfloor$ where γ is the number of unique columns in an approximated query. We limit the resulting number to a range between 6 and 20. By retrieving at column level, a table is selected if any of its columns are selected. We merge retrieved schema elements with schema elements from the approximated query to construct a sub-schema. To further increase the recall, we add additional primary keys and foreign keys that are not selected but valid based on selected tables, except for experiments where only approximated queries are used (see Table 4). In such cases, however, if the SQL query involves only tables (e.g. `SELECT * FROM books`), primary keys of selected tables are still included to ensure that corresponding `CREATE TABLE` statements (see Table 8) are meaningful and consistent.

For selecting values, similarly, we match the input question and the set of values for each (selected) column that has a non-numeric type. The top 3 results are added to the prompt as exemplified in Table 8. The same setting of schema and value selection is used for all datasets we experimented with except CSPIDER. Due to the cross-lingual nature of CSPIDER, schema selection and value selection are simply disabled.

Training FastRAT$_{\text{ext}}$ We follow the original hyper-parameters provided by (Vougiouklis et al., 2023) for training FastRAT$_{\text{ext}}$. The monolingual version of FastRAT is based on BERT$_{\text{LARGE}}$ while its cross-lingual variant on XLM-RoBERTa-large.

C.3 OpenAI Models

We use `gpt-4` (`gpt-4-0613`) and `gpt-3.5-turbo` (`gpt-3.5-turbo-0613`) for our experiments. For decoding, sampling is disabled and the maximum number of tokens to generate is set to 256. A single experiment, on the SPIDER development set using our approach with FastRAT$_{\text{ext}}$ as the approximator and dynamic database context selection costs around \$0.8 and \$16.5 in the case of `gpt-3.5-turbo-0613` and `gpt-4-0613` respectively.

C.4 Experiments with Open-Source LLMs

We further conduct experiments with open-source models from the DeepSeek family[12], that specialise in code generation. Prompting and decoding setups remain consistent across all LLMs (cf. Section C of the Appendix). Table 9 summarises the results. We see that our approach can generalise even in the case of open-source LLM alternatives. Interestingly, our scores using `deepseek-coder-33b-instruct` are comparable to the scores when using `gpt-3.5-turbo-0613` across all approximators. Inference experiments are conducted on a machine using 8×NVIDIA-V100 32G GPUs.

[9] https://github.com/explosion/spaCy

[10] https://www.nltk.org/api/nltk.stem.porter.html

[11] https://github.com/dorianbrown/rank_bm25

[12] We use the implementations provided by https://huggingface.co/deepseek-ai.

Model	Spider		Spider-DK		Spider-Real		Spider-Syn		CSpider
	EX	EM	EX	EM	EX	EM	EX	EM	EM
`deepseek-coder-6.7b-instruct`									
Ours (w/ $\text{FastRAT}_{\text{ext}}$)	78.6	64.8	66.7	46.4	73.2	55.7	66.0	49.5	53.0
Ours (w/ Graphix-T5)	79.5	64.8	–	–	–	–	–	–	–
`deepseek-coder-33b-instruct`									
Ours (w/ $\text{FastRAT}_{\text{ext}}$)	81.5	62.1	70.5	46.4	77.4	59.3	68.7	49.5	55.9
Ours (w/ Graphix-T5)	83.4	64.7	–	–	–	–	–	–	–

Table 9: Execution (EX) and exact match (EM) accuracy scores of our approach using DeepSeek family models, on the development splits of Spider and CSpider, and the Spider-DK, Spider-Real and Spider-Syn test splits. CSpider results are using only $\text{FastRAT}_{\text{ext}}$ as approximator.

D Spider and CSpider Experiments

We report experiments on CSpider (Min et al., 2019) and Spider (Yu et al., 2018), which contain database schema information and examples in Chinese and English respectively. Since CSpider is a translated version of the Spider dataset, the characteristics of the two with respect to structure and number of examples are identical. Both datasets contain 8,659 examples of questions and SQL queries along with their relevant SQL schemata (i.e. 146 unique databases). The development and test[13] sets consist of 1,034, on 20 unique databases and 2,147, on 40 unique databases, respectively, and none of the relevant databases are seen in the training set. Due to the scarcity of works reporting test scores on these benchmarks, we chose not to include our results in the main body of our manuscript. Table 10 shows the performance of our framework with respect to execution and exact match accuracy scores on the test splits of Spider and CSpider.

E Baselines

We compare the performance of our approach against several baselines. We dichotomize the landscape of baselines in fine-tuning- and prompting-based baselines.

Fine-tuning-based **GraPPa** uses synthetic data constructed via induced synchronous context-free grammar for pre-training an MLM on the SSP-label classification; **DG-MAML** applies meta-learning targeting zero-shot domain generalization. **FastRAT** incorporates a decoder-free framework, by directly predicting SQL queries from SSP labels; **Graphix-T5** inserts a graph-aware layer into T5 (Raffel et al., 2020) to introduce structural inductive bias; **RESDSQL** decouples schema linking and SQL skeleton parsing using a framework based on a ranking-enhanced encoder and skeleton-aware decoder; **HG2AST** proposes a framework to integrate dedicated structure knowledge by transforming heterogeneous graphs to abstract syntax trees.

Prompting-based Zero-shot prompting with LLMs has been explored by Guo et al.; Liu et al.; Pourreza and Rafiei; **C3** introduces calibration bias prompting to alleviate LLMs' biases; **DIN-SQL** uses chain-of-thought prompting with pre-defined prompting templates tailored for the assessed question hardness; **DAIL-SQL** uses query approximation and SQL skeleton-based similarities for example selection; **SQL-PaLM** proposes a framework for *soft* column selection and execution-based refinement; **RAG w/ Rev. Chain** augments the input prompt with question skeleton-based example retrieval and an execution-based revision chain.

[13] Since the 1st of March 2024, the test sets of both Spider and CSpider have become publicly available.

Model	Easy		Medium		Hard		Extra		All	
	EX	**EM**	**EX**	**EM**	**EX**	**EM**	**EX**	**EM**	**EX**	**EM**
					SPIDER					
FastRAT$_{\text{ext}}$	86.2	81.3	72.2	66.0	60.0	51.6	48.5	33.9	68.7	60.9
`deepseek-coder-33b-instruct`										
Ours (w/ Graphix-T5)	89.6	85.5	88.6	66.4	73.9	50.1	58.8	28.0	80.7	60.7
`gpt-4`										
Ours (w/ Graphix-T5)	91.9	87.4	90.3	80.4	81.2	66.1	74.2	47.6	86.0	73.4
					CSPIDER					
FastRAT$_{\text{ext}}$	–	67.2	–	49.9	–	41.5	–	12.9	–	45.5
`deepseek-coder-33b-instruct`										
Ours (w/ FastRAT$_{\text{ext}}$)	–	79.7	–	58.2	–	40.2		22.4	–	52.9
`gpt-4`										
Ours (w/ FastRAT$_{\text{ext}}$)	–	81.2	–	67.7	–	53.2	–	29.9	–	61.1

Table 10: Execution (EX) and exact match (EM) accuracy scores of our framework, on the test splits of SPIDER and CSPIDER.