SYNQL: Synthetic Data Generation for In-Domain, Low-Resource Text-to-SQL Parsing

Denver Baumgartner
Semiotic Labs / Los Altos, CA, USA
denver@semiotic.ai

Tomasz Kornuta
Semiotic Labs / Los Altos, CA, USA
tomasz@semiotic.ai

Abstract

We address the challenge of generating high-quality data for text-to-SQL parsing in low-resource, in-domain scenarios. Although leveraging large language models (LLMs) and in-context learning often achieves the best results in research settings, it is frequently impractical for real-world applications. Therefore, fine-tuning smaller, domain-specific models provides a viable alternative. However, the scarcity of training data frequently constrains it. To overcome this, we introduce SYNQL, a novel method for synthetic text-to-SQL data generation tailored for in-domain contexts. We demonstrate the effectiveness of SYNQL on the KaggleDBQA benchmark, showing significant performance improvements over models fine-tuned on original data. Additionally, we validate our method on the out-of-domain Spider dataset. We open-source the method and both synthetic datasets.

1 Introduction

Semantic parsing Kamath and Das [2018] is the process of mapping natural language to machine-interpretable representations. Text-to-SQL Zelle and Mooney [1996], Qin et al. [2022], Katsogiannis-Meimarakis and Koutrika [2023] is a specialized form of semantic parsing that converts natural language questions into SQL queries. Recent progress has been enabled by the emergence of several large-scale benchmarks [Zhong et al., 2017, Yu et al., 2018, Li et al., 2023] and marked by two main research directions: In-context learning [Brown et al., 2020, Pourreza and Rafiei, 2024] and Fine-tuning Raffel et al. [2020]. In-context learning, primarily relying on the latest large language models (LLMs) such as GPT-4 [Achiam et al., 2023] or Llama 2 [Touvron et al., 2023], is currently demonstrating superior performance on most benchmarks [Chang and Fosler-Lussier, 2023, Wang et al., 2024]. However, this approach is impractical for many real-world applications due to high inference costs and/or latency. Because of this, many have focused on fine-tuning smaller models tailored to a given domain and database(s). It appears that in most cases, people encounter a *low-resource scenario*, where high-quality, domain-specific data is scarce, and only a limited number of Question-Query Pairs (QQPs) is initially available. Given that manual data generation is both expensive and labor-intensive, alternative methods involving Synthetic Data Generation (SDG) and data augmentation techniques for the text-to-SQL domain [Yu et al., 2021, Wu et al., 2022, Hu et al., 2023] have become increasingly essential. In this paper, we focus on this problem and propose a new method for generating synthetic text-to-SQL data. Contributions of this paper are as follows:

- We present a systematic comparison of text-to-SQL SDG methods and highlight the need for more diverse data.
- To address this, we propose SYNQL, a novel synthetic data generation method for text-to-SQL that leverages in-context learning and introduces 'Topics'—a new type of contextual information enhancing the diversity of generated QQPs.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Paper / Method	SQL Synthesis	NLQ Synthesis
Wang et al. [2021] SQL→NLQ	**Algorithm:** 1) Construct dataset-specific PCFG from original dataset 2) Generate SQLs using the constructed PCFG **Contextual data:** Grammar/PCFG extracted from dataset	**Algorithm:** 1) Fine-tune BART on original data for SQL-to-NLQ 2) Prompting: generated SQLs-to-NLQs **Contextual data:** Generated SQLs
Wu et al. [2022] SQL→NLQ	**Algorithm:** 1) Construct an abstract syntax tree grammar (ASTG) 2) Generate SQL using ASTG until 80% coverage of the original dataset's SQL Templates occurs **Contextual data:** Target SQL Template Distribution	**Algorithm:** 1) SQLs are decomposed and represented as segments 2) Train a copy-based seq2seq model for SQL-to-NLQ 3) Generate & compose NLQs from generated SQLs **Contextual data:** Generated SQLs
Hu et al. [2023] SQL→IR→NLQ	**Algorithm:** 1) Extract SQL Templates from original dataset 2) Fill templates using schema-weighted column sampling **Contextual data:** Extracted SQL Templates	**Algorithm:** 1) Fine-tune T5/Large models for SQL-to-IR & IR-to-NLQ 2) Prompting: T5/Large models for SQL→IR & IR→NLQ **Contextual data:** Generated SQLs
	NLQ Synthesis	**SQL Synthesis**
Yang et al. [2021] (H-NEURSYN) ES→NLQ→SQL	**Algorithm:** 1) Extract Entity Sequences (ES) from SQL 2) Train Schema-to-ES & ES-to-NLQ models (T5) 3) Sample ESs & generate NLQs **Contextual data:** Database Schemas	**Algorithm:** 1) Train a NLQ→SQL T5 model 2) Generate QQPs from NLQs: beam search w/ execution 3) Deduplicate resulting data (including origin) **Contextual data:** Generated NLQs
Ye et al. [2023] (SEMGEN) NLQ→SQL	**Algorithm:** Prompting: Codex/davinci-002 **Contextual data:** - Database Schemas (+ ST + KR) - 3 exemplary rows per table (pulled from database) - 10-shot NLQs (randomly sampled from original dataset)	**Algorithm:** Prompting: Codex (+ filter: valid execution) **Contextual data:** - Database Schemas (+ ST + KR) - NQL (synthesized) - 3 exemplary rows per table (pulled from database) - 10-shot QQPs (randomly sampled from original dataset)
	Joint SQL+NLQ Synthesis	
Yu et al. [2021] (GRAPPA) NLQ+SQL	**Algorithm:** 1) Induce grammer/SCFG based on the original dataset 2) Select 90 predominant SQL Template programs	3) For each SQL program select 4 associated NLQ templates 4) Fill template pairs, sampling the SCFG given a database **Contextual data:** Extracted grammar rules (SCFG)
Kuznia [2023] NLQ+SQL	**Algorithm:** Prompting: GPT-3 (+filter: non-NULL exec.) **Contextual data:** - Database Schemas (+ ST + KR)	- 3 exemplary rows per table (pulled from database) - 5-shot QQPs (manually selected & curated)
Ours (SYNQL) T→SQL+NQL	**Algorithm:** 1) Extraction of SQL Templates from the original dataset 2) Prompting: GPT-4 (Topic generation) 3) Prompting: GPT-4 (+ filter: valid execution)	**Contextual data:** - Database Schemas (+ ST + KR) - Extracted SQL Templates - Synthesized Topics

Table 1: Comparison of selected state-of-the-art Synthetic Data Generation methods. Arrow → in **Method** indicates order of main synthesis steps, similarly as numbers in particular algorithms. (+ ST + KR) next to Database Schema indicates that the used Schema includes Strong Typing (ST) and Key Relationships (KR). (+ filter: valid execution) next to Prompting indicates an additional step of checking if synthesized SQL is valid and can be executed.

- In order to assess the quality of our method, we experiment with KaggleDBQA Lee et al. [2021], an established low-resource benchmark, and demonstrate that models trained on SYNQL-KaggleDBQA exceed the performance of those trained on the original data.
- Additionally, to better understand the properties of SYNQL data, we generate a synthetic equivalent of the Spider dataset Yu et al. [2018] and analyze model performance when trained on both original and synthetic data.
- Finally, we open-source both method [1] and synthetic datasets [2][3] for further research.

2 Related work

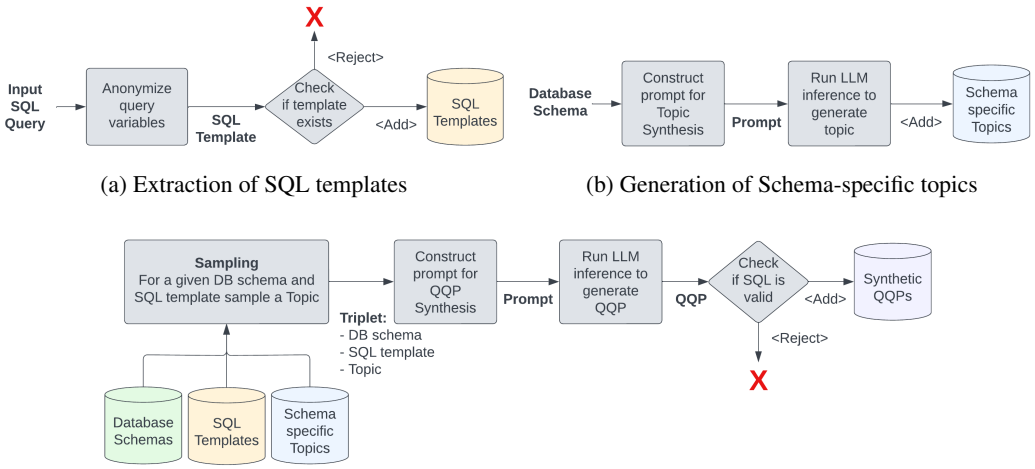
Synthetic data generation for text-to-SQL involves synthesizing two components: a Natural Language Question (NLQ) and a corresponding SQL query. Hence, at a high level, one can distinguish three types of methods: 1. SQL→NLQ: methods that first synthesize query and then question, 2. NLQ→SQL: methods that first synthesize question and then query, 3. SQL+NLQ: methods that synthesize both query and question at the same time.

Tab. 1 utilizes this categorization and presents a selection of recent methods for text-to-SQL synthesis, emphasizing the most critical steps of a method (algorithm) and the contextual data provided as input to the algorithm. Note that a clear trend is visible: deep learning-based methods leveraging LLMs are replacing other synthesis methods. This shift follows a broader trend of utilizing LLMs for SDG in other fields of NLP, with the Phi family of Small Language Models (SLMs) being the most prominent example of this process [Gunasekar et al., 2023, Javaheripi and Bubeck, 2023, Abdin et al., 2024].

[1] `https://github.com/semiotic-ai/SynQL`

[2] `https://huggingface.co/datasets/semiotic/SynQL-Spider-Train`

[3] `https://huggingface.co/datasets/semiotic/SynQL-KaggleDBQA-Train/`



(a) Extraction of SQL templates

(b) Generation of Schema-specific topics

(c) Joint generation of Question-Query Pairs (QQPs)

Figure 1: Steps of the SYNQL data generation method. See the appendices for exemplary data.

For NLQ synthesis, solutions leveraging question templates Yu et al. [2021] were replaced by fine-tuning (BERT-derived) encoder-(domain-specific)decoder models [Wang et al., 2021, Wu et al., 2022], and most recently by generative decoders with In-Context Learning [Ye et al., 2023, Kuznia, 2023].

A similar process occurs with SQL synthesis, progressing from earlier works that build on domain-specific grammars [Wang et al., 2021, Yu et al., 2021, Wu et al., 2022], to methods utilizing template or pattern-based filling and/or sampling [Hu et al., 2023], and finally to generative decoders [Ye et al., 2023, Kuznia, 2023].

Additionally, a few past solutions tried to bridge the SQL-NLQ gap by, e.g., leveraging Entity Sequences [Yang et al., 2021] or Intermediate Representations [Hu et al., 2023]. This problem is naturally addressed in methods leveraging joint SQL+NLQ synthesis. According to our knowledge, Ye et al. [2023] and Kuznia [2023] are the first to utilize a purely prompt-based generative approach for text-to-SQL SDG, relying upon in-context learning to guide QQP generation. Broadly, the primary components of these prompts are: 1. Task definition and rules, 2. Contextual info (database schemas, demonstrations, etc.), 3. Output formatting. Both utilize few-shot demonstrations when synthesizing QQPs and provide database schemas and sampled database content as contextual information. Chang and Fosler-Lussier [2023], Wang et al. [2024], Gao et al. [2023] and others follow a related line of research, investigating the optimal selection of demonstrations for inference.

3 The SYNQL Method

The core motivation that led to the development of SYNQL was to create a simple yet effective way of controlling the diversity of generated questions and their SQL query counterparts across both the Question and Query spaces. For that purpose, as indicated in Tab. 1, we leverage three types of contextual data: 1. SQL Templates 2. Topics 3. Database Schemas.

By changing the number and diversity of the former two, one can control the size and diversity of generated data. SQL Templates serve as anonymized examples to guide SQL generation. Topics' main role is guiding NLQ generation. They help instantiate SQL queries while also bridging the gap between NLQ and SQL for a given pair. Finally, Database Schemas ground the generative process in a given domain and ensure the correct names of tables, columns, and leveraged table relations. Fig. 1 presents the three main steps of SYNQL, two of which are responsible for pre-processing inputs and preparing contextual data, which is then used to generate QQPs.

3.1 Contextual Data

Database Schemas: SYNQL uses a rich schema representation of the underlying database(s), formatted as a series of `CREATE` statements, including column types and key relationships, indicating

Dataset/Split	# Databases	# Tables/DB	# QQPs	# Topics	# SQL Templates
Spider/train	140	5.26	7,000	–	912
Spider/dev	11	4.05	1,034	–	254
SYNQL-Spider/train	140	5.26	114,955	764	15,775
KaggleDBQA/fewshot	8	2.25	87	–	50
KaggleDBQA/test	8	2.25	185	–	84
SYNQL-KaggleDBQA/train	8	2.25	1,638	37	319

Table 2: Statistics of original and corresponding SYNQL- splits for both Spider and KaggleDBQA.

primary and foreign keys. For LLM-based SDG, this representation has yielded the best results [Ye et al., 2023, Hu et al., 2023, Kuznia, 2023]. In particular, providing column types helps to mitigate errors, such as performing arithmetic functions on text-based values (e.g., `SUM(student.name)`), whereas providing key relationships helps to ensure `JOIN(s)` occur on the correct fields. Appendices A.3 and A.4 show an exemplary schema and model response to potential type errors.

SQL Templates: We process input SQL queries to extract a set of unique SQL Templates, as presented in Fig. 1a. Our process follows Zhong et al. [2020b] and anonymizes all variables of a given SQL query, leaving the operations untouched. We perform this for all the collected SQL queries and store all unique, extracted templates.

Schema-specific Topics: A **Topic** is a sentence that describes a subject related to a given database schema. It serves as a contextual hint derived from the schema, helping guide the creation of relevant and diverse questions focused on specific aspects of the schema. The process for generating topics is outlined in Fig. 1b. For a given database schema, we create a prompt instructing the model to generate distinct topics relevant to that schema[4], and repeat this for every database schema.

3.2 Algorithm

Fig. 1c outlines the process of joint generation of SQL and NLP pairs[5]. Given the three contextual data components (Database Schemas, SQL Templates and Topics) we construct prompts utilizing a combination of every Topic and SQL Template. As Topics are inherently tied to Database Schemas, we utilize the Database Schema associated with the selected Topic for any given prompt. After running LLM inference given a prompt, we check the correctness of the generated QQP by ensuring it properly executes against the underlying database, similarly to Ye et al. [2023].

Note that this method can be adjusted to utilize a subset of the prompt components, reducing the total amount of generated QQPs. For example, for the data-rich Spider dataset, we randomly sample one topic for every Database/Query Template pair, whereas for the low-resource KaggleDBQA dataset, we generate QQPs for every possible combination of Topic and SQL Template. A data-efficiency analysis, further exploring the impact of dataset size on the fine-tuned model performance, can be found in Appendix A.6.

Dataset/Split	Easy	Medium	Hard	Extra
Spider/train	8.9%	33.3%	18.3%	39.5%
SYNQL-Spider/train	2.2%	16.6%	16.1%	65.1%
KaggleDBQA /fewshot	28.0%	32.0%	18.0%	22.0%
SYNQL-KaggleDBQA/train	16.6	32.9%	24.2%	26.3%

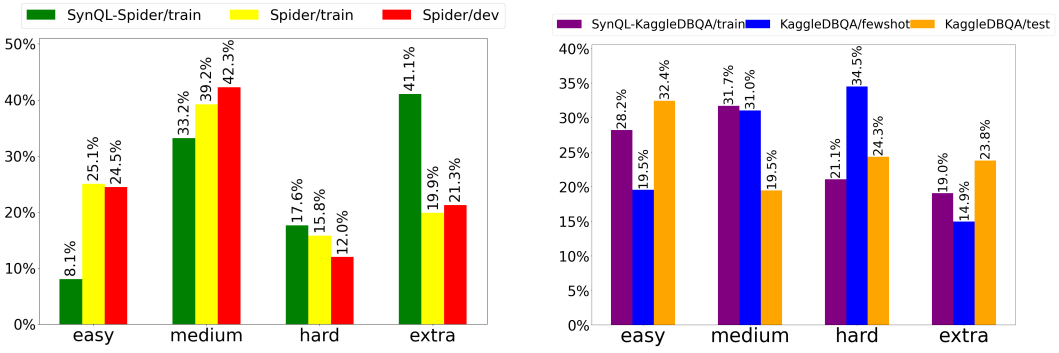
Table 3: Distribution of SQL Template hardness across original and generated datasets and splits.

[4]The prompt for topic generation, along with example topics, can be found in appendix A.5.

[5]The associated prompt can be found in appendix A.7.

4 SYNQL datasets

Using the method described in the previous section, we have generated two synthetic datasets associated with two different benchmarks: Spider and KaggleDBQA. The core difference between those two benchmarks is that Spider aims at "cross-domain generalization" (training and testing on different database schemas), whereas KaggleDBQA tests "in-domain generalization," hence this pair became a *de facto* standard for testing various text-to-SQL parsing methods (e.g. Gao et al. [2023], Chang and Fosler-Lussier [2023], Wang et al. [2024]). In Tab. 2, we present a comparison of original datasets and splits with their synthetic SYNQL- counterparts.



(a) Spider and SYNQL-Spider splits

(b) KaggleDBQA and SYNQL-KaggleDBQA splits

Figure 2: Composition of SQL structure hardness for queries in different dataset splits. (a) Distribution for Spider and SYNQL-Spider. (b) Distribution for KaggleDBQA and SYNQL-KaggleDBQA.

4.1 SYNQL-Spider

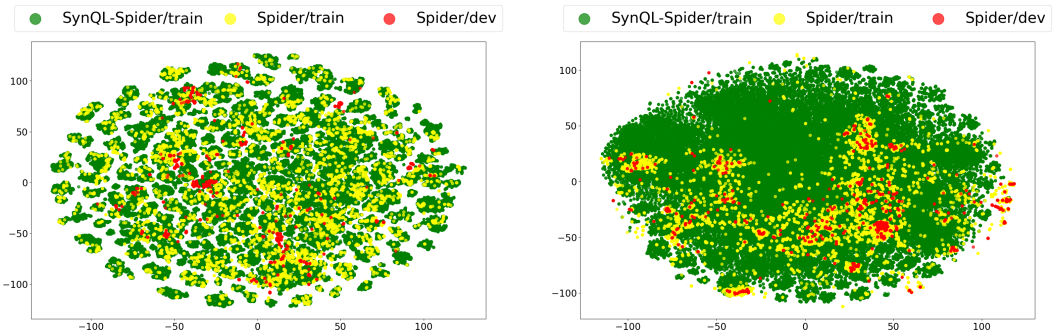
Spider Yu et al. [2018] is one of the most established benchmarks for semantic parsing/text-to-SQL, designed to facilitate the training and evaluation of models that can generalize across different database schemas. It consists of over 10,000 QQPs across 200 databases, making it (still) one of the most comprehensive benchmarks in SQL parsing. We first investigate this dataset because it allows for a deeper analysis of the properties of synthetically generated data compared to the original dataset, which is abundant with data, enabling us to draw more general conclusions.

We derived our synthetic split from **Spider/train**. We started with 140 training database schemas and generated topics for each, resulting in 764 topics. Next, we extracted query templates from all training queries, resulting in 912 templates. Finally, we used database schemas, topics, and query templates to generate question-query pairs, resulting in 127,680 QQPs. After filtering the invalid queries (ensuring they can properly execute against the underlying database), we received 114,955 QQPs, forming the **SYNQL-Spider/train** split.

The first interesting observation goes to the diversity of generated queries, which is reflected by the fact that from those 115k queries, we extracted (following procedure from Fig. 1a) more than 15k unique SQL templates. This is also visible in t-SNE visualization of SQL embeddings space on Fig. 3b, where generated queries (green) cover large fragments of spaces not covered by the **Spider/train** split from which the templates were derived (yellow). We also note that the SYNQL method covered most of them without access to queries from **Spider/dev** split (red).

Regarding the NLQ (question) space presented in Fig. 3a, we can see hundreds of small clusters, in most cases, formed around original questions. This is an intriguing property, especially since no natural questions from the original Spider splits were used to generate SYNQL NLQs. We hypothesize that this is a direct consequence of injecting topics into prompts during generation.

Finally, when comparing the hardness of original and synthetic queries (Fig. 2a), we notice that for the Spider dataset, the SYNQL method has a tendency towards generating more complex queries, with **extra** hard queries constituting almost half of the generated split (41.1% vs 19.9%), and heavily under-representing the queries falling into **easy** category (8.1% vs 25.1%). To understand this better, we compare this distribution with the distribution of SQL templates as presented in Tab. 3, and note



(a) NLQ (question) embedding space

(b) SQL (query) embedding space

Figure 3: t-SNE visualizations of Spider and SYNQL-Spider splits in SQL and NLQ embedding spaces. Details of procedures we used for embedding generation can be found in Appendix A.1.

that, on average, our SynQL method generated a QQP that matched the hardness of the SQL Template used as contextual information 85.1% of the time, indicating an impressive ability to generate data matching the original distribution in terms of hardness.

4.2 SYNQL-KaggleDBQA

In contrast to Spider, KaggleDBQA [Lee et al., 2021] is a smaller dataset (in terms of both number of QQPs and databases), developed to bridge the gap between academic datasets (such as Spider) and industrial databases. The databases used in KaggleDBQA are from Kaggle, with minimalistic preprocessing and data cleansing. It features abbreviated and obscure naming of tables, columns, and domain-specific categorical values, often accrued from legacy development or migrations. Despite simpler database schemas, the dataset offers more complex SQL structures compared to those from Spider. A visualization of this dataset embedding space can be found in Appendix A.2.

We treat KaggleDBQA as our benchmark as it is closer to our considered low-resource scenario, where the organization collects a small number of new QQPs from its production environment (from its users) and wants to quickly fine-tune its model to perform better on those (and similar) queries.

Starting from the **KaggleDBQA/fewshot** split, created by the authors of Lee et al. [2021] and available at the project website; we generated 37 distinctive topics given 8 input database schemas. Next, we took 87 queries and extracted 50 unique templates. Finally, we generated 1,850 QQPs and filtered the invalid ones, resulting in 1,638 QQPS. Those queries formed **SYNQL-KaggleDBQA/train**.

Please note that despite the relatively large number of synthetic SQL templates (319, compared to 50 SQL templates in the original **KaggleDBQA/fewshot**), in this case, the SYNQL method generated a more balanced split from the point of view of query hardness (Fig. 2). As in the case of Spider, we find the resulting distribution of query hardness follows the distribution of SQL templates extracted from **KaggleDBQA/fewshot** split and used as contextual data during generation, with the generated QQP matching in hardness 86.4% of the time.

5 Experimental Results

5.1 Training procedure

In our experiments we utilize the training framework provided by Scholak et al. [2021] to fine-tune T5-Large and T5-3B base checkpoints [Raffel et al., 2020]. We follow the same procedure for each training, reporting the average results of three runs utilizing random seeds. We use Adafactor [Shazeer and Stern, 2018] optimizer with a learning rate of 10-4 and gradient accumulation of batch size of 2048 and 32 for Spider and KaggleDBQA, respectively. We train for 50 epochs with early stopping based on dev/test EX values. For input data formatting, we follow Scholak et al. [2021] and use the schema encoding from Shaw et al. [2020].

Model/Checkpoint	Fine-tuned on Dataset/split	Evaluated on Spider/dev EM [%]	EX [%]	TS-EX [%]
		Previous works with In-context Learning		
LLaMA-13B (0-shot) ♢	–	2.4	–	20.3
LLaMA-13B + 5-shot DAIL$_s$ ♢	–	16.2	–	32.4
GPT-4 (0-shot) ♢	–	22.1	–	72.3
GPT-4 + 5-shot DAIL$_s$ ♢	–	71.9	–	**82.4**
		Previous works with Fine-tuning		
LLaMA-13B (0-shot) ♢	Spider/train (SFT)	62.7	–	67.0
LLaMA-13B + 5-shot DAIL$_s$ ♢	Spider/train (SFT)	61.3	–	66.4
T5-3B/base ♣	Spider/train	71.5	74.4	**68.38**
		Previous works with Fine-tuning on Synthetic Data		
T5-3B/base ♠	Spider/train + Synthetic	74.5	**78.6**	–
T5-3B/base ♡	SYMGEN (140db, 71k)	48.55	–	61.03
T5-3B/base ♡	SYMGEN (160db, 103k)	48.55	–	69.25
		This work		
T5-3B/base	SYNQL-Spider/train	33.88 ± 1.40	69.05 ± 1.12	57.74 ± 2.76
T5-3B/base	SYNQL-Spider/train	–	75.05	62.28

Table 4: Performance of selected solutions on Spider. Reporting Exact Set Match Accuracy (EM), Execution Accuracy (EX) and Test-Suite Execution Accuracy (TS-EX) Zhong et al. [2020a]. ♢ indicates results reported in Gao et al. [2023], results with ♣ come from Scholak et al. [2021], ♠ indicate results from Hu et al. [2023] and ♡ come from Ye et al. [2023].

5.2 Results on SYNQL-Spider

The most recent state-of-the-art results in the Spider leaderboard are obtained by leveraging In-context Learning with LLMs. This is also reflected in Tab. 4, where we present selected results from several papers. In particular, we observed that the best results, in terms of Test-Suite Execution Accuracy (TS-EX) [Zhong et al., 2020a] that we treat as our primary indicator of model quality, are achieved using In-context Learning with GPT-4, utilizing five demonstrations selected with DAIL$_s$ Gao et al. [2023]. Note that GPT-4 is impractical for many scenarios, e.g., applications requiring fast responses, whereas pure In-context Learning with smaller models, such as LLaMA-13B, yields worse results.

When it comes to results obtained with LLaMA, we can clearly see a huge improvement when leveraging Supervised Fine-Tuning (SFT) for both zero-shot and few-shot settings, with DAIL$_s$ used for selecting relevant demonstration examples. Still, the achieved scores are far from the recent state of the art, and even with five demonstrations, LLaMA-13B is still under-performing compared to the scores achieved when fine-tuning much smaller T5-3B/base model (GOOGLE-T5/T5-3B checkpoint from HuggingFace) on the same **Spider/train** split. Hence, we decided to use this architecture (and checkpoint) in our experiments.

Next, comparing the performance of T5-3B models when trained on synthetic data, it is evident that purely synthetic data for Spider resulted in worse performance than the original **Spider/train** data, and the only improvement over the baseline comes from augmenting the original **Spider/train** split with synthetic data Hu et al. [2023].

Finally, an interesting observation is associated with the gap between EM and EX/TS-EX scores achieved by T5-3B/base fine-tuned on our **SYNQL-Spider/train** data. We conclude this to be a property resulting from the data generated by GPT-4 and derived straight from the knowledge embedded in this model. Note that this is consistent with the results seen with zero-shot prompting obtained with GPT-4, suggesting that the model generates SQL queries that, despite being significantly different from the queries in the original split, can still yield correct data from the database.

5.3 Results on SYNQL-KaggleDBQA

Experimental results with KaggleDBQA are presented in Tab. 5. Similarly, as for Spider, the current SOTA performance on the KaggleDBQA dataset is achieved using In-context Learning, this time by the Codex model (14.8 billion parameters) and eight demonstrations selected by the ODIS method Chang and Fosler-Lussier [2023]. Moreover, analogically, smaller models like CodeLlama-7B with In-context Learning achieve worse results than model fine-tuning. In particular, the best

Model/Checkpoint	Fine-tuned on Dataset/Split	Evaluated on KaggleDBQA/test EM[%]	EX[%]
	Previous works with In-context Learning		
CodeLlama-7B (0-shot) ♣	–	–	9.9
CodeLlama-7B + 10-shot FUSED ♣	–	–	22.8
Codex/davinci-002 (0-shot) ♡	–	–	26.8
Codex/davinci-002 + 8-shot ODIS ♡	–	–	**54.8**
	Previous works with Fine-tuning		
SmBoP ♡	KaggleDBQA/fewshot	–	27.2
T5-3B/base + Picard ♡	KaggleDBQA/fewshot	–	29.8
	This work		
T5-Large/base	KaggleDBQA/fewshot	12.79 ± 0.62	19.82 ± 0.82
T5-Large/base	SYNQL-KaggleDBQA/train	12.61 ± 1.74	33.15 ± 1.12
T5-3B/base	KaggleDBQA/fewshot	18.66 ± 1.17	24.50 ± 0.31
T5-3B/base	SYNQL-KaggleDBQA/train	12.43 ± 1.43	$\mathbf{35.14 \pm 0.0}$

Table 5: Performance of selected solutions on KaggleDBQA. ♣ denotes results reported in Wang et al. [2024], whereas ♡ indicates results from Chang and Fosler-Lussier [2023].

score reported before our work was T5-3B/base+PICARD, achieving 29.8% EX when fine-tuned on the original **KaggleDBQA/fewshot**.

Remarkably, our results indicate that we can surpass this score by over 5 points using the T5-3B/base model, even without employing PICARD. When we compare the performance of fine-tuning of T5-3B/base models on **KaggleDBQA/fewshot** and **SYNQL-KaggleDBQA/train** splits, we see an increase of over 10 points (on average!) in relative performance when utilizing the latter. Furthermore, we observe a similar improvement with a smaller model, T5-Large (770M parameters). Finally, we observe lower EM scores for models trained on synthetic and original data and huge gaps between EM and EX scores for models trained on **SYNQL-KaggleDBQA/train**. We find those consistent with results achieved on Spider.

These findings underscore the potential of our SYNQL method to enhance model performance significantly in smaller, resource-constrained scenarios. We think the ability to improve model performance on in-domain problem settings is associated with injecting topics as contextual information and leveraging GPT-4 for generating more diverse, although still domain-specific queries.

6 Summary

In this work, we introduced SYNQL, an LLM-prompting-based SDG approach for joint synthesis of QQP pairs for low-resource scenarios. We reviewed the leading SDG text-to-SQL methods and built upon their work by introducing additional generation steps to expand the available contextual information used during generation with Topics. We demonstrated the ability of this method to improve model performance in low-resource settings, utilizing KaggleDBQA as a benchmark for performance. Additionally, to better understand the properties of the SYNQL method, we applied it to the data-rich Spider benchmark and demonstrated the method's ability to generate data of high diversity, one of the primary objectives of SDG for NLP.

Our future work will be focused on three areas: 1. more granular control of synthetic data distributions, 2. injecting richer contextual information during generation, 3. utilization of contextual information for post-generation filtering. The first direction is motivated by the observation that sometimes our method does not fully cover the query space (e.g., right-hand side of Fig. 3b). This is indicative of the larger challenge of guiding the distribution of QQPs in a granular fashion and warrants further investigation. Second, we see room for the injection of more contextual information, starting from SQL Templates from external sources to using records sampled from the database [Kuznia, 2023], to leveraging the latest demonstration selection algorithms [Chang and Fosler-Lussier, 2023, Wang et al., 2024] for In-context learning. Given the limitless ability to generate valid SQL, the challenge becomes selecting the optimal components given a domain and problem space. Finally, more complex filtering and selection of generated QQPs, e.g., ensuring that SQL does not return NULL upon execution, can help align generated data with external requirements or user preferences.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Shuaichen Chang and Eric Fosler-Lussier. Selective demonstrations for cross-domain text-to-sql. *arXiv preprint arXiv:2310.06302*, 2023.

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

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Yiqun Hu, Yiyun Zhao, Jiarong Jiang, Wuwei Lan, Henghui Zhu, Anuj Chauhan, Alexander Hanbo Li, Lin Pan, Jun Wang, Chung-Wei Hang, Sheng Zhang, Jiang Guo, Mingwen Dong, Joseph Lilien, Patrick Ng, Zhiguo Wang, Vittorio Castelli, and Bing Xiang. Importance of synthesizing high-quality data for text-to-SQL parsing. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1327–1343, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.86. URL https://aclanthology.org/2023.findings-acl.86.

Mojan Javaheripi and Sebastien Bubeck. Phi-2: The surprising power of small language models. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/, 2023.

Aishwarya Kamath and Rajarshi Das. A survey on semantic parsing. In *Automated Knowledge Base Construction (AKBC)*, 2018.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936, 2023.

Kirby Kuznia. Using language models to generate text-to-sql training data an approach to improve performance of a text-to-sql parser, 2023. URL https://hdl.handle.net/2286/R.2.N.187426.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 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, August 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-long.176.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface. *A big bench for large-scale database grounded text-to-sqls. CoRR abs/2305.03111*, 2023.

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

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*, 2022.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*, 2020.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

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

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. Learning to synthesize data for semantic parsing. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.220. URL `https://aclanthology.org/2021.naacl-main.220`.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. Improving demonstration diversity by human-free fusing for text-to-sql. *arXiv preprint arXiv:2402.10663*, 2024.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing, 2022.

Wei Yang, Peng Xu, and Yanshuai Cao. Hierarchical neural data synthesis for semantic parsing. *arXiv preprint arXiv:2112.02212*, 2021.

Jiacheng Ye, Chengzu Li, Lingpeng Kong, and Tao Yu. Generating data for symbolic language with large language models. *arXiv preprint arXiv:2305.13917*, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Department of Computer Science, Yale University*, 2018. URL `https://arxiv.org/abs/1809.08887`.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing, 2021.

John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites. In *EMNLP 2020 Long Paper*, 2020a. URL `https://arxiv.org/abs/2010.02840`.

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

Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. Grounded adaptation for zero-shot executable semantic parsing. *arXiv preprint arXiv:2009.07396*, 2020b.

A Appendix

A.1 Generation of SQL and NLQ embeddings

To generate the SQL embeddings, we utilized the code2seq model [Alon et al., 2018], trained on a combination of Spider [Yu et al., 2018] and WikiSQL data [Zhong et al., 2017]. Our 'code2seq' model takes the abstract syntax tree (AST) representation of SQL and predicts the corresponding NLQ, using a branch based attention mechanism to learn the common logical structures of SQL.

To generate the NLQ embeddings, we utilize the most recent version of 'Sentence-Transformers', all-mpnet-base-v2, from Reimers and Gurevych [2019].

A.2 t-SNE Visualizations

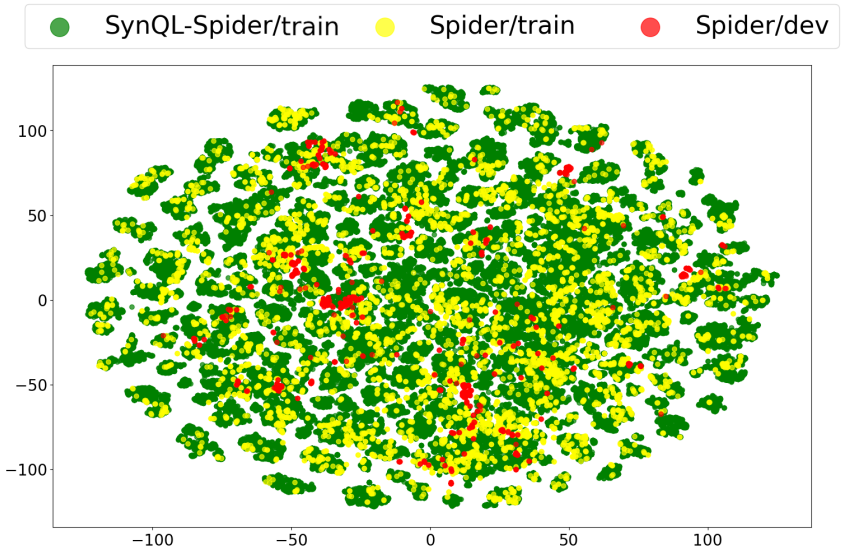


Figure 4: t-SNE visualization of Spider and SYNQL-Spider splits in NLQ (question) embedding space.

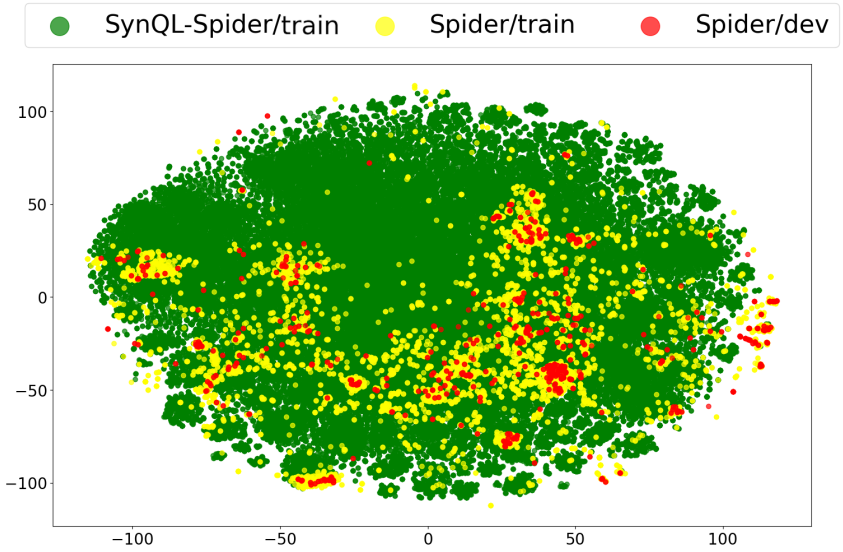


Figure 5: t-SNE visualization of Spider and SYNQL-Spider splits in SQL (query) embedding space.

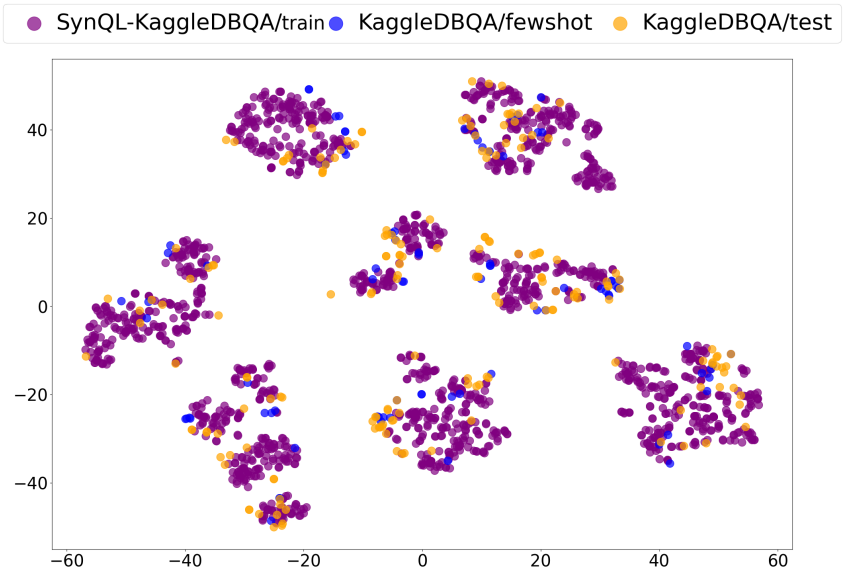


Figure 6: t-SNE visualization of KaggleDBQA and SYNQL-KaggleDBQA splits in NLQ (question) embedding space.

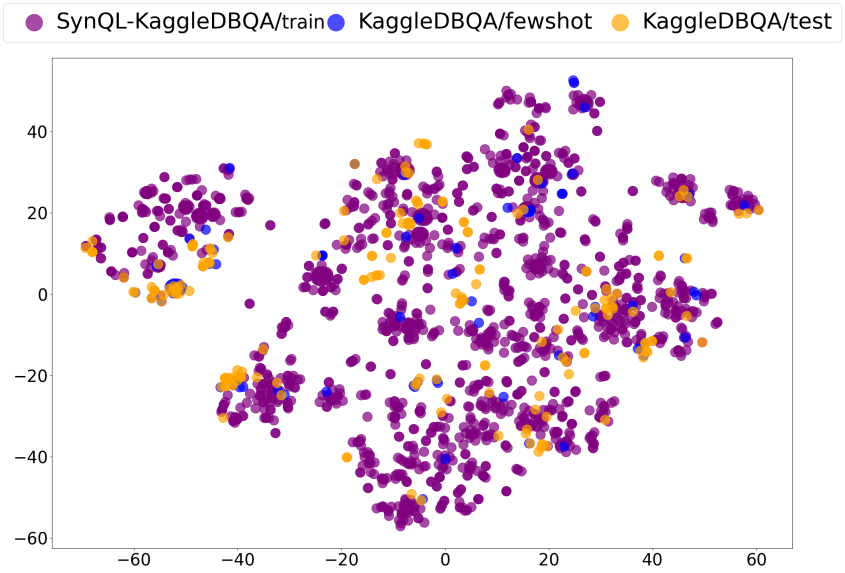


Figure 7: t-SNE visualization of KaggleDBQA and SYNQL-KaggleDBQA splits in SQL (query) embedding space.

A.3 Exemplary Database Schema

Example is shown in listing 1.

Listing 1: "Exemplary database schema"

```
CREATE TABLE "Web_client_accelerator" (
    "id" int,
    "name" text,
    "Operating_system" text,
    "Client" text,
    "Connection" text,
    PRIMARY key("id")
```

```
)
CREATE TABLE "browser" (
    "id" int,
    "name" text,
    "market_share" real,
    PRIMARY key("id")
)
CREATE TABLE "accelerator_compatible_browser" (
    "accelerator_id" int,
    "browser_id" int,
    "compatible_since_year" int,
    PRIMARY key("accelerator_id", "browser_id"),
    FOREIGN KEY ("accelerator_id") REFERENCES `
        Web_client_accelerator`("id"),
    FOREIGN KEY ("browser_id") REFERENCES `browser`("id")
)
```

A.4 Example of Prompt and Response in Case of Invalid Combination

Exemplary prompt for QQP generation presented in Fig A.7. Retrieved response regarding the invalid combination is presented in "Model Response".

Schema:

```
CREATE TABLE "Web_client_accelerator" (
    "id" int,
    "name" text,
    "Operating_system" text,
    "Client" text,
    "Connection" text,
    PRIMARY key("id")
)
CREATE TABLE "browser" (
    "id" int,
    "name" text,
    "market_share" real,
    PRIMARY key("id")
)
CREATE TABLE "accelerator_compatible_browser" (
    "accelerator_id" int,
    "browser_id" int,
    "compatible_since_year" int,
    PRIMARY key("accelerator_id", "browser_id"),
    FOREIGN KEY ("accelerator_id") REFERENCES `
        Web_client_accelerator`("id"),
    FOREIGN KEY ("browser_id") REFERENCES `browser`("id")
)
```

Topic: `Web Client Accelerator Information (Questions specifically related to the web client accelerator. Avoid questions related to browser or compatibility)`

SQL Template: `SELECT SUM(?) FROM ? ORDER BY ? DESC LIMIT ?;`

Model Response: `The provided query structure is not applicable to the given topic and the schema. The structure suggests an aggregate function (SUM) on a table and sorting the result in descending order, which is not applicable to the Web Client Accelerator schema provided. It is recommended to revise the query structure to align with the database schema.`

A.5 Topic Generation Prompt and Examples

Exemplary prompt for topic generation is presented in Fig 8, whereas it's response is presented in Fig 9.

System Prompt:
Your job is to provide distinct topics that encapsulate a specific subset of questions that can be asked about a database.

The goal is to generate distinct topics for the database. The topics should not overlap. Please return the topics as a numbered list within a JSON object. The list should have integer keys for each topic and the value should be the topic description itself. Ensure that the numbering starts at 1 and each entry is unique.
{
"1": "Topic Description for the first topic",
"2": "Topic Description for the second topic",
"3": "Topic Description for the third topic",
}
The topics should be relevant to typical questions that would be asked about the database, so try to avoid details that are too specific to the database (such as column names). Try to make the topics distinct from each other, so that a large area of potential questions is covered.

User Prompt:
Provided here are the create statements for tables in database browser_web. Your job is to return distinct topics of questions that can be asked about the database browser_web.

schema:
```
CREATE TABLE "Web_client_accelerator" (
    "id" int,
    "name" text,
    "Operating_system" text,
    "Client" text,
    "Connection" text,
    PRIMARY key("id")
)
CREATE TABLE "browser" (
    "id" int,
    "name" text,
    "market_share" real,
    PRIMARY key("id")
)
CREATE TABLE "accelerator_compatible_browser" (
    "accelerator_id" int,
    "browser_id" int,
    "compatible_since_year" int,
    PRIMARY key("accelerator_id", "browser_id"),
    FOREIGN KEY ("accelerator_id") REFERENCES `Web_client_accelerator`("id"),
    FOREIGN KEY ("browser_id") REFERENCES `browser`("id")
)
```

Example Topics:
{
"1": "College Information (Questions specifically related to the colleges. Avoid questions related to players or tryouts)",
"2": "Player Information (Questions specifically related to players. Avoid questions related to colleges or tryouts)",
"3": "Tryout Information (Questions specifically related to tryouts. Avoid questions related to colleges or players)",
}
Only respond with the topic of the question, not the question itself, formatted as the Example Topics are. Please return the topics as a numbered dictionary within a JSON object.

Figure 8: Topic Generation Prompt

Example Generated Topics:
"browser_web":
{
"1": "Web Client Accelerator Information (Questions specifically related to the web client accelerator. Avoid questions related to browser or compatibility)",
"2": "Browser Information (Questions specifically related to the browser. Avoid questions related to the web client accelerator or compatibility)",
"3": "Compatibility Information (Questions specifically related to the compatibility of browsers and web client accelerators. Avoid questions related to the browser or the web client accelerator individually)"
}

Figure 9: Topic Generation Prompt Response

A.6 Synthetic Data Efficiency Analysis

Model performance tends to improve in a somewhat predictable fashion as the size of training data increases (a phenomena known as *scaling laws* [Kaplan et al., 2020]). As SynQL enables one to can control the size of the generated split in several ways (the simplest being increasing/decreasing the number of topics that we generate and/or decreasing the number of extracted templates by sub-sampling), we decided to test the impact of size on the model performance.

Model	Fine-tuned On	Size (QQPs)	EX [%]
T5-3B	Spider/train (Baseline)	N=7,000	71.41 ($\pm$ 0.51)
T5-3B	SynQL-Spider/train	114,955	69.05 ($\pm$ 1.12)
T5-3B	SynQL-Spider/train-3N	21,000	66.05 ($\pm$ 0.84)
T5-3B	SynQL-Spider/train-2N	14,000	65.93 ($\pm$ 0.30)
T5-3B	SynQL-Spider/train-N	7,000	61.70 ($\pm$ 0.60)
T5-3B	KaggleDBQA/fewshot (Baseline)	N=87	24.50 ($\pm$ 0.25)
T5-3B	SynQL-KaggleDBQA/train	1,638	35.14 ($\pm$ 0.00)
T5-3B	SynQL-KaggleDBQA/train-3N	261	27.03 ($\pm$ 0.88)
T5-3B	SynQL-KaggleDBQA/train-2N	174	24.86 ($\pm$ 0.44)
T5-3B	SynQL-KaggleDBQA/train-N	87	13.87 ($\pm$ 0.67)

Table 6: Impact of synthetic dataset size on T5-3B model performance. Models trained on varying sizes of SYNQL-Spider and SYNQL-KaggleDBQA datasets are evaluated on their respective test sets (**Spider/dev** and **KaggleDBQA/test**). EX represents Execution Accuracy with standard deviations.

We investigate the impact of dataset size on model performance by fine-tuning our base model on varying sized subsets of our SYNQL-Spider and SYNQL-Kaggle datasets, using their associated manually generated datasets as the determiner of split size. We evaluate across four dataset splits:

1. **All**: The entire SYNQL dataset,
2. **3N**: A random split of SYNQL, 3 times the size of original dataset training/fine-tuning split,
3. **2N**: A random split of SYNQL, 2 times the size,
4. **N**: A random split of SYNQL, same size as original dataset training/fine-tuning split.

Across our runs, presented in Tab. 6, we affirm the assumption that larger splits will result in improved downstream model performance. In terms of data quality, this further validates the ability of our SYNQL method to generate both diverse and high quality data. Given the ability to cheaply and rapidly generate a datasets orders of magnitude greater than that observed in our experiments, we expect that further improvement in downstream model performance will be observed simply by scaling the quantity of synthetically generated data.

A.7 QQP Generation Prompt and Examples

Exemplary prompt for generation of QQPs is presented in Fig 10, with several generated responses presented in Fig 11.

System Prompt:
Your task is to create a SQL query and an associated question based on a given subject, query structure, and schema. ∗∗The query must strictly adhere to the provided query structure and be a valid SQL query. The question should be relevant to the subject and accurately answered by the query∗∗. Follow these guidelines:

1) The query must be valid and logical SQL.
2) The query must match the query structure exactly.
3) The question must match the topic of the subject.
4) The query must answer the question.
5) The query must comply with the given table schema.
6) Do not ask overly vague or specific questions that a user would not typically ask.

Do not modify the query structure. Do not keep any placeholder ('?') values. For example:
Query Structure: SELECT ? FROM ? WHERE ? = ?;
Generated Query: SELECT column_one FROM table_one WHERE column_two = 1

The response must be in the following JSON format:
Response Format: {"question": "<generated question>", "query": "<generated query>"}

User Prompt:
Given the following topic, query structure, and schema, generate a unique question and SQL query. The generated SQL query must strictly adhere to the provided query structure and be valid, logical, SQL. The question should be relevant to the topic, and the query should accurately answer the question using the given schema.
∗∗Do not generate low-quality questions or queries∗∗. These include queries that have irrelevant structure, such as unnecessary joins. ∗∗The SQL query must be valid∗∗, both in its syntax and relation to the database schema.
- Schema:
CREATE TABLE "Web_client_accelerator" (
"id" int,
"name" text,
"Operating_system" text,
"Client" text,
"Connection" text,
PRIMARY key("id")
)
CREATE TABLE "browser" (
"id" int,
"name" text,
"market_share" real,
PRIMARY key("id")
)
CREATE TABLE "accelerator_compatible_browser" (
"accelerator_id" int,
"browser_id" int,
"compatible_since_year" int,
PRIMARY key("accelerator_id", "browser_id"),
FOREIGN KEY ("accelerator_id") REFERENCES ‘Web_client_accelerator‘("id"),
FOREIGN KEY ("browser_id") REFERENCES ‘browser‘("id")
)
- Question Topic: Web Client Accelerator Information (Questions specifically related to the web client accelerator. Avoid questions related to browser or compatibility)
- Query Structure: SELECT COUNT(DISTINCT columnOne) FROM tableOne WHERE columnTwo = 1

Response Format: {question: <generated question>, query: <generated query>}

Figure 10: QQP Generation Prompt

Example Generated QQPs:


```
{
    "question": "What are the names of the web client accelerators that run on the Windows operating system,
sorted by their      names?",
    "query": "SELECT name FROM Web_client_accelerator WHERE Operating_system LIKE '%Windows%'
ORDER BY      name",
}
```


Figure 11: QQP Generation Prompt Response