

Searching for Better Database Queries in the Outputs of Semantic Parsers

Anton Osokin*
HSE University
Yandex

Irina Saparina*
HSE University
Yandex

Ramil Yarullin*
HSE University
Yandex

Abstract

The task of generating a database query from a question in natural language suffers from ambiguity and insufficiently precise description of the goal. The problem is amplified when the system needs to generalize to databases unseen at training. In this paper, we consider the case when, at the test time, the system has access to an external criterion that evaluates the generated queries. The criterion can vary from checking that a query executes without errors to verifying the query on a set of tests. In this setting, we augment neural autoregressive models with a search algorithm that looks for a query satisfying the criterion. We apply our approach to the state-of-the-art semantic parsers and report that it allows us to find many queries passing all the tests on different datasets.

1 Introduction

Generating a database query from a natural-language description of the user’s intent is a long-standing and important task. In the recent years, most of the community focus was on the Spider dataset (Yu et al., 2018), which poses the task in the zero-shot regime, meaning that a method has to generalize to databases unseen at training. The Spider dataset contains English questions and SQL queries. The progress has been remarkable, and the accuracy has moved from below 30% to above 70%. A part of this success can be attributed to the adoption of pre-trained transformer models like BERT (Devlin et al., 2019) into most of the pipelines.

Given such progress, it is natural to ask whether we are getting closer to solving the problem. Several recent studies have noted that the task might be harder than it looks. Finegan-Dollak et al. (2018) found that many single-database datasets had identical queries in both train and test sets and showed that using such splits effectively reduced the problem to classifying the queries from the train set.

Shaw et al. (2021) continued the study in the multi-database setting and showed that the compositional generalization was hard to achieve, and even to measure it, one should be very careful with splits. In a different line of thought, Suhr et al. (2020) examined how the models trained on Spider generalize to other datasets and reported that generalization was challenging. Even within one dataset, many questions have several interpretations leading to different queries, and annotation policies do not cover these ambiguities or cover them differently.

Acknowledging that the zero-shot setting might be too difficult to tackle as is, we aim to better define and simplify the problem to achieve better results in terms of the number of correctly generated queries. In this task, most modern models produce a distribution over all possible outputs, which can guide the search at the test time.

We observe that if the search algorithm has access to a criterion that can evaluate the output by treating it as a database query, the overall method can produce much better results. We consider the following criteria ordered by their “strength”: a query is executed without errors, a query produces output from correct columns, a query produces a correct result on one test database, and a query produces correct results on a set of test databases. We experiment with different search methods and report that the complete anytime beam search (Zhang, 1998) outperforms sampling-based alternatives.

Many practical cases arise when the user is willing to trade off some of their time to improve the output query. Our approach allows the user to obtain a better query by interactively guiding the search via providing the target output columns or the answer on one or more test databases. The user is expected to supply this information without the gold query. Notably, the execution criterion does not require extra user input but relies on executing generated queries.

In addition, the test time database can be out of

* Equal contribution.

domain w.r.t. the training set. One can annotate gold queries for fine-tuning on new databases to improve out-of-domain performance, but this can be prohibitively expensive. Our approach gives a way to improve out-of-domain results without extra annotated training data. We view our approach as a way to control the trained model leading to a more reliable and responsible query synthesis.

For our studies, we used three state-of-the-art models in terms of execution accuracy (with publicly available implementations): T5-3B (Raffel et al., 2020) fine-tuned to Spider by Scholak et al. (2021), BRIDGE (Lin et al., 2020) and SQ-QDMR (Saparina and Osokin, 2021).

In this paper, we make several observations. The complete anytime beam search works with different search criteria and with all the three models. Reasonable results can be obtained with the maximum beam size of 100, which fits on a single modern GPU. Searching with the execution criterion can significantly improve the quality of the decoders that generate output as an unconstrained token sequence, e.g., T5, and using such criterion does not require extra user input. Searching for the queries that return a correct output on one database allows finding many queries that provide a correct output on that database. Therefore, such searching w.r.t. one database can result in false positives, and it is important to evaluate the queries on a set of databases. Based on the method of Zhong et al. (2020), we built the test suite of databases for the evaluation. With these test suites, we show that there are multiple false positives among the queries that pass one test, while searching for the queries that pass the test suite produces the outputs of higher quality.

Finally, we experiment with the generalization of the models trained on Spider to the Geo-Query (Zelle and Mooney, 1996), IMDB, Yelp (Yahmazadeh et al., 2017) and Academic (Li and Jagadish, 2014) datasets. We show that searching w.r.t. different criteria still works under this distribution shift, and searching w.r.t. the criteria with tests is often comparable to fine-tuning the network to a test dataset directly.

This paper is organized as follows. In Section 2, we review our setting. In Section 3, we provide the details of our method. Section 4 provides the details of the test-suite construction procedure, Section 5 describes the experimental setup. In Section 6, we provide the experimental results and

discussion. We review some related works in Section 7 and conclude in Section 8.

2 Preliminaries

We consider the problem of generating queries to databases given the description of the user’s intent in natural language in the cross-database setting where the train, validation and test splits contain different databases. Models trained in this setting, in theory, can be evaluated on any database.

A typical model for the cross-database setting is an encoder-decoder neural network. Encoders typically consist of a pre-trained BERT-like transformer followed by a specialized encoder that can incorporate the database structure in some form (Guo et al., 2019; Wang et al., 2020; Cao et al., 2021; Cai et al., 2021). Sometimes the BERT part is further fine-tuned on database-related objectives (Yu et al., 2021; Deng et al., 2021). The encoder input is a concatenation of the tokenized question and a sentence representation of the database schema separated by the special delimiter token. The representation of the database schema consists of the tokenized table and column names and values related to the question. These values are commonly extracted by string matching with question tokens (Lin et al., 2020). Decoders are typically autoregressive based on LSTM or transformers. Some decoders do not check the syntactic correctness of the output and its consistency with the database. Some provide output w.r.t. a grammar (Yin and Neubig, 2017); some use post-hoc checks with parsers.

In this paper, we experiment with three models: T5-3B fine-tuned on the Spider dataset by Scholak et al. (2021), BRIDGE (Lin et al., 2020) and SQ-QDMR (Saparina and Osokin, 2021). We provide a detailed description of these models in Section 5.3.

3 Search with Models

We now describe our approach to searching for queries on top of a learned model. We first generate full query candidates using a search method and then select the first one that passes the selected search criterion. We show possible search criteria in Section 3.1 and search methods in Section 3.2.

3.1 Search Criteria

Execution criterion. To avoid syntactically incorrect queries, we can prune the search with the execution criterion. The query passes this criterion

if it can be executed on the input database without errors. In particular, the query has to contain valid table and column names. These properties are not guaranteed for the unconstrained decoders as T5. Thus the execution criterion can be extremely useful for such models.

Output column match. With this criterion, we compare the output columns (that the query will select) with the correct ones. Firstly, wrong output columns is a common mistake in the text-to-SQL parsers (Guo et al., 2019; Lin et al., 2020; Suhr et al., 2020). Secondly, the output columns can provide domain knowledge and shed some light on the user intent in a realistic scenario when the input question is ambiguous (Suhr et al., 2020; Lee et al., 2021).

One test. This criterion compares the result of query execution on a given database (one test case) with the correct one. With this criterion, we search for a correct query in terms of execution accuracy on the input database (Section 5.2).

Test suite. This criterion checks if a query passes a set of tests. Each test case corresponds to a particular database, and all test databases share the same schema. This criterion is inspired by a test suite of databases with high code coverage proposed by Zhong et al. (2020). The set of databases is designed to distinguish the correct queries from potential false positives. Searching with this criterion is equivalent to the search for a correct query in terms of test-suite accuracy (Section 5.2).

3.2 Search Methods

Top- k and Top- p (Nucleus) Sampling (Fan et al., 2018; Holtzman et al., 2020) draw samples from the truncated distribution: the probability mass is re-weighted between the k most probable elements in top- k sampling and between the elements with cumulative probability mass exceeding p in top- p sampling (k, p are hyperparameters).

UniqueRandomizer (Shi et al., 2020) is a method to incrementally sample sequences without replacement. The samples are drawn until the stopping condition is reached (one of the search criteria in our case). The probabilities of selected elements are reduced after each iteration of sampling to improve diversity in samples.

Complete Anytime Beam (CAB) Search of Zhang (1998) extends the regular beam search by running it several times with increasing beam sizes. Importantly, the beams produced by beam search

Table 1: Comparison of the test suites’ statistics. **NoEmpty** is the percentage of SQL queries for which at least one test database with non-empty execution result is found; **Cover** is the percentage of neighbor queries distinguished by the test databases; **Tests** is the average number of test databases per query; **Time** is the average wall-clock execution time per query; **Size** is the total size of all test databases.

Dataset		NoEmpty (%) ↑	Cover (%) ↑	Tests (Num)	Time (Sec) ↓	Size (GB) ↓
Spider	1 test	96.7	89.1	1	0.1	3.25
	Orig.	98.2	96.1	675	3.6	0.16
	Our	99.3	98.8	1.8	0.2	0.04
GeoQuery	1 test	94.7	93.2	1	0.1	10⁻⁴
	Orig.	66.7	52.6	108	12.6	0.01
	Our	100	98.9	1.6	0.2	0.05
IMDB	1 test	75.2	14.9	1	8.7	1.49
	Orig.	79.2	79.4	200	17.7	0.04
	Our	100	99.6	2	0.3	0.03
Yelp	1 test	36.5	15.8	1	6.4	2.21
	Orig.	84.1	80.6	282	30.9	0.03
	Our	99.2	97.0	3.1	0.4	0.03
Academic	1 test	51.1	5.50	1	44.2	4.33
	Orig.	97.9	92.1	411	44.3	0.07
	Our	96.8	95.2	2.3	0.4	0.07

are known to have little diversity because of the peaks in the softmax scores. We follow the approach of Zohar and Wolf (2018); Shrivastava et al. (2021), who recently have used CAB to search for programs on top of neural autoregressive models. In these works, the authors limit the number of hypotheses coming from each element of the previous beam (we will refer to this upper bound as the width of the beam search). Between outer CAB iterations, we also increase the width by a constant value and multiply the beam size by a constant factor. The schedules of the beam size and beam-search width are important hyperparameters.

4 Test Suite Construction

Testing on one database is generally not enough to ensure the semantic correctness of the generated query, but running the query on too many databases can be computationally inefficient. The inefficiency problem is especially acute in our task due to the large number of query candidates that should be tested and several rounds of the searching process.

We build our test suites by modifying the method of Zhong et al. (2020), which relies on generating the so-called neighbor queries from a given set of gold queries and randomly sampling databases to distinguish gold queries from as many neighbors

as possible. We observed two key drawbacks of the test databases generated by [Zhong et al. \(2020\)](#). First, the test suites contained many databases, and some were unnecessarily large, which resulted in very long testing on them. Second, outputs of many queries were often empty (or zero for queries with aggregators) on these test databases. If the output of the gold query is empty on all the elements of the test suite, it cannot be distinguished from trivial dummy queries. This effect is more salient if the gold query returns empty output on the original database. We alleviate these issues by independently generating test databases for each gold query, explicitly limiting the number of rows in each table and putting extra effort into generating at least one database where the gold query returns a non-empty output. The details of our procedure are provided in [Appendix A](#).

We compare our test suites with the original ones of [Zhong et al. \(2020\)](#) on the five considered datasets described in [Section 5.1](#). For a fair comparison, we generate the independent sets of neighbor queries for each gold query. These neighbor queries are different from the neighbors generated in the process of creating the test suites. In [Table 1](#), we compare the initial databases (1 test), original test suites and our test suites. It can be seen that, in the space of neighbor queries, our test suites have higher code coverage (Cover) than the original databases and the original test suites. With new test suites, the average query execution time (on all the corresponding tests) is reduced 50x across the datasets. We release the test suites we created.¹

5 Experiment Setup

5.1 Data

We conduct our experiments on five text-to-SQL datasets: multi-database Spider ([Yu et al., 2018](#)) and single-database GeoQuery ([Zelle and Mooney, 1996](#); [Iyer et al., 2017](#); [Finegan-Dollak et al., 2018](#)), IMDB, Yelp ([Yaghmazadeh et al., 2017](#)) and Academic ([Li and Jagadish, 2014](#)).

Spider. We use dev and test sets (451 and 521 examples) from the work of [Saparina and Osokin \(2021\)](#): they are parts of the original Spider dev, but some examples (from dev) were repaired.

GeoQuery, IMDB, Yelp and Academic. We use query splits created by [Finegan-Dollak et al. \(2018\)](#), additionally filtered from duplicates and

examples with gold SQL queries that crash or execute longer than 5 minutes with the Python package `sqlite3`. The dataset statistics are provided in [Table 7](#) of [Appendix D](#).

We do not consider more single-database datasets because ATIS ([Price, 1990](#); [Dahl et al., 1994](#)), Scholar ([Iyer et al., 2017](#)) and Advising ([Finegan-Dollak et al., 2018](#)) database schemes exceed 512 token limits of pre-trained encoders, Restaurants ([Popescua et al., 2003](#); [Tang and Mooney, 2001](#)) contains too many duplicates.

5.2 Evaluation Metrics

Exact-set Match ([Yu et al., 2018](#)) is an SQL-to-SQL comparison metric that reflects the fraction of the predicted queries matching the ground-truth queries. In the matching process, each query is decomposed into fragments that are compared individually so that the metric is not too sensitive to the ordering of independent clauses. This metric does not take into account predicted values and can give a high score to incomplete queries. As SQ-QDMR model produces queries in SPARQL, we cannot use the exact-set match as a primary evaluation metric.

Execution accuracy is designed to compare the queries by their execution output on an original database. In contrast to the exact-set match, this prevents false-negative queries but leaves space for potential false positives. The version provided by [Yu et al. \(2018\)](#) for Spider evaluation has issues in SPARQL-SQL comparison, so we use the version provided by [Saparina and Osokin \(2021\)](#) unless explicitly mentioned otherwise.

Test-suite accuracy ([Zhong et al., 2020](#)) approximates the semantic accuracy of the query synthesis models. This metric refers to the share of predicted queries producing the correct answers on all databases from the test suite. We build the test suites for Spider dev and test sets and for all the queries in the other four considered datasets.

5.3 Models

We consider three models: T5-3B fine-tuned on Spider ([Scholak et al., 2021](#)), BRIDGE ([Lin et al., 2020](#)) and SQ-QDMR ([Saparina and Osokin, 2021](#)). These models have top execution accuracy among publicly available models on Spider. We also tried to search under our search criteria on top of the bottom-up semi-autoregressive model of [Rubin and Berant \(2021\)](#), but we could not make the search increase the number of correct queries.

¹github.com/ramild/TestSuite

For evaluation on Spider, we use the released checkpoints of the best models. BRIDGE training data included question splits of single-database datasets, so we re-train it on Spider-only data to evaluate on query splits of these datasets. For re-training BRIDGE and fine-tuning all models, we use official implementations (see Appendix E).

T5 (Raffel et al., 2020) is a pre-trained seq2seq model based on Transformer. Recently, Shaw et al. (2021); Scholak et al. (2021) successfully applied T5 for the text-to-SQL task. The input sequence contains question tokens and tokens of column and table names. The database values matched with the question tokens are appended to the corresponding column names (Lin et al., 2020). The output of the T5 model is the sequence of tokens representing the SQL query. Note that this model generates output sequence without explicitly considering the SQL grammar and schema consistency.

BRIDGE (Lin et al., 2020) consists of the BERT-based encoder and pointer-generator decoder. The input sequence is formed from the concatenation of question, table and column names, and relevant database values separated by special token and encoded with BERT. The relevant values are selected with fuzzy string matching between question and database values. Column encodings are further enriched with meta-data features such as primary or foreign keys and data types obtained from the feed-forward layer. The LSTM-based decoder with multi-head attention at each step copies question or schema tokens or generates the SQL keywords. During decoding, model chooses columns only from the predicted table to provide schema consistency. An additional static SQL analyzer filters incorrect output queries.

SQ-QDMR (as we refer to the model of Saporina and Osokin (2021)) contains RAT-transformer, GraPPa encoders (Wang et al., 2020; Yu et al., 2021) and a grammar-guided LSTM-based decoder (Yin and Neubig, 2017). The SQ-QDMR decoder produces output in the form of grounded intermediate representations derived from QDMRs of the Break dataset (Wolfson et al., 2020). The grounded QDMRs are not directly related to any execution engine and cannot be executed as is, but Saporina and Osokin (2021) implemented a non-trainable translator from grounded QDMR to the SPARQL query language, in which queries can be executed. We can think of grounded QDMRs augmented with this translator as executable database queries.

Table 2: Comparing the execution accuracy of different search approaches under the 1-test criterion on the Spider dev split.

Model	Greedy	CAB	Sample	Top-p	UniRand
T5-3B	77.0	94.4	93.0	93.0	94.1
BRIDGE	66.8	91.0	87.1	84.9	86.2
SQ-QDMR	80.4	98.0	98.0	98.0	98.2

6 Results & Analysis

6.1 Impact of the Search Methods

We compare different decoding strategies in our setting (Table 2): top- k and top- p (nucleus) sampling (Fan et al., 2018; Holtzman et al., 2020), UniqueRandomizer (Shi et al., 2020) and CAB search (Zhang, 1998). We measure the execution accuracy of these search methods under the 1-test criterion on Spider dev. We use the same sampling budgets (1000 for BRIDGE and SQ-QDMR and 800 for T5-3B due to the memory limits), tune p in top- p sampling and the temperature for all methods, more implementation details in Appendix B).

The results demonstrate that a significant number of output queries pass one test after searching with any of these methods, so different decoding strategies can be compatible with our approach. UniqueRandomizer is very time-consuming since it generates samples sequentially in contrast to other methods that generate beams of samples in parallel. CAB search is demanding in terms of the device memory as it has to process the whole beam jointly. For further experiments, we choose CAB search because it works best for two models.

6.2 Impact of the Search Criteria

We apply search under different selection criteria (execution, output column match, test on one database) to T5-3B, BRIDGE and SQ-QDMR on Spider dataset and compare with the greedy and beam search baselines. Table 3 shows the results measured with execution accuracy. **Searching on top of all models with different selection criteria increases execution accuracy in almost all cases.**

One exception is the search with the execution criterion on top of BRIDGE and SQ-QDMR, the results of which are close to the greedy decoding. The outputs of these systems are almost always executable because BRIDGE runs a static SQL analyzer for filtering, SQ-QDMR decodes according to the QDMR grammar and both models have schema-consistent decoding. The T5 model, in contrast, does not have any grammar or schema

Table 3: Execution accuracy of search under different selection criteria (execution, output column match and 1 test) on Spider; beam s. refers to beam search.

Model	Split	Greedy	Beam S.	Exec	Cols	1 Test
T5-3B		77.0	77.4	83.1	84.2	94.4
BRIDGE	dev	66.8	68.8	68.4	72.0	91.0
SQ-QDMR		80.4	80.6	80.4	83.6	98.0
T5-3B		70.8	71.0	73.7	77.4	90.7
BRIDGE	test	64.0	66.2	64.6	67.9	83.4
SQ-QDMR		65.6	65.8	65.6	68.9	86.7

constraints in decoding. **For language model decoders that do not explicitly check grammar and schema consistency, the execution criterion can significantly improve the quality.**

For T5-3B, we also compare the results of the search with PICARD (Scholak et al., 2021), our search with the execution criterion, and greedy decoding. For a fair comparison with Scholak et al. (2021), we use the same data, the official Spider dev set, and the same metrics: exact-set matching accuracy and execution accuracy provided by Yu et al. (2018):

Method	EM	Exec
Greedy	71.5	74.4
PICARD	75.5	79.3
CAB+execution	74.5	78.7

The results obtained with PICARD and with searching under the execution criterion are comparable – our search gets most of the benefit over the baseline. PICARD provides slightly better quality and works with smaller beams but requires more effort to incorporate because it is tightly connected with the decoder output vocabulary and grammar. For both approaches, the percent of output queries with execution errors is around 2% in contrast to the baseline T5-3B decoding with 12%.

The search criterion based on the matching of output columns provides even better results. As Table 3 shows, all models benefit from this criterion: execution accuracy increases by 3-4% nearly everywhere. This criterion largely simplifies the task with extra information at the test time.

Search for the queries that pass one test allows finding a significant number of such queries. Passing one test means correct execution result on the input database, so all the queries found with this criterion are correct in terms of the execution accuracy. Thus, these results indicate that our searching approach works, and we can find the correct queries with the corresponding criterion. However, we ob-

Table 4: Test-suite accuracy of search under different selection criteria (execution, output column match, 1 test and test suite) on Spider.

Model	Split	Greedy	Exec	Cols	1 Test	Test Suite
T5-3B		72.2	77.7	78.3	86.9	90.1
BRIDGE	dev	63.2	64.8	68.2	81.9	84.0
SQ-QDMR		72.5	72.2	75.4	84.0	94.4
T5-3B		68.9	73.7	75.4	84.7	86.8
BRIDGE	test	61.9	62.5	65.4	76.6	77.8
SQ-QDMR		62.3	62.1	65.4	75.2	84.1

serve that more than 10% of queries found with one-test criterion are false-positive according to the test-suite accuracy (Table 6 in Appendix C).

These results motivate us to evaluate the test-suite accuracy of criterion-guided search. Table 4 confirms our findings: searching with the execution criterion helps T5-3B, and searching for correct output columns improves the results of all models. **Search for the queries that pass one test results in a significant number of false-positive queries. The correct queries can be found by searching with the test-suite criterion directly.**

6.3 Efficiency

Time Measurements. The running time during the search is dominated by the time of the decoder for all three models: executing each considered query takes 3% of the decoder time for T5-3B, which is 0.01 sec per run; 53%, 0.02 sec – for BRIDGE; 72%, 0.03 sec – for SQ-QDMR. The total running time depends on the effective beam size used during the search.

The T5-3B model with the execution criterion on top runs in 1.7 sec compared to 3.1 sec reported by the PICARD paper, where both systems were run on 1 NVIDIA A100 GPU. The main reason is that due to CAB, we do not set one beam size in advance and thus, process at least 70% of examples with an effective beam size of 1.

Impact of the Maximum Size of Beam. The maximum size of the beam is an important parameter. Figure 1 shows the dependence between the obtained test-suite accuracy on the Spider dev set and the maximum beam size in the search under the test suite criterion. For all models, we start with the maximum beam size equal to 1, which is equivalent to the greedy decoding and finish with the maximum beam size allowed by our implementation and hardware: 10k for BRIDGE and SQ-QDMR on 1 NVIDIA V100 GPU and 800 for T5-3B on 8 NVIDIA A100 GPUs.

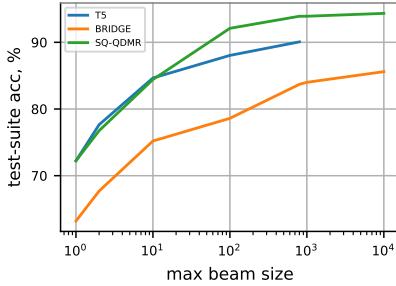


Figure 1: Test-suite accuracy on the dev set for the T5-3B, BRIDGE and SQ-QDMR models tested with CAB for the test-suite criterion.

Test-suite accuracy improves as the maximum beam size increases. BRIDGE with the maximum beam size equal to 10k achieves 86% of test-suite accuracy and SQ-QDMR — 94%. However, the search works well enough even with smaller beams: with the maximum beam size of 100, BRIDGE achieves almost 80%, SQ-QDMR — 92%, and T5-3B achieves 88% (with 800, T5-3B achieves 90%). Importantly, the search with the beam size of 100 does not require multiple GPUs for T5-3B.

6.4 Experiments on Single-Database Data

To show more benefits of searching under selection criteria, we evaluate it on single-database datasets, GeoQuery, IMDB, Yelp and Academic, with test-suite accuracy (Table 5; see Appendix F for execution accuracy). We use query splits of Finegan-Dollak et al. (2018) and two model types: trained on Spider only and fine-tuned on a particular dataset. The Academic database is very large, so we cannot evaluate one-test criterion on this dataset and fine-tune SQ-QDMR (other datasets do not have QDMR annotation required for fine-tuning). More fine-tuning details are in Appendix E.

The results show that models trained on Spider struggle to generalize to other datasets, which is consistent with the findings of Suhr et al. (2020). **More information about the problem (in the form of additional train data or selection criteria) helps improve the quality.**

IMDB, Yelp and Academic are more challenging datasets for cross-database semantic parsers than GeoQuery, but they are significantly smaller (Table 7), and the models are less stable while testing on them (with all random seeds fixed). Stronger criteria, such as passing one test or test suite, do work even with datasets of such difficulty, when weaker criteria fail. On GeoQuery, the search under the one-test and test-suite criteria leads to even better

Table 5: Different search criteria (execution, output column match, 1 test and test suite) on top of pre-trained models on the query test splits of different datasets with the **test-suite accuracy**.

Dataset (test size)	Model	Greedy	Exec	Cols	1 test	Test Suite
GeoQuery (182)	T5-3B	55.5	56.6	63.7	67.6	72.5
	+ fine-tune	64.3	70.9	85.2	88.5	94.5
	BRIDGE	55.9	50	62.6	74.7	74.7
	+ fine-tune	65.4	66.5	81.9	86.8	91.8
	SQ-QDMR	37.4	37.4	41.8	65.9	81.9
	+ fine-tune	56.0	56.0	59.9	76.9	83.0
IMDB (17)	T5-3B	5.9	11.8	17.6	29.4	41.2
	+ fine-tune	52.9	52.9	52.9	52.9	58.8
	BRIDGE	11.8	11.8	17.6	17.6	17.6
	+ fine-tune	52.9	52.9	52.9	52.9	52.9
Yelp (10)	T5-3B	20	20	10	30	10
	+ fine-tune	20	30	20	50	40
	BRIDGE	0	0	0	20	10
	+ fine-tune	30	40	50	60	70
Academic (15)	T5-3B	6.7	13.3	26.7	-	33.3
	+ fine-tune	60	53.3	53.3	-	80
	BRIDGE	0	0	6.7	-	20
	+ fine-tune	33	40	40	-	80
	SQ-QDMR	13.3	13.3	6.7	-	66.7

quality than fine-tuning. Our test-suite criterion is especially useful when one test is difficult to run on the large original database, e.g., on Academic.

As a result, we conclude that criterion-guided search on top of a pre-trained model is a good alternative to fine-tuning in cases when training data is not available, but the user is ready to provide more information on each test question.

7 Related Works

Search for Database queries. The task of translating NL questions into database queries implies the ability to query databases with natural language. To ensure this, it is essential to generate syntactically correct queries that refer to valid table and column names for the given database schema. Wang et al. (2019) noticed that a partially decoded SQL query can be executed, and thus, the result of this execution can guide the decoding process. At each decoding step, partial queries that crush or give an empty result during the execution are removed from beams. In this work, we also consider the execution criterion of search but apply it to the finished hypotheses, which allows us to search on top of the models with different output formats, including

intermediate representations.

Lin et al. (2020) generated SQL queries in the execution order to keep the consistency between the predicted database entities and checked output correctness with the static SQL parser. Suhr et al. (2020) executed the top-10 generated queries in beam search to filter the inexecutable ones, which is close to checking the execution criterion in our work but differs by the search method.

Task-specific decoders such as autoregressive grammar-based (Yin and Neubig, 2017; Lin et al., 2018) and tree decoders (Dong and Lapata, 2016), semi-autoregressive decoder (Rubin and Berant, 2021) provide some guarantees as they control the output structure. However, as noticed by Scholak et al. (2021), these decoding methods are incompatible with pre-trained decoders of language models. These pre-trained decoders, like the one of T5, can also be successfully applied to the text-to-SQL task (Shaw et al., 2021). Scholak et al. (2021) proposed to check hypotheses in beam search on the lexical and grammatical levels at each step of the beam search. However, compared to us, their approach required heuristics to prune incomplete queries.

The concurrent work by Wolfson et al. (2022) uses several components similar to ours but in a very different way. They use the QDMRs of Wolfson et al. (2020) with textual arguments as a form of weak supervision to generate SQL queries for the training set. Their synthesis process results in many candidate SQL queries and relies on tests to select the one as an annotation. Such a process is similar to the method of Saparina and Osokin (2021) for constructing groundings of QDMR arguments. However, the search process of Wolfson et al. (2022) is not connected to any neural model and is not used at the test time.

Search for Programs with Neural Networks.

Our approach to searching for queries is closely related to the field of program synthesis if we interpret queries as programs. Recently, neural networks have been applied in a wide range of program synthesis tasks, see the excellent work of Chaudhuri et al. (2021) for a recent review.

When programs are synthesized from large language models generating multiple outputs, selecting the one that, e.g., passes some or all tests is a common practice. For example, the Codex model (Chen et al., 2021) for synthesizing Python code includes some sample tests into the input prompt to give the model more information to de-

fine the user intent. Chen et al. (2021) following Kulal et al. (2019), among others, also uses the pass@k metric, which effectively means that the model generates k outputs, and the best ones are selected based on tests. The pass@k metric can be interpreted as test-suite accuracy after search w.r.t. the tests with the beam of size k .

Overall, it is widely accepted that tests are useful to precisely define the user intent. However, they are hard to collect at a large scale, especially when coupled with a description in natural language. Because of this, large-scale benchmarks related to code, e.g., CodeXGLUE (Lu et al., 2021), primarily used text-based metrics like BLEU. The attempts to specialize BLEU to code by combining it with abstract syntax trees extracted from code, like CodeBLEU (Ren et al., 2020), are in some sense similar to the SQL-based exact match metric of the Spider dataset (Yu et al., 2018).

Approaches to Simplify the Cross-database Setting. The community has made multiple attempts to modify the cross-domain setting to make the problem easier to solve. Yu et al. (2019b) collected the SParC dataset with coherent question sequences, which can allow sharing of information between the sequences. Yu et al. (2019a) collected the CoSQL dataset with an interactive conversational setting with SQL queries, making it possible to explore user interactions with the system. Lee et al. (2021) collected KaggleDBQA with database documentation in the form of textual description for database columns that can potentially allow language models to provide outputs better corresponding to the user’s intent. Our approach provides users with a way to interact with the model by supplying tests. Given the initial question in natural language, the users can provide extra tests until they are satisfied with the generated query.

8 Conclusion

We studied the search over the outputs of the neural autoregressive models for better database query generation. We considered three state-of-the-art models: T5-3B, BRIDGE and SQ-QDMR. We observed that the search algorithms work with multiple criteria for selecting the output query. We also compared the search-augmented methods with the fine-tuned models on the GeoQuery, IMDB, Yelp and Academic datasets (under the distribution shift) and observed that the method with search can sometimes work even better than fine-tuning. Com-

pared to fine-tuning, the search-based method does not require additional training data but relies on additional information on each test example. With such properties, our search based methods might be helpful for use cases like interactive query generation or annotating new datasets.

Limitations

In our experiments, we work only with the datasets where the user question was written in English. This might have simplified the task for T5 as the keywords and entity names of the query languages were also in English.

The test suites we built were still not perfect. In particular, it was hard to generate a test database such that the query `SELECT year FROM concert GROUP BY year HAVING count(*) >= 50` had non-empty output because it needed at least 50 rows with the same value in the column `year`. We also noticed that the value 1 as the gold query output also caused many false positives and should probably have been considered the empty value for the queries outputting the count aggregator.

The results of the search methods were also not perfect for out-of-domain data, even with strong search criteria based on tests. One reason for that was that we started to hit the limitations of the models, which were built with mostly Spider in mind. In particular, the database preprocessing stage to select values for the query was, in some cases, slow and inaccurate.

References

- Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. SADGA: Structure-aware dual graph aggregation network for Text-to-SQL. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.
- Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. 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 2541–2555, Online. Association for Computational Linguistics.
- Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. 2021. Neurosymbolic programming. *Foundations and Trends in Programming Languages*, 7(3).
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. [Evaluating large language models trained on code](#). *arXiv*, 2107.03374v2.
- Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *In Proceedings of the Workshop on Human Language Technology of Association for Computational Linguistics*.
- Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. [Structure-grounded pretraining for text-to-SQL](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: Pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Li Dong and Mirella Lapata. 2016. [Language to logical form with neural attention](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.
- Angela Fan, Mike Lewis, and Yann Dauphin. 2018. [Hierarchical neural story generation](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.
- Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui

- Zhang, and Dragomir Radev. 2018. [Improving text-to-SQL evaluation methodology](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.
- Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. [Towards complex text-to-SQL in cross-domain database with intermediate representation](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. [The curious case of neural text degeneration](#). In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. [Learning a neural semantic parser from user feedback](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.
- Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy Liang. 2019. SPoC: Search-based pseudocode to code. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.
- Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. [KaggleDBQA: Realistic evaluation of text-to-SQL parsers](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.
- Fei Li and H. V. Jagadish. 2014. [Constructing an interactive natural language interface for relational databases](#). *Proc. VLDB Endow.*
- Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2018. [Grammar-based neural text-to-sql generation](#). *arXiv*, 1905.13326.
- Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. [Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. [CodeXGLUE: A machine learning benchmark dataset for code understanding and generation](#). *arXiv*, 2102.04664v2.
- Ana-Maria Popescua, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the International Conference on Intelligent User Interfaces*.
- P. J. Price. 1990. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop*.
- 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](#). *Journal of Machine Learning Research*, 21(140):1–67.
- 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](#). *arXiv*, 2009.10297v2.
- Ohad Rubin and Jonathan Berant. 2021. [SmBoP: Semi-autoregressive bottom-up semantic parsing](#). In *Proceedings of the 5th Workshop on Structured Prediction for NLP (SPNLP 2021)*, pages 12–21, Online. Association for Computational Linguistics.
- Irina Saparina and Anton Osokin. 2021. [SPARQLing database queries from intermediate question decompositions](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8984–8998, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. [PICARD: Parsing incrementally for constrained auto-regressive decoding from language models](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. [Compositional generalization and natural language variation: Can a semantic parsing approach handle both?](#) 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 922–938, Online. Association for Computational Linguistics.
- Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the International Conference on Machine Learning (ICML)*.

- Kensen Shi, David Bieber, and Charles Sutton. 2020. [Incremental sampling without replacement for sequence models](#). In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2021. Learning to combine per-example solutions for neural program synthesis. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.
- Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. [Exploring unexplored generalization challenges for cross-database semantic parsing](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.
- Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the European Conference on Machine Learning*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Aleksandr 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.
- Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Aleksandr Polozov, and Rishabh Singh. 2019. [Robust text-to-SQL generation with execution-guided decoding](#). *arXiv*, 1807.03100.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. [Transformers: State-of-the-art natural language processing](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly supervised text-to-sql parsing through question decomposition. In *Findings of the Association for Computational Linguistics: NAACL 2022*.
- Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. [Break it down: A question understanding benchmark](#). *Transactions of the Association for Computational Linguistics*, 8:183–198.
- Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*
- Pengcheng Yin and Graham Neubig. 2017. [A syntactic neural model for general-purpose code generation](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.
- 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 *Proceedings of International Conference on Learning Representations (ICLR)*.
- Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. [CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. [Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.
- Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. [SPaC: Cross-domain semantic parsing in context](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.
- John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence (AAAI)*.
- Weixiong Zhang. 1998. Complete anytime beam search. In *Proceedings of the national conference on artificial intelligence (AAAI)*.
- Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. [Semantic evaluation for text-to-SQL with distilled test suites](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

Amit Zohar and Lior Wolf. 2018. Automatic program synthesis of long programs with a learned garbage collector. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

A Construction of Test Suites

The method of Zhong et al. (2020) relies on generating the so-called neighbor queries from a given set of gold queries Q . A set of neighbors N_q is obtained through slightly changing the original query $q \in Q$. Now, a database w distinguishes q and $g \in N_q$ if their execution results on w are different. The suite S is formed greedily: a new sampled database is added to S if it distinguishes a pair in $\mathcal{N} = \{(q, g) \mid q \in Q, g \in N_q\}$ that is not distinguished by any database previously added to S .

In the algorithm above, to sample the databases, the original queries are parsed to derive the constant values and the corresponding columns from the WHERE clauses. For instance, in a query `SELECT * from cars WHERE mpg > 20`, the constants 20 and 21 are assigned some probability to be generated as values of the `cars.mpg` column. We improve the query parser so that it accounts for common cases when table aliases are used (e.g., `SELECT * from cars as T WHERE T.mpg > 20`). Still, if a gold query has too many WHERE clauses, its execution on a randomly generated database is likely to be an empty table or, in the case of SELECT-ing an aggregate function, zeros or NULL values. This issue causes many potential false positives.

For this reason, for every query, we first search for a test database on which the query execution is not empty and only then proceed to check if the database distinguishes the query and its neighbors. This way, the chances are higher to sample at least one database yielding non-empty output on a gold query, leading to better code coverage. We also propose to build a separate test suite for each query. This modification allows dramatically reducing the time of the test-suite evaluation of a query.

Additionally, we make several important changes to the original implementation:

- We limit the number of rows in the tables to 100. As a result, the test suite databases are smaller in size and faster in evaluation.
- We adjust the types of the columns in the original and the sampled databases so that they match the type of the corresponding values. In the float type columns, we also reduce the precision of numbers to 16-bit. These changes are crucial for

SPARQL language, as it is more sensitive to data types than SQL.

- If a column has only unique values in the original database and its name contains special substrings such as 'Name', 'ID', and 'Phone', we treat it as a unique key and generate its values accordingly. This heuristic allows performing the GROUP BY operations on any unique key.

B Implementation Details of Search

For the search on top of T5-3B, we use the (Scholak et al., 2021) model provided in the Transformers library (Wolf et al., 2020). To search with large beam sizes, we modify the Transformers implementation of T5 inference: instead of caching keys and values in attention blocks, we cache the attention outputs, which reduces memory usage at the cost of speed.

For working with SQL queries outside the Spider dataset, we had to replace the SQL parsing coming with Spider due to its limited functionality. We use the `mo-sql-parsing` library instead.² We execute all SQL queries in the `sqlite3`³ package for python. For executing SPARQL queries, we use the open-source version of the Virtuoso system.⁴ During our search, the system generated many cumbersome queries, so we had to impose a strict time limit for each query and make our implementation robust to the database engine crashes. The time limit for Spider queries is 30 seconds and for other datasets is 300 seconds.

The grid of beam sizes for CAB search on top of T5-3B is 2, 10, 100, 800, the corresponding widths are 2, 2, 2, 5, the grid for BRIDGE is 1, 10, 100, 1000 and the widths are 1, 2, 2, 5, the grid for SQ-QDMR is 1, 100, 1000 and the widths are 1, 5, 10. We infer BRIDGE and SQ-QDMR on one NVIDIA V100 GPU and T5-3B on 8 NVIDIA A100 GPUs (while searching with maximum beam size). For the search with the sampling methods, we use the same schema: we run sampling several times, increasing the number of samples (k in top- k) until the selection criterion is passed. The grids for all the methods are the same. For searching with UniqueRandomizer, we run the methods until the selection criterion is passed or the maximum number of iterations is reached (equal to the maximum beam size: 800 for T5-3B and 1000 for BRIDGE and SQ-QDMR). We tune the temperature and p

²github.com/klahnakoski/mo-sql-parsing

³docs.python.org/3/library/sqlite3.html

⁴github.com/openlink/virtuoso-opensource

on Spider dev (the grid for temperature tuning was from 0.5 to 3.5 with 0.25 step; the grid for p was 0.85, 0.9, 0.95). The best $p = 0.95$ and the best temperature values are the following:

Model	CAB	Sample	Top-p	UniRand
T5-3B	2.7	2.4	3.1	2.7
BRIDGE	2.6	3.4	3.7	3.5
SQ-QDMR	1.0	1.3	1.9	1.2

While testing on single-database data, we use the same temperature parameters for the models trained on Spider only. For the fine-tuned models, we also tune temperature on dev sets and choose the values 2.0 for BRIDGE and T5-3B and 1.0 for SQ-QDMR.

C Execution (1-test) Accuracy vs. Test-suite Accuracy

In Table 6, we show the results of the search on Spider with two criteria: passing one test and passing the test suite. For both criteria, we compute execution accuracy (checks the execution result on one database) and test-suite accuracy (checks the execution results on the test suite). For all the models, test-suite accuracy is significantly lower than execution accuracy for the search with the one-test criterion.

Table 6: Execution (EX) and test-suite (TS) accuracy of search with 1-test and test-suite criteria on Spider dev and test.

Model	Split	1 test		Test Suite	
		EX	TS	EX	TS
T5-3B	dev	94.4	86.9	91.4	90.1
BRIDGE		91.0	81.9	87.4	84.0
SQ-QDMR		98.0	84.0	93.7	94.4
T5-3B	test	90.7	84.7	85.5	86.8
BRIDGE		83.4	76.6	77.4	77.8
SQ-QDMR		86.7	75.2	79.7	84.1

D Single-database Datasets

We use the query splits provided by [Finegan-Dollak et al. \(2018\)](#) for four single-database datasets: GeoQuery, IMDB, Yelp and Academic. We choose query splits because they are more difficult than question splits, according to the findings of [Finegan-Dollak et al. \(2018\)](#). We exclude duplicates and examples with gold SQL queries that crash or execute longer than 5 minutes with sqlite3. The statistics of resulted datasets are presented in Table 7.

Table 7: Statistics of single-database data.

Dataset	Train	Dev	Test
GeoQuery	536	159	182
IMDB	103	9	17
Yelp	104	11	10
Academic	142	18	15

E Fine-tuning Details

For fine-tuning on single-database data, we use official implementations of all the models.⁵ We experimented with different training strategies: initializing from released checkpoints of the models trained on Spider and training from scratch (in this case, models contain transformers pre-trained on textual data). We choose the best approach for each model and refer to it as fine-tuning.

We fine-tune T5-3B pre-trained on textual data ([Raffel et al., 2020](#)) for 300 epochs on one NVIDIA A100 GPU with the same parameters as [Scholak et al. \(2021\)](#) used: Adafactor optimizer ([Shazeer and Stern, 2018](#)) with learning rate 1e-4 and batch size 625.

The released checkpoint of the BRIDGE model was trained on data that includes question splits of single-database data that we consider. We re-train this model on Spider-only data to evaluate it on query splits of single-database datasets. We use the same training parameters as [Lin et al. \(2020\)](#) used: Adam optimizer ([Kingma and Ba, 2014](#)) with the same scheduler (L-inv learning rate decay) and batch size 32. We choose the best checkpoint on the development set as [Lin et al. \(2020\)](#) did in their work (execution accuracy and test-suite accuracy of our and authors' checkpoints are the same on Spider dev and test). We use the same training procedure for fine-tuning on single-database data but create training data from both Spider train set and the train set of a particular dataset.

For fine-tuning SQ-QDMR on GeoQuery, we use the corresponding part of the Break dataset ([Wolfson et al., 2020](#)) and generate the 366 train groundings using the automatic annotation model of [Saparina and Osokin \(2021\)](#). We cannot fine-tune on the Academic dataset because its database is large and preprocessing of [Saparina and Osokin \(2021\)](#) failed. QDMR forms for other datasets were not provided in the Break dataset, so we could not fine-tune on them. For fine-tuning on GeoQuery, we start with the released checkpoint and param-

⁵github.com/ElementAI/picard;
github.com/salesforce/TabularSemanticParsing;
github.com/yandex-research/sparqling-queries;

eters saved on 73000 iterations of Spider training and continue up to 81000 iterations with GeoQuery train data. We also use the same parameters as [Saparina and Osokin \(2021\)](#) used: the optimizer is Adam ([Kingma and Ba, 2014](#)) with polynomial decay scheduler used by ([Wang et al., 2020](#)), the batch size is 24.

We use 1 NVIDIA A100 GPU for training T5-3B, 1 NVIDIA V100 GPU for BRIDGE and 4 NVIDIA V100 GPUs for SQ-QDMR.

F Execution Accuracy of Search on Single-Database Data

Table 8 shows execution accuracy of search on different single-database datasets. We consider two types of models: trained only on Spider data and fine-tuned on single-database data. Comparing with Table 5, the figures are higher because many false-positive queries pass one test (the execution accuracy metric) and do not pass the test suite. For this reason, the results of searching with the test-suite criterion are lower in terms of execution accuracy: if no tested query satisfies the test-suite criterion, the system defaults to the result of the greedy decoding, which may fail one test, while the one-test criterion would select a false positive.

Table 8: Different search criteria (execution, output column match, 1 test and test suite) on top of pre-trained models on the query test splits of different datasets with **execution accuracy**.

Dataset (test size)	Model	Greedy	Exec	Cols	1 test	Test Suite
GeoQuery (182)	T5-3B	56.6	59.3	69.8	80.2	74.7
	+ fine-tune	69.8	76.4	88.5	97.8	97.3
	BRIDGE	57.6	51.1	63.7	86.8	74.7
	+ fine-tune	71.4	72.5	86.3	96.7	93.4
	SQ-QDMR	40.7	40.7	45.6	92.3	75.8
	+ fine-tune	61.5	61.5	64.8	90.7	84.1
IMDB (17)	T5-3B	5.9	17.6	17.6	35.3	35.3
	+ fine-tune	52.9	52.9	52.9	58.8	52.9
	BRIDGE	17.6	17.6	23.5	35.3	23.5
	+ fine-tune	52.9	52.9	52.9	58.8	52.9
	SQ-QDMR	11.8	11.8	11.8	41.2	35.3
Yelp (10)	T5-3B	30	60	50	10	30
	+ fine-tune	40	50	40	10	70
	BRIDGE	10	30	30	90	30
	+ fine-tune	40	80	80	100	70
	SQ-QDMR	40	40	40	80	40
Academic (15)	T5-3B	6.7	13.3	26.7	-	26.7
	+ fine-tune	53.3	53.3	53.3	-	73.3
	BRIDGE	6.7	6.7	6.7	-	20
	+ fine-tune	33	40	40	-	80
	SQ-QDMR	5.6	5.6	11.1	-	55.6