A GENERIC PROMPT FOR AN LLM THAT ENABLES NL-TO-SQL ACROSS DOMAINS AND COMPOSITIONS

Aseem Arora, Shabbirhussain Bhaisaheb, Manasi Patwardhan,
Lovekesh Vig & **Gautam Shroff**
TCS Research, India
{aseem.arora, shabbirhussain.b, manasi.patwardhan,
lovekesh.vig, gautam.shroff}@tcs.com

ABSTRACT

Large Language Models (LLMs) pretrained for code generation have demonstrated remarkable performance for multiple high level reasoning tasks. One such challenging task is Cross-Domain and Cross-Compositional generalization of Text-to-SQL semantic parsing, where the model is expected to exhibit generalization to novel compositions not seen during training. In this paper, we evaluate the capabilities of Codex, the GPT3 model pretrained with code, in both zero-shot and few-shot settings. Existing LLM (Codex) based approaches for this task rely on inference-time retrieval of similar few-shot samples from the training set, to build a run-time prompt for test time SQL query generation. In contrast, we devise an algorithm to come up with a minimal set of few-shots from the available data with complete coverage of SQL clauses, operators and functions and maximum coverage of available domains. We combine these few-shots with the out-of-distribution test query to define what we term as a *Generic Prompt (GP)*, which is further used to generate the corresponding SQL. The *GP*, being common across distinct test queries, not only provides us with a more efficient solution avoiding test time retrieval, but also yields SOTA few-shot cross-domain generalization results with an execution accuracy of 70.64% on the Spider-Dev set. We also evaluate the *Generic Prompt* on the App-Dev split of Spider-CG dataset for compositional generalization and Kaggle DBQA zero-shot cross-domain generation dataset obtaining few shot execution accuracy of 55.41% and 32.61%, respectively, with 3.25% and 11.41% absolute improvement over the zero-shot performance with Codex. We also showcase that for all the three datasets, the *GP* leads to a better performance than a prompt constructed by equal number of randomly selected exemplars from the available data.

1 INTRODUCTION

Recently, Large Language Models (LLMs) such as T5Roberts et al. (2019), GPT3Brown et al. (2020a), CodexChen et al. (2021b), PaLMChowdhery et al. (2022), pretrained with massive volumes of data have shown to improve performance for multiple reasoning tasks using in-context learning Brown et al. (2020b); Huang & Chang (2022), including program synthesis Austin et al. (2021); Jain et al. (2021); Nijkamp et al. (2022) and semantic parsing Shin & Durme (2021); Drozdov et al. (2022); Shin & Durme (2021); Shin et al. (2021). There are a few recent approaches where LLMs are specifically used for Text-to-SQL semantic parsing in (i) zero-shot setting Rajkumar et al. (2022a) where only the test Natural Language (NL) query constitutes the prompt, (ii) few-shot setting where exemplars similar to the test query in the target domain (SPIDER Yu et al. (2018b) Dev Split) are retrieved from the available training data (SPIDER Yu et al. (2018b) train split). Note that the database domains for the exemplars may be distinct from the target domain Poesia et al. (2022a) and (iii) few-shot setting where the exemplars (termed templates) are selected from a small amount of data from the target-domain ensuring maximum coverage of compositions Rajkumar et al. (2022a); Qiu et al. (2022); Hosseini et al. (2022); Yang et al. (2022) (For example, the GeoQuery dataset Tang & Mooney (2001); Zelle & Mooney (1996)). In few-shot settings, the selected exemplars, along with the test NL query, form the prompt for the LLM.

In this paper, we are mainly interested in (ii) i.e. cross-domain generalization, where the test queries may not have the set-of compositions covered in the training data (cross-composition generalizability). Moreover, considering a purely cross-domain setting, as opposed to (iii) above, we assume no availability of in-domain few-shots (exemplars belonging to the target databases). Synchromesh Poesia et al. (2022a) has a similar setting, except they retrieve exemplars during run-time (at inference) by (a) selecting semantically similar NL queries as exemplars and (b) Target Similarity Tuning (TST) where the exemplars with similar target programs, irrespective of differences in the surface natural language features, are selected. This reliance on inference-time retrieval of similar few-shot samples from the available data to build a run-time prompt and generate SQL for a test query, results in a less efficient solution. As opposed to this, we devise an algorithm to come up with a minimal set-of few-shot exemplars from the available data where the set-of selected exemplars ensures complete coverage of SQL clauses, operators and functions and maximizes coverage of available domains. We combine these few-shots with the out-of-distribution test query to define what we term as a *Generic Prompt (GP)*, which is further used to generate the corresponding SQL. As the generation of the *GP*, which is common across distinct test queries, is offline, it provides us with a more time-efficient solution obviating the need for real time retrieval. Moreover, with this prompt we obtain state-of-the-art few-shot cross-domain generalization results on the Spider dataset with an execution accuracy of 70.64%. We further obtain an execution accuracy of (i) 55.41% on the more complex Spider-CG dataset for queries on unseen databases with unseen compositions and (ii) 32.26% on the Kaggle-DBQA Lee et al. (2021b) zero-shot cross-domain generalization dataset. We establish that these results are better than zero-shot results and few-shot results with the same number of randomly selected exemplars as in *GP* (we term this as Random Prompt - RP).

2 Related Work

Rajkumar et al. (2022a) has attempted to use LLM for Text-to-SQL semantic parsing task in zero-shot as well as few-shot settings. For zero-shot setting, they experiment with various formats of the test query along with the database schema with which the LLMs are prompted, such as the APIDocs schema format (explained in detail in section 4 and Table 4) or via SQL '*CREATE TABLE*' commands, with and without randomly selected data rows from the table. For the few-shot setting, Rajkumar et al. (2022a); Qiu et al. (2022); Hosseini et al. (2022); Yang et al. (2022) focus on cross-composition generalization and provide the queries which are posed on the target database itself as exemplars (cross-domain setting is not considered). Thus, the assumption is that few queries are available for a new database. They work with datasets such as GeoQuery (Tang & Mooney (2001); Zelle & Mooney (1996)) with data in US geography domain, Scholar (Iyer et al. (2017)) with data in academic publications or a dataset designed for queries in E-commerce domain (Yang et al. (2022)). In our approach, we assume no availability of annotated data in terms of SQL programs for the given textual (natural language) queries and, thus, completely a cross-domain setting.

Sychromesh (Poesia et al. (2022a)) assumes a cross-domain setting and, for the test query in the target domain, similar queries from the source domain are used as exemplars. 5-shot samples with NL queries and their corresponding SQLs are extracted as exemplars based on (i) the semantic similarity of the natural language queries with the test query, and (ii) Target Similarity Tuning (TST) where the exemplars with similar target programs, irrespective of differences in the natural language queries, are selected. In addition to TST, the authors perform constrained semantic decoding (CSD), which reduces the implementation errors in the generated SQLs by ensuring that the generated tokens lead to correct programs following a pre-specified grammar. For every new test query, the few-shots in the prompt have to be retrieved during run-time to match the given query. This reliance on inference-time retrieval to build a run-time prompt and generate SQL for a test query, leads to an inefficient solution. As opposed to this, we devise an algorithm to come up with a minimal set-of few-shot exemplars from the available data, where the set-of selected exemplars ensures complete coverage of SQL clauses, operators and functions and maximal coverage of domains. We combine these few-shots with the out-of-distribution test query to define what we term as a *Generic Prompt (GP)*, which is fed to the LLM to generate the corresponding SQL. As the generation of the *GP*, which is common across distinct test queries, is offline, it provides us with a more time-efficient solution. Also, with the well-curated few-shots our approach yields better performance than all the above discussed approaches.

Rajkumar et al. (2022a) have addressed the concerns around possible memorization of existing datasets such as SpiderYu et al. (2018a) by large language models such as Codex, which are trained on code data. The possibility of memorization arises as the the *Spider Dev* split file (*dev.sql*) resides on Github[1]. However, prompting Codex with verbatim fragments of this file leads to generations which do not match with the file contents. For example, given a question in the format specified in the file, the table aliasing strategy followed in the generated SQLs does not match with the gold SQLs provided in the file. On the similar lines of Rajkumar et al. (2022a), our prompting format of text queries (APIDocs+Values: Explained in Section 4 and Table 4 in detail) is completely different than the format in which NL-SQL pairs are stored in the *Spider Dev* split file. With the unique *GP* based approach, in the paper, along with the Spider dataset Yu et al. (2018b), we show performance improvements over zero-shot setting for other datasets such as *Spider-CG* Gan et al. (2022) and *Kaggle DBQA* Lee et al. (2021a), for which the evaluation files are not residing on Github[2], for the LLMs (Codex in our case). This eliminates the possibility of memorization by models like Codex for these datasets.

3 DATASET

We use three Text-to-SQL datasets for carrying out the experiments with our proposed approach, *namely*, Spider Yu et al. (2018a), Spider-CG Gan et al. (2022), and the Kaggle-DBQA Lee et al. (2021a) datasets.

Spider dataset: *Spider* is a human-annotated, large-scale, complex, and cross-domain Text-to-SQL benchmark dataset. The dataset consists of a total of 200 databases with 140, 20, 40 databases in the training, development and test splits, respectively. The respective splits contain 7000, 1034, and 2147 Text-to-SQL pairs however the Spider test-split is not available publicly. We analyse the performance of our model on the Spider development split (*Spider-Dev*).

Spider-CG dataset: The *Spider-CG* is designed for measuring the compositional generalization performance of Text-to-SQL models. The authors first modify the Spider dataset Yu et al. (2018a) to obtain the Spider-SS dataset where the Text-to-SQL pairs from *Spider-Train* are transformed to corresponding sub-sentences and NatSQL pairs. The sub-sentences are obtained using a sentence-split algorithm and the corresponding NatSQL is manually annotated. Based on Spider-SS, Spider-CG is constructed by either substituting sub-sentences with those from other samples (CG-SUB), or composing two sub-sentences to form a more complicated sample (CG-APP). The CG-SUB consists of a train split (CG-SUB_T) and a development split (CG-SUB_D). Similarly, CG-APP is divided in a train split (CG-APP_T) with 18,793 samples and a development split (CG-APP_D) with 3,237 samples.As our focus is mostly on cross-domain generalization, we evaluate our approach on the CG-APP_D. This is a more difficult split to test on than (*Spider-Dev*) as along with cross-domain, it has out-of-distribution queries with respect to the compositions.

Kaggle-DBQA dataset: This is a cross-domain Text-to-SQL evaluation dataset of real Web databases. The dataset covers a total of 8 databases and for each database there is a set of fine-tuning and test examples. The 8 databases are (i) Nuclear with 10 and 22, (ii) Crime with 9 and 18, (iii) Pesticide with 16 and 34, (iv) MathScore with 9 and 19, (v) Baseball with 12 and 27, (vi) Fires with 12 and 25, (vii) WhatCD with 13 and 28, and (viii) Soccer with 6 and 12 fine-tuning and test examples, respectively.

4 APPROACH

We use LLMs in both zero-shot and few-shot settings. In the zero-shot setting the prompt only has the test query along with its database schema in a specific format, whereas for the few-shot setting we have come up with an algorithm which selects exemplars from *Spider-Train* dataset with complete coverage of SQL clauses, operators and functions and maximum coverage of domains (databases). We use these exemplars appended with the test query in a specific format to generate the *Generic Prompt (GP)*. In few-shot settings, the *GP* is provided as an input to the LLM to generate the SQL. We perform post-processing on the generated queries and then execute them on the database to get the

[1] https://github.com/taoyds/spider/tree/master/evaluation_examples

[2] https://github.com/chiahsuan156/KaggleDBQA; https://github.com/ygan/SpiderSS-SpiderCG

predicted answer. The predicted answer is compared with the ground truth (GT) answer to compute the execution accuracy. The details of the LLM we use, the algorithm and the formats of the test query and the exemplars are discussed in detail below.

Algorithm 1: Generic Prompt Creation

Input : $D = \{db_j, \{t_{ij}, s_{ij}, a_{ij}\}_{i=1}^{N_j}\}_{j=1}^{M}$ ▷ Available Dataset with database, text query, SQL query, answer tuples,
$T = \{db_l, \{t_{kl}, s_{kl}, a_{kl}\}_{k=1}^{K}{}_l\}_{l=1}^{L}$ ▷ TestSet,
$O = \{Operators, Clauses, Functions\}$
Output : GP ▷ Generic Prompt
Initial Stage : $E \leftarrow \Phi$ ▷ Exemplars
$i \leftarrow 1$,
while $i \neq N_m * M$ **do**
  $j \leftarrow 1$
  while $j \neq M$ **do**
    $O_e \leftarrow$ *Extract_Operators*(s_{ij}) ▷ Operator extracted from SQL
    if $O_e \in O$ **then**
      $E \leftarrow E + \{db_j, t_{ij}, s_{ij}\}$ ▷ Add tuple as an Exemplar if Operators were not Covered
      $O \leftarrow O - O_e$ ▷ Remove covered operators
      for *x in E* **do**
        if *Extract_Operators*$(s_x \in x) \in O_e$ **then**
        end if
        $E \leftarrow E - \{db_x, t_x, s_x\}$ ▷ Remove Exemplar Query if Extracted Operators are super-set of Exemplar Query Operators
      end for
    end if
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while
$GP \leftarrow E + \{db_l, t_{kl}, s_{kl}\} \in T$

```
### SQLite SQL tables, with their properties:
#
# Addresses('address_id', 'line_1', 'line_2', 'city', 'zip_postcode',
'state_province_county', 'country')
# People('person_id', 'first_name', 'middle_name', 'last_name',
'cell_mobile_number', 'email_address', 'login_name', 'password')
# Students('student_id', 'student_details')
# Courses('course_id', 'course_name', 'course_description', 'other_details')
# People_Addresses('person_address_id', 'person_id', 'address_id',
'date_from', 'date_to')
# Student_Course_Registrations('student_id', 'course_id', 'registration_date')
# Student_Course_Attendance('student_id', 'course_id', 'date_of_attendance')
# Candidates('candidate_id', 'candidate_details')
# Candidate_Assessments('candidate_id', 'qualification', 'assessment_date',
'asessment_outcome_code')
#
### which course has most number of registered students?
SELECT T1.course_name FROM courses AS T1 JOIN student_course_registrations AS T2 ON T1.course_id = T2.course_Id GROUP BY T1.course_id ORDER BY count(*) DESC LIMIT 1;

.....


### SQLite SQL tables, with their properties:
#
# Rooms('RoomId', 'roomName', 'beds', 'bedType', 'maxOccupancy',
'basePrice', 'decor')
# Reservations('Code', 'Room', 'CheckIn', 'CheckOut', 'Rate', 'LastName',
'FirstName', 'Adults', 'Kids')
#
### Find the first and last names of people who payed more than the rooms'
base prices.
SELECT T1.firstname , T1.lastname FROM Reservations AS T1 JOIN Rooms
AS T2 ON T1.Room = T2.RoomId WHERE T1.Rate - T2.basePrice > 0;
```

Table 1: Generic Prompt (Partial for Illustration)

Large Language Model: We choose Codex Chen et al. (2021a) (Code-Da-Vincci) Chen et al. (2021b) from OpenAI with 175B parameters as the LLM, because it provides few-shot state-of-the-art performance on most of the semantic parsing tasks including Text-to-SQL Rajkumar et al. (2022b), as well as showcases compositional generalization capabilities Qiu et al. (2022); Hosseini et al. (2022). We access the OpenAI Codex using its API and use in-context learning to generate SQLs for the given NL query with our uniquely defined prompt.

Prompt Design Algorithm: For prompt design, we assume to have a dataset $D = \{db_j, \{t_{ij}, s_{ij}, a_{ij}\}_{i=1}^{N_j}\}_{j=1}^{M}$, where t and s are the text-SQL query pairs posed on database db and a are the answers to the SQL queries after execution, N_j are total number of query pairs belonging to database db_j, M are total number of databases.$T = \{db_l, \{t_{kl}, s_{kl}, a_{kl}\}_{k=1}^{K_l}\}_{l=1}^{L}$ of $\sum_{l=1}^{L} K_l$ query pairs and L databases, such that $\{db_j\}_{j=1}^{M} \cap \{db_l\}_{l=1}^{L} = \phi$. As we consider the cross-domain setting we do not have any overlap between the training and test set databases.

We manually collect SQL operators, clauses and functions to form a set-of primitive operations O. The final set consists of operations covered by queries $\{q_i\} \in D$. The primitive operations include (i) SQL Clauses such as 'FROM', 'HAVING', 'WHERE', 'ORDER BY', etc, (ii) SQL Operators such as arithmetic (+, -, *, /, %), comparison (=, !=, $<$, $>$, etc) and logical (ALL, AND, ANY, LIKE, etc) operators and (iii) SQL Functions such as (AVG, COUNT, MAX, MIN, etc). To select the few-shot exemplars E for the *GP* we traverse the query pairs database wise, such that the first NL-SQL query pairs corresponding to every database are traversed first followed by the 2^{nd} and so on. This allows for maximum coverage of the databases. As we traverse samples in D, a sample $\{db_j, t_i, s_i\}$ becomes part of E, if s_i covers at least one uncovered primitive operation in O. If currently traversed query s_i covers a super-set of primitive operations of a query s_x in the existing exemplar $\{db_x, t_x, s_x\} \in E$ then $\{db_i, t_i, s_i\}$ replaces $\{db_x, t_x, s_x\}$ in E. The algorithm terminates when all the queries are visited once and all the possible primitive operations in O are covered. This allows us to choose a minimal set of samples as exemplars to cover the complete set of primitive operations. The detailed algorithm is explained in Algorithm 1. We use the *Spider-train* set as D and *Spider-dev* set as T. Our algorithm yields 18 exemplars as few-shots covering a total of 15 databases. The selected queries cover a total of 32 SQL operators and clauses. We use API-Doc format for the exemplars illustrated

```
### SQLite SQL tables with their properties:
#
# AREA_CODE_STATE('area_code', 'state')
# range of values of column area_code (201, 989)
# unique values of column state ('TN', 'NY', 'RI', 'KS', 'CT', 'VA')
# VOTES('vote_id', 'state', 'contestant_number')
# range of values of column vote_id (1, 5)
# unique values of column state ('NY', 'CA', 'NJ')
# range of values of column contestant_number (2, 5)
#
### Return the names of the contestants whose names contain the substring
'Al' , ordered by contestant name descending .
SELECT
```

Table 2: Zero-shot query example

Before post-processing	After post-processing
SELECT Airline FROM airlines WHERE Abbreviation = ' UAL ' AND Country = ' USA ';	SELECT Airline FROM airlines WHERE Abbreviation = 'UAL' AND Country = 'USA';
SELECT Airline FROM flights GROUP BY Airline HAVING count(*) >= 10 UNION SELECT Airline FROM airlines WHERE Airline = ' JetBlue Airways ' OR Airline = ' SpiceJet Airways';	SELECT Airline FROM flights GROUP BY Airline HAVING count(*) >= 10 UNION SELECT Airline FROM airlines WHERE Airline = 'JetBlue Airways' OR Airline = 'SpiceJet Airways';

Table 3: Example of a query with post-processing

in Table 1. API Docs consist of the sequence of tables along with the column names followed by the text (NL) query, followed by its SQL.

Test Query Format: The test query, which is a sample $\{db_l, t_{lk}\} \in T$ is added to the prompt in a specific format depicted in Table 2. It is database schema followed by the text (NL) query in the same API Doc format. However, to make the model better understand the details of the schema, we also add the column values in the given format. For categorical columns, we add unique column values while for the numerical columns of type *int* or *float*, we add the existing range of values in the column. Due to the limitation on the number of maximum token length allowable by the OpenAI Codex which is 8000 tokens, we threshold the unique values for categorical columns to T per column. We empirically found the value of $T = 10$. In zero-shot settings the prompt consists of only the test sample in the specified format. Whereas, in the few-shot setting the test sample is appended to the few-shot exemplars to form the *GP* as outlined in Algorithm 1.

Post-processing: We post-process the generated SQL queries. We notice that for some of the generated queries, an extra space is generated at the beginning and/or at the end of the quotes depicting column values of the query, as shown in the examples illustrated in Table 3. We remove these trailing spaces as the post-processing step.

5 Results and Discussion

For bench-marking, we use existing zero-shot Rajkumar et al. (2022b) and few-shot Poesia et al. (2022b) approaches which have used Codex for generating SQL from Natural Language Queries. We also compare our results with state-of-the-art supervised approaches in the literature. We use execution accuracy as the metric to compare the results.

As illustrated in the Table 4, Rajkumar et al. (2022b) prompts Codex with the test sample in distinct formats: (i) Only NL question, (ii) *API Docs*: which is same as our format represented in Figure 2 and explained in section 4, except the column values are not added to the table schema, (iii) *API Docs + Select X*: Add 'X' rows of each table to (ii), (iv) *Create Table*: SQL table creation command (v) *Create Table + Select X*: Add 'X' rows of each table to (iv). For *Spider-Dev*, Rajkumar et al. (2022b) have their best zero-shot performance with *API DOCS + Select 3*, which yields 60.3% execution accuracy. Our zero-shot prompt ((*API Docs + Values*) setting demonstrates superior performance of 68.31% as compared to the closest format of *API Docs + Select 10* by Rajkumar et al. (2022b), which yields them 60.8% execution accuracy. Additionally, with post-processing (*API Docs + Values + PoPr*) we get further improvement to 70.16%.

The few-shot approach discussed in Synchromesh Poesia et al. (2022b) which uses 5 most similar exemplars. As constraint decoding is not the focus our approach we compare our performance with the execution accuracy by Synchromesh Poesia et al. (2022b) using Target Semantic Tuning (TST) and without Constraint Semantic Decoding (CSD), which is 60% with Codex for *Spider-Dev* dataset. As compared to their approach, our approach achieves an execution accuracy of 69.48% in the setting *Generic prompt + API docs + Values*, showing an absolute improvement of 9.48% execution accuracy. When we additionally use post-processing (*Generic prompt + API docs + Values + PoPr*), the performance further improves to 70.64%, showing an absolute improvement of 10.64% over the TST setting in Poesia et al. (2022b). Though we use 18-shots in our *GP* based approach, which is higher than the 5-shots used in Poesia et al. (2022b); these 18-shots are consistent across all the test queries. As Poesia et al. (2022b) use similarity based exemplar retrieval approach, for them Top-K,

Approach	Model	Prompt Setting	Description	Supervised / Zero / Few Shot	Spider-Dev	CG-APP-D	Kaggle-DBQA
					Execution Accuracy		
Gan et al. (2022)	RATSQL	-	Models trained with *Spider-Train*	Supervised	76.70%	75.10%	13.56%†
Rubin & Berant (2020)	SmBOP	-		Supervised	74.40%	59.60%	26.49%
Scholak et al. (2021)	Picard	-		Supervised	74.18%	59.65%	26.48%
Qi et al. (2022)	RASAT	-		Supervised	76.60%	59.90%	27.47%
Poesia et al. (2022b)	GPT-3 13B	NLS	NLS: Natural Language based Similarity TST: Target Similarity Tuning	Few(5)	16%	-	-
		TST		Few(5)	14%	-	-
	GPT-3 175B	NLS		Few(5)	28%	-	-
		TST		Few(5)	31%	-	-
	Codex 175B	NLS		Few(5)	56%	-	-
		TST		Few(5)	60%	-	-
Rajkumar et al. (2022b)	Codex 175B	Question	NL query	Zero	8.30%	-	-
		API Docs	NL query + tables + columns	Zero	56.8%	-	
		+ Select 1	Select X: API docs + X rows sampled from each table	Zero	60.9%	-	-
		+ Select 3			60.3%	-	-
		+ Select 5			60.5%	-	-
		+ Select 10			60.8%	-	-
		Create Table	Table Creation Commands	Zero	59.9%	-	-
		+ Select 1	Create Table + X rows sampled from each table	Zero	64.8%	-	-
		+ Select 3			67.0%	-	-
		+ Select 5			65.3%	-	-
		+ Select 10			63.3%	-	-
Our approach	Codex 175B	API Docs + Values	API Docs + R unique values for categorical columns and range for numerical columns	Zero	68.31%	50.50%	21.20%
		*Random prompt + API docs + Values	Randomly selected exemplars in API docs format + Test Query in API docs + Values format	Few(18)	66.52%	51.55%	29.83%
		Generic prompt + API docs + Values	Generic Prompt with a well curated set of exemplars selected by an algorithm in API docs format + Test Query in (API docs + Values) format	Few(18)	**69.48%**	**53.59%**	**32.61%**
		API Docs + Values + PoPr	PoPr: Post-processing	Zero	70.16%	52.16%	21.20%
		*Random prompt + API docs + Values + PoPr		Few (18)	66.52%	51.55%	29.83%
		Generic prompt + API docs + Values + PoPr			**70.64%**	**55.41%**	**32.61%**

Table 4: Execution Accuracy Results. (†) indicates Exact Match results from Lee et al. (2021c). (*) indicates results are averaged over Three runs

.

Prompt setting	Spider-Dev				
	Execution Accuracy				
	Easy	Medium	Hard	Extra	All
API docs + Values	81.05%	72.75%	56.90%	49.40%	68.31%
*Random prompt + API docs + Values	81.25%	70.27%	56.61%	44.88%	66.52%
Generic prompt + API docs + Values	79.44%	73.65%	63.79%	49.40%	69.48%
API Docs + Values + PoPr	83.87%	75.00%	57.47%	50.00%	70.16%
*Random prompt + API docs + Values	81.25%	70.27%	56.61%	44.88%	66.52%
Generic prompt + API docs + Values + PoPr	81.85%	74.77%	64.37%	49.40%	70.64%

Table 5: Results of our approach on *Spider-Dev* based on hardness of the queries. (*) indicates results are averaged over Three runs

where $K > 5$, may not yield better results than $K = 5$. Thus, number of few-shots used by our *GP* approach are not directly comparable with the number of few-shots selected by Poesia et al. (2022b).

For *CG-APP-D* and *Kaggle-DBQA*, Rajkumar et al. (2022b) and Poesia et al. (2022b) haven't reported any results. Overall, for all three datasets our *GP* based approach demonstrates an improvement over the zero-shot with (average absolute improvement of 5.04% execution accuracy across the datasets) and without (average absolute improvement of 5.22% execution accuracy across the datasets) post-processing.

We believe that in retrieval based few-shot techniques like Synchromesh Poesia et al. (2022b) the number of exemplars > 5 may not yield better performance. In Synchromesh, similar exemplars are selected using a neural network based scoring function, trained with the tree-edit distance based similarity between the corresponding SQLs of the natural language query pairs. Top-K exemplars, which serve as few-shots, are retrieved. This ensures that the resulting few-shots follow the same SQL query composition, as that of the test natural language query having maximally overlapping operators, clauses and functions. These retrieval based approaches do not ensure comprehensive coverage of all SQL operators, clauses and functions, but coverage of only the ones that fit into the composition of the test query. Also, Synchromesh must have empirically found the optimal value of K to be 5, yielding the best performance for the task. To support our above discussed hypothesis empirically, as the code of Synchromesh is not available, we have followed their setup and computed results with selecting 18 similar exemplars as few-shots. On KaggleDBQA test set Lee et al. (2021a) , this yields an execution accuracy of 29.83%, which is lower than the *GP* accuracy (32.61% 4). Similarly for

Spider Dev Yu et al. (2018a) dataset, with similar 18 few-shots, the execution accuracy is 69.19%, which is inferior to *GP* based approach (70.64% 4). These results after post-processing.

To fairly demonstrate the effectiveness of the *GP* created with our novel algorithm, we also report the results obtained with using a *Random Prompt (RP)* replacing the *GP* with the same number (18) of randomly selected exemplars as that of *GP* from the *Spider-Train* dataset. We create three distinct *RP*s by randomly sampling the queries from *Spider-Train* with replacement. We observe that the results of each *RP* is inferior to the *GP* results for the respective datasets. However, in Table 4 we illustrate the performance averaged over the three iterations of the RP for each dataset. Overall, for all three datasets our *GP* based approach demonstrates an improvement over the RP both with (average absolute improvement of 3.59% execution accuracy across the datasets) and without (average absolute improvement of 2.59% execution accuracy across the datasets) post-processing. This demonstrates that the well-curated *GP* carries information necessary to address more queries than the information carried by the *RP* which consists of same number of exemplars. We also observe that for the *Spider-Dev* dataset *RP* leads to inferior results as compared to zero-shot acting as noise, whereas for *Spider-CG* dataset, the performance of *RP* is comparable to zero-shot demonstrating no additional benefit of using the randomly selected exemplars as few-shots in the prompt. This also illustrates the efficacy of our *GP* based approach and the novelty of our algorithm used to choose the exemplars to construct the *GP*.

As constraint decoding is not the focus of this paper, for fair comparison, we compute the execution accuracy results (depicted in Table 4) without constraint decoding for the supervised models Picard Scholak et al. (2021) and RASAT Qi et al. (2022). As opposed to the our zero-shot results, our well-curated *GP* with only 18 exemplars provide comparable results (achieving 70.64% and 55.41% execution accuracy on *Spider-Dev* and *CG-APP-Dev*) with the state-of-the-art supervised approaches such as SmBOP Rubin & Berant (2020), Picard Scholak et al. (2021) and RASAT Qi et al. (2022). The Results with RATSQL Gan et al. (2022) are not directly comparable with the other supervised approaches as it uses additional NATSQL based supervision. For *KaggleDBQA* our zero-shot results (with 21.20% execution accuracy) are inferior to the supervised approaches, however the *GP* results (with 32.61% execution accuracy) are superior to the supervised approaches. This shows for our of domain queries *GP* setting works better than the zero-shot and yields at-par or better performance than supervised setting. Also, though *GP* yields better performance, the overall low execution accuracy numbers on *KaggleDBQA* and *CG-APP-Dev* datasets demonstrates the scope for Out-of-Distribution setting in terms of addressing in cross-domain and cross-composition queries. Further, Table 5 provides results on the *Spider-Dev* dataset based on the query hardness annotation provided in the dataset. It was observed that without post-processing the *GP* yields performance improvement for harder queries.

6 Qualitative Analysis

We perform qualitative analysis on a set of sampled queries as shown in Table 6. We divide our analysis into three cases:(i) Samples for which SQLs generated by zero-shot prompt are rectified by the *GP*, (ii) Samples for which both zero-shot and *GP* fail to generate correct SQLs and (iii) Samples for which SQLs generated by zero-shot are correct, but by the *GP* are incorrect. We randomly sample 40 queries for each case (120 queries in total) for manual analysis.

For the first case, we classify the samples in three categories (Table 6). The percentage of queries belonging to each category is provided in the column '%'. The first category is 'Over-simplified' (example 1), where zero-shot generates over-simplified SQLs by not involving the required 'JOINs' in the query, but the *GP* fixes the same. The second category is 'Over-complicated' (example 2) where, an unnecessary 'ORDER BY' clause is observed in the zero-shot generated query further rectified by *GP*. The third category is 'Logically Incorrect', where the generated zero-shot query does not semantically match with the given NL description. For example in example 3 of table 6, the zero-shot SQL incorrectly puts a condition on 'trainscript_id = 1', but gets rectified by the *GP*.

For the samples belonging to the second case, we find that there is a set of queries, both zero-shot and *GP* generated, which are 'Over-simplified' (example 4), and specifically, use no JOIN, which is required. We also find a set of queries where 'Redundant columns' are used (example 5). The third set of queries are of category 'Incorrect Tables' where, wrong tables are used in the queries (example

	Sr. No.	Category	Sample	ZS	GP	%
Case 1	1	Over-simplified; No JOIN used	***NL**: Which city has most number of arriving flights?* ***GT**: SELECT T1.City FROM AIRPORTS AS T1 JOIN FLIGHTS AS T2 ON T1.AirportCode = T2.DestAirport GROUP BY T1.City ORDER BY count(*) DESC LIMIT 1* ***ZS**: SELECT DestAirport, COUNT(*) AS count FROM flights GROUP BY DestAirport ORDER BY count DESC LIMIT 1* ***GP**: SELECT T1.City FROM airports AS T1 JOIN flights AS T2 ON T1.AirportCode = T2.DestAirport GROUP BY T1.City ORDER BY count(*) DESC LIMIT 1*	✗	✓	30%
	2	Over-complicated; Additional ORDER BY	***NL**: What are the names of countries that speak more than 2 languages, as well as how many languages they speak?* ***GT**: SELECT COUNT(T2.Language) , T1.Name FROM country AS T1 JOIN countrylanguage AS T2 ON T1.Code = T2.CountryCode GROUP BY T1.Name HAVING COUNT(*) ¿ 2* ***ZS**: SELECT CountryCode, COUNT(*) AS Languages FROM countrylanguage GROUP BY CountryCode HAVING COUNT(*) ¿ 2 ORDER BY Languages DESC* ***GP**: SELECT T1.name , count(*) FROM country AS T1 JOIN countrylanguage AS T2 ON T1.code = T2.countrycode GROUP BY T1.code HAVING count(*) ¿ 2*	✗	✓	22.5%
	3	Logically incorrect	***NL**: When is the first transcript released? List the date and details.* ***GT**: SELECT transcript_date, other_details FROM Transcripts ORDER BY transcript_date ASC LIMIT 1* ***ZS** SELECT transcript_date, other_details FROM Transcripts WHERE transcript_id = 1* ***GP**: SELECT transcript_date , other_details FROM transcripts ORDER BY transcript_date LIMIT 1*	✗	✓	47.5%
Case 2	4	Over-simplified; No JOIN used	***NL**: Which region is the city Kabul located in?* ***GT**: SELECT Region FROM country AS T1 JOIN city AS T2 ON T1.Code = T2.CountryCode WHERE T2.Name = "Kabul"* ***ZS**: SELECT District FROM city WHERE Name = 'Kabul'* ***GP**: SELECT district FROM city WHERE name = 'Kabul'*	✗	✗	45%
	5	Redundant columns	***NL**: What is the most common nationality of people?* ***GT**: SELECT Nationality FROM people GROUP BY Nationality ORDER BY COUNT(*) DESC LIMIT 1* ***ZS**: SELECT Nationality, COUNT(Nationality) AS Nationality_Count FROM people GROUP BY Nationality ORDER BY Nationality_Count DESC LIMIT 1* ***GP**: SELECT Nationality, count(*) FROM people GROUP BY Nationality ORDER BY count(*) DESC LIMIT 1*	✗	✗	32.5%
	6	Incorrect Table	***NL**: When did the episode "A Love of a Lifetime" air?* ***GT**: SELECT Air_Date FROM TV_series WHERE Episode = "A Love of a Lifetime"* ***ZS**: SELECT Original_air_date FROM Cartoon WHERE Title = 'A Love of a Lifetime'* ***GP**: SELECT Original_air_date FROM Cartoon WHERE Title = 'A Love of a Lifetime'*	✗	✗	12.5%
	7	Logically incorrect	***NL**: What is the maximum capacity and the average of all stadiums?* ***GT**: select max(capacity), average from stadium* ***ZS**: SELECT MAX(Capacity), AVG(Average) FROM stadium* ***GP**: SELECT MAX(capacity), AVG(capacity) FROM stadium*	✗	✗	10%
Case 3	8	Over-complicated; Unnecessary JOIN	***NL**: What are the emails of the professionals living in either the state of Hawaii or the state of Wisconsin?* ***GT**: SELECT email_address FROM Professionals WHERE state = 'Hawaii' OR state = 'Wisconsin'* ***ZS**: SELECT email_address FROM Professionals WHERE state = 'Hawaii' OR state = 'Wisconsin'* ***GP**: SELECT email_address FROM professionals JOIN locations ON professionals.professional_id = locations.professional_id WHERE state = 'Hawaii' OR state = 'Wisconsin';*	✓	✗	32.5%
	9	Incorrect table	***NL**: How many different degrees are offered?* ***GT**: SELECT count(DISTINCT degree_summary_name) FROM Degree_Programs* ***ZS**: SELECT COUNT(DISTINCT degree_summary_name) FROM Degree_Programs* ***GP**: SELECT count(DISTINCT degree_program_id) FROM Student_Enrolment*	✓	✗	30%
	10	Logically incorrect	***NL**: How many different series and contents are listed in the TV Channel table?* ***GT**: SELECT count(DISTINCT series_name), count(DISTINCT content) FROM TV_Channel* ***ZS**: SELECT COUNT(DISTINCT series_name) AS series_name, COUNT(DISTINCT Content) AS Content FROM TV_Channel* ***GP**: SELECT count(DISTINCT series_name) + count(DISTINCT content) FROM TV_Channel*	✓	✗	37.5%

Table 6: Qualitative analysis. NL:Natural Language, GT: Ground Truth, ZS: Zero-Shot, ✓indicates the query is correct and ✗indicates the query is incorrect. % of sampled 40 queries under each case

6). While the fourth category is 'Logically Incorrect' queries. An instance is shown in example 10 where 'AVG' function is used unnecessarily.

As part of the analysis for the third case, we observe that the queries broadly belong to three categories. The first one being 'Over-complicated', where an unnecessary JOIN is used in the *GP* generated query (example 8). The second category where incorrect tables are used in the *GP* generated query (example 9) and the third category is where the *GP* generated query is logically incorrect. For instance, in example 10 addition '+' operator is used which is not correct.

7 CONCLUSION

Semantic parsing of Text-to-SQL in a cross-domain setting poses a challenging problem due to a lack of availability of Text-SQL pairs in the new domain with unseen compositions. In this paper, we leverage the pre-trained Codex model in zero-shot and few-shot settings. As opposed to prior approaches which rely on inference-time retrieval of exemplars, we devise an algorithm which constructs a well-curated Prompt, which we term as the *Generic Prompt* as it acts as a common prompt across every test sample obviating the need for dynamic exemplar retrieval. Our novel algorithm performs offline retrieval to obtain a set of exemplars based on complete coverage of the SQL operators, clauses and functions and maximal coverage of databases to form the *GP*. Experiments demonstrate that the *GP* achieves state-of-the-art results on three distinct cross-domain datasets, surpassing the prior zero-shot and few-shot approaches and yielding performance comparable with supervised approaches.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, abs/2108.07732, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020a.

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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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*, abs/2005.14165, 2020b.

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. Evaluating large language models trained on code, 2021a. URL `https://arxiv.org/abs/2107.03374`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura 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, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021b.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Andrew Drozdov, Nathanael Scharli, Ekin Akyuurek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. *ArXiv*, abs/2209.15003, 2022.

Yujian Gan, Xinyun Chen, Qiuping Huang, and Matthew Purver. Measuring and improving compositional generalization in text-to-sql via component alignment. *arXiv preprint arXiv:2205.02054*, 2022.

Arian Hosseini, Ankit Vani, Dzmitry Bahdanau, Alessandro Sordoni, and Aaron C. Courville. On the compositional generalization gap of in-context learning. *ArXiv*, abs/2211.08473, 2022.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. 2022.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 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)*, pp. 963–973, Vancouver,

Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1089. URL https://aclanthology.org/P17-1089.

Naman Jain, Skanda Vaidyanath, Arun Shankar Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram K. Rajamani, and Rahul Sharma. Jigsaw: Large language models meet program synthesis. *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 1219–1231, 2021.

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)*, pp. 2261–2273, Online, August 2021a. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-long.176.

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)*, pp. 2261–2273, Online, August 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.176. URL https://aclanthology.org/2021.acl-long.176.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. *arXiv preprint arXiv:2106.11455*, 2021c.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Haiquan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. 2022.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *ArXiv*, abs/2201.11227, 2022a.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*, 2022b.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *ArXiv*, abs/2205.06983, 2022.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. Evaluating the impact of model scale for compositional generalization in semantic parsing. *ArXiv*, abs/2205.12253, 2022.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *ArXiv*, abs/2204.00498, 2022a.

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

Adam Roberts, Colin Raffel, Katherine Lee, Michael Matena, Noam Shazeer, Peter J. Liu, Sharan Narang, Wei Li, and Yanqi Zhou. Exploring the limits of transfer learning with a unified text-to-text transformer. Technical report, Google, 2019.

Ohad Rubin and Jonathan Berant. Smbop: Semi-autoregressive bottom-up semantic parsing. *arXiv preprint arXiv:2010.12412*, 2020.

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

Richard Shin and Benjamin Van Durme. Few-shot semantic parsing with language models trained on code. *ArXiv*, abs/2112.08696, 2021.

Richard Shin, C. H. Lin, Sam Thomson, Charles C. Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jas' Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. *ArXiv*, abs/2104.08768, 2021.

Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, 2001.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 49–60, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.5. URL https://aclanthology.org/2022.findings-naacl.5.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018a.

Tao Yu, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z 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 *Conference on Empirical Methods in Natural Language Processing*, 2018b.

John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*, 1996.