How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot, Single-domain, and Cross-domain Settings

Shuaichen Chang and **Eric Fosler-Lussier**
The Ohio State University
{chang.1692, fosler-lussier.1}@osu.edu

Abstract

Large language models (LLMs) with in-context learning have demonstrated remarkable capability in the text-to-SQL task. Previous research has prompted LLMs with various demonstration-retrieval strategies and intermediate reasoning steps to enhance the performance of LLMs. However, those works often employ varied strategies when constructing the prompt text for text-to-SQL inputs, such as databases and demonstration examples. This leads to a lack of comparability in both the prompt constructions and their primary contributions. Furthermore, selecting an effective prompt construction has emerged as a persistent problem for future research. To address this limitation, we comprehensively investigate the impact of prompt constructions across various settings and provide insights into prompt constructions for future text-to-SQL studies.

1 Introduction

Text-to-SQL models enable users to query databases using natural language questions (NLQs) without developing the underlying SQL query. Over the past few decades, neural models with supervised learning have achieved impressive performance on the text-to-SQL task, which are trained on a large training set and then evaluated on test examples [30, 31, 25, 27, 13, 16].

Recently, large language models (LLMs) have excelled in various language understanding and generation tasks through in-context learning [2, 5, 9], including on the text-to-SQL task [24, 4, 19]. Instead of extensive training on text-to-SQL data, in-context learning allows LLMs to convert a test NLQ into a SQL query using a prompt text. This prompt text includes essential components such as the test database and question. These are accompanied by zero or a few demonstrations: NLQ-SQL pairs corresponding to either the test database (single-domain) or different databases (cross-domain). Figure 1 provides the prompt of an example in the Spider dataset [32] for single-domain text-to-SQL.

Previous research has augmented the text-to-SQL capability of LLMs with demonstration-retrieval strategies [20, 29], intermediate reasoning steps [8, 6, 21], and self-debugging ability [6, 21]. However, those studies often employ different prompt strategies that include various key components of text-to-SQL: database schema and content, and demonstration examples. The difference in prompt constructions makes it difficult to directly compare two studies on their main contribution Moreover, the creation of prompt text for structured databases remains a challenging issue for future research due to the absence of a comprehensive study on the matter.

In this paper, we evaluate various strategies for prompt construction in three commonly employed text-to-SQL settings: zero-shot, single-domain, and cross-domain. We assess LLMs on text-to-SQL, considering various database prompt constructions in all three settings. Additionally, in the cross-domain scenario, we investigate the strategy for constructing demonstrations. Through our evaluation, we aim to gain insights into the effectiveness of these prompt construction strategies. Our findings can be summarized as follows:

Table Representation Learning Workshop at NeurIPS 2023.

Database
```
CREATE TABLE Highschooler (
ID int primary key,
name text,
grade int
);
/*
3 example rows:
SELECT * FROM Highschooler LIMIT 3;
ID    name    grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/
```
Task Instruction
```
-- Using valid SQLite, answer the following questions for the tables provided above.
```
Demonstration
```
Question: What is Kyle's id?
SELECT ID FROM Highschooler WHERE name = "Kyle";
```
Test Question
```
Question: How many high schoolers are there?
SELECT
```

Figure 1: A prompt example of 1-shot single-domain text-to-SQL for a snippet of the database `Network_1` and a question in Spider [32].

Table(Columns) [19]
```
Highschooler(ID, name, grade);
Friend(student_id, friend_id);
```
Columns=[] [21]
```
Table Highschooler, Columns = [ID, name, grade];
Table Friend, Columns = [student_id, friend_id];
```
+ForeignKey [21]
```
Foreign_keys = [Friend.student_id = Highschooler.ID, Friend.friend_id = Highschooler.ID];
```
CreateTable [24]
```
CREATE TABLE Highschooler (
ID int primary key,
name text,
grade int
);
CREATE TABLE Friend (
student_id int,
friend_id int,
primary key (student_id,friend_id),
foreign key(student_id) references Highschooler(ID),
foreign key (friend_id) references Highschooler(ID)
);
```

Figure 2: Examples of the different database schema constructions for a snippet of database `Network_1` in Spider.

- Table relationship and table content play a crucial role in effectively prompting LLMs. However, it is essential to carefully consider their representation in the prompt, as LLMs are sensitive to the specific presentation in the zero-shot and cross-domain settings.
- In-domain demonstration examples can mitigate LLMs' sensitivity to different representations of database knowledge but they cannot replace table content knowledge.
- The length of the prompt has a significant impact on the LLMs' performance in the cross-domain setting. We discovered a preferred prompt length that leads to improved performance.

2 In-context Learning for Text-to-SQL

In the text-to-SQL task, a database and a natural language question (NLQ) are provided as input for generating an output SQL query. In-context learning allows pretrained large language models (LLMs) to perform text-to-SQL by providing either zero or a few training examples (NLQ-SQL pairs) as demonstrations. This section introduces three widely used settings for in-context learning in text-to-SQL. Prompt examples in these settings can be found in Appendix 8.1.

Zero-shot Text-to-SQL This setting evaluates the text-to-SQL capability of pretrained LLMs to directly infer the NLQ-SQL relationship from a table without any demonstration examples. The input includes a task instruction and a test question with its corresponding database. Zero-shot text-to-SQL is used to directly assess the text-to-SQL capability of LLMs [24, 4, 19].

Single-domain Few-shot Text-to-SQL This setting is designed for applications or domains where it is easy to construct examples, such as booking flights [22, 10] and querying geographic information [33]. It tests the ability of LLMs to adapt with a few in-domain demonstration examples, which are collected from the same database as the test question. The goal is to evaluate how well the LLMs can perform text-to-SQL with minimal in-domain training data [24].

Cross-domain Few-shot Text-to-SQL This setting evaluates the generalization capability of models to new domains by learning from out-of-domain demonstrations. In this scenario, the demonstration NLQ-SQL pairs correspond to one or multiple demonstration databases that are different from the test database. Cross-domain few-shot text-to-SQL assesses how well LLMs can apply their learned knowledge from demonstrations to new databases [20, 6].

InsertRow [6]

```
INSERT INTO Highschooler (ID, name, grade)
VALUES (1510, "Jordan", 9);
INSERT INTO Highschooler (ID, name, grade)
VALUES (1689, "Gabriel", 9);
INSERT INTO Highschooler (ID, name, grade)
VALUES (1381, "Tiffany", 9);
```

SelectRow [24]

```
/*
3 example rows:
SELECT * FROM Highschooler LIMIT 3;
ID    name    grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/
```

SelectCol (Ours)

```
/*
Columns in Highschooler and 3 distinct
examples in each column:
ID: 1025, 1101, 1247
name: "Jordan", "Gabriel", "Tiffany"
grade: 9, 10, 11
*/
```

Figure 3: Examples of the different database content constructions for the `Highschool` table.

Unnormalized database and SQL

```
-- Database Schema

CREATE TABLE Highschooler(
        ID int primary key,
        name text,
        grade int);



-- SQL Query

SELECT count( * ) FROM Highschooler WHERE
Name = "Kyle";
```

Normalized database and SQL

```
-- Database Schema

create table highschooler (
id int primary key,
name text,
grade int
);



-- SQL Query

select count(*) from highschooler where name
= 'Kyle';
```

Figure 4: An example of the normalization for database and SQL prompts.

3 Prompt Construction

A text-to-SQL prompt typically comprises four components: a task instruction, a test database, a test NLQ, and optional demonstrations, as illustrated in Figure 1. While the task instruction and test NLQ are easily presented in natural language, there are various strategies for representing the databases and incorporating demonstrations. In this section, we explore different prompt constructions for databases and demonstrations.

3.1 Database Prompt

A relational database consists of the database schema and database content. The database schema encompasses the schemas (headers) of tables and the relationship among tables, and database content refers to the data stored in the tables.

Database Schema Figure 2 illustrates various prompt constructions for the database schema that have been utilized in previous studies: (1) `Table(Columns)` [19] lists each table along with its columns inside parentheses to represent the table schemas; (2) `Columns=[]` [21] represents each table along with a list of its columns using an equation-like notation; (3) `+ForeignKey` [21] further adds foreign keys to indicate the relationships between tables; (4) `CreateTable` [24] employed the "`Create Table`" statement to display the table schemas and relationships.

To ensure consistency in the prompt text and accommodate the case-insensitivity of SQL keywords and the database schema, we unify the space and line break in the prompt text and convert all words to lowercase, except for the database content. This normalization process helps to standardize the prompt text. An example is shown in Figure 4.

Database content Previous research shows that being aware of database content can improve model performance by exposing models to the specific format of values in each column [30, 18, 24]. For instance, the phrase "American student" could be converted to "`WHERE country = 'USA'`" or "`WHERE country = 'The United States of America'`" depending on the contents of the `country` column.

Figure 3 summarizes different approaches for showcasing the content of a database. (1) `InsertRow` [6]: This method displays R rows of each table by utilizing R "`INSERT INTO`" statements. (2) `SelectRow` [24]: This approach employs the "`SELECT * FROM Table LIMIT R`" query to display the first R rows of each table. (3) `SelectCol`: Instead of presenting table content in a row-wise manner, an alternative method is to use a column-wise format. As there may be duplicated content across different rows, presenting the content column-wise ensures the provision of distinct values within each column to expose LLMs to a broader range of content. We propose using the query "`SELECT DISTINCT [Column] FROM [Table] LIMIT R`" to list R distinct cell values in each column.

3.2 Demonstration Prompt

In few-shot settings, LLMs are provided with demonstrations within the prompt text. In the single-domain few-shot setting, we incorporate a few NLQ-SQL pairs as demonstrations inserted between the test database and question, following previous work [24]. In the cross-domain few-shot setting, we use both out-of-domain NLQ-SQL pairs (demonstration examples) and corresponding databases (demonstration databases) placed before the test database and question. Prior research in the N-shot setting either uses one demonstration database with N examples [21] or employs N demonstration databases, each with a single NLQ-SQL pair [20, 6]. In contrast, we consider a more general scenario where the demonstrations comprise M databases, each with K NLQ-SQL pairs, with $M \times K = N$. We list the examples of 4-shot single-domain and cross-domain demonstrations in Appendix 8.1.

Additionally, we normalize demonstration SQL queries by first parsing the SQL queries and unifying their format, such as using lowercase for SQL keywords and database schema and unifying the space around punctuation. Figure 4 provides an example of SQL normalization.

4 Experiments

Data & Evaluation For our experiments, we utilize the Spider dataset [32], a cross-domain benchmark for the text-to-SQL task. We conduct our experiments on the development set of Spider (Spider-dev) as the test set is not publicly available. Spider-dev consists of 20 databases with 1034 pairs of NLQ and SQL in total. We evaluate models with execution accuracy (EX) which compares the execution results of a predicted SQL and a gold SQL.

In the cross-domain setting, we use the training set of Spider to select demonstrations. As a few databases contain long schema that may cause the prompt to exceed the token limits of LLMs, we only use the databases with fewer than 1000 tokens when constructing the `CreateTable` prompt. This results in a total of 130 databases being used as demonstration databases.

Models We used GPT-3 Codex [5] and ChatGPT due to their demonstrated performance and prevalence in the field.[1]

Experiment Setup For the zero-shot setting, we construct each prompt text with a task instruction, a test database, and a test question. We include $R = 3$ table rows in the database prompt, which has been discovered as the optimal number in previous work [24]. For the few-shot settings, we incorporate N demonstration examples in addition to the zero-shot prompt text.

In the single-domain scenario, we use a leave-one-out split, as some databases in Spider-dev contain a small number of examples. When evaluating one example, we regard all other examples from the same database as the training set and randomly retrieve N examples from them. Since Spider contains multiple NLQs corresponding to the same SQL query, we avoid having the same SQL template in both the training and test sets, again following previous work [12].

In the cross-domain scenario, we randomly select M demonstration databases, each with K NLQ-SQL pairs ($M \times K = N$) from the Spider training set. Incorporating multiple demonstration databases in a prompt text significantly increases its length. Hence, we use Codex and ChatGPT-16K for the cross-domain experiments, which have a context length limit of 8K, and 16K, respectively. In both single-domain and cross-domain settings, we compare different prompt construction methods using the same few-shot examples to make a fair comparison. We repeat our experiments three times and present the average results.

5 Results

In this section, we present our empirical findings in the areas of zero-shot, single-domain, and cross-domain text-to-SQL. Through our experiments, we aim to answer a few crucial research questions in each setting and provide insightful strategies for future studies on effective prompting.

[1]We employ the Code-davinci-002 version of Codex across all settings. In zero-shot and single-domain setups, we utilize the `gpt-3.5-turbo-0301` version of ChatGPT. For cross-domain experiments involving ChatGPT-16K, we turned to `gpt-3.5-turbo-16k-0613` due to its extended context length.

		Codex		ChatGPT	
Database Prompt Construction		# Tokens (U|N)	EX (U|N)	# Tokens (U|N)	EX (U|N)
`Table Schema`	`Table(Columns)`	148 | 147	69.0 | 71.9	118 | 115	68.8 | 70.5
	`Columns=[]`	169 | 167	70.2 | 71.8	137 | 135	68.3 | 69.1
`+Relationship`	`Columns=[]+ForeignKey`	226 | 223	72.3 | 73.1	178 | 174	72.9 | 71.2
	`CreateTable`	474 | 356	71.8 | 73.1	339 | 254	70.7 | 71.7
`+Relationship+Content`	`CreateTable+InsertRow 3`	1089 | 1013	70.9 | 71.9	964 | 872	71.8 | 71.8
	`CreateTable+SelectRow 3`	820 | 770	73.3 | 74.1	761 | 674	71.8 | 72.1
	`CreateTable+SelectCol 3`	958 | 831	**75.0** | **75.7**	799 | 712	**73.3** | **73.6**

Table 1: Zero-shot results of Codex and ChatGPT using different database prompt constructions. `Table Schema` (upper part) contains prompts that solely include the schema of tables, while `+Relationship` (middle part) incorporates foreign keys as the table relationships and `+Relationship+Content` (lower part) adds table content as well. # Tokens is the average token counts in the prompts and EX represents the execution accuracy of SQLs. U|N represents the results of unnormalized prompts and normalized prompts, respectively. The underlines highlight the lower number of tokens and higher accuracies when comparing unnormalized and normalized prompts and the highest accuracy achieved among all prompts is highlighted in bold.

5.1 Zero-shot Text-to-SQL

In the zero-shot setting, we focus on comparing different prompt constructions for databases. Table 1 shows the average prompt length and execution accuracy of Codex and ChatGPT using various database prompt constructions.

Q1: How does normalized database prompt perform compared to unnormalized ones? Normalized schemas are found to have a reduced token count in comparison to unnormalized schemas across all database constructions. The normalization also tends to yield slightly better performance. As for Codex, normalized schemas show improvement in all prompts. For ChatGPT, normalized schemas either improve accuracy or achieve the same accuracy or achieve the same level of accuracy as unnormalized schemas in 6 out of 7 schema constructions. The tests of statistical significance are presented in Appendix 8.2.

Q2: What database knowledge is crucial for effectively prompting LLMs? Our experiments indicate that table relationships and content are important. The `Columns=[]` prompt includes only the table schema, while the `Columns=[]+ForeignKey` prompt contains the additional relationship among tables shown as foreign keys. Including such information improves the performance of both Codex (71.8 -> 73.1) and ChatGPT (69.1 -> 71.2). Moveover, exposing LLMs to database content with the `SelectRow` and `SelectCol` prompts further enhances the performance of both Codex and ChatGPT, while the `InsertRow` prompt does not seem to be beneficial. We believe that database content is valuable, but its representation needs to be carefully chosen.

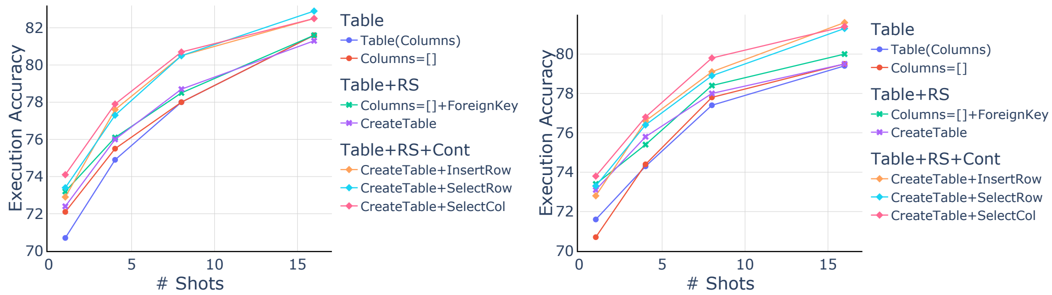
Q3: How does Codex perform compared to ChatGPT? While we do not focus on comparing different LLMs on the text-to-SQL tasks in this paper, it is worth noting that Codex consistently outperforms ChatGPT on zero-shot text-to-SQL using various prompt constructions.

Based on all the findings above, we would recommend using Codex in conjunction with normalized `CreateTableSelectCol` prompt construction for zero-shot text-to-SQL.[2]

5.2 Single-domain Text-to-SQL

In the zero-shot text-to-SQL setting, we discovered that the prompt constructions of databases impact the performance of LLMs. This discovery naturally raises the question of whether the introduction of in-domain demonstrations affects the performance of LLMs to different database prompts.

[2]To simplify our experiments and ensure consistent prompts, we adopt normalization for single-domain and cross-domain experiments.



(a) Codex

(b) ChatGPT

Figure 5: Execution accuracy of Codex and ChatGPT for single-domain text-to-SQL with 1, 4, 8, and 16 in-domain examples. `RS` and `Cont` correspond to table relationship and table content, respectively. Detailed results can be found in Table 3 and 4.

Q1: Does the use of in-domain demonstrations enhance LLM's performance? Figure 5 depicts the performance of Codex and ChatGPT using different database prompt constructions with respect to different numbers of in-domain demonstration examples. For all database prompts, the performance of LLMs experiences a notable improvement when in-domain examples are presented. Furthermore, the performance continues to enhance as the number of in-domain examples increases.

Q2: What database knowledge is important when presenting in-domain demonstrations? While we have observed that the presence of table relationships and table content enhanced LLMs' performance in the zero-shot scenario, it is not clear whether they are still important in the single-domain setting. A hypothesis is that table relationship and table content knowledge can be acquired from in-domain examples as they may appear in SQL clauses `JOIN` and `WHERE`.

For table relationships, we compare two database prompt constructions `Columns=[]` and `Columns=[]+ForeignKey`. Both construct the table schema in the same way while the latter includes foreign keys as table relationships. In the zero-shot scenario, `Columns=[]+ForeignKey` outperforms `Columns=[]` by 1.3 and 2.1 for Codex and ChatGPT, respectively. However, as increasing the number of in-domain examples, we notice a gradual reduction in the performance gap between these two prompts. With the utilization of 16 in-domain examples, the gap completely disappears for Codex, while ChatGPT exhibits a marginal difference of only 0.5%.

For table content, we compare `CreateTable` with `CreateTable+SelectCol`. Both contain the same prompts for presenting the table schema and relationship, while the latter additionally includes table content. In the zero-shot scenario, `CreateTable+SelectCol` outperforms `CreateTable` by 2.0% for Codex and 1.7% for ChatGPT. As we proceed to increase the number of in-domain examples, we observe that the performance gap between these two prompts does not exhibit a significant reduction. Even with 16 in-domain examples, the gap still persists at 1.3 for Codex and 1.9 for ChatGPT.

These results indicate LLMs are able to quickly learn table relationships from a small number of in-domain demonstrations, however, it is more challenging to obtain table content knowledge from demonstration examples. Consequently, the inclusion of table content remains crucial for achieving satisfactory performance in the single-domain text-to-SQL scenario.

Q3: Can in-domain demonstrations alleviate the sensitivity of LLMs to the representation of table content? In the zero-shot setting, we observe that LLMs are sensitive to how the table content is presented. Specifically, `SelectCol 3` outperforms `InsertRow 3` by a substantial margin of 3.8 for Codex and 1.8 for ChatGPT. However, as we expose LLMs to in-domain demonstrations, LLMs become less sensitive to the specific representation of table content. The performance disparities among the three table content prompts become marginal. Notably, with only 4 examples, the performance difference between `SelectCol 3` and `InsertRow 3` diminishes to 0.3 for Codex and 0.2 for ChatGPT.

To summarize, in single-domain text-to-SQL, we recommend incorporating a greater number of in-domain examples whenever feasible. It is also essential to ensure the presence of table content in conjunction with the table schema while the specific choice of table content construction is less crucial compared to the zero-shot scenario.

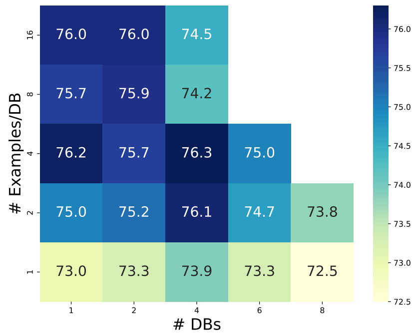


Figure 6: A heat map of Codex's execution accuracy using `CreateTable+SelectRow 3` for different numbers of databases and examples per database in the demonstration. Darker color indicates higher accuracy.

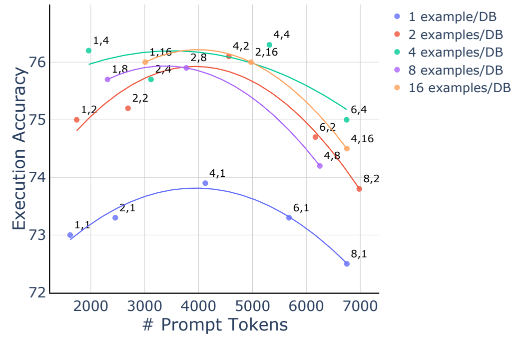


Figure 7: Execution accuracy of Codex in relation to the length of prompts. Each dot represents a demonstration construction, with the m, k denoting the number of databases and examples per database. The lines represent second-degree polynomial trendlines fitted to the results.

5.3 Cross-domain Text-to-SQL

In this section, we present the results to answer a series of questions regarding the demonstration and database prompt construction.

5.3.1 Impact of Demonstration Prompt

To investigate the impact of the number of databases and examples per database in demonstrations, we conduct experiments encompassing various combinations. Specifically, our demonstrations are composed of M demonstration databases, each containing K NLQ-SQL pairs. We consider scenarios with up to 8 databases and 16 examples per database for Codex as long as the combination does not exceed the prompt length limit. We opt to use the database prompt `CreateTable+SelectRow 3` as it contains fewer tokens compared to `InsertRow` and `SelectCol` while encompassing all valuable database knowledge. Due to the paper space constraints, we present the experiments involving ChatGPT-16K in Appendix 8.4.1 which show similar results as Codex.

Q1: Does increasing demonstration examples enhance LLMs' performance? Figure 6 presents the accuracy of Codex corresponding to different combinations of the number of databases and the number of examples per database used as demonstrations. We analyze the results from two perspectives. Firstly, for a fixed number of databases, we observe an initial improvement in Codex's performance as the number of examples per database increases. However, this improvement plateaus or declines once 4 examples per database are provided. Surprisingly, when using 4 databases, employing 8 or 16 examples per database leads to a significant decrease in the Codex's performance compared to using 2 or 4 examples per database. Secondly, for a fixed number of examples per database, we observe an initial increase in Codex's performance as the number of databases increases, however, this improvement is followed by a significant decrease once the number of databases reaches a certain threshold (either 4 or 6).

Q2: Why does increasing the number of databases decrease LLMs' performance? As depicted in Figure 6, presenting more databases does not always lead to improved performance. In fact, there is a significant decline in performance, once it surpasses a threshold. We hypothesize that this phenomenon is attributed to the length of the prompt text. To test this hypothesis, we analyze the results in relation to the prompt length.

Figure 7 shows the relationship between the accuracy of different demonstration prompts and their prompt lengths. Notably, the performance of Codex exhibits an inverted-U shape as the prompt length increases for each number of examples per database. Additionally, we observe a substantial drop in performance once the prompt text length exceeds approximately 5500 tokens. Similarly, Figure 9 shows that the performance of ChatGPT-16K starts to decrease when prompt text length exceeds 11K tokens. Based on these observations, we conjecture that LLMs may have a sweet spot in terms of prompt length, potentially influenced by factors such as their model architecture or training data. This indicates that even though LLMs are capable of handling long contexts, they may not necessarily perform better with excessively long prompts.

5.3.2 Impact of Database Prompt

Since incorporating demonstration databases may cause a decrease in Codex's performance, we focus our database prompt experiments on using one demonstration database in combination with varying quantities of demonstration examples. Table 5 presents the execution accuracy of Codex using different database prompts.

Q3: Do different database prompts show similar trends with the number of demonstration examples? We observe an initial performance increase for all database prompts. However, once more than 4 examples are provided, the improvement starts to level off, indicating that the different database prompts exhibit similar trends in relation to the number of demonstration examples.

Q4: Can out-of-domain demonstrations alleviate the sensitivity of LLMs to database prompts? First, we observe that table relationships and content in the prompts remain crucial for effectively prompting Codex in the cross-domain setting. This is not surprising, as Codex cannot directly learn knowledge specific to the test database from the out-of-domain demonstrations. Furthermore, we find that Codex continues to exhibit sensitivity to the representation of table content. Despite having demonstration databases that mirror the construction of the test database, Codex still displays a preference for`SelectRow` and `SelectCol` when presenting table content, compared to `InsertCol`.

In conclusion, while out-of-domain demonstrations enhance LLMs' capabilities in text-to-SQL, they do not provide database-specific knowledge. Consequently, careful construction of database prompts remains crucial, aligning with the observations made in the zero-shot setting.

6 Related Work

LLMs for Text-to-SQL In recent years, there has been significant progress in leveraging LLMs for the text-to-SQL task. Various methods have been proposed to enhance the capabilities of LLMs. For example, demonstration retrieval with similarity and diversity has demonstrated effectiveness in the cross-domain setting [26, 20, 15]. Furthermore, intermediate reasoning steps and self-correction have been incorporated to improve LLMs in text-to-SQL [21, 6].

In contrast to these approaches, our focus lies in conducting a comprehensive evaluation of prompt representations across different text-to-SQL settings. While there are similar motivations to the work by Rajkumar et al. [24], which analyzes the performance of CodeX on Spider for the zero-shot setting and on two databases for the single-domain setting, we aim to provide more general findings by evaluating across a wider range of databases and considering all three text-to-SQL settings.

Table Representation Encoding structured databases with neural models has been a persistent challenge. To encode database schema, graph neural networks are utilized to represent the relationships among tables [1, 7]. Alternatively, other studies [14, 18, 28] have converted table schemas into a sequence to effectively leverage pretrained language models, such as BERT [11] and T5 [23]. In such cases, table relationships can be encoded as meta-data features [18] or used as a guide for attention mechanism [30, 3, 17].

To incorporate table content into neural models, prior supervised methods provide question-specific table content by identifying the relevant table content mentioned in the question through string matching [18, 28]. However, Chang et al. [4] have revealed the vulnerability of string matching to perturbations. Given that LLMs with in-context learning support longer input sequences compared to supervised methods, we follow previous work to provide table content without explicitly considering the questions [24, 6].

7 Conclusions

In this paper, we investigate effective prompting strategies in the text-to-SQL task. We thoroughly compare various prompt construction strategies for databases and demonstrations in the zero-shot, single-domain, and cross-domain text-to-SQL. Through our investigation, we uncover the critical database knowledge and optimal representations for effective prompting. Additionally, an interesting finding is the existence of a sweet spot in terms of prompt length for Codex in the cross-domain setting. Overall, we believe that our findings will provide valuable guidance for future research in the field of text-to-SQL with LLMs.

References

[1] Ben Bogin, Matt Gardner, and Jonathan Berant. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*, 2019.

[2] 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, 2020.

[3] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*, 2021.

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

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[7] Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. Shadowgnn: Graph projection neural network for text-to-sql parser. *arXiv preprint arXiv:2104.04689*, 2021.

[8] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.

[9] 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.

[10] Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, pages 43–48, 1994.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*, 2018.

[13] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. Natural sql: Making sql easier to infer from natural language specifications. *arXiv preprint arXiv:2109.05153*, 2021.

[14] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[15] Itay Levy, Ben Bogin, and Jonathan Berant. Diverse demonstrations improve in-context compositional generalization. *arXiv preprint arXiv:2212.06800*, 2022.

[16] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Decoupling the skeleton parsing and schema linking for text-to-sql. *arXiv preprint arXiv:2302.05965*, 2023.

[17] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *arXiv preprint arXiv:2301.07507*, 2023.

[18] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 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, 2020.

[19] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*, 2023.

[20] 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*, 2022.

[21] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*, 2023.

[22] Patti Price. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

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

[25] Ohad Rubin and Jonathan Berant. Smbop: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, 2021.

[26] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*, 2021.

[27] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 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, 2021.

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

[29] Peng Shi, Rui Zhang, He Bai, and Jimmy Lin. Xricl: Cross-lingual retrieval-augmented in-context learning for cross-lingual text-to-sql semantic parsing. *arXiv preprint arXiv:2210.13693*, 2022.

[30] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

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

[32] 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*, 2018.

[33] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055, 1996.

8 Appendix

8.1 Prompt Examples

Below contains an example of a zero-shot normalized prompt, which contains the database `Network_1` from Spider [32], a task instruction "Using valid SQLite, answer the following questions for the tables provided above.", and a test question "How many high schoolers are there?".

Zero-shot normalized prompt

```
create table highschooler (
id int primary key,
name text,
grade int
);
/*
3 example rows:
select * from highschooler limit 3;
id    name    grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/

create table friend (
student_id int,
friend_id int,
primary key (student_id,friend_id),
foreign key(student_id) references highschooler(id),
foreign key (friend_id) references highschooler(id)
);
/*
3 example rows:
select * from friend limit 3;
student_id    friend_id
1510    1381
1510    1689
1689    1709
*/

create table likes (
student_id int,
liked_id int,
primary key (student_id, liked_id),
foreign key (liked_id) references highschooler(id),
foreign key (student_id) references highschooler(id)
);
/*
3 example rows:
select * from likes limit 3;
student_id    liked_id
1689    1709
1709    1689
1782    1709
*/

-- Using valid SQLite, answer the following questions for the tables provided above.

Question: How many high schoolers are there?
select
```

Below contains an example of a 4-shot single-domain normalized prompt, which contains a database prompt and 4 demonstration examples ahead of the test question.

4-shot single-domain normalized prompt

```
create table highschooler (
id int primary key,
name text,
grade int
);
/*
3 example rows:
select * from highschooler limit 3;
id    name    grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/

create table friend (
student_id int,
friend_id int,
primary key (student_id,friend_id),
foreign key(student_id) references highschooler(id),
foreign key (friend_id) references highschooler(id)
);
/*
3 example rows:
select * from friend limit 3;
student_id    friend_id
1510    1381
1510    1689
1689    1709
*/

create table likes (
student_id int,
liked_id int,
primary key (student_id, liked_id),
foreign key (liked_id) references highschooler(id),
foreign key (student_id) references highschooler(id)
);
/*
3 example rows:
select * from likes limit 3;
student_id    liked_id
1689    1709
1709    1689
1782    1709
*/

-- Using valid SQLite, answer the following questions for the tables provided above.

Question: What is Kyle's id?
select id from highschooler where name = 'Kyle';
Question: Return the names of friends of the high school student Kyle.
select t3.name from friend as t1 join highschooler as t2 on t1.student_id = t2.id join
highschooler as t3 on t1.friend_id = t3.id where t2.name = 'Kyle';
Question: Show names of all high school students who do not have any friends.
select name from highschooler except select t2.name from friend as t1 join highschooler as t2
 on t1.student_id = t2.id;
Question: What are the names and grades for each high schooler?
select name, grade from highschooler;
Question: How many high schoolers are there?
select
```

Below contains an example of a 4-shot cross-domain prompt, which contains 2 demonstration databases, each with 2 demonstration examples ahead of the test database and question.

4-shot cross-domain prompt

```
create table publication (
publication_id int,
book_id int,
publisher text,
publication_date text,
price real,
primary key (publication_id),
foreign key (book_id) references book(book_id)
);
/*
3 example rows:
select * from publication limit 3;
publication_id    book_id    publisher    publication_date    price
1    1    Pearson    August 2008    15000000.0
2    3    Thomson Reuters    March 2008    6000000.0
3    4    Wiley    June 2006    4100000.0
*/

create table book (
book_id int,
title text,
issues real,
writer text,
primary key (book_id)
);
/*
3 example rows:
select * from book limit 3;
book_id    title    issues    writer
1    The Black Lamb    6.0    Timothy Truman
2    Bloody Mary    4.0    Garth Ennis
3    Bloody Mary : Lady Liberty    4.0    Garth Ennis
*/

-- Using valid SQLite, answer the following questions for the tables provided above.

Question: List the writers of the books in ascending alphabetical order.
select writer from book order by writer asc;
Question: How many books are there?
select count(*) from book;


create table race (
race_id int,
name text,
class text,
date text,
track_id text,
primary key (race_id),
foreign key (track_id) references track(track_id)
);
/*
3 example rows:
select * from race limit 3;
race_id    name    class    date    track_id
1    Rolex 24 At Daytona    DP/GT    January 26 January 27    1
2    Gainsco Grand Prix of Miami    DP/GT    March 29    2
3    Mexico City 250    DP/GT    April 19    2
*/

create table track (
track_id int,
name text,
location text,
seating real,
year_opened real,
primary key (track_id)
);
/*
3 example rows:
select * from track limit 3;
track_id    name    location    seating    year_opened
1    Auto Club Speedway    Fontana, CA    92000.0    1997.0
2    Chicagoland Speedway    Joliet, IL    75000.0    2001.0
3    Darlington Raceway    Darlington, SC    63000.0    1950.0
*/

-- Using valid SQLite, answer the following questions for the tables provided above.
Question: Show the name and location for all tracks.
select name, location from the track;
Question: Show the name of track and the number of races in each track.
```

```
select t2.name, count(*) from race as t1 join track as t2 on t1.track_id = t2.track_id group
by t1.track_id;


create table highschooler (
id int primary key,
name text,
grade int
);
/*
3 example rows:
select * from highschooler limit 3;
id    name    grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/

create table friend (
student_id int,
friend_id int,
primary key (student_id,friend_id),
foreign key(student_id) references highschooler(id),
foreign key (friend_id) references highschooler(id)
);
/*
3 example rows:
select * from friend limit 3;
student_id    friend_id
1510    1381
1510    1689
1689    1709
*/

create table likes (
student_id int,
liked_id int,
primary key (student_id, liked_id),
foreign key (liked_id) references highschooler(id),
foreign key (student_id) references highschooler(id)
);
/*
3 example rows:
select * from likes limit 3;
student_id    liked_id
1689    1709
1709    1689
1782    1709
*/

-- Using valid SQLite, answer the following questions for the tables provided above.

Question: How many high schoolers are there?
select
```

8.2 Tests of Significance

Table 1 contains the performance of Codex and ChatGPT using different database prompt constructions in the zero-shot setting. We observe that the normalization results in slightly improved performance for all database prompt constructions with Codex and 6 out of 7 database prompt constructions with ChatGPT. It is important to note, however, that when comparing normalized and unnormalized database prompt constructions using the same method, the results did not demonstrate statistical significance in McNemar's test, with p-values greater than 0.05. Nevertheless, the primary advantage of normalization lies in its ability to reduce variations among different databases and minimize the overall prompt length.

When evaluating various prompt constructions, we note the advantages gained from incorporating both table relationships (`Columns=[]+ForeignKey` vs `Columns=[]`) and table content (`CreateTable+SelectCol 3` vs `CreateTable`) are mostly statistically significant in McNemar's test, with p-values smaller than 0.05. Table 2 displays the results of the significant tests. The performance of `Columns=[]+ForeignKey` compared to `Columns=[]` is statistically significant in all cases, except for codex with normalized prompts. Likewise, the performance of `CreateTable+SelectCol 3` is statistically significant for both Codex and ChatGPT, with both normalized and unnormalized prompts, when compared to `CreateTable`. These significant findings highlight the effectiveness of incorporating table relationships and database content.

Prompt 1	Prompt 2	LLM	Normalization	Significant Test
Columns=[]	Columns=[]+ForeignKey	Codex	U	✓
		Codex	N	✗
		ChatGPT	U	✓
		ChatGPT	N	✓
CreateTable	CreateTable+SelectCol 3	Codex	U	✓
		Codex	N	✓
		ChatGPT	U	✓
		ChatGPT	N	✓

Table 2: Tests of Statistical Significance for comparing different prompt constructions. Prompt 1 and Prompt 2 were used to represent two distinct methods of constructing prompts in McNemar's test. The prompts were categorized as U and N, representing unnormalized and normalized database prompts, respectively. The ✓ symbol indicates that the p-value is smaller than 0.05, indicating statistical significance, while the ✗ symbol indicates p-values greater than 0.05, indicating a lack of statistical significance.

8.3 Detailed Single-domain Results

Tables 3 and 4 provide detailed results of Codex and ChatGPT in the single-domain setting, respectively. The performance of both models is also illustrated in Figure 5.

Database Prompt Construction		0-shot	1-shot	4-shot	8-shot	16-shot
Table Schema	Table(Columns)	71.9	70.7	74.9	78.0	81.6
	Columns=[]	71.8	72.1	75.5	78.0	81.6
+Relationship	Columns=[]+ForeignKey	73.1	73.2	76.1	78.5	81.6
	CreateTable	73.1	72.4	76.0	78.7	81.3
+Relationship+Content	CreateTable+InsertRow 3	71.9	72.9	<u>77.6</u>	<u>80.5</u>	82.5
	CreateTable+SelectRow 3	<u>74.1</u>	<u>73.4</u>	77.3	<u>80.5</u>	**82.9**
	CreateTable+SelectCol 3	**75.7**	**74.1**	**77.9**	**80.7**	<u>82.5</u>

Table 3: Single-domain results of Codex using different prompt constructions for database schema and content. The best and second-best results for each shot are highlighted in bold and underlined.

Database Prompt Construction		0-shot	1-shot	4-shot	8-shot	16-shot
Table Schema	Table(Columns)	70.5	71.6	74.3	77.4	79.4
	Columns=[]	69.1	70.7	74.4	77.8	79.5
+Relationship	Columns=[]+ForeignKey	71.2	<u>73.4</u>	75.4	78.4	80.0
	CreateTable	71.7	73.1	75.8	78.0	79.5
+Relationship+Content	CreateTable+InsertRow 3	71.8	72.8	<u>76.6</u>	<u>79.1</u>	**81.6**
	CreateTable+SelectRow 3	<u>72.1</u>	73.3	76.4	78.9	81.3
	CreateTable+SelectCol 3	**73.6**	**73.8**	**76.8**	**79.8**	<u>81.4</u>

Table 4: Single-domain results of ChatGPT using different prompt constructions for database schema and content. The best and second-best results for each shot are highlighted in bold and underlined.

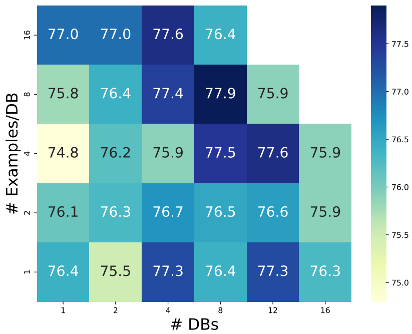


Figure 8: A heat map of ChatGPT-16K's execution accuracy using `CreateTable+SelectRow 3` for different numbers of databases and examples per database in the demonstration. Darker color indicates higher accuracy.

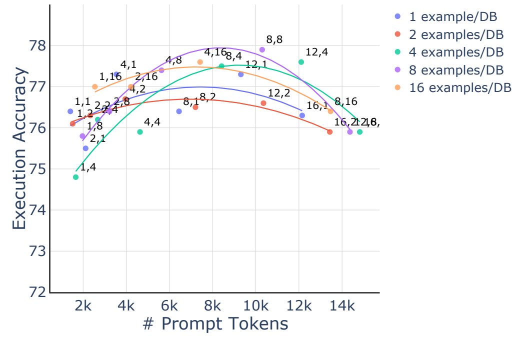


Figure 9: Execution accuracy of ChatGPT-16K in relation to the length of prompts. Each dot represents a demonstration construction, with the m, k denoting the number of databases and examples per database. The lines represent second-degree polynomial trendlines fitted to the results.

8.4 Detailed Cross-domain Results

8.4.1 Impact of Demonstration Prompt for ChatGPT-16K

Figure 8 presents the accuracy of ChatGPT-16K corresponding to different combinations of the number of databases and the number of examples per database used as demonstrations. Similar to our findings with Codex, presenting more databases does not always lead to improved performance for ChatGPT-16K. For a fixed number of examples per database, we observe an initial increase in its performance as the number of databases increases, however, this improvement is followed by a decrease once the number of databases reaches a certain threshold. To understand this phenomenon, we analyze the results in relation to the prompt length.

Figure 9 shows the relationship between the accuracy of different demonstration prompts and their prompt lengths. Similar to Codex, the performance of ChatGPT-16K also exhibits an inverted-U shape as the prompt length increases for each number of examples per database. Additionally, we observe the performance starts to decrease once the prompt text length exceeds approximately 11K tokens.

While Codex supports 8K tokens and ChatGPT-16K supports 16K tokens, we notice that their performance tends to decline when dealing with demonstrations that exceed approximately 70% of the maximum prompt length.

8.5 Impact of Database Prompt

Database Prompt Construction		0-shot	1-shot	2-shot	4-shot	8-shot	16-shot
`Table Schema`	`Table(Columns)`	71.9	72.0	73.0	73.2	72.8	73.9
	`Columns=[]`	71.8	71.9	73.6	74.2	73.7	74.4
`+Relationship`	`Columns=[]+ForeignKey`	73.1	<u>73.3</u>	74.5	74.9	74.9	75.2
	`CreateTable`	73.1	72.1	73.4	73.7	74.1	75.1
`+Relationship+Content`	`CreateTable+InsertRow 3`	71.9	72.2	74.1	74.9	74.9	74.8
	`CreateTable+SelectRow 3`	<u>74.1</u>	73.0	<u>75.0</u>	<u>76.2</u>	<u>75.7</u>	<u>76.0</u>
	`CreateTable+SelectCol 3`	**75.7**	**74.4**	**75.5**	**76.5**	**76.8**	**76.5**

Table 5: Cross-domain results of Codex using different database prompt constructions. Only one demonstration database is included in a prompt, N-shot represents N examples corresponding to the demonstration database. The best and second-best results for each shot are highlighted in bold and underlined.