Text-to-SQL tasks with small language models using recursive distillation

Chung-Yu Wang

York University, Toronto, Canada

Abstract. With the advancement of natural language processing, large language models have shown their capabilities on various natural language tasks. One of the natural language tasks is Text-to-SQL, which provides a database schema and a natural language question to the language model and asks the model to generate a format-correct and executable SQL query to the problem. However, existing methods of dealing with Text-to-SQL tasks have some drawbacks such as requiring human efforts in the training process and the computational cost of training large language models is not affordable for people. Therefore, the study proposes a framework to resolve those disadvantages by using small language models and recursive distillation.

Keywords: Text-to-SQL · SQL distillation · Recursive learning.

1 Introduction

1.1 Motivation

The rapid advancement of artificial intelligence has brought us to an era where machines can engage in conversations, understand context, and even generate code. In this context, large language models like ChatGPT, developed by OpenAI, have emerged as powerful tools capable of producing code snippets based on user prompts [1].

Given the adeptness demonstrated by language models in code generation, there is a growing interest among researchers and practitioners in exploring their capability to generate SQL and focusing on the exploration across various scenarios, including SQL-to-Text [2] and Text-to-SQL [3]. SQL is a powerful programming language commonly used for managing and manipulating relational databases. However, writing SQL queries requires a certain level of expertise, and many users, especially those who are not familiar with database systems, may find it challenging to formulate precise SQL commands [4].

Text-to-SQL tasks bridge this gap by enabling users to interact with databases using natural language queries instead of traditional SQL syntax. The goal is to develop machine learning models that can understand and interpret user-generated text and then convert it into SQL queries that can be executed against a database.

This task is essential for various applications, including natural language interfaces for databases, voice-activated database querying, and enhancing the accessibility of database systems for users with limited SQL knowledge. It sits at the intersection of natural language processing (NLP) and database management, presenting unique challenges such as understanding the semantics of user queries, handling ambiguity, and generating accurate and efficient SQL statements.

1.2 Contributions

To address these challenges, researchers and practitioners in the field of NLP and database systems have been actively working on developing advanced models and techniques. Due to the remarkable capabilities demonstrated by language models in the contemporary era, state-of-the-art models leverage large language models to learn the mapping between natural language and SQL representations. However, existing methods of dealing with Text-to-SQL tasks have some drawbacks such as requiring human efforts in the training process and the computational cost of training large language models is not affordable for people.

Regarding the drawbacks of existing approaches, our study has two main contributions:

- Proposed a framework has the potential to automate and streamline the training process, mitigating the need for extensive human intervention and concurrently diminishing computational costs associated with model training.
- Illustrated the implementation of utilizing critic metrics to filter out inferior-quality SQL queries.

2 Related Work Survey

2.1 Text-to-SQL tasks

Text-to-SQL task is defined as the transformation of a natural language (NL) question related to database items into its corresponding structured query language (SQL), which can subsequently be executed against a relational database. Technically, when provided with a NL question Q and the corresponding database schema S, the objective is to generate an SQL query Y that is both executable and correct, thereby producing the answer to Q [3].

2.2 Solutions of Text-to-SQL Tasks

Although various neural generation models have been developed to deal with the Text-to-SQL task, the state-of-the-art techniques leverage large language models (LLMs) to achieve better performance on the task [5] [6] [7] [8]. We categorize them into two main approaches for utilizing large language models: fine-tuning

and in-context learning.

Fine-tuning: Similar to various natural language tasks, the integration of large language models into the Text-to-SQL domain holds promise for enhancing performance on this specific downstream task. In the context of most Text-to-SQL tasks, the conventional approach involves leveraging a large language model alongside a set of training data annotated by humans. This training data typically comprises natural language questions paired with their corresponding database queries. The primary objective during supervised fine-tuning is to minimize the disparity between the generated query by the language model and the ground truth query provided by human annotations. [7]

In the context of the Text-to-SQL task, earlier research endeavors involved the fine-tuning of T5 transformer-based models, yielding an accuracy of 68.3% [9]. Additionally, the GPT-3 model achieved an accuracy of 51.3% on the same task [10]. Both reported results are grounded in test-suite accuracy, a metric that offers a more refined evaluation of the quality of generated SQL queries.

In-context learning: The fundamental idea behind in-context learning is to enable the language model to comprehend and generate desired outcomes by imbuing it with task-related context, such as examples or domain knowledge, through prompt engineering. This is achieved through two primary methods of forming prompts with context: zero-shot prompting and few-shot prompting. [7]

For zero-shot prompting, previous efforts involve providing only the question as a prompt, without including any database information. However, the performance of zero-shot prompting tends to be suboptimal, primarily attributed to the absence of information regarding the database schema and the correct format of SQL queries [10]. To enhance performance on the task, an intuitive approach is to furnish language models with database information and SQL domain knowledge, a strategy commonly referred to as few-shot prompting.

For few-shot prompting, prompts are structured into three distinct parts. The initial segment entails furnishing information about the database and its schema, succeeded by the presentation of query examples, and culminating with the inclusion of the natural language question in the final part [11] [12] [10]. The interpretation of prompts can vary across different formats, presenting challenges and research problems associated with prompt engineering. For instance, the performance of language models may exhibit variations when employing different quantities of examples [10], or when utilizing diverse formats to illustrate the database schema [12].

2.3 Evaluation Metrics of Text-to-SQL Tasks

Three commonly used metrics for evaluating SQL query quality are exact set match accuracy (EM), execution accuracy (EX), and test-suite accuracy (TS).

Exact set match accuracy, without considering values, is determined by comparing the ground-truth SQL query with the predicted SQL query [3]. EM assesses model performance by strictly scrutinizing differences in SQL. However,

human SQL annotations often exhibit bias, as a natural language question may correspond to multiple SQL queries.

Execution accuracy, which does consider values, is computed by comparing the output results of executing the ground-truth SQL query and the predicted SQL query on the database contents provided in the test set [3].

To address potential issues arising from SQL execution on finite-sized databases and to mitigate false positives and false negatives, test-suite execution accuracy [13] extends the execution evaluation to multiple database instances per schema. Specifically, the test-suite distills a small database from randomly generated databases, aiming to achieve high code coverage. This approach enables the provision of the best approximation of semantic accuracy.

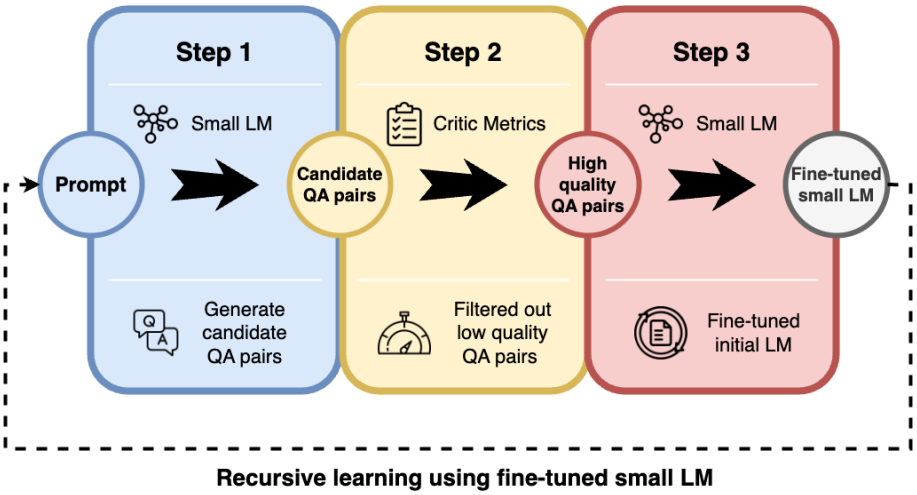

Fig. 1. The framework of the paper consists of three sequential steps. Step 1: Generate candidate QA pairs. **Step 2:** Filtered out low-quality QA pairs. **Step 3:** Fine-tuned initial small language models. Through recursive iterations of this process, the small language model autonomously learns Text-to-SQL.

3 Background

3.1 Formal Problem Statement

Drawbacks of fine-tuning: In the realm of Text-to-SQL tasks, it is customary to employ benchmark datasets such as WikiSQL[14] and SPIDER [15] for the fine-tuning of language models. However, this process necessitates a substantial annotated dataset for effective training. The annotation of data represents a considerable commitment of human resources and time. The challenge lies in the

absence of an efficient method to autonomously generate high-quality Text-to-SQL datasets without human annotation, particularly when access to established large-scale benchmark datasets is unavailable.

Drawbacks of few-shot prompting: The practice of providing few-shot prompts to large language models and provoking them to generate corresponding SQL queries for natural language problems is a prevalent approach in Text-to-SQL tasks. However, the computational expenses associated with interacting with large language models may be prohibitively high, particularly considering the extensive tokens in the prompts and the considerable volume of prompt requests.

3.2 Architecture Description

Figure 1 illustrates the comprehensive process of the framework proposed in the paper. It aims to resolve the above two problems of the existing methodologies. Using self-produced data with critic metrics can generate high-quality data for fine-tuning language models without human involvement. Utilizing a small language model as our fundamental model can reduce the computational cost and have access to control its training progress.

The framework consists of three steps. In **Step 1**, a few-shot prompt is provided to a small language model, instructing it to produce multiple candidate question-answer (QA) pairs. **Step 2** involves the application of critic metrics to eliminate QA pairs of inferior quality. In **Step 3**, the small language model undergoes fine-tuning using the identified high-quality QA pairs and is subsequently employed as the small language model in the initial step. Through iterative iterations of this process, the small language model autonomously acquires proficiency in Text-to-SQL tasks.

4 Methodology

The framework used in the study is inspired by the I2D2 framework [16]. It aims to automatic training a small pretrained language model for Text-to-SQL tasks. Our choice of initial small language model is Incoder-1B [17]. However, it can be replaced by any language model that pretrained on code. Given that the majority of code-pretrained models exhibit an inherent understanding of query structures owing to their training data, it can be inferred that they possess an implicit knowledge of SQL.

We trained Incoder-1B in three stages. In the first stage, the procedure involves the acquisition of the database schema along with associated exemplary queries and related questions. Subsequently, a predetermined template is employed to prompt the initial language model, eliciting the generation of candidate question-query pairs for each query(Section 4.1). During the second stage, erroneous and non-executable queries will be filtered and excluded through the application of critical metrics(Section 4.2). In the final stage, the language model

undergoes iterative recursive distillation learning, wherein it is fine-tuned using its own high-quality generations, as determined by the critic metrics(Section 4.3).

Schema	TABLE head(head_ID int, name text, born_state text, age real);
Example	Question: How many heads of the departments are age 56? Answer: SELECT count(*) FROM head WHERE age = 56;
Activate phrase	Question:

Table 1. An example of few-shot prompting. Using a fixed template start with the schema of the database followed by some examples of question-answer pairs. Conclude the prompt with an activation phrase.

4.1 Prompt Construction

We utilize the Spider dataset [15] as the primary raw data source, consisting of 200 database schemas and 5693 distinct complex SQL queries spanning a diverse array of domains. To optimize the performance of the language model while mitigating potential influences from prompts, we adopt a prompt template employed in a previous study [11]. This template demonstrates that language models can enhance their capabilities by internalizing contextual knowledge embedded within the prompting structure.

Essentially, our prompt formulation involves initially presenting the database schema, followed up by multiple question-query pairs as examples for the model. Consequently, appending a simple term such as "Question" serves to elicit the generation of additional question-query pairs by the language model. The entirety of this prompt template is illustrated in Table 1 for reference.

4.2 Critic Metrics

Why needed critic metrics: We discovered that the employment of smaller language models exhibits poor performance in generating candidate question-query pairs, particularly those involving executable and format-correct SQL queries. The provision of data of inferior quality adversely impacts the model's outcomes in each iteration. Consequently, it becomes imperative to implement a robust filtration mechanism to eliminate erroneous and non-executable queries from the pool of candidate question-query pairs generated by these smaller language models.

How to comprise critic metrics: Critic metrics may encompass various policies and evaluation metrics, with the primary goal of identifying and filtering out non-executable and erroneous queries. The composition of critic metrics involves

the integration of multiple policies and evaluation metrics. However, determining the quality of a query poses challenges, given that even minor syntax errors can render an entire query non-executable or yield inaccurate responses to the posed question. Therefore, it is imperative for critic metrics to possess the capability to either exclude non-executable queries or discern semantically similar queries to the gold query that, despite their resemblance, may lead to incorrect outcomes.

Why using execution accuracy and test suite accuracy: In our approach, we employ two widely recognized evaluation metrics, namely execution accuracy (EX) and test-suite accuracy (TS), as the primary critic metrics to identify and eliminate queries of inferior quality. The execution accuracy (EX) metric measures whether the SQL execution outcome aligns with the ground truth. On the other hand, test-suite accuracy (TS) involves subjecting each query to multiple tests against a randomly generated database with the same schema. Unlike EX, which evaluates based on a single test, TS helps mitigate false positives and, consequently, enhances precision. It is noteworthy that the study does not incorporate the commonly used exact match evaluation metric, as a single query may have multiple correct SQL formulations.

Algorithm 1 Pseudo code of the framework

Input:
P_0: A language model pretrained on code
X: Set of prompts
$\epsilon()$: Execution accuracy metric
$\tau()$: Test-suite accuracy metric
N: Number of Iterations

for $k = 0, 1, ..., N\text{-}1$ **do** ▷ Each Iteration
  $D_k \leftarrow \{\}$
  for $X_i \in X$ **do**
    $[(Q, A)] = P_k(Q, A)|X_i$ ▷ Generate QA pairs
    for $(Q_i, A_i) \in [(Q, A)]$ **do**
      if $\epsilon(A) = 1$ and $\tau(A) = 1$ **then** ▷ Apply Critic Metrics
        $D_k \leftarrow (Q_i, A_i)$
  $P_{k+1} \leftarrow argmax_\theta E \log P_k(Q_n, A_n)|(Q_1, A_1), ..., (Q_{n-1}, A_{n-1}); (Q, A) \in D_k$ ▷ Fine-tuned LM on high-quality QA pairs

4.3 Recursive distillation Learning

Why recursive distillation learning: The prior study explores recursive variants [18], showcasing that language models possess the capacity to iteratively enhance their performance without the need for maintaining distinct sets of parameters [11]. Intuitively, we hypothesize that fine-tuning the language model based on its own high-quality generations can enhance its performance for generating question-answer pairs. This process involves learning policies and formats

that align with higher-quality samples, thereby contributing to improved model proficiency in question-answer pair generation.

Recursive distillation learning in the study: Distillation refers to the process of creating a dataset without any human annotations, wherein a language model is employed to generate both task inputs and their corresponding labels [19]. In the context of recursive distillation learning, the incorporation of context allows for the internalization of training examples, presenting a potential alternative to the conventional approach of direct learning from these examples through gradient descent [11]. In this particular study, we task a language model with generating candidate task inputs in a question-query format and utilize recursive learning by fine-tuning the original language model with self-produced, high-quality inputs. In order to mitigate the overfitting concern during training on self-generated data, we initially partitioned the raw data into multiple batches. Subsequently, prompts from each batch were systematically provided to the model during individual iterations. This approach facilitates incremental learning of the task, thereby acting as a preventive measure against overfitting. Our approach enables a smaller language model to excel in Text-to-SQL tasks without the need for few-shot prompting and human-annotated data. Our framework is formally described in Algorithm 1.

5 Use cases

In this study, the SPIDER dataset[15] serves as the foundational raw data, comprising 10,181 natural language questions and 5,693 unique SQL queries. The dataset spans 200 databases across 138 distinct domains. To showcase the applicability of our approach, we demonstrate the process of constructing few-shot prompts and employ execution accuracy and test-suite accuracy as the designated critic metrics for evaluation purposes.

head_ID	name	born_state	age
1	Tiger Woods	Alabama	67
2	Sergio Garcia	California	68
3	K.J.Choi	Alabama	69
4	Dudley Hart	California	52
5	Jeff Maggert	Delaware	53
6	Billy Mayfair	California	69
7	Stewart Cink	Florida	50
8	Nick Faldo	California	56
9	Fadraig Harrington	Connecticut	43
10	Franklin Langham	Connecticut	67

Table 2. The structure and data in Table head of SPIDER dataset.

NL question	How many heads of the departments are age 56?
Correct SQL	SELECT count(*) FROM head WHERE age = 56;

Table 3. A Natural language questions and its corresponding correct SQL based on Table head of SPIDER dataset.

5.1 Prompt Template Example

In this study, a fixed prompt template is employed as a means to standardize the quality of candidate question-answer pairs, thereby mitigating the potential impact of varying prompt structures. The methodology involves illustrating the process of constructing a prompt based on a given database schema and multiple associated natural language questions aligned with the schema. The final prompt illustrated in Table 1 is formed with the database schema in Table 2 and the QA pair in Table 3.

Candidate QA pairs	Execution accuracy	Test-suite accuracy
How many heads of the departments are age 56? SELECT count(*) FROM head WHERE age <56;	✘	✘
How many heads of the departments are age 56? SELECT count(*) FROM head WHERE age >56;	✘	✘
How many heads of the departments are age 56? SELECT count(*) FROM head WHERE age = 56;	✔	✔
How many heads of the departments are age 56? SELECT count(head.name) FROM head WHERE age = 56;	✔	✔

Table 4. Using execution accuracy and test-suite accuracy to filter out low-quality candidates.

5.2 Critic Metrics Demonstration

Subsequent to the formulation of a few-shot prompt, the next step involves its presentation to a small language model, prompting the model to generate multiple candidate question-answer pairs. Given the potential presence of inferior-quality pairs within the generated candidates, a filtering process is instituted. This involves the utilization of execution accuracy and test-suite accuracy as critic metrics, enabling the identification and subsequent exclusion of pairs failing to meet the established metric thresholds. The whole process is illustrated in Table 4.

Through the application of execution accuracy, we effectively identified and filtered out low-quality candidates, specifically those generating non-correct SQL (Candidate 1 & 2) in response to the natural language question. Moreover, the utilization of test-suite accuracy ensures the correctness of a candidate, even if it does not precisely match the ground truth (Candidate 4). It is noteworthy that if exact set match accuracy were employed as the sole critic metric, candidates semantically equivalent to the ground truth would not be considered high-quality, underscoring the importance of employing a more nuanced evaluation criterion.

6 Summary

6.1 Conclusions

Through the utilization of small language models and critic metrics within the training loop, the capability to instruct models in Text-to-SQL tasks without direct human involvement is facilitated, concurrently mitigating the computational costs associated with training. This approach is envisioned to extend beyond the confines of SQL or code generation, finding applicability in diverse domains. In a landscape where there is a prevalent inclination towards leveraging large-scale language models, assuming scale as the sole determinant for optimal task performance, it is imperative to recognize the potential efficacy of small language models. Indeed, these smaller models could represent an alternative avenue for achieving efficiency in various natural language tasks.

6.2 Future work

In the context of future work, our research delineates two primary directions for exploration. Firstly, although our study predominantly relies on the simplistic few-shot prompting methodology, there exists untapped potential in the exploration of more innovative approaches, such as Chain-of-Thought [20] or Step-back prompting [21]. These techniques involve the strategic composition of prompts to provoke more desirable outcomes, termed as prompt engineering. Therefore, one direction for future investigation involves determining whether alternative prompt organization methods can yield improved model performance.

Secondly, our research has focused on a limited set of critic metrics for SQL evaluation. In future endeavors, there is an opportunity to delve deeper into the exploration and refinement of critic metrics. While our study demonstrates the efficacy of two metrics as evaluative metrics, applying multiple metrics in SQL evaluation could enhance the overall quality of question-answer pairs. This multifaceted evaluation approach would provide a more comprehensive evaluation not only for Text-to-SQL tasks but also for other SQL-related tasks, such as SQL-to-Text.

References

1. Jiho Shin, Clark Tang, Tahmineh Mohati, Maleknaz Nayebi, Song Wang, and Hadi Hemmati. Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks. *arXiv preprint arXiv:2310.10508*, 2023.
2. Da Ma, Xingyu Chen, Ruisheng Cao, Zhi Chen, Lu Chen, and Kai Yu. Relation-aware graph transformer for sql-to-text generation. *Applied Sciences*, 12(1):369, 2021.
3. Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*, 2022.
4. Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. Expert perspectives on student errors in sql. *ACM Transactions on Computing Education*, 23(1):1–28, 2022.
5. Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*, 2023.
6. Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*, 2023.
7. Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023.
8. Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. Prompting gpt-3.5 for text-to-sql with de-semanticization and skeleton retrieval. In *Pacific Rim International Conference on Artificial Intelligence*, pages 262–274. Springer, 2023.
9. Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.
10. Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*, 2022.
11. Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.
12. Ruoxi Sun, Sercan Ö Arik, Rajarishi Sinha, Hootan Nakhost, Hanjun Dai, Pengcheng Yin, and Tomas Pfister. Sqlprompt: In-context text-to-sql with minimal labeled data. *arXiv preprint arXiv:2311.02883*, 2023.
13. Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites. *arXiv preprint arXiv:2010.02840*, 2020.
14. Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.
15. 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.
16. Chandra Bhagavatula, Jena D Hwang, Doug Downey, Ronan Le Bras, Ximing Lu, Keisuke Sakaguchi, Swabha Swayamdipta, Peter West, and Yejin Choi. I2d2:

Inductive knowledge distillation with neurologic and self-imitation. *arXiv preprint arXiv:2212.09246*, 2022.
17. Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*, 2022.
18. Eunbi Choi, Yongrae Jo, Joel Jang, and Minjoon Seo. Prompt injection: Parameterization of fixed inputs. *arXiv preprint arXiv:2206.11349*, 2022.
19. Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. Towards zero-label language learning. *arXiv preprint arXiv:2109.09193*, 2021.
20. Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
21. Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*, 2023.