# Accurate and Regret-Aware Numerical Problem Solver for Tabular Question Answering

**Yuxiang Wang, Jianzhong Qi*, Junhao Gan**

School of Computing and Information Systems, The University of Melbourne
yuxiang.wang8@student.unimelb.edu.au, {jianzhong.qi, junhao.gan}@unimelb.edu.au

## Abstract

Question answering on free-form tables (a.k.a. *TableQA*) is a challenging task because of the flexible structure and complex schema of tables. Recent studies use Large Language Models (LLMs) for this task, exploiting their capability in understanding the questions and tabular data, which are typically given in natural language and contain many textual fields, respectively. While this approach has shown promising results, it overlooks the challenges brought by numerical values which are common in tabular data, and LLMs are known to struggle with such values. We aim to address this issue, and we propose a model named TabLaP that uses LLMs as a planner rather than an answer generator. This approach exploits LLMs' capability in multi-step reasoning while leaving the actual numerical calculations to a Python interpreter for accurate calculation. Recognizing the inaccurate nature of LLMs, we further make a first attempt to quantify the trustworthiness of the answers produced by TabLaP, such that users can use TabLaP in a *regret-aware* manner. Experimental results on two benchmark datasets show that TabLaP is substantially more accurate than the state-of-the-art models, improving the answer accuracy by 5.7% and 5.8% on the two datasets, respectively.

**Code** — https://github.com/yxw-11/TabLaP

## Introduction

Tables are a commonly used data format to organize and present information. *Table Question Answering* (TableQA) aims to answer questio ns based on data in tables. It arises as an important problem to automatically extract information from free-form tables for non-experts (Ye et al. 2023a). In a typical TableQA task, questions are given in natural language. The input tables are in free-form and may not have a well-defined schema, i.e., they are not necessarily relational tables, such as web tables (Cafarella et al. 2008). In other words, these tables may have mixed data types in columns and non-predefined logical forms (Jin et al. 2022). In this case, structural query language-based solutions (Pasupat and Liang 2015; Zhong, Xiong, and Socher 2017; Pourreza and Rafiei 2023; Gao et al. 2024) are not always applicable. See Figure 1a for an example of the TableQA task.

Recent studies (Cheng et al. 2023; Ye et al. 2023b; Liu, Wang, and Chen 2024; Patnaik et al. 2024) use Large Language Models (LLMs), exploiting their semantic capabilities to analyze textual questions as well as tabular data, where many fields are often in text. Unfortunately, these studies have overlooked a notorious issue of the LLMs – their limited capability in handling numerical data (Frieder et al. 2024). As Figure 1b shows, prompting an LLM (GPT-3.5 Turbo) directly with a table and a question for numerical calculation could lead to an inaccurate answer.

To address this issue, we study how to strengthen the capability of LLMs to handle numerical questions. A crucial observation behind our proposed solution is that while LLMs struggle with carrying out numerical calculations, they are capable of producing plans to execute such calculations. As Figure 1c shows, using the same LLM but prompting it to generate a calculation plan in Python, the generated Python script can be executed and yield the correct answer.

Based on this observation, we propose a TableQA model with an LLM as a planner, *TabLaP*. We call the planner LLM *NumSolver* and design a prompt to guide it to generate a plan that decomposes a complex numerical question into sequential steps (in Python script) based on Chain-Of-Thought (Wei et al. 2022). The script is then executed by a Python interpreter to produce an answer to the question.

Moreover, to largely retain the strong capability of LLMs to process non-numerical questions, TabLaP takes a dual-branch structure, where NumSolver forms a branch and a state-of-the-art (SOTA) TableQA model forms the other.

As each of the two branches produces an answer, to integrate the answers from them, we exploit a third LLM (named *AnsSelector*) – an open-source one (LlaMa 3-8B-Instruct) which allows for fine-tuning – to take the question and answers from both model branches (including the reasoning texts) as input and selects a branch to trust. The answer from the selected branch is then returned as the final answer.

Recognizing the inaccurate nature of LLMs, we further quantify the *trustworthiness* of the answers generated by TabLaP. We propose a module named *TwEvaluator* based on yet another LLM and Multi-Arm Bandit (MAB) (Vermorel and Mohri 2005). TwEvaluator tracks the answer correctness of both TabLaP branches over time and yields a trustworthiness label of the final answer accordingly. This label enables users to consume the answer in a *regret-aware* manner.

*Corresponding author

Overall, this paper makes the following contributions:

- We propose TabLaP, a multi-LLM-based model for TableQA tasks with both numerical and non-numerical questions. The core contribution of TabLaP lies in its holistic system design that exploits the strength of each model forming a module of TabLaP, rather than forcing a model onto an unsuitable task.
- We propose NumSolver, an LLM-based module to process numerical questions. We further fine-tune an open-source LLM-based AnsSelector module to decide whether to use the answer generated by NumSolver or a SOTA TableQA model (e.g., Liu, Wang, and Chen 2024), exploiting their capabilities in answering numerical and non-numerical questions, respectively.
- We propose a regret-aware scheme enabled by an LLM-and-MAB-based module named TwEvaluator that tracks the correctness of the answer generation modules and produces a trustworthiness label for the answers.
- We conduct experiments to test the effectiveness of TabLaP on `WikiTableQuesetions` (Pasupat and Liang 2015) and `FTQ`. `WikiTableQuesetions` is a public dataset, while `FTQ` is adapted by us from the `FeTaQA` dataset (Nan et al. 2022) by removing answer tokens non-directly relevant to the questions. The experimental results show that TabLaP outperforms SOTA TableQA models on both datasets by 5.7% and 5.8% in accuracy, respectively. Meanwhile, the answer trustworthiness labels generated by TabLaP help reduce the user regret ratio on using the model generated answers by 19.6% and 20.6% on the two datasets, respectively, compared with always trusting the model generated answers.

## Related Work

Studies on TableQA are mainly driven by designing models that can understand questions in natural language and tabular data. These studies can be categorized into *semantic parsing-based*, *pre-trained language model (PLM)-based*, and *large language model (LLM)-based*.

**Semantic parsing-based methods.** Semantic-parsing-based methods transform natural language questions into a logical form (e.g., SQL) that machines can understand and execute. There are two sub-categories of methods: (i) *weakly-supervised* (Pasupat and Liang 2015; Neelakantan et al. 2017; Yu et al. 2018a), and (ii) *fully-supervised*, such as NL-to-SQL (Zhong, Xiong, and Socher 2017; Pourreza and Rafiei 2023; Gao et al. 2024). For weakly supervised methods, a semantic parser generates the logical form based on an input question, a table, and the answer. There is no pre-defined ground-truth logical form. Fully supervised methods, on the other hand, further take a ground-truth logical form as their input. Both sub-categories focus on analyzing the questions. They are less effective on tables with complex structures and data types (Hong et al. 2024).

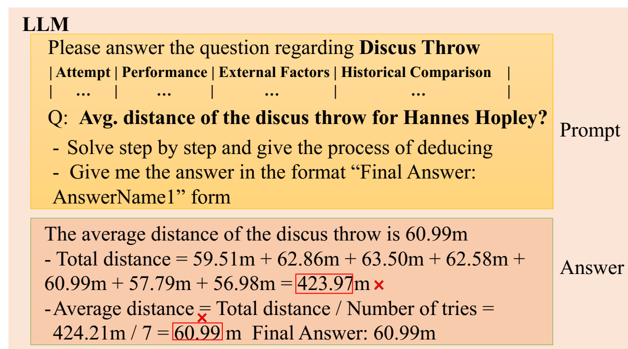
**PLM-based methods.** Language model-based methods, including PLM-based ones, focus on guiding the models to understand tabular data. There are two directions: (i) Tailoring Transformer (Vaswani et al. 2017) model structure for better tabular data understanding. For example,

**Table Title:** Hannes Hopley Discus Throw

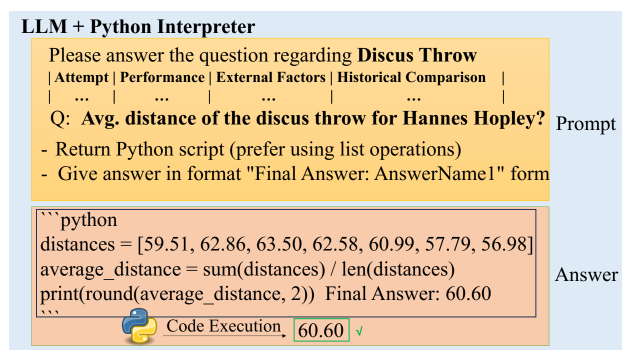
| Attempt Details | Performance Data | External Factors | Historical Comparison |
|---|---|---|---|
| 1st Try | Distance: **59.51m** Accuracy: 85% | Wind: Light breeze | Last: 60.02m Average: 59.80m |
| 2nd Try | Distance: **62.86m** Speed: 45 rpm | Wind: Slight increase at 2 m/s | Last: 61.30m Best: 62.90m |
| 3rd Try | Distance: **63.50m** Release Angle: 35° | Wind: Calm Humidity: 60% | Last: 62.00m Best: 63.00m |
| 4th Try | Distance: **62.58m** Power: 92% | Wind: Crosswind increased | Last: 62.70m Average: 62.50m |
| 5th Try | Distance: **60.99m** Fatigue: High | Wind: Gusty, challenging conditions | Last: 61.80m Below Average |
| 6th Try | Distance: **57.79m** Stability: Poor | Wind: Strong gusts | Last: 58.50m Worst Attempt: 57.50m |
| 7th Try | Distance: **56.98m** | Wind: Unfavorable | Last : 59.00m |

**Question:** The average distance of the discus throw for Hannes Hopley?

(a)




(b)




(c)

Figure 1: A TableQA example with a numerical question: (a) Input table and question; (b) Answer from prompting an LLM (GPT-3.5 Turbo) directly; (c) Answer from prompting the LLM to generate a Python script for the calculation (we prompt the LLM to ignore the unit to suit the evaluation procedure of the benchmark datasets, which can be kept by adapting the prompt easily).

TAPAS (Herzig et al. 2020) masks table cells and extends BERT (Devlin et al. 2019) by adding column, row, and rank embeddings. TUTA (Wang et al. 2021) further masks columns or table segments and designs a special attention mechanism with a tree structure to capture the hierarchical relationships and dependencies for tabular data. TaBERT (Yin et al. 2020) extends BERT with *vertical self-attention* and combines text and table representations in a unified framework. (ii) Pre-training and fine-tuning models for end-to-end TableQA. For example, TAPEX (Liu et al. 2022) pre-trains BART (Lewis et al. 2020) with a large synthetic dataset derived from the `WikiTableQuestions` dataset. CABINET (Patnaik et al. 2024) fine-tunes content relevance-based modules that reduce the weight of irrelevant rows or columns to guide the PLM focus more on the key information. OmniTab (Jiang et al. 2022) also leverages BART as its backbone model, while it is pre-trained using both real and synthetic data – these include SQL queries from the Spider dataset (Yu et al. 2018b) and synthetic natural language sentences converted from the SQL queries using their proposed SQL2NL model. The aim is to enhance the model capability to perform few-shot learning for TableQA. The PLM models are smaller than the latest LLMs. They have limited capabilities in understanding tabular semantics, leading to sub-optimal TableQA results, as shown empirically.

**LLM-based methods.** Recent studies exploit the semantic and context (i.e., table) tracking capabilities of LLMs.

In-context learning is a common method, where a few TableQA examples are fed into an LLM together with an input question and a table to prompt the LLM for answer generation. For example, Chain-of-Table (Wang et al. 2024) solves TableQA problems step by step and obtains a sub-table at each step using a GPT-based model. DATER (Ye et al. 2023b) decomposes tables and questions into sub-tables and sub-questions at each step to help LLMs understand the question and data more easily.

Mix-SC (Liu, Wang, and Chen 2024), the SOTA model, generates a few answers for each question exploiting the stochastic nature of LLMs. It uses two types of prompts: prompting the LLM to run as a Python agent to execute Python scripts directly and Chain-of-Thoughts prompting to ask the LLM to solve problems stey by step (Wei et al. 2022). The best answer is returned, which is selected using a *self-consistency* method, i.e., taking the most frequent answer. The models above generate answers by LLMs directly, while we use LLMs to generate an answer calculation plan. These models can form a branch in our model. We use Mix-SC for its SOTA performance.

Binder (Cheng et al. 2023) uses an LLM to generate an initial program for an input question and identify portions of the program that are difficult to solve. It then re-invokes the LLM to supplement these parts. Finally, the refined program is executed by an interpreter to obtain the TableQA answer. LEVER (Ni et al. 2023) exploits an LLM as a planner to generate multiple possible SQL queries, executes them to obtain candidate answers, and employs a verification model to select the best answer. TabLaP differs from Binder in that it uses the Chain-of-Thought method, allowing the LLM to generate problem-solving scripts step by step and enables the LLM to directly analyze and produce an answer, mitigating the impact of code execution errors. In comparison, Binder's answers heavily rely on the quality of the initially generated program, while its design does not account for handling errors in the program. Therefore, TabLaP better leverages the analytical capabilities of LLMs and provides a more reliable solution. Compared with LEVER, TabLaP generates Python scripts instead of SQL queries, which are more flexible for non-relational tables. Further, TabLaP's answer selector examines not only the answers (as done by LEVER) but also the reasoning process of the answers, leading to more accurate answer selection.

**Numerical Reasoning for LLMs.** Understanding numerical data and performing calculations are known issues with LLMs (Didolkar et al. 2024; Frieder et al. 2024).

Efforts have been made to improve LLMs' mathematical capability. DELI (Zhang et al. 2023), which utilizes an LLM to generate the solution process for mathematical problems, identifies mathematical expressions within the solution, and then calls external tools to execute the calculation. PAL (Gao et al. 2023) breaks down a mathematical problem into multiple steps, generates executable code for each step to produce intermediate results, and combines these intermediate answers to calculate the final answer. Inspired by this latter study, we generate multiple answers for an input question and train a classifier to select the best answer. Our solution differs from both studies above, in that we use an LLM to generate Python scripts and execute them end-to-end, effectively reducing the impact of errors produced by the intermediate steps. Additionally, by analyzing the reasoning process to select the best answer, TabLaP addresses scenarios where the calculation is correct while the reasoning is flawed.

# Methods

Given a table $T$ and a question $Q$ regarding the data in $T$, our goal is to design a model that produces an accurate answer for $Q$. Here, $Q = (q_1, q_2, \ldots, q_{|Q|})$ is given in natural language, where $q_i \in Q$ ($i \in [1, |Q|]$) is a token (e.g., a word), $T$ is also represented as a sequence of tokens in natural language, where the cells are separated by special characters such as '|' while the rows are separated by newline characters. The effectiveness of a model is measured by a token-based comparison between the answer generated by the model and the ground truth.

In this paper, we are particularly interested in multi-step numerical questions which require applications of two or more basic arithmetic and relational operators $\{+, -, *, /, >, <\}$. These operators are common in our experimental datasets, while the techniques proposed can be extended to support more relational operators such as $\geq$ and $\leq$ straightforwardly. It is known that existing LLMs are less effective in handling this type of questions.

## Model Overview

As outlined in Algorithm 1, there are three stages in the question answering process of our TabLaP model: (i) *answer generation*, (ii) *answer selection*, and (iii) *answer trustworthiness evaluation*.

In the answer generation stage, two models are adopted. One is a SOTA model (Liu, Wang, and Chen 2024) (detailed in Related Work), denoted by $M_{SOTA}$, and the other is our numerical question answering module NumSolver, denoted by $M_{NS}$ (detailed in Answer Generation). These two mod-

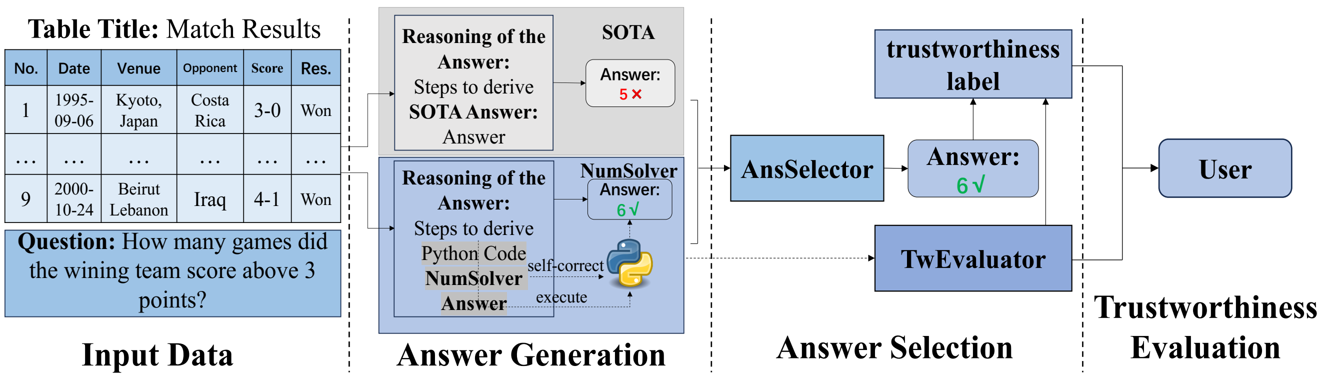



Figure 2: Overview of TabLaP. The model has three stages: answer generation, answer selection, and trustworthiness evaluation. (i) The answer generation stage uses both a SOTA TableQA model (Liu, Wang, and Chen 2024) and our NumSolver module to generate answers for an input question and a table, where NumSolver focuses on numerical questions. (ii) The answer selection stage selects answers from the generated ones, based on the question and the reasoning steps generated by the two models. (iii) The trustworthiness evaluation stage tracks the success rates of both the SOTA model and NumSolver, and generates a trustworthiness label that is presented to users together with the selected answer.

**Algorithm 1:** Table Question Answering with TabLaP

**Require:** Table $T$, questions $Q$, answer generation models $M_{SOTA}$ and NumSolver $M_{NS}$, AnsSelector, TwEvaluator

1: **Answer Generation:**
(1) Generate prompt$_{SOTA}$ and prompt$_{NS}$ for $M_{SOTA}$ and $M_{NS}$, respectively
(2) Obtain answer and reasoning pairs $(\text{Ans}_A, \text{Rsn}_A)$ and $(\text{Ans}_B, \text{Rsn}_B)$ by feeding prompt$_{SOTA}$ and prompt$_{NS}$ to $M_{SOTA}$ and $M_{NS}$, respectively

2: **Selection by AnsSelector:**
(1) Combine table $T$ schema and $Q$ with $(\text{Ans}_A, \text{Rsn}_A)$ and $(\text{Ans}_B, \text{Rsn}_B)$ to form answer selection prompt$_{\text{sel}}$
(2) $\text{Ans}_{\text{best}} \leftarrow \text{AnsSelector}(\text{prompt}_{\text{sel}})$

3: **Verification by TwEvaluator:**
(1) Combine table $T$ schema and $Q$ with $(\text{Ans}_A, \text{Rsn}_A)$ and $(\text{Ans}_B, \text{Rsn}_B)$ to form verification prompt$_{\text{twe}}$
(2) $\text{Label}_{\text{true/false}} \leftarrow \text{TwEvaluator}(\text{prompt}_{\text{twe}})$
(3) Update TwEvaluator based on $\text{Ans}_{\text{best}}$, $\text{Label}_{\text{true/false}}$, and ground truth answer

4: **return** $\text{Ans}_{\text{best}}$, $\text{Label}_{\text{true/false}}$

els take the input pair $(T, Q)$ and individually generate answers (together with their reasoning steps) for $Q$.

In the answer selection stage, our AnsSelector module (detailed in Answer Selection) takes as input the question $Q$, the schema of $T$ and the two answers along with the corresponding reasoning steps provided by the two answer generation models in the previous stage. It decides an answer from the two candidates.

In the answer trustworthiness evaluation stage, our TwEvaluator (detailed in Trustworthiness Evaluation) tracks the reliability of the answers generated by the two answer generation models and produces a trustworthiness label for the answer selected in the previous stage.

## Answer Generation

We use NumSolver and a SOTA model, Mix-SC (Liu, Wang, and Chen 2024), as two separate branches to answer $Q$.

In NumSolver, we prompt a backbone LLM to answer $Q$ step by step and generate the reasoning process with intermediate results. Besides, we also prompt the LLM to write down the Python scripts to answer $Q$. By executing the Python scripts with a Python interpreter, question answers are obtained, which are found to be more accurate than those obtained from the direct LLM reasoning process, especially for multi-step numerical questions. When the Python-based answers are different from the LLM-generated ones, and they contain numerical values, priority is given to the Python-based answers. When errors occur during Python execution, the answers obtained directly from the LLM are used. This process is called *self-correctness*.

The prompts used in our models (including the ones introduced below) are available in our online technical report (Wang, Qi, and Gan 2024).

## Answer Selection

AnsSelector decides the best answer from those generated by the two model branches in the answer generation stage, based on the question, table schema, generated answers, and the reasoning process. We use a fine-tuned Llama3-8B-Instruct as AnsSelector, which returns a label of either `[A]` or `[B]`, indicating that the answer from the SOTA TableQA branch or NumSolver is preferred, respectively.

## Trustworthiness Evaluation

TwEvaluator trains an LLM (a fine-tuned Llama3-8B-Instruct) with a similar input to that of AnsSelector. It returns an answer of either `[True]` or `[False]`, indicating whether TabLaP has answered the input question correctly.

To enhance the reliability of TwEvaluator, we use two methods: the Expanding Window (EW) method and the Multi-arm Bandits (MAB) with Upper Confidence Bound (UCB) (Slivkins 2019), as *rejection filters* that filter potential false rejections of the TwEvaluator.

The EW method. The EW method starts by accepting the TwEvaluator LLM's predictions for $t$ TableQA instances, to calculate an initial accuracy of the LLM, denoted by $A(t)$:

$$A(t) = \frac{\text{num\_correct}(M_{Tw}, t)}{t}. \quad (1)$$

Here, num_correct$(M_{Tw}, t)$ denotes the number of correct predictions made by the TwEvaluator LLM for the $t$ test instances. From the $(t+1)$-th test instance, the EW method rejects the LLM's *unreliable* labels with a probability of $P(t) = 1 - A(t)$, and updates $A(t)$ to $A(t+1)$.

The MAB method. The MAB method aims to balance exploration and exploitation by selecting actions that maximize expected rewards while considering uncertainty. We use an MAB of two arms representing either to accept or to reject an *unreliable* label of the TwEvaluator LLM. We aim to select the arm that maximizes the TwAccuracy of TabLaP, as guided by the equation below:

$$\hat{\mu}_i(t) = \frac{\sum_{n=1}^{t-1} r_i(n) \cdot \mathbb{I}(a(n) = i)}{N_i(t-1)}, \quad (2)$$

where $\hat{\mu}_i(t)$ is the estimated mean reward of arm $i$ at time (i.e., test instance) $t$; $r_i(n)$ is the reward received when arm $i$ is selected at time $n$; $\mathbb{I}(a(n) = i)$ is the indicator function, which equals to 1 if arm $i$ is selected at time $n$, and 0 otherwise; and $N_i(t)$ is the number of times arm $i$ has been selected up to time $t$.

To balance exploitation (i.e., to follow the arm with a larger estimated mean reward $\hat{\mu}_i(t)$) and exploration (i.e., to try the other arm and accumulate more accurate mean reward estimates for the arm), we use the *Upper Confidence Bound* (UCB) algorithm:

$$UCB_i(t) = \hat{\mu}_i(t) + c \cdot \sqrt{\frac{\ln t}{N_i(t)}}, \quad (3)$$

where $UCB_i(t)$ is the UCB for arm $i$ at time $t$; $c$ is the exploration parameter that controls the balance between exploration and exploitation; and $\ln$ is the natural logarithm function. We select the arm with a larger UCB value at time $t$.

## Fine-tuning for AnsSelector and TwEvaluator

We adopt Low-Rank Adaptation (LoRA) (Hu et al. 2022), to fine-tune our AnsSelector and TwEvaluator with Llama3-8B-Instruct as the LLMs. The forward pass for both AnsSelector and TwEvaluator LLMs is represented as:

$$\mathbf{h} = \underbrace{\mathbf{W}_0\mathbf{x}_{(T,Q,A)} + \Delta\mathbf{W}\mathbf{x}_{(T,Q,A)}}_{\text{gradient descent update}} = \underbrace{\mathbf{W}_0\mathbf{x}_{(T,Q,A)} + \mathbf{A}\mathbf{B}\mathbf{x}_{(T,Q,A)}}_{\text{LoRA update}}, \quad (4)$$

where $\mathbf{W_0}$ is the initial parameter weights for the Transformers; $\mathbf{x}_{(T,Q,A)}$ is a tuple of input table, question, and model answer sets, including answers and their corresponding reasoning processes; and $\mathbf{\Delta W}$ is the gradient descent update for parameters which is decomposed into low-rank matrices $\mathbf{A}$ and $\mathbf{B}$: $\mathbf{\Delta W} \in \mathbb{R}^{m \times d}$, $\mathbf{A} \in \mathbb{R}^{m \times k}$, and $\mathbf{B} \in \mathbb{R}^{k \times d}$. Matrices $\mathbf{A}$ and $\mathbf{B}$ have much fewer parameters compared with matrix $\mathbf{W}$, as $k \ll \min(m, d)$.

| Dataset | # QA Pairs | | # Numerical Questions | |
|---|---|---|---|---|
| | Training | Testing | Training | Testing |
| `WTQ` | 11,321 | 4,344 | 5,461 | 2,148 |
| `FTQ` | 2,000 | 1,245 | 417 | 182 |
| `TabFact_small` | 92,283 | 2,024 | 16,956 | 368 |

Table 1: Dataset statistics. The number of QA pairs includes both numerical and non-numerical questions. Besides, the average # tokens per table for `WTQ`, `FTQ`, and `TabFact_small` are 662.6, 297.0, and 317.5. The average # tokens per answer are 1.7, 5.1, and 1.0, respectively.

Since Llama3-8B-Instruct is used as the backbone model, the fine-tuning of AnsSelector and TwEvaluator LLMs is a fully supervised classification task that is posed as an utterance prediction task of the label words. The aim of the LLMs is to maximize $P_\theta(x_t^{(i)} \mid x_1^{(i)}, x_2^{(i)}, \ldots, x_{t-1}^{(i)})$, i.e., given previous tokens $(x_1^{(i)}, x_2^{(i)}, \ldots, x_{t-1}^{(i)})$, we maximize the conditional probability of $x_t^{(i)}$, where $\theta$ is the model parameter. Therefore, the loss functions for this fine-tuning task are:

$$\mathcal{L}_{ans} = -\sum_{i=1}^{V_{ans}} y_{m,i} \log(\hat{y}_{m,i}); \mathcal{L}_{twe} = -\sum_{i=1}^{V_{twe}} y_{n,i} \log(\hat{y}_{n,i}) \quad (5)$$

$$\mathcal{L} = -\log(\hat{y}_{m,y_m}) - \alpha \log(\hat{y}_{n,y_n}) = \mathcal{L}_{ans} + \alpha\mathcal{L}_{twe}. \quad (6)$$

Here, $y_{m,i}$ and $y_{n,i}$ are the ground-truth labels for the $m$th and $n$th tokens at position $i$ in one-hot encoding; $\hat{y}_{m,i}$ and $\hat{y}_{n,i}$ are the model predicted probabilities for the $m$th and $n$th tokens at position $i$; $V_{ans}$ and $V_{twe}$ are the dictionary lengths of the AnsSelector and TwEvaluator LLMs, respectively; $\mathcal{L}_{ans}$ and $\mathcal{L}_{twe}$ are the loss functions for the two LLMs, respectively; and $\alpha$ is a coefficient used to control the influence of the TwEvaluator LLM on the result, with its default value being 1.

# Experiments and Results

Next, we present experimental results to verify the effectiveness of TabLaP. We aim to answer the following questions: **(Q1)** How does TabLaP compare with SOTA models and the latest LLMs, in terms of accuracy to process TableQA tasks? **(Q2)** How effective is TabLaP in tracking the trustworthiness of its answers? **(Q3)** How effective is TabLaP in handling numerical problems? **(Q4)** How much do the modules contribute to the overall accuracy of TabLaP?

## Experimental Setup

**Datasets.** We use a public benchmark dataset named `WikiTableQuestions` (denoted as **`WTQ`**) (Pasupat and Liang 2015) and a dataset named **`FTQ`** which we adapted from the `FeTaQA` dataset (Nan et al. 2022). Besides, we use a third dataset named **`TabFact`** (Chen et al. 2020) to showcase the general applicability of TabLaP. Due to space limit, results on this third dataset is only included in our online technical report (Wang, Qi, and Gan 2024).

`FeTaQA` is also a TableQA dataset. Its answers contain descriptive content that may not be directly relevant to the

| Model | | Accuracy (%) WTQ | Accuracy (%) FTQ | TwAccuracy (%) WTQ | TwAccuracy (%) FTQ |
|---|---|---|---|---|---|
| Fine-tuned PLMs | TAPEX-Large | 57.5 | 19.8 | – | – |
| | OmniTab-Large | 63.3 | 25.2 | – | – |
| Zero-shot LLMs | GPT-3.5 Turbo | 50.9 | 38.1 | – | – |
| | GPT-4o | 58.1 | <u>41.7</u> | – | – |
| | Mix-SC | <u>72.5</u> | 41.6 | – | – |
| Few-shot LLMs | Binder | 64.6 | – | – | – |
| | DATER | 65.9 | – | – | – |
| | Chain-of-Table | 67.3 | – | – | – |
| Ours | TabLaP-EW | **76.6** | **44.1** | 77.6 | **53.8** |
| | TabLaP | **76.6** | **44.1** | **77.9** | 53.7 |
| | $\Delta$ | +5.7 | +5.8 | | |

Table 2: TableQA performance results on WTQ and FTQ datasets. Best results are in **boldface**, while second best results are <u>underlined</u>; $\Delta$ (%) denotes the performance gain of TabLaP comparing with the best baseline results.

questions. We remove such noisy information from the answers and only retain the entities that answer the questions, to form the FTQ dataset. Our technical report (Wang, Qi, and Gan 2024) includes an example question-answer pair from the original FeTaQA dataset and its cleaned-up version in our FTQ dataset. We use GPT-4o to process the FeTaQA dataset and extract the entities that may form answers to the questions, which are refined manually to produce the final question-answer pairs in FTQ. The prompt used by GPT-4o to extract the entities and further details on the construction of FTQ are also included in our technical report.

Table 1 summarizes the dataset statistics. WTQ is a larger dataset with longer tables, while FTQ has longer answers on average, making it a more challenging dataset (because its questions may contain multiple sub-questions). TabFact_small is a subset of the TabFact dataset for table-based fact verification, where the task is to determine whether each statement is entailed or refuted.

The numerical questions are counted by keyword matches. Each keyword may suggest single- or multi-step calculations, while a question with multiple keyword matches typically involve multi-step calculations. Refer to our technical report for further details on the keywords used for numerical question counting and typical examples of multi-step numerical questions.

**Model input data preparation.** Following a common practice of the literature (Yin et al. 2020; Liu, Wang, and Chen 2024; Wang et al. 2024), tables are flattened and converted into a sequence to form part of the prompts to the LLMs. Table cells are separated by '|' characters, while rows are separated by line breaks. Questions in natural language are added directly into the prompts, and answers (which are also in natural language) are only used for training the AnsSelector and the TwEvaluator modules. Details of the ground-truth construction for fine-tuning AnsSelector and TwEvaluator are included in our technical report.

For the AnsSelector and TwEvaluator LLMs, they take the question, the table titles and headers but not the contents as input, together with the answers and reasoning process, such that they can focus on analyzing the generated answers instead of attempting to answer the questions again. As mentioned earlier, the prompts for the LLMs of TabLaP, AnsSelector, and TwEvaluator can be found in our technical report.

**Competitors.** We compare our models TabLaP-EW (which uses the EW method for TwEvaluator) and TabLaP with three categories of baseline models including SOTA in each category: (1) Fine-tuned PLM-based models: TAPEX-Large (Liu et al. 2022) and OmniTab-Large (Jiang et al. 2022) (SOTA); (2) Zero-shot LLM-based models: GPT-3.5-Turbo (OpenAI 2024a), GPT-4o (OpenAI 2024b), and Mix-SC (Liu, Wang, and Chen 2024) (SOTA); (3) Few-shot LLM-based models: Binder (Cheng et al. 2023), DATER (Ye et al. 2023b), and Chain-of-Table (Wang et al. 2024) (SOTA). For Binder, DATER, and Chain-of-Tables, we use the best results reported in their papers. For the other models, we rerun the experiments on the three datasets (with fine-tuning if applicable).

**Implementation details.** We use Mix-SC for the SOTA branch of TabLaP and GPT-3.5 Turbo as the backbone model of NumSolver. We fine-tune the AnsSelector and TwEvaluator LLMs with the AdamW optimizer (Loshchilov and Hutter 2019) using a learning rate of 0.0002 and a weight decay 0.001. The maximum number of input tokens is 5,000, and the maximum number of epochs is 20. More details of the hyper-parameters are in our technical report.

**Evaluation metrics.** We report both the exact-match answer **Accuracy** for each model, as well as the accuracy (**TwAccuracy**) of the trustworthiness label generated by our TwEvaluator module. TwAccuracy is calculated as the number of times when the trustworthiness label is correctly predicted, divided by the total number of test instances.

All experiments are run with two NVIDIA A100 80 GB GPUs on a cloud GPU server.

## Results

**Overall results (Q1).** Table 2 reports the overall performance of the models. Our TabLaP models outperform all competitors on both datasets, improving TableQA accuracy by 5.7% and 5.8%, respectively. This confirms the effectiveness of our dual-branch model structure to exploit both a SOTA TableQA model and our NumSolver to yield accurate answers for more questions. We run chi-squared tests comparing TabLaP with SOTA (Mix-SC on WTQ and GPT-4o on FTQ), yielding p-val of 0.002 and 0.24, respectively. This confirms that the result on WTQ is statistically significant, while the larger p-val on FTQ is due to the smaller test set size. The accuracy of TabLaP-EW and TabLaP are the same because they share the answer generation modules and differ only in the TwEvaluator module (detailed next).

Among the baseline models, Mix-SC has the best overall results, for its self-consistency-based method to choose the most likely answer from multiple answers. On FTQ, GPT-4o is slightly better than Mix-SC, because Mix-SC uses a less advanced backbone, GPT-3.5 Turbo, while we have used its default prompts which might not be optimized for FTQ.

The PLM-based models are uncompetitive. Their smaller model sizes limited their semantic understanding capability.

The few-shot LLM-based models Binder, DATER, and Chain-of-Tables do *not* run directly on FTQ. Since they are

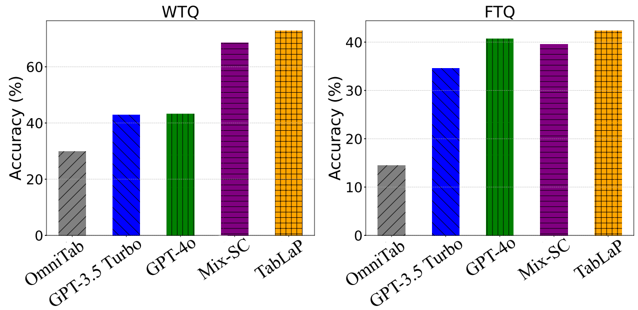



Figure 3: Model accuracy on numerical questions.

outperformed by Mix-SC, we did not adapt their implementation for comparison on FTQ.

**Effectiveness in tracking answer trustworthiness (Q2).** Table 2 also shows the accuracy (i.e., TwAccuracy) of TabLaP to report the trustworthiness of its answers. On WTQ, TwAccuracy of the TabLaP models is close to 80%, meaning that users can follow the trustworthiness labels to either consume or reject our models' answers, without any regret for four out of five questions asked to TabLaP on average.

On FTQ, TwAccuracy of the TabLaP models is not as high, because the dataset is more difficult, such that it is also challenging to predict the trustworthiness of the model answers. Note that TwAccuracy of TabLaP is now much higher than the answer accuracy (53.7% vs 44.1%), meaning that it is more beneficial to follow our trustworthiness labels than blindly trusting the answers.

None of the baseline models offer any trustworthiness labels and hence they do not have TwAccuracy results. If users simply accept the answers generated by these models, the TwAccuracy of these models will be the same as their answer accuracy. Let the *user regret ratio* of a model be $1 - TwAccuracy$. Then, the lowest regret ratios of the baselines are $1 - 72.5\% = 27.5\%$ and $1 - 41.7\% = 58.3\%$ on the two datasets, respectively, while those of TabLaP are $1 - 77.9\% = 22.1\%$ and $1 - 53.7\% = 46.3\%$, which are 19.6% and 20.6% lower, respectively.

**Performance on numerical questions (Q3).** Figure 3 reports accuracy on numerical questions. We show the results of the SOTA PLM-based model (OmniTab) and zero-shot LLMs (GPT-4o and Mix-SC). We also include GPT-3.5 Turbo since it is the backbone of our NumSolver.

TabLaP has the highest accuracy on both datasets, now outperforming the best baselines by 6.3% (72.8% vs. 68.5% of Mix-SC) and 6.8% (42.3% vs. 39.6% of GPT-4o) on the two datasets, respectively. This result confirms that exploiting the reasoning and planning capabilities of the backbone LLM to process numerical questions is more effective than prompting the LLM to generate the results directly.

**Ablation study (Q4).** We conduct an ablation study with four model variants: (1) NumSolver; (2) TabLaP-w/o-NumSolver, where NumSolver is replaced with GPT-3.5 Turbo; (3) TabLaP-w/o-AnsSelector, where AnsSelector is replaced with a random selection of the answers from the two branches of TabLaP; and (4) TabLaP.

As Table 3 shows, using just NumSolver causes substantial accuracy drops, as there are many non-numerical questions in the datasets which NumSolver is not designed for. Meanwhile, removing either NumSolver or AnsSelector from TabLaP also leads to lower accuracy. This confirms the effectiveness of both modules in contributing to the overall accuracy of TabLaP. We also replace our NumSolver and TwEvaluator with two out-of-box LLMs, Llama3-8B-Instruct and GPT4o-mini, which result in lower Accuracy as well (see our technical report), confirming the need for the fine-tuned modules.

| Model | Accuracy (%) WTQ | Accuracy (%) FTQ |
|---|---|---|
| NumSolver | 64.7 | 42.6 |
| TabLaP-w/o-NumSolver | 62.2 | 43.5 |
| TabLaP-w/o-AnsSelector | 68.5 | 42.2 |
| TabLaP | **76.6** | **44.1** |

Table 3: Ablation study results.

We then study the effectiveness of TwEvaluator with EW and MAB. As shown in Table 2, both TabLaP-EW and TabLaP (with MAB) share similar TwAccuracy. While TabLaP-EW is more accurate on FTQ, TabLaP outperforms on WTQ. This is because TabLaP-EW can estimate the accuracy of the TwEvaluator LLM quickly by the simple design of EW, which better suits the smaller test set of FTQ, while MAB takes more test instances to learn an (more precise) estimation of the accuracy of the TwEvaluator LLM – refer to our technical report for additional results.

**Case study.** In our technical report, we also include three typical numerical questions that fail the SOTA model Mix-SC, while TabLaP successfully answers those questions.

## Conclusion

We proposed TabLaP, an accurate and regret-aware multi-LLM-based model for the TableQA task. TabLaP uses an LLM as a planner to generate calculation plans for numerical questions, exploiting the reasoning capability of LLMs while avoiding their limitations in carrying out the actual calculations. TabLaP comes with a module based on multi-arm bandit to quantify the trustworthiness of the answers generated by the model, enabling users to consume the answers in a regret-aware manner for the first time. We verified the effectiveness of TabLaP on a public benchmark dataset WikiTableQuestions and an adapted dataset FTQ. The results show that TabLaP outperforms SOTA TableQA models in accuracy by 5.7% and 5.8% on the two datasets, respectively. Meanwhile, the answer trustworthiness labels generated by TabLaP help reduce the user regret ratio on consuming the model generated answers by 19.6% and 20.6% on the two datasets, respectively, compared with always trusting the model generated answers.

## Acknowledgments

This work is in part supported by the Australian Research Council (ARC) via Discovery Projects DP230101534 and DP240101006. Jianzhong Qi is supported by ARC Future

Fellowship FT240100170. Junhao Gan is in part supported by ARC Discovery Project DP230102908.

## References

Cafarella, M. J.; Halevy, A.; Wang, D. Z.; Wu, E.; and Zhang, Y. 2008. WebTables: Exploring the Power of Tables on the Web. *Proceedings of the VLDB Endowment*, 1(1): 538–549.

Chen, W.; Wang, H.; Chen, J.; Zhang, Y.; Wang, H.; Li, S.; Zhou, X.; and Wang, W. Y. 2020. TabFact: A Large-scale Dataset for Table-based Fact Verification. In *ICLR*.

Cheng, Z.; Xie, T.; Shi, P.; Li, C.; Nadkarni, R.; Hu, Y.; Xiong, C.; Radev, D.; Ostendorf, M.; Zettlemoyer, L.; Smith, N. A.; and Yu, T. 2023. Binding Language Models in Symbolic Languages. In *ICLR*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 4171–4186.

Didolkar, A. R.; Goyal, A.; Ke, N. R.; Guo, S.; Valko, M.; Lillicrap, T. P.; Rezende, D. J.; Bengio, Y.; Mozer, M. C.; and Arora, S. 2024. Metacognitive Capabilities of LLMs: An Exploration in Mathematical Problem Solving. In *AI for Math Workshop @ ICML*.

Frieder, S.; Pinchetti, L.; Griffiths, R.-R.; Salvatori, T.; Lukasiewicz, T.; Petersen, P.; and Berner, J. 2024. Mathematical Capabilities of ChatGPT. In *NeurIPS*, 27699–27744.

Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; and Zhou, J. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment*, 17(5): 1132–1145.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. PAL: Program-aided Language Models. In *ICML*, 10764–10799.

Herzig, J.; Nowak, P. K.; Müller, T.; Piccinno, F.; and Eisenschlos, J. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*, 4320–4333.

Hong, Z.; Yuan, Z.; Zhang, Q.; Chen, H.; Dong, J.; Huang, F.; and Huang, X. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*.

Jiang, Z.; Mao, Y.; He, P.; Neubig, G.; and Chen, W. 2022. OmniTab: Pretraining with Natural and Synthetic Data for Few-shot Table-based Question Answering. In *NAACL*, 932–942.

Jin, N.; Siebert, J.; Li, D.; and Chen, Q. 2022. A Survey on Table Question Answering: Recent Advances. In *China Conference on Knowledge Graph and Semantic Computing*, 174–186.

Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; and Zettlemoyer, L. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL*, 7871–7880.

Liu, Q.; Chen, B.; Guo, J.; Ziyadi, M.; Lin, Z.; Chen, W.; and Lou, J.-G. 2022. TAPEX: Table Pre-training via Learning a Neural SQL Executor. In *ICLR*.

Liu, T.; Wang, F.; and Chen, M. 2024. Rethinking Tabular Data Understanding with Large Language Models. In *NAACL*, 450–482.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *ICLR*.

Nan, L.; Hsieh, C.; Mao, Z.; Lin, X. V.; Verma, N.; Zhang, R.; Kryściński, W.; Schoelkopf, H.; Kong, R.; Tang, X.; et al. 2022. FeTaQA: Free-form Table Question Answering. *Transactions of the Association for Computational Linguistics*, 10: 35–49.

Neelakantan, A.; Le, Q. V.; Abadi, M.; McCallum, A.; and Amodei, D. 2017. Learning a Natural Language Interface with Neural Programmer. In *ICLR*.

Ni, A.; Iyer, S.; Radev, D.; Stoyanov, V.; Yih, W.-t.; Wang, S.; and Lin, X. V. 2023. LEVER: Learning to Verify Language-to-Code Generation with Execution. In *ICML*, 26106–26128.

OpenAI. 2024a. GPT-3.5 Turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo. Accessed: 2025-01-20.

OpenAI. 2024b. GPT-4o. https://openai.com/index/hello-gpt-4o/. Accessed: 2025-01-20.

Pasupat, P.; and Liang, P. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *ACL/IJCNLP*, 1470–1480.

Patnaik, S.; Changwal, H.; Aggarwal, M.; Bhatia, S.; Kumar, Y.; and Krishnamurthy, B. 2024. CABINET: Content Relevance-based Noise Reduction for Table Question Answering. In *ICLR*.

Pourreza, M.; and Rafiei, D. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *NeurIPS*, 359–371.

Slivkins, A. 2019. Introduction to Multi-Armed Bandits. *Foundations and Trends in Machine Learning*, 12(1-2): 1–286.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *NeurIPS*, 5998–6008.

Vermorel, J.; and Mohri, M. 2005. Multi-armed Bandit Algorithms and Empirical Evaluation. In *ECML*, 437–448.

Wang, Y.; Qi, J.; and Gan, J. 2024. Accurate and Regret-aware Numerical Problem Solver for Tabular Question Answering. Technical report, The University of Melbourne. https://arxiv.org/abs/2410.12846.

Wang, Z.; Dong, H.; Jia, R.; Li, J.; Fu, Z.; Han, S.; and Zhang, D. 2021. TUTA: Tree-Based Transformers for Generally Structured Table Pre-training. In *KDD*, 1780–1790.

Wang, Z.; Zhang, H.; Li, C.-L.; Eisenschlos, J. M.; Perot, V.; Wang, Z.; Miculicich, L.; Fujii, Y.; Shang, J.; Lee, C.-Y.; and Pfister, T. 2024. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. In *ICLR*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought

Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*, 24824–24837.

Ye, J.; Chen, X.; Xu, N.; Zu, C.; Shao, Z.; Liu, S.; Cui, Y.; Zhou, Z.; Gong, C.; Shen, Y.; Zhou, J.; Chen, S.; Gui, T.; Zhang, Q.; and Huang, X. 2023a. A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models. *CoRR*, abs/2303.10420.

Ye, Y.; Hui, B.; Yang, M.; Li, B.; Huang, F.; and Li, Y. 2023b. Large Language Models are Versatile Decomposers: Decomposing Evidence and Questions for Table-based Reasoning. In *SIGIR*, 174–184.

Yin, P.; Neubig, G.; Yih, W.-t.; and Riedel, S. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*, 8413–8426.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018a. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In *NAACL*, 588–594.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; and Radev, D. 2018b. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*, 3911–3921.

Zhang, B.; Zhou, K.; Wei, X.; Zhao, X.; Sha, J.; Wang, S.; and Wen, J.-R. 2023. Evaluating and Improving Tool-Augmented Computation-Intensive Math Reasoning. In *NeurIPS*, 21582–21597.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.