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 the 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 contains 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, while LLMs are known to struggle with such values. We aim to address this issue and answer numerical questions. We propose a model named TabLaP that uses LLMs as a planner rather than an answer generator, exploiting 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.

Introduction

Tables are a commonly used data format to organize and present information. *Table Question Answering* (TableQA), which aims to answer questions based on data in tables, arises as an important problem to automatically extract and analyze meaningful 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, that is, they are not necessarily relational tables, e.g., 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), such that 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 (Ye et al. 2023b; Cheng et al. 2023; Patnaik et al. 2024; Liu, Wang, and Chen 2024) use Large Language Models (LLMs), exploiting their semantic understanding capability 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 the design of our proposed solution is that while LLMs struggle with carrying out numerical calculations, they are capable of producing plans to execute such calculations. To see this, as shown in Figure 1c, using the same LLM but prompting it to generate a calculation plan expressed in Python, the Python code generated can actually 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 generated 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 *AnsSelecter*) – 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 the answers generated by 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) that together tracks the correctness of the answers from both branches of TabLaP over time and yields a trustworthiness flag of the final answer based on its source branch's accuracy. This trustworthiness flag enables users to consume the answers in a *regret-aware* manner.

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 a task which it is not good at.
- We propose NumSolver, an LLM-based module to process numerical questions. We further fine-tune an open-source LLM-based AnsSelecter module to decide whether to use the answer generated by NumSolver or an SOTA TableQA model (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 flag for the answers.
- We evaluate the performance of TabLaP on two real datasets, `WikiTableQuesetions` (Pasupat and Liang 2015) and `FTQ`. `WikiTableQuesetions` is a public benchmark dataset, while `FTQ` is adapted by us from the `FeTaQA` dataset (Nan et al. 2022) by removing the 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 flags generated by TabLaP help reduce the user regret ratio on using the model generated answers by 30.9% and 20.6% on the two datasets, respectively, compared with always trusting the model generated answers.

Related Work

TableQA has attracted much attention in recent years. Existing research is mainly driven by studies to design 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) which machines can easily understand and execute. There are two sub-categories of methods: (i) *weakly-supervised* (Pasupat and Liang 2015; Yu et al. 2018a; Neelakantan et al. 2017), and (ii) *fully-supervised*, such as NL-to-SQL (Zhong, Xiong, and Socher 2017; Pourreza and Rafiei 2023; Gao et al. 2024). In weakly supervised methods, the 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).

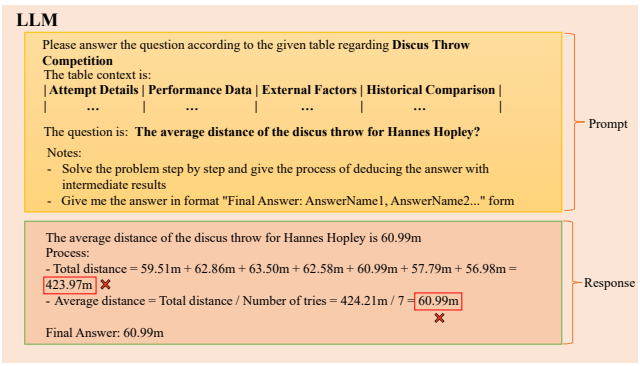
PLM-based methods. Language model-based methods, including PLM-based ones, focus on guiding the models to understand tabular data. There are two directions: (i) Tailoring Transfomer (Vaswani et al. 2017) model structure

Table Title: Hannes Hopley Discus Throw

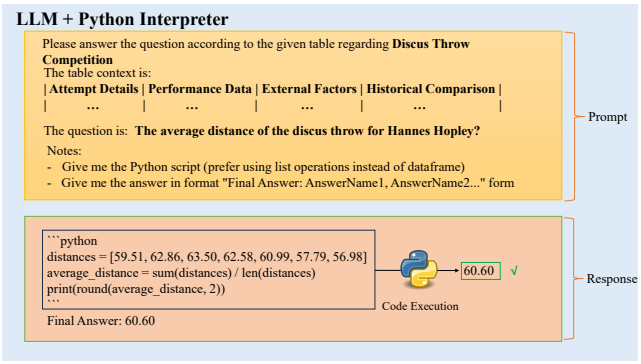
Attempt Details	Performance Data	External Factors	Historical Comparison
1st Try	Distance: **59.51m** Accuracy: 85%	Wind: Light breeze Temperature: 22 °C	Last Competition: 60.02m Average: 59.80m
2nd Try	Distance: **62.86m** Rotation Speed: 45 rpm	Wind: Slight increase, crosswind at 2 m/s	Last Competition: 61.30m Personal Best: 62.90m
3rd Try	Distance: **63.50m** Release Angle: 35 °	Wind: Calm Humidity: 60%	Last Competition: 62.00m Season Best: 63.00m
4th Try	Distance: **62.58m** Power Output: 92%	Wind: Crosswind increased, slightly affecting the throw	Last Competition: 62.70m Average: 62.50m
5th Try	Distance: **60.99m** Fatigue Level: High	Wind: Gusty, challenging conditions	Last Competition: 61.80m Below Average
6th Try	Distance: **57.79m** Form Stability: Poor	Wind: Strong gusts	Last Competition: 58.50m Worst Attempt: 57.50m
7th Try	Distance: **56.98m**	Wind: Unfavorable	Last Competition: 59.00m

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

(a)



(b)



(c)

Figure 1: 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.

for better tabular data understanding. For example, TAPAS (Herzig et al. 2020) masks cells in the tables and extends the BERT (Devlin et al. 2019) structure 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 *verti-*

cal self-attention and combines text and table representations in a unified framework. (ii) Pre-training and fine-tuning models for end-to-end TableQA problem solving. For example, TAPEX (Liu et al. 2022) pre-trains BART (Lewis et al. 2020) with a large synthetic dataset derived from the `WikiTableQuestions` dataset (Pasupat and Liang 2015). OmniTab (Jiang et al. 2022) also leverages BART as its backbone model, while it is pre-trained using both real and synthetic data, including SQL queries from the Spider dataset (Yu et al. 2018b) and synthetic natural language sentences obtained by using their proposed SQL2NL model to convert the SQL queries in the Spider dataset to natural language sentences, to enhance its capability to perform few-shot learning for the TableQA task. These PLM-based solutions have relatively small model sizes comparing with the latest LLM-based ones, which limit their capabilities in understanding the semantics of tabular data, leading to sub-optimal TableQA results, as shown in our experimental study.

LLM-based methods. More recent studies exploit the strong capabilities of LLMs in semantic understanding and tracking long contexts (i.e., large tables).

In-context learning is a common approach, 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. CABINET (Patnaik et al. 2024) proposes a content relevance-based approach that filters out irrelevant rows or columns to reduce the amount of data to be examined by the LLM.

Mix-SC (Liu, Wang, and Chen 2024), the SOTA model, generates a few answers for each question exploiting the stochastic nature of answers generated by an LLM. It uses two types of prompts: prompting the LLM to run as a Python agent to execute Python scripts directly and Chain-of-Thoughts prompt to ask the LLM 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 TabLaP. 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 the LLM 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 numerical/mathematical calculations are a known issue with LLMs (Frieder et al. 2024; Didolkar 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 (Zhang et al. 2023; Gao et al. 2023) 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 calculation 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 which 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 the token-based comparison between the answer generated by the model and the ground truth.

In this paper, we are particular 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 on handling this type of questions.

Model Overview

There are three stages in the question answering process of our TabLaP : (i) *answer generation*, (ii) *answer selection*, and (iii) *answer trustworthiness evaluation*.

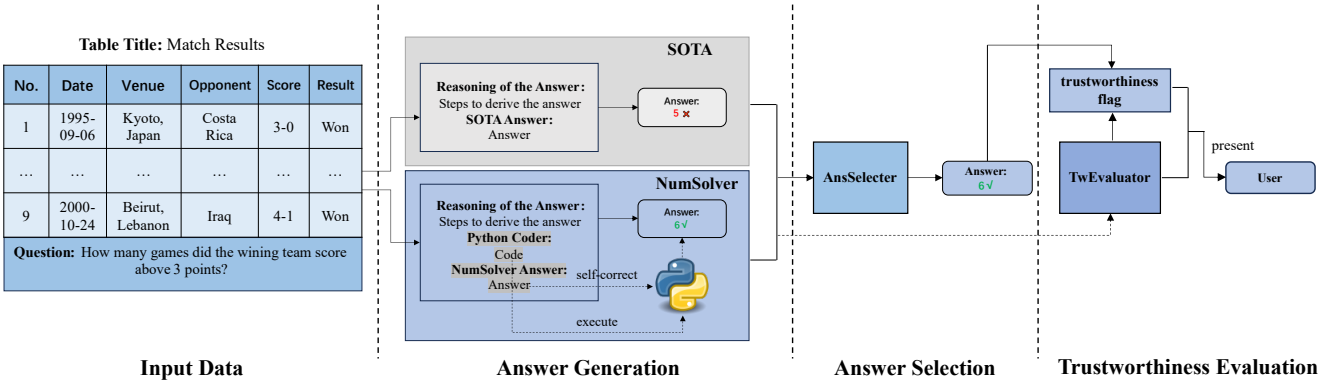


Figure 2: Overview of TabLaP. Our TabLaP model has three stages: answer generation, answer selection, and trustworthiness evaluation. (i) The answer generation stage uses both an SOTA TableQA model (Liu, Wang, and Chen 2024) and our NumSolver module to generate answers for an input question and a table,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

In the answer generation stage, two models are adopted. One is an SOTA TableQA model (Liu, Wang, and Chen 2024) (detailed in the Related Work), denoted by M_{sota}, and the other is our numerical question answering module NumSolver (detailed in the next subsection). These two models take the input pair (T, Q) and individually generate answers (together with their reasoning steps) for the question Q.

In the answer selection stage, our AnsSelecter module (detailed in the Answer Selection Stage) takes the input table T and the two answers along with the corresponding reasoning steps provided by the two answer generation models in the previous stage. Then AnsSelecter decides and returns an answer from the two candidates.

In the answer trustworthiness evaluation stage, our TwEvaluator (detailed in the Answer Trustworthiness Evaluation) tracks the reliablity of the answers generated by the two answer generation models and produces a trustworthiness flag for the returned answer in the answer selection stage.

Answer Generation Stage

In this stage, we use two models: NumSolver and an SOTA model (Mix-SC), as two separate branches to answer the question. The details of Mix-SC is explained in the Related Work section.

In NumSolver, we force a backbone LLM to answer an input question Q step by step and generate the reasoning process with the intermediate results. Besides, we also ask the LLM to write down the Python Scripts, which can be used to answer the question, and the answer directly obtained through the table reasoning. By executing the Python scripts with a Python Interpreter, question answers are obtained which are found more accurate, especially for those multi-step numerical questions, than those obtained with the direct reasoning methods. When the Python-execution answers are different from the answers generated by the LLM directly and they contain numerical values, priority is given to them. When errors occur during executing the Python scripts, the answers obtained directly through reasoning are used as a backup. This process is called *self-correctness*.

Algorithm 1: Table Question Answering with TabLaP

Require: Flattened table T, questions Q, answer A, answer generation models M_{sota} and NumSolver (NS), classifier AnsSelecter, answer evaluator TwEvaluator

1: **for** each triplet (T, Q, A) **do**
2: Generate prompt_{SOTA}, prompt_{NS} for M_{sota} and NumSolver, respectively
3: Obtain Answer and Reasoning pairs $(\text{Ans}_A, \text{Rsn}_A)$, $(\text{Ans}_B, \text{Rsn}_B)$ by feeding prompt_{SOTA}, prompt_{NS} to $M_{SOTA}, NumSolver$
4: **Selection by AnsSelecter:**
(1) Combine table structure T with $(\text{Ans}_A, \text{Rsn}_A)$, $(\text{Ans}_B, \text{Rsn}_B)$ to form classification $\text{prompt}_{\text{cls}}$
(2) Get the best answer
$\text{Ans}_{\text{best}} \leftarrow \text{AnsSelecter}(\text{prompt}_{\text{cls}})$
5: **Verification by TwEvaluator:**
(1) Combine table structure T with $(\text{Ans}_A, \text{Rsn}_A)$, $(\text{Ans}_B, \text{Rsn}_B)$ to form verification $\text{prompt}_{\text{verif}}$
(2) Determine whether Q can be correctly answered, $\text{Label}_{\text{true/false}} \leftarrow \text{TwEvaluator}(\text{prompt}_{\text{verif}})$
6: **end for**

Answer Selection Stage

In this stage, AnsSelecter utilizes the answers generated by the two branches in the answer generation stage to decide the best solution for the given problem. To achieve this, AnsSelecter first analyzes the reasoning steps of the two models, and then makes a decision based on the question, generated answers and their corresponding reasoning process.

Answer Trustworthiness Evaluation

In the trustworthiness evaluation stage, unlike the existing TableQA models, TwEvaluator returns a trustworthiness flag which indicates whether TabLaP is able to provide a reliable answer to the given question. TwEvaluator determines the reliability of the answer by analyzing the logical rationality and the accuracy of the models. When the returned flag suggests that the answer provided by TabLaP is reliable, then users can use the answer given by the AnsSelecter with certain confidence. Conversely, when the returned label suggests that TabLaP lacks ability to solve the problem, users will be informed that the answer to the question might be wrong.

To enhance the reliability of TwEvaluator, we incorporate two methods: the Expanding Window method and the Multi-arm Bandits with Upper Confidence Bound (UCB) (Slivkins 2019). The Expanding Window method involves initially establishing a set of cases to calculate the accuracy of TwEvaluator's results and then using this accuracy to determine the probability of accepting TwEvaluator's output. As new cases are encountered, the probability is updated accordingly with respect to the accumulating data over time. On the other hand, the Multi-arm Bandits with UCB method balances the exploration and exploitation by selecting actions (in this context, decisions regarding TwEvaluator's outputs) that maximize the expected reward with consideration on the uncertainty of the outcomes. The UCB component specifically allows the model to prioritize actions that are less certain but potentially more rewarding, thus improving the overall performance and the robustness of TwEvaluator in providing accurate and reliable answers.

Fine-tuning for AnsSelecter and TwEvaluator

We adopt Low-Rank Adaptation (LoRA) (Hu et al. 2022), to fine-tune our AnsSelecter and TwEvaluator with Llama3-8B-Instruct as a backbone model. LoRA is a parameter-efficient fine-tuning method for low-rank decomposition of the gradient update matrices in transformers. The effectiveness of LoRA is proven by (Aghajanyan, Gupta, and Zettlemoyer 2021) that pre-trained language models can still learn efficiently despite a random projection to a smaller subspace. Thus, the forward pass for both AnsSelecter and TwEvaluator can be represented as:

$$\begin{aligned}\mathbf{h} &= \underbrace{\mathbf{W}_0\mathbf{x}_{(T,Q,A)} + \Delta\mathbf{W}\mathbf{x}_{(T,Q,A)}}_{\text{gradient descent update}} \\ &= \underbrace{\mathbf{W}_0\mathbf{x}_{(T,Q,A)} + \mathbf{A}\mathbf{B}\mathbf{x}_{(T,Q,A)}}_{\text{LoRA update}} \end{aligned} \tag{1}$$

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

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

$$\mathcal{L}_{cls} = -\sum_{i=1}^{T} y_{m,i}\log(\hat{y}_{m,i}) \tag{2}$$

$$\mathcal{L}_{verif} = -\sum_{i=1}^{V} y_{n,i}\log(\hat{y}_{n,i}) \tag{3}$$

$$\begin{aligned}\mathcal{L}_{total} &= -\log(\hat{y}_{m,y_m}) - \alpha\log(\hat{y}_{n,y_n}) \\ &= \mathcal{L}_{cls} + \alpha\mathcal{L}_{verif}\end{aligned} \tag{4}$$

where $y_{m,i}$ and $y_{n,i}$ are the ground-truth labels for the mth and nth tokens at position i in one-hot encodings. $\hat{y}_{m,i}$ and $\hat{y}_{n,i}$ are the model predicted probabilities for the mth and nth tokens at position i. T and V are the dictionary lengths of the AnsSelecter and TwEvaluator, respectively. α is a coefficient used to control the influence of TwEvaluator on the result, and its default value is 1.

Dataset	# QA Pairs		# Numerical Questions		Avg # Tokens	Avg # Tokens
	Training	Testing	Training	Testing	per Table	per Answer
WTQ	11,321	4,344	5,461	2,148	2,083.5	1.7
FTQ	2,000	1,245	417	182	1,007.7	5.1

Table 1: Dataset statistics. The number of QA pairs includes both numerical and non-numerical questions.

Experiments and Results

This section presents experimental results to verify the effectiveness of TabLaP. We aim to answer the following questions: **(Q1)** How does TabLaP compare with SOTA models and 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 does 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), to showcase the applicability of TabLaP across datasets.

`FeTaQA` is also a TableQA dataset. Its answers contain descriptive contents that may not be directly relevant to the questions. We remove such noisy information from the answers and only retain the entities that answer the questions, to form the `FTQ` dataset. Appendix A.1 shows 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 by the authors to produce the final question-answer pairs in FTQ. Appendix A.1 shows the prompt used by GPT-4o to extract the entities and further details on the construction of FTQ.

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). 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. Appendix A.2 shows further details on the keywords used for numerical question counting, and Appendix A.3 shows typical examples of multi-step numerical questions.

	Model	Accuracy (%)		TwAccuracy (%)	
		WTQ	FTQ	WTQ	FTQ
Fine-tuned	TAPEX-Large	57.5	19.8	–	–
	OmniTab-Large	63.3	25.2	–	–
Zero-shot	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	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
	Δ	+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>; Δ (%) denotes the performance gain of TabLaP comparing with the best baseline results.

Model input data preparation. Following a common practice of the literature (Yin et al. 2020; Wang et al. 2024; Liu, Wang, and Chen 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 AnsSelecter and the TwEvaluator modules. Details of the ground-truth construction for fine-tuning AnsSelecter and TwEvaluator are included in Appendix A.4.

For AnsSelecter and TwEvaluator, they take 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.

The prompts for TabLaP, AnsSelecter, and TwEvaluator are shown in Appendices A.5, A.6, and A.7, respectively.

Competitors. We compare our models TabLaP-EW 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) (STOA). For Binder, DATER, and Chain-of-Tables, we use the results reported in their papers. For the other models, we rerun the experiments on our two 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 use Llama3-8B-Instruct as the backbone model for AnsSelecter and TwEvaluator. We fine-tune these modules 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 shown in Appendix A.8.

Evaluation metrics. We report both the exact-match answer **Accuracy** for each model, as well as the accuracy (**TwAccuracy**) of the trustworthiness flag generated by our TwEvaluator module. TwAccuracy is calculated as the number of times when the trustworthiness flag 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 computer.

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 an SOTA TableQA model and our NumSolver to yield accurate answers for more questions. We note that the accuracy of TabLaP-EW and TabLaP is 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 outperformed by Mix-SC, we did not adapt their implementation for 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 flags to either consume or reject our models answers, without any regret for four out of five questions asked to TabLaP on average.

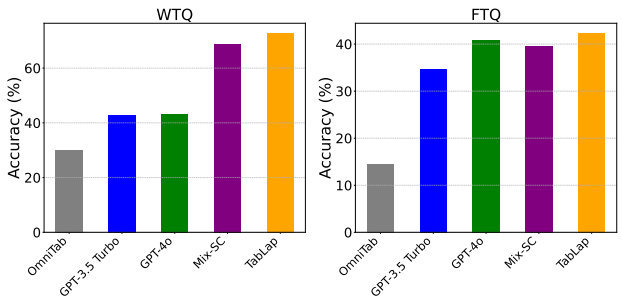


Figure 3: Model accuracy on numerical questions.

On FTQ, TwAccuracy of the TabLaP models is not as high, because the dataset is very difficult as discussed above, 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 flags than blindly trusting the answers.

None of the baseline models offer any trustworthiness flags 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 $1 - 72.5\% = 17.5\%$ and $1 - 41.7\% = 58.3\%$ on the two datasets, respectively, while those of TabLaP are $1 - 77.9\% = 12.1\%$ and $1 - 53.7\% = 46.3\%$, which are 30.9% and 20.6% lower, respectively.

Performance on numerical questions (Q3). Figure 3 reports the 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 6.1% (72.8% vs. 68.5% of Mix-SC) and 7.0% (42.3% vs. 39.6% of GPT-4o) on the two datasets, respectively. This result confirms that exploiting the reasoning and planning capability of the backbone LLM to process numerical questions is a more effective approach than prompting the LLM to generate the calculation results directly.

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

Table 3: Ablation study results.

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-AnsSelecter, where AnsSelecter is replaced with a random selection of the answers from the two branches of TabLaP; (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 good at. Meanwhile, removing either NumSolver or AnsSelecter from TabLaP also leads to lower accuracy. This confirms the effectiveness of both modules in contributing to the overall accuracy of TabLaP.

We then study the effectiveness of TwEvaluator with the expanding window (EW) technique and the MAB. As shown in Table 2, both TabLaP-EW and TabLaP(with MAB) share similar TwAccuracy. While TabLaP-EW is more accuracy on FTQ, TabLaP outperforms on WTQ. This is because TabLaP-EW can estimate the accuracy of the LLM model used for the TwEvaluator module quickly by the simple design of the EW technique, 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 LLM model used for TwEvaluator.

Case study. In Appendix A.9, we show 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 flags generated by TabLaP helps reduce the user regret ratio on using the model generated answers by 30.9% and 20.6% on the two datasets, respectively, compared with always trusting the model generated answers.

References

Aghajanyan, A.; Gupta, S.; and Zettlemoyer, L. 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In *ACL/IJCNLP*, 7319–7328.

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.

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. Large language model.

OpenAI. 2024b. GPT-4o. https://openai.com/index/hello-gpt-4o/. Large language model.

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, 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.

Appendix A

A.1

Additional Dataset Details Our dataset FTQ is a cleaned-up version of the FeTaQA dataset (Nan et al. 2022) where we remove tokens from the ground-truth answers that are not directly relevant to the questions. This makes automatic evaluation of model accuracy on the dataset easier and more precise (instead of using fuzzy metrics such as the ROUGE score). We use GPT-4o to extract key entities from the answers that are relevant to the questions, which recorded an accuracy of 67.7%. We manually check the extracted entities and correct them if necessary to ensure the data quality. The prompt used is shown in Figure 4.

The original FeTaQA dataset contains questions on Wikipedia tables where the ground-truth answers are in free-form text, with an average of 18.9 tokens per answer. After our clean-up process, the ground-truth answers in our FTQ dataset consist of multiple entities that directly answer the questions, with an average of 5.1 tokens per answer. Table 4 shows an example of a question and its answer in FeTaQA and FTQ.

```
Please filter out entities that can answer the
question from the given answer sentence.
The question is: [QUESTION]
The answer is: [ANSWER]

Notes:
- Give me the entities in format "Final Answer:
AnswerName1, AnswerName2..." form
- Keep entities as short as possible and don't include
any explanations
- Try not to include the subject of the question in
entities
```

Figure 4: Prompt for GPT-4o to extract key entities from the FeTaQA dataset.

Dataset	Answer
FeTaQA	In 2019, Shagun Sharma played in the roles as Pernia in Laal Ishq, Vikram Betaal Ki Rahasya Gatha as Rukmani/Kashi and Shaadi Ke Siyape as Dua.
FTQ	Laal Ishq, Vikram Betaal Ki Rahasya Gatha, Shaadi Ke Siyape

Table 4: Comparison of answers in FeTaQA and our FTQ datasets, for the question "What TV shows was Shagun Sharma seen in 2019?".

Figure 5 shows the distribution of the table length (i.e., number of tokens per table) of both the WTQ and FTQ datasets. The tables in WTQ are larger, about half of which have over 1,000 tokens, while 65.8% of the tables in FTQ have fewer than 1,000 tokens.

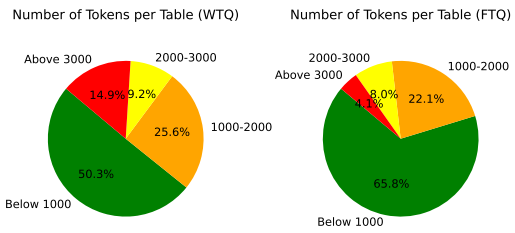


Figure 5: Number of tokens per table of the WTQ and FTQ datasets.

A.2

Numerical Question Filtering We collect frequently used keywords for the numerical questions, including "how many", "number", "the most", "difference", "count", "highest", "average", "at least", "rank", "lowest", "percentage", "sum", "compare", and "frequency" to extract potential numerical questions. The keyword counts are shown in Table 5.

A.3

Examples of Multi-hop Numerical Problems Below, we show three typical types of multi-hop numerical questions found in the datasets, the answers of which cannot be obtained through a one-step calculation.

1. Ordering comparison after some calculation:
   Which country had the most riders that placed in the top 20 of the 1971 trans-ama final standings?
2. Difference comparison after some calculation:
   What's the difference between the highest score in a game and the lowest score in a game?
3. Average (i.e., sum and division) calculation:
   What is the average number of draws?

Among the WTQ and FTQ datasets, there are 814 and 16 Type-1 questions, 410 and 4 Type-2 questions, and 153 and 10 Type-3 questions, respectively.

A.4

Ground-truth Label Construction for Training AnsSelecter and TwEvaluator For AnsSelecter and TwEvaluator training, we collect the training output of both model branches (i.e., SOTA TableQA model and our NumSolver). Given a training instance, if only one model branch yields the correct result, its ground-truth label for AnsSelecter is that branch, and the ground-truth label for TwEvaluator is a [True] flag (detailed later). If both branches return the correct answer, we randomly label one of the branches as the ground-truth label for the AnsSelecter based on the overall accuracy of the two branches, while the ground-truth label for TwEvaluator is again the [True] flag. If both branches return a wrong answer, we discard this training instance when training AnsSelecter, while the ground-truth label of this instance for TwEvaluator is [False].

A.5

Prompt for NumSolver For NumSolver, we ask GPT-3.5 Turbo to generate the reasoning process, Python script, and

Dataset	Numerical question related keywords													
	how many	number	the most	difference	count	highest	average	at least	rank	lowest	percentage	sum	compare	frequency
WTQ	4,097	1,912	814	410	399	215	153	149	145	81	73	48	16	4
FTQ	369	43	16	4	12	18	10	3	13	3	51	43	26	-

Table 5: Numerical question related keyword counts of the WTQ and FTQ datasets.

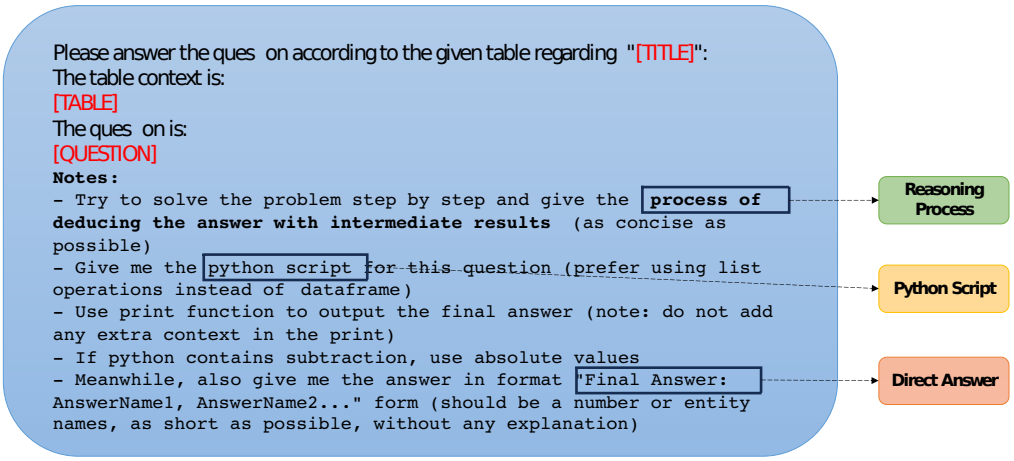


Figure 6: Prompt for the NumSolver.

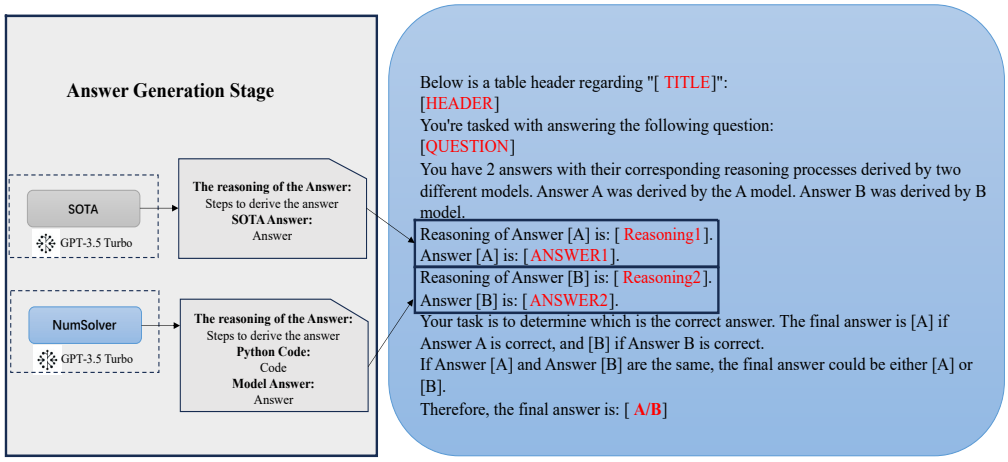


Figure 7: Prompt for the AnsSelecter.

a question answer (obtained with Chain-of-Thought). The prompt template is shown in Figure 6.

A.6

Prompt for AnsSelecter For AnsSelecter, we prompt a fine-tuned Llama3-8B-Instruct with the reasoning process obtained from the answer generation stage, the answers, and the table information. The prompt template is shown in Figure 7. AnsSelecter returns a label of either [A] or [B], indicating that the answer from the SOTA TableQA branch or our NumSolver is preferred, respectively.

A.7

TwEvaluator We provide additional details about TwEvaluator in this section, including the prompt template used by the LLM of TwEvaluator, design details of the module, and additional experimental results for the module.

Prompt. The LLM (i.e., a fine-tuned Llama3-8B-Instruct) of TwEvaluator uses a similar input to that of AnsSelecter, as shown in Figure 8. It returns an answer of either [True] or [False], indicating whether TabLaP has answered the input question correctly.

Design details of TwEvaluator. Empirically, we observe that using yet another LLM to evaluate the outcome of AnsSelecter (which is also an LLM) is highly accurate when it predicts [True] but is less accurate when it predicts

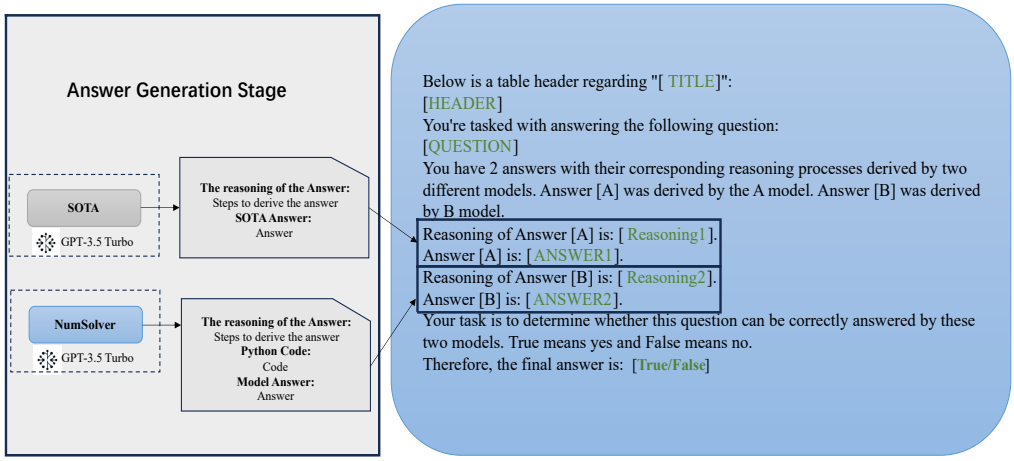


Figure 8: Prompt for the TwEvaluator.

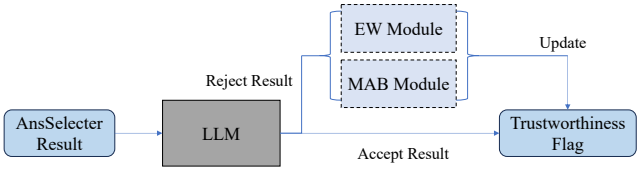


Figure 9: Structure of the TwEvaluator module.

`[False]`. This means that when the TwEvaluator LLM predicts `[True]`, users can trust the answers given by TabLaP with little regret. On the other hand, if the TwEvaluator LLM predicts `[False]`, TabLaP might have actually produced a correct answer which should not be dismissed.

To recover from the `[False]` cases of the TwEvaluator LLM, we attach to it a *rejection filter* that filters potential false rejections of the TwEvaluator LLM, as illustrated by Figure 9. For efficiency, we use two simple probabilistic models for the implementation of the rejection filter – *Expanding Window* (EW) and *Multi-armed Bandits* (MAB).

The Expanding Window method. The EW method starts by accepting the TwEvaluator LLM predictions for t TableQA instances, to calculate an initial accuracy of the TwEvaluator LLM, denoted by $A(t)$:

$$A(t) = \frac{\text{num\_correct}(M_{Tw}, t)}{t}. \tag{5}$$

Here, num_correct(M_{Tw}, t) denotes the number of correct predictions made by the TwEvaluator LLM for the t test instances, $\mathcal{M}_{Tw}$, and t is system parameter. We empirically set the value of t as 100 and 50 for the `WTQ` and `FTQ` datasets, respectively.

From the $(t+1)$-th test instance, the EW method rejects the TwEvaluatorLLM's `[FALSE]` prediction with a probability of $P(t) = 1 - A(t)$, while $A(t+1)$ is updated to:

$$A(t+1) = \frac{\text{num\_correct}(M_{Tw}, t+1)}{t+1}. \tag{6}$$

The Multi-armed Bandit 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 a `[FALSE]` prediction of the TwEvaluator LLM. We aim to select the arm that maximizes the TwAccuracy of TabLaP, as guided by the equation below:

$$\hat{\mu}_i(t) = \frac{\sum_{n=1}^{t-1} r_i(n) \cdot \mathbb{I}(a(n) = i)}{N_i(t-1)}, \tag{7}$$

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. As we aim to maximize the TwAccuracy of TwEvaluator, we set the reward $r_i(n)$ as 1 if choosing arm i is consistent with the ground-truth, and -1 otherwise.

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:

$$UCB_i(t) = \hat{\mu}_i(t) + c \cdot \sqrt{\frac{\ln t}{N_i(t)}}, \tag{8}$$

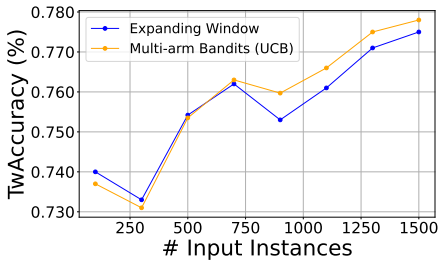
where $UCB_i(t)$ is the UCB for arm i at time t; c is the exploration parameter that controls the balance between exploration and exploitation (we empirically set $c = \sqrt{2}$); and ln is the natural logarithm function. For arm selection, we use:

$$a(t) = \arg\max_i \left(UCB_i(t)\right), \tag{9}$$

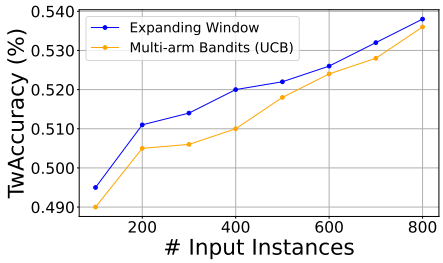
where $a(t)$ is the arm to be selected at time t, when TwEvaluator LLM predicts `[FALSE]` for test instance t. The MAB method accepts or rejects the prediction following $a(t)$.

Effectiveness of EW and MAB. Figure 10 shows the performance of TwEvaluator using the EW and the MAB methods, respectively. The EW method helps achieve high TwAccuracy with fewer test instances, while the TwAccuracy of MAB grows slower but may reach even higher values when

more test instances are seen. This is consistent with the design goal of the MAB, i.e., to obtain overall better results through a progressive exploration and exploitation process. This result also suggests an optimization opportunity: to start with EW while training MAB in the background; once MAB has gained higher accuracy, we switch to it.



(a) WTQ



(b) FTQ

Figure 10: TwEvaluator's TwAccuracy with the EW and MAB methods over different numbers of test instances.

A.8

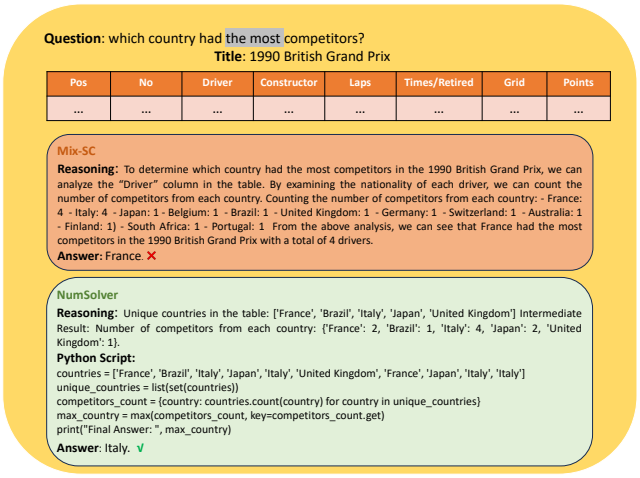
Model Fine-tuning Hyper-parameters Table 6 summarizes the hyper-parameter values used for training AnsSelecter and (the LLM of) TwEvaluator. We share the parameter values between both modules for consistency and reproducibility.

Hyper-parameter	Value
lora_alpha	64
r	16
optimizer	paged_adamw_32bit
learning rate	2e-4
weight decay	0.001
$max.$ gradient norm	0.3
warm-up ratio	0.03

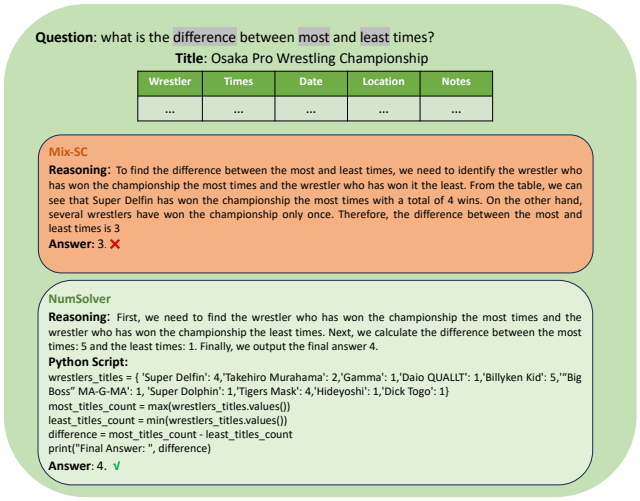
Table 6: Module fine-tuning hyper-parameters for AnsSelecter and TwEvaluator.

A.9

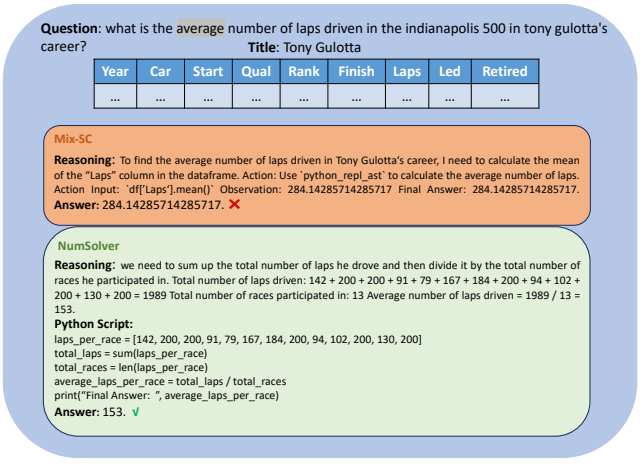
Case Study Three examples of multi-hop numerical questions are shown in Figure 11. The SOTA TableQA model, Mix-SC, failed to answer all these examples, while our NumSolver has successfully answered them.



(a) A question to compare order after summation.



(b) A question to calculate difference after summation.



(c) A question to calculate average (i.e., summation and division).

Figure 11: Three examples of multi-hop numerical questions and answers generated by the SOTA model Mix-SC and by our NumSolver.