Rethinking Table Pruning in TableQA: From Sequential Revisions to Gold Trajectory-Supervised Parallel Search

Yu Guo[1*], Shenghao Ye[1*], Shuangwu Chen[1†], Zijian Wen[1], Tao Zhang[1], Qirui Bai[1]
Dong Jin[2†], Yunpeng Hou[2], Huasen He[1], Jian Yang[1], Xiaobin Tan[1]
[1]University of Science and Technology of China
[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
{yukariguo, ssh0321y, wzj20020304, zhangtaolqy, bqr135, hyp314}@mail.ustc.edu.cn
{chensw, kingdon, hehuasen, jianyang, xbtan}@ustc.edu.cn

Abstract

Table Question Answering (TableQA) benefits significantly from table pruning, which extracts compact sub-tables by eliminating redundant cells to streamline downstream reasoning. However, existing pruning methods typically rely on sequential revisions driven by unreliable critique signals, often failing to detect the loss of answer-critical data. To address this limitation, we propose TabTrim, a novel table pruning framework which transforms table pruning from sequential revisions to gold trajectory-supervised parallel search. TabTrim derives a gold pruning trajectory using the intermediate sub-tables in the execution process of gold SQL queries, and trains a pruner and a verifier to make the step-wise pruning result align with the gold pruning trajectory. During inference, TabTrim performs parallel search to explore multiple candidate pruning trajectories and identify the optimal sub-table. Extensive experiments demonstrate that TabTrim achieves state-of-the-art performance across diverse tabular reasoning tasks: TabTrim-8B reaches 73.5% average accuracy, outperforming the strongest baseline by 3.2%, including 79.4% on WikiTQ and 61.2% on TableBench.

1 Introduction

Table Question Answering (TableQA), which enables users to retrieve information from tabular data via natural language (Zhang et al., 2025), remains a non-trivial challenge, as numerous irrelevant rows and columns in raw tables often disrupt reasoning process (Chen et al., 2024). Against this backdrop, table pruning (Wu et al., 2023; Lin et al., 2023), which seeks to extract compact sub-tables by removing redundant cells, has emerged as a promising approach to streamline downstream TableQA.

Existing pruning methods generally fall into two paradigms, as illustrated in Fig.1. Program-based

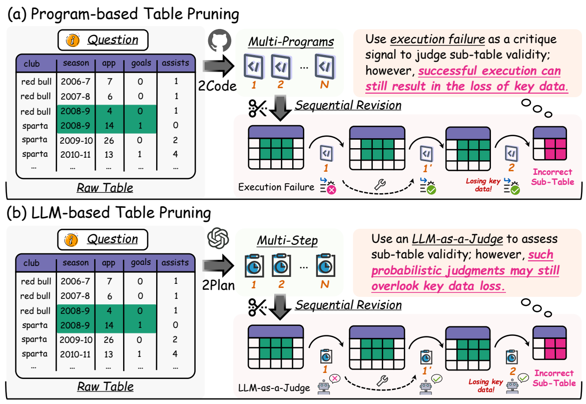

Figure 1: Illustration of (a) Program-based and (b) LLM-based pruning paradigms. ▬ represents table headers, ▬ represents gold cells related to answer and ▬ represents incomplete gold cells.

methods (Cheng et al., 2023; Zhang et al., 2024; Nahid and Rafiei, 2024) parse input query and invoke executable programs (e.g., SQL and Python) to convert pruning tasks into procedural operations. In contrast, LLM-based methods (Ye et al., 2023; Wang et al., 2024) use Chain-of-Thought or multi-agent planning to perform pruning through multi-step reasoning. Despite promising results, both paradigms are prone to errors in pruning, particularly for intricate queries or tables where the implicit associations between query semantics and tabular data become obscure. These errors may propagate and accumulate in multi-step pruning, significantly degrading TableQA performance (Yu et al., 2025).

To tackle this issue, recent studies introduce critique signals to refine pruning decisions at each step for both paradigms. For program-based pruning, execution failures are used to trigger program repair or regeneration (Jin et al., 2025). They can capture syntactic or runtime errors yet scarcely pinpoint semantic errors, as a program may execute successfully but remove answer-critical data. For LLM-based pruning, an LLM-as-a-Judge is of-

[*]Equal contribution
[†]Corresponding authors

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12960–12976
July 2-7, 2026 ©2026 Association for Computational Linguistics

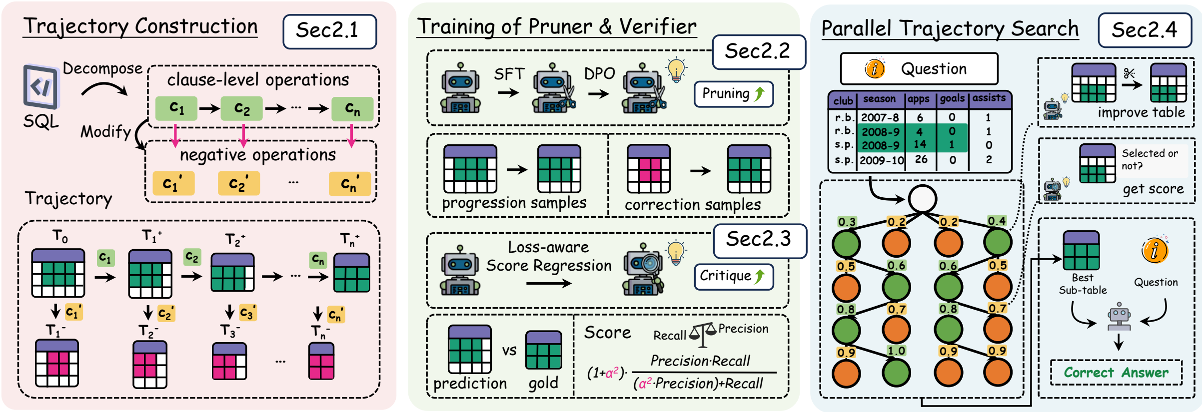

Figure 2: Overview of TabTrim. TabTrim constructs gold sub-table trajectories to provide supervision for two core components: a Trajectory-supervised Pruner and a Loss-aware Verifier. During inference, TabTrim performs Parallel Trajectory Search with a step-wise generate–score–select procedure: the pruner generates candidate pruned sub-tables, the verifier assesses sub-tables with loss-aware scores, and the search retains the top candidates, finally outputting the highest-scoring sub-table for downstream reasoning.

ten employed to check whether the pruned cells are redundant at each reasoning step (Yu et al., 2025). However, mandating LLMs to perform self-reflection readily induces subjective bias, as models tend to either rationalize their erroneous reasoning or over-criticize their valid steps (Huang et al., 2024), leading to inconsistent judgments.

Rethinking table pruning in TableQA, we attribute these limitations to two fundamental reasons: **(1) Unreliable Critique.** Without an explicit objective grounding, existing critique mechanisms can barely verify whether we have over-pruned critical cells or under-pruned redundant content at each step, thus undermining the reliability of table pruning. **(2) Sequential Revision.** Existing multi-step table pruning is inherently a sequential revision process that iteratively refines pruning results based on the reflection of preceding iterations. Such refinement is usually confined to a single pruning trajectory, unable to backtrack and explore alternatives, thereby readily getting trapped in suboptimal pruning results.

To address these challenges, we identify two key insights: (1) The intermediate sub-tables yielded from gold SQL execution for Text-to-SQL tasks are grounded on verified query results, and thus can serve as a reliable gold pruning trajectory for critiquing each pruning step. (2) By generating multiple candidate pruning trajectories in parallel and searching for the optimal one, the exploration space can be effectively expanded, thereby avoiding local optima.

In light of the above insights, we propose **TabTrim**, a novel gold trajectory-supervised parallel search framework for table pruning. Specifically, TabTrim derives a gold pruning trajectory using the intermediate sub-tables in the execution process of gold SQL queries, and trains a pruner and a verifier to make the step-wise pruning result align with the gold pruning trajectory. During inference, TabTrim performs parallel search to explore multiple candidate pruning trajectories and identify the optimal sub-table.

Our Contributions. (1) *New Insight.* We rethink table pruning in TableQA by transforming it from sequential revisions to gold trajectory-supervised parallel search. (2) *Novel Framework.* We propose TabTrim, a novel table pruning framework which leverages gold trajectory as a reliable critique for each pruning step and employs parallel search to find the optimal sub-table. (3) *SOTA Performance.* Extensive experiments demonstrate that TabTrim achieves state-of-the-art performance across diverse tabular reasoning tasks. TabTrim-8B reaches 73.5% average accuracy, outperforming the strongest baseline by 3.2%, including 79.4% on WikiTQ and 61.2% on TableBench.

2 TabTrim

In this section, we introduce TabTrim, a framework that performs step-wise pruning with reliable critique through parallel search, reducing answer-critical data loss while extracting simplified sub-tables to enhance downstream reasoning. We begin by introducing the trajectories construction (Sec.2.1), which serves as the supervision for the two core components of TabTrim, as shown in Fig.2: (1) Trajectories-supervised Pruner, which generates pruned sub-tables step by step (Sec.2.2); and (2) Loss-aware Verifier, which assesses each sub-table with a loss-aware score (Sec.2.3). Finally, TabTrim performs inference via Parallel Trajectory Search, which invokes the pruner to generate multiple candidate sub-tables, rejects low-quality ones using verifier scores, and outputs the highest-scoring sub-table for downstream table reasoning (Sec.2.4).

2.1 Trajectory Construction

Since existing TableQA datasets only provide final answers and lack step-level sub-table annotations, we derive sub-table trajectories from single-table Text-to-SQL data by executing a sequence of decomposed gold SQL clause-level operations. The constructions of gold sub-table trajectories and off-trajectory negatives are detailed as follows.

Gold Trajectories via SQL Decomposition. We leverage existing Text-to-SQL datasets $\mathcal{D} = \{(Q, SQL_{\text{gold}}, T_{\text{raw}})\}$ as data sources, where each instance consists of a natural language question Q, an annotated gold SQL query SQL_{gold}, and the corresponding raw table T_{raw}. Exploiting the compositional structure of SQL, we decompose the gold query SQL_{gold} into a sequence of clause-level operations $(c_1, c_2, \ldots, c_n)$ following the query's logical execution order. Here, we focus on clause-level operations that directly affect answer-critical data, such as row filtering and column projection (where projection retains columns referenced by `SELECT`, `GROUP BY`, `HAVING`, or aggregate expressions). By sequentially executing these operations, we obtain a gold sub-table trajectory $\tau^+ = (T_0, T_1^+, \ldots, T_n^+)$, where T_t^+ denotes the gold sub-table after executing c_t at step $t = 1, \ldots, n$ and T_0 is initialized by T_{raw}. This procedure produces gold sub-table trajectories without requiring additional manual annotation. More details about SQL decomposition are shown in Appendix A.1.

Off-trajectory Negatives Construction. To improve model's robustness against erroneous pruning, we construct negative sub-tables off the gold trajectory by modifying gold clause-level operations. Executing the modified operations on T_{t-1}^+ yields non-trivial but incorrect sub-tables T_t^-. The training samples are formed as progression dataset $\mathcal{D}^+ = \{(Q, T_0, T_{t-1}^+, T_t^+)\}$ and correction dataset $\mathcal{D}^- = \{(Q, T_0, T_{t-1}^-, T_t^+)\}$. More details about SQL modification are shown in Appendix A.2.

2.2 Gold Trajectory-supervised Pruner

We use the trajectories constructed above as supervision signals to train the pruner M_θ, which generates the next pruned sub-table T_t conditioned on the question Q, the raw table T_{raw}, and the current sub-table T_{t-1}. The training process is conducted in two distinct stages.

Stage 1: Supervised Fine-Tuning. To teach the pruner to align with the gold pruning trajectory and equip it with the ability to recover from erroneous pruning, we include both progression samples from $\mathcal{D}^+$ and correction samples from $\mathcal{D}^-$. We then optimize the pruner with the following supervised fine-tuning objective:

$$\begin{aligned}\mathcal{L}_{\text{SFT}}(\theta) = -&\sum_{(Q,T_0,T_{t-1}^+,T_t^+)\in\mathcal{D}^+} \log P_\theta(T_t^+ \mid Q, T_0, T_{t-1}^+) \\ -\lambda &\sum_{(Q,T_0,T_{t-1}^-,T_t^+)\in\mathcal{D}^-} \log P_\theta(T_t^+ \mid Q, T_0, T_{t-1}^-),\end{aligned} \tag{1}$$

where θ denotes the model parameters and λ weights the correction term. The first term of loss function trains the pruner to follow the gold trajectory from T_{t-1}^+ to T_t^+, while the second term trains it to recover to the same target T_t^+ from a modified sub-table T_{t-1}^-. The sums range over all training instances and time steps t.

Stage 2: Direct Preference Optimization. Supervised fine-tuning alone may still yield sub-tables with subtle semantic errors. We therefore employ Direct Preference Optimization (DPO) (Rafailov et al., 2023) to enforce a preference for the gold next sub-table T_t^+ over the incorrect next sub-table T_t^- under the same context. The loss is:

$$\begin{aligned}\mathcal{L}_{\text{DPO}}(\theta) = -\log\sigma\Big(&\beta\log\frac{P_\theta(T_t^+ \mid Q, T_0, T_{t-1})}{P_{\text{ref}}(T_t^+ \mid Q, T_0, T_{t-1})} \\ &-\beta\log\frac{P_\theta(T_t^- \mid Q, T_0, T_{t-1})}{P_{\text{ref}}(T_t^- \mid Q, T_0, T_{t-1})}\Big),\end{aligned} \tag{2}$$

where P_θ denotes the target model to be optimized and P_{ref} is a frozen reference model. The two log-ratio terms compare how much the target model increases the likelihood of T_t^+ and decreases the likelihood of T_t^- relative to the reference model under the same context. β controls the strength of this preference, and $\sigma(\cdot)$ is the logistic sigmoid. This stage pushes the model to prefer the gold pruning process $T_{t-1} \to T_t^+$ over the negative pruning process $T_{t-1} \to T_t^-$, reducing subtle errors.

2.3 Loss-aware Verifier

To quantify how far a pruned sub-table deviates from the optimal one, we introduce a Loss-aware Verifier g_ϕ that assesses each sub-table with a quality score.

Loss-aware Quality Score. To obtain an objective supervision, we compare each intermediate sub-table T_t with the optimal reference sub-table, defined as the final sub-table T_n^+ of the gold trajectory constructed in Sec. 2.1.

To make this comparison computable, we canonicalize the raw table T_{raw} into a set of indexed cells and represent its sub-table as a subset of canonical cell set. Concretely, each cell is identified by its row index r, column index c, and value v_{rc}. Each sub-table is represented by a set of selected cells $E(T) = \{(r, c, v_{rc})\}$. More details about canonicalization are shown in A.3.

We then define Precision and Recall by cell-set overlap between the intermediate sub-table and the optimal reference sub-table:

$$\begin{aligned} \text{Precision} &= \frac{|E(T_t) \cap E(T_n^+)|}{|E(T_t)|}, \\ \text{Recall} &= \frac{|E(T_t) \cap E(T_n^+)|}{|E(T_n^+)|}. \end{aligned} \tag{3}$$

Intuitively, Recall measures how much answer-critical data is preserved, while Precision reflects the amount of redundant cells retained. Since losing answer-critical data is typically more detrimental to downstream reasoning, we define the loss-aware quality score as an α-weighted F-score:

$$S(T_t) = (1+\alpha^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\alpha^2 \cdot \text{Precision}) + \text{Recall}}. \tag{4}$$

Here, α controls the penalty on answer-critical data loss. In particular, choosing $\alpha > 1$ emphasizes recall, so missing answer-critical data leads to a larger score drop.

Training. We train the verifier to regress from (Q, T_0, T_t) to the loss-aware quality score $S(T_t)$. Specifically, we construct a verifier training set $\mathcal{D}_{\text{ver}}$ using sub-tables from both (1) gold trajectories $\{T_t^+\}$ and (2) modified off-trajectory negatives $\{T_t^-\}$ (Sec. 2.1). For each sub-table T_t, we compute its score by comparing it with T_n^+, and then optimize the verifier with the following objective:

$$\mathcal{L}_{\text{ver}}(\phi) = \sum_{(Q,T_0,T_t)\in\mathcal{D}_{\text{ver}}} \big(g_\phi(Q, T_0, T_t) - S(T_t)\big)^2. \tag{5}$$

At inference time, the learned verifier predicts step-level loss-aware scores for sub-tables without access to T_n^+.

2.4 Parallel Trajectory Search

TabTrim performs inference via parallel search, which incrementally constructs pruning trajectories as sequences of sub-tables. Starting from the initial table T_0, at each step the pruner M_θ generates multiple candidate next sub-tables for each currently retained sub-table. The verifier g_ϕ then assesses a step-level loss-aware score to each candidate, and TabTrim selects a small set of top-ranked candidates to continue expanding in the next step. Unlike sequential revisions that commit to single pruning trajectory, TabTrim maintains multiple competing sub-table trajectories and performs a generate–score–select procedure step by step. This design reduces the risk of early pruning errors by allowing the search to discard low-quality branches while preserving promising alternatives.

Specifically, we implement parallel search process via beam search. Let k denote the beam width, b the branching factor, and $D_{\max}$ the maximum depth. We initialize $\mathcal{B}_0 = \{T_0\}$. For $t = 0$, we warm-start by sampling $k \cdot b$ candidates from T_0 and selecting the top-k sub-tables; for $t \geq 1$, each beam sub-table proposes b candidates. For each step $t = 0, \dots, D_{\max} - 1$, we form the candidate pool

$$\mathcal{U}_{t+1} = \{\tilde{T}_{t+1}^{j,i} = M_\theta(Q, T_0, \tilde{T}_t^j) \mid \tilde{T}_t^j \in \mathcal{B}_t,\ i = 1, \dots, b\}, \tag{6}$$

score each candidate with the verifier

$$\hat{S}_\phi(\tilde{T}_{t+1}^{j,i}) = g_\phi(Q, T_0, \tilde{T}_{t+1}^{j,i}), \tag{7}$$

and keep the top-k candidates as the next beam:

$$\mathcal{B}_{t+1} = \text{TopK}(\mathcal{U}_{t+1}, k; \hat{S}_\phi). \tag{8}$$

Finally, we output the highest-scoring sub-table among all explored beams:

$$\hat{T} = \arg \max_{\tilde{T} \in \bigcup_{t=0}^{D_{\max}} \mathcal{B}_t} \hat{S}_\phi(\tilde{T}), \tag{9}$$

and feed $(Q, \hat{T})$ to an LLM to generate the answer A.

3 Experiments

3.1 Experimental Setup

Models. We construct over 80K training samples from WikiSQL (Zhong et al., 2017) and SQUALL

Method	WikiTQ	TabFact	TB-NR	TB-FC	TB-DA	Average
			Direct QA			
Qwen3-4B (Yang et al., 2025a)	50.8	74.1	61.7	70.8	19.2	55.3
Qwen3-8B (Yang et al., 2025a)	52.2	76.7	64.5	72.9	23.0	57.9
GPT-4o-mini (Hurst et al., 2024)	54.3	77.4	65.5	76.0	25.1	59.8
			Program-based methods			
Binder (Cheng et al., 2023)	54.8	83.3	66.8	67.7	26.8	59.9
TabSQLify (Nahid and Rafiei, 2024)	68.7	78.3	65.2	76.0	28.0	63.2
			LLM-based methods			
Dater (Ye et al., 2023)	65.8	83.6	65.0	69.8	28.6	62.6
Chain-of-Table (Wang et al., 2024)	67.5	88.9	68.5	78.1	30.3	66.7
			Critique methods			
TALON (Jin et al., 2025)	70.7	87.6	67.3	77.1	28.9	66.3
Table-Critic (Yu et al., 2025)	72.6	<u>90.6</u>	73.0	<u>81.3</u>	<u>33.8</u>	70.3
			Ours			
TabTrim-4B	<u>76.8</u>	89.4	<u>76.3</u>	79.2	32.1	<u>70.8</u>
TabTrim-8B	**79.4**	**91.2**	**78.8**	**83.3**	**34.7**	**73.5**

Table 1: Main performance comparison on WikiTQ, TabFact and TableBench benchmarks. **Bold** denotes the best performance and <u>underline</u> denotes the second-best performance.

(Shi et al., 2020) using the data construction procedure described in Sec.2.1. Based on this data, we train Qwen3-4B and Qwen3-8B as pruners, and train Qwen3-0.6B as the verifier with $\alpha = 1.5$. Unless otherwise stated, we set the beam width and branching factor to $k = b = 2$ and the maximum search depth to $D_{\max} = 4$, which yields an upper bound of $O(k \cdot b \cdot D_{\max})$ pruner and verifier calls per example. This results in TabTrim-4B and TabTrim-8B. More training and implementation details are provided in Appendix B.1.

Datasets. We evaluate our method on three representative and challenging benchmarks spanning diverse tabular reasoning tasks, including (1) WikiTQ (Pasupat and Liang, 2015), a table reasoning dataset with 4,344 samples from 421 Wikipedia tables. (2) TabFact (Chen et al., 2020), a fact verification benchmark in table reasoning with 2,024 test samples from 298 tables. (3) TableBench (TB) (Wu et al., 2025a), a complex tabular reasoning benchmark with 886 questions covering tasks of numerical reasoning (NR), fact checking (FC) and data analysis (DA). More information about datasets is shown in Appendix B.2.

Evaluation Metric. For WikiTQ and TableBench datasets, we use exact match accuracy (EM) to check whether the predicted answer matches the ground truth. For TabFact, we adopt binary classification accuracy as evaluation metric.

Compared Methods. We compare TabTrim with baselines from four categories: **(1) Direct QA:** Qwen3 (Yang et al., 2025a) and GPT-4o-mini (Hurst et al., 2024). **(2) Program-based methods:** Binder (Cheng et al., 2023) and TabSQLify (Nahid and Rafiei, 2024). **(3) LLM-based methods:** Dater (Ye et al., 2023) and Chain-of-Table (Wang et al., 2024). **(4) Critique methods:** TALON (Jin et al., 2025) and Table-Critic (Yu et al., 2025). Since many baselines couple pruning and reasoning within closed-source frameworks, we standardize our evaluation by executing their complete pipelines with GPT-4o-mini. For each method, we strictly follow its original settings to ensure peak performance. For TabTrim, we also use GPT-4o-mini to generate the final answer from the selected sub-table to ensure a fair comparison.

3.2 Main Results

Tab.1 reports the performance of TabTrim and all baselines on WikiTQ, TabFact and TableBench. Our comprehensive evaluation reveals several key findings. First, TabTrim achieves the best overall performance across all benchmarks. TabTrim-8B reaches an average accuracy of 73.5%, outperforming the strongest non-TabTrim baseline (Table-Critic, 70.3%) by 3.2%, while TabTrim-4B also

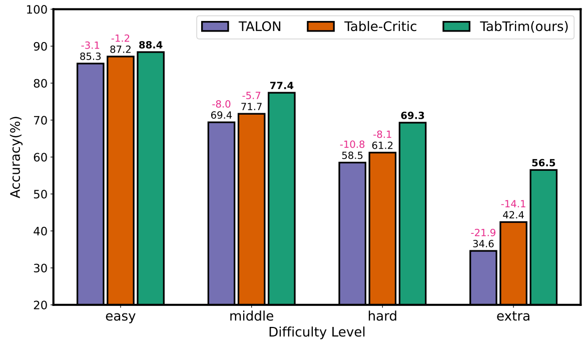

Figure 3: Accuracy comparison of TabTrim, Table-Critic, and TALON across different difficulty levels in the WikiTQ dataset.

Method	WikiTQ	▽	TableBench	▽
TabTrim	**79.4**	–	**61.2**	–
w/o DPO	78.1	-1.3	58.6	-2.6
w/o Correction Samples	74.8	-4.6	55.4	-5.8
w/o Training	54.7	-24.7	49.6	-11.6

Table 2: Ablation on the training pipeline of the sub-table pruner. ▽ denotes absolute accuracy degradation.

Method	WikiTQ	▽	TableBench	▽
Loss-aware score	**79.4**	–	**61.2**	–
Balanced score	77.8	-1.6	58.3	-2.9

Table 3: Effect of loss-aware verification. ▽ denotes absolute accuracy degradation.

attains a strong 70.8% average and already surpasses all non-TabTrim methods. Second, TabTrim delivers consistent gains across diverse tabular reasoning tasks. On WikiTQ, TabTrim-8B achieves 79.4%, exceeding the best baseline by 6.8%, highlighting that our step-wise pruning coupled with reliable verification is especially effective for complex compositional questions. On TabFact, TabTrim-8B achieves high performance of 91.2%, indicating that the advantage of improved pruning extends beyond only hard instances. Third, TabTrim generalizes well to TableBench, where it achieves the best performance on all subtasks. Compared with the strongest baseline, it improves TB-NR by 5.8%, TB-FC by 2.0%, and TB-DA by 0.9%. The especially large gain on TB-NR suggests that TabTrim is particularly effective for multi-hop numerical reasoning, where correct answering is highly sensitive to retaining the right operands and critical data. Finally, we observe a clear scaling effect with stronger pruners. Upgrading the pruner from 4B to 8B consistently improves performance across all datasets, yielding a 2.5% average gain, which suggests that better proposal quality further amplifies the benefit of Parallel Trajectory Search under the verification.

3.3 Fine-grained Analysis on Hardness

To probe how TabTrim behaves on complex questions, we follow prior work (Ye et al., 2025) and stratify WikiTQ questions into difficulty levels based on the empirical success rate of GPT-4o. Concretely, for each question we sample 100 independent answers from GPT-4o and count how many are correct, then assign a hardness label: Easy (90–100 correct), Medium (60–89), Hard (10–59), or Extra Hard (0–9). Fig.3 compares TabTrim-8B against Table-Critic and TALON under these hardness buckets. TabTrim-8B consistently achieves higher accuracy across all levels, with the most pronounced improvements in the Extra Hard regime where baseline performance drops sharply. This trend suggests that TabTrim is especially beneficial for challenging questions, as it more reliably preserves answer-critical data needed for downstream multi-step reasoning.

3.4 Ablation Study

We conduct ablation studies to quantify the contribution of each component in TabTrim. Unless otherwise stated, all ablations are conducted on TabTrim-8B under the same inference configuration as the main results, and we report accuracy on WikiTQ and TableBench (total EM).

Effect of Pruner Training. We first ablate the training recipe of the pruner. Starting from the full model, we progressively remove (1) DPO, (2) correction samples constructed from modified off-trajectory sub-tables, and (3) all training (i.e., using the base model without fine-tuning). Tab.2 shows a consistent degradation as supervision is removed: dropping DPO reduces performance on both datasets, and removing correction samples further harms accuracy, indicating that robustness to off-trajectory sub-tables is crucial for reliable multi-step pruning. When training is removed entirely, performance collapses (-24.7% on WikiTQ and -11.6% on TableBench), approaching the direct QA baseline. This confirms that process-supervised trajectories are essential for learning effective pruning behaviors.

Effect of Loss-aware Verification. Next, we examine whether the verifier's loss-aware scoring is necessary. We compare our default set-

Method	WikiTQ	▽	TableBench	▽
Rank by Verifier	**79.4**	–	**61.2**	–
Rank by Likelihood	74.2	-5.2	56.7	-4.5
Sequential Revisions	72.9	-6.5	55.1	-6.1

Table 4: Ablation on search and ranking signal in score-guided inference. ▽ denotes absolute accuracy degradation.

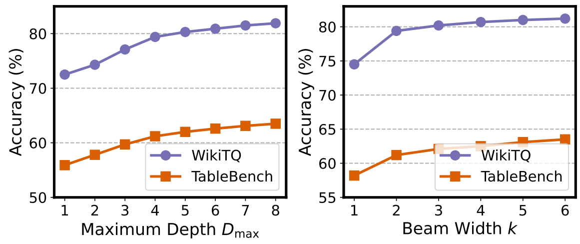

Figure 4: Scaling behavior of TabTrim-8B with increased inference-time search budget on WikiTQ and TableBench. Left: varying maximum depth $D_{\max}$ with $k = b = 2$. Right: varying beam width k with $D_{\max} = 4$ and $b = 2$.

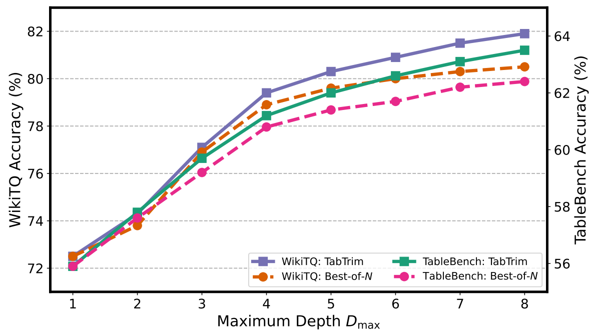

Figure 5: TabTrim vs. Best-of-N sampling under a matched compute budget. We fix $k = b = 2$ for TabTrim and set Best-of-N to $N = k \cdot b = 4$ and vary the maximum depth $D_{\max}$. We report accuracy on WikiTQ and TableBench.

ting ($\alpha = 1.5$) with a balanced variant ($\alpha = 1$) that reduces the emphasis on retaining answer-critical data. As shown in Tab.3, using the balanced score consistently degrades performance (-1.6% on WikiTQ and -2.9% on TableBench), indicating that a loss-aware quality score provides a stronger keep/discard criterion for multi-step pruning.

Effect of Parallel Trajectory Search. Finally, we analyze the role of Parallel Trajectory Search and its ranking signal. Tab.4 compares (1) the full TabTrim that performs beam search and ranks candidates by the verifier score, (2) a variant that keeps the same beam search but ranks by the pruner likelihood, and (3) the sequential revisions that disables search (i.e., $k = b = 1$) with a matched call budget by increasing the maximum depth accordingly. Replacing verifier-based ranking with likelihood ranking causes substantial drops on both datasets (-5.2% on WikiTQ and -4.5% on TableBench), suggesting that likelihood is not a reliable proxy for sub-table quality. Moreover, disabling search also degrades performance (-6.5% on WikiTQ and -6.1% on TableBench), confirming that expanding beyond sequential revisions is important for recovering from early pruning errors.

3.5 Analysis of Scaling

We analyze how TabTrim scales with increased inference-time compute. Fig.4 reports the performance of TabTrim-8B on WikiTQ and TableBench under varying search configurations. When we fix the beam width and branching factor ($k = b = 2$) and gradually increase the maximum depth $D_{\max}$, accuracy improves monotonically from the shallow setting ($D_{\max} = 1$) to deeper searches on both datasets. Similarly, when we fix $D_{\max} = 4$ and branching factor $b = 2$ and increase the beam width (from $k = 1$ to larger values), performance also improves, indicating that TabTrim can reliably convert additional search budget into accuracy gains rather than becoming unstable.

We further compare Parallel Trajectory Search with naive sampling under a matched compute budget. Specifically, we construct a Best-of-N baseline that uses the same pruner and verifier, but replaces beam search with N independent $D_{\max}$-step pruning rollouts and returns the rollout with the highest verifier score (we set $N = k \cdot b$ to match the per-depth expansion budget). As shown in Fig.5, TabTrim consistently outperforms Best-of-N on both WikiTQ and TableBench across all depths. This indicates that maintaining and selecting sub-tables step-by-step during the search is more effective than allocating the same calls to independent rollouts and only selecting at the end. Overall, these results demonstrate that TabTrim scales positively with inference-time compute and uses the search budget more efficiently than naive sampling.

3.6 Analysis of Plug-and-Play Gains

As TabTrim is an open-source, plug-and-play pruning module, we evaluate its plug-and-play gains on downstream open-source reasoners. Concretely, we run Qwen3-8B and Table-R1-7B (Yang et al., 2025c) on (1) raw tables and (2) TabTrim(8B)-

Method	WikiTQ	△	TableBench	△
Qwen3	52.2	–	48.4	–
w/ TabTrim	78.1	+25.9	59.2	+10.8
Table-R1	77.5	–	34.3	–
w/ TabTrim	84.9	+7.4	43.1	+8.8

Table 5: Plug-and-play gains from using TabTrim-selected sub-tables for open-source downstream reasoners. △ denotes absolute accuracy gain.

selected sub-tables, and report accuracy on WikiTQ and TableBench. Tab.5 shows that TabTrim consistently boosts both reasoners across datasets, indicating that TabTrim can serve as an effective front-end that reduces distracting noise while preserving answer-critical data for downstream reasoning.

3.7 Analysis of Computational Cost

We evaluate the computational cost of TabTrim using token counts (in millions) for both input and output, as shown in Tab.6. Compared to the Table-Critic, TabTrim achieves significantly higher efficiency, consuming substantially fewer input tokens. Although TabTrim generates more output tokens due to its candidate generation mechanism, its total token consumption remains markedly lower, ranging from $0.56\times$ to $0.66\times$ of the TableCritic's total. These results demonstrate that TabTrim not only provides superior pruning performance but also maintains a much higher aggregate compute efficiency.

4 Related Work

Table Reasoning. Early work on table reasoning develops task-specific models with table-aware pretraining or fine-tuning objectives (Herzig et al., 2020; Yin et al., 2020; Deng et al., 2022; Liu et al., 2022; Gu et al., 2022). With the rise of general-purpose LLMs, recent methods often serialize tables into text and perform open-text reasoning (Zhang et al., 2025) and introduce decomposition strategies to handle complex queries (Zhao et al., 2024; Wu and Feng, 2024). However, purely textual reasoning can be brittle as irrelevant noisy content can distract the model and undermine reasoning.

Table Pruning. To filter out noise before reasoning, recent work increasingly focuses on table pruning. Program-based methods rely on executable SQL or Python to filter rows and columns (Cheng et al., 2023; Zhang et al., 2024; Nahid and Rafiei, 2024; Jin et al., 2025), and LLM-based methods perform pruning through iterative reasoning (Ye et al., 2023; Wang et al., 2024; Yu et al., 2025). However, these methods are typically error-sensitive, where an early-stage pruning error can propagate and lead to reasoning failure.

Dataset	Table-Critic Input (M)	Table-Critic Output (M)	TabTrim Input (M)	TabTrim Output (M)	Ratio
WikiTQ	135.5	3.8	63.8	14.5	0.56×
TabFact	62.1	2.4	32.9	9.7	0.66×

Table 6: Token usage over WikiTQ and TabFact.

Critique Mechanism. To alleviate error propagation, recent work explores critique mechanisms (Madaan et al., 2023; Yang et al., 2025b) to evaluate intermediate steps and revise subsequent decisions. In table pruning, program-based methods often rely on execution feedback to trigger repair or regeneration (Jin et al., 2025), while LLM-based methods commonly use LLM-as-a-Judge to assess intermediate outputs (Yu et al., 2025). However, these critique signals are not reliable with true pruning errors, and these methods typically applied in a sequential revision process. As a result, erroneous pruning decisions may go undetected, and once answer-critical data is discarded, later revisions are often unable to recover it. To address these limitations, TabTrim derives step-wise supervision from sub-table trajectories to train a pruner and a verifier, and performs inference via parallel search to maintain multiple competing pruning trajectories and discard low-quality branches.

5 Conclusion

In this paper, we propose TabTrim, a framework that performs step-wise pruning with reliable critique through parallel search, reducing answer-critical data loss while extracting simplified sub-tables to enhance downstream reasoning. TabTrim derives step-wise supervision from sub-table trajectories to train a pruner that generates pruned sub-tables and a verifier that scores intermediate sub-tables. At inference, TabTrim performs parallel search to maintain multiple competing pruning trajectories and discard low-quality branches. Experiments on WikiTQ, TabFact, and TableBench demonstrate that TabTrim consistently outperforms strong baselines, effectively improving table pruning and enhancing downstream table reasoning.

Limitations

Our experimental scope is currently bounded by computational resource constraints, confining our evaluation to the 4B and 8B parameter regimes. While TabTrim demonstrates superior performance at this scale, the substantial computational expenditure required for fine-tuning massive architectures precludes the extension of our analysis to the larger range. Consequently, the scaling laws and performance ceiling of TabTrim with stronger base models remain to be fully characterized, an investigation we leave for future work.

Ethics Statement

Our work aims to enhance the reliability of multi-step table pruning in TableQA. However, like any system based on LLMs, it still entails the risk of generating factually incorrect answers or incomplete sub-tables. We strongly advise users to exercise caution and verify critical outputs when deploying TabTrim in real-world scenarios. Furthermore, our research builds upon open-source advancements, specifically utilizing models such as Qwen3 and frameworks including PyTorch and Hugging Face. We strictly adhere to the respective licenses and usage policies of these resources, acknowledging their pivotal contribution to the community.

Acknowledgement

This work was financially supported by National Key Research and Development Program of China, No.2024YDLN0004 and the National Natural Science Foundation of China (U23A20275).

References

Shengnan An, Xunliang Cai, Xuezhi Cao, Xiaoyu Li, Yehao Lin, Junlin Liu, Xinxuan Lv, Dan Ma, Xuanlin Wang, Ziwen Wang, and 1 others. 2025. Amo-bench: Large language models still struggle in high school math competitions. *arXiv preprint arXiv:2510.26768*.

Si-An Chen, Lesly Miculicich, Julian Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. Tablerag: Million-token table understanding with language models. *Advances in Neural Information Processing Systems*, 37:74899–74921.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact : A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. *ICLR*.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *SIGMOD Rec.*, 51(1):33–40.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-operations aware fact verification via sentence-table cloze pre-training. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4971–4983, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ruochun Jin, Xiyue Wang, Dong Wang, Haoqi Zheng, Yunpeng Qi, Silin Yang, and Meng Zhang. 2025. TALON: A multi-agent framework for long-table exploration and question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 27385–27401, Suzhou, China. Association for Computational Linguistics.

Honglin Lin, Zheng Liu, Yun Zhu, Chonghan Qin, Juekai Lin, Xiaoran Shang, Conghui He, Wentao Zhang, and Lijun Wu. 2026a. Mmfinereason: Closing the multimodal reasoning gap via open data-centric methods. *arXiv preprint arXiv:2601.21821*.

Honglin Lin, Chonghan Qin, Zheng Liu, Qizhi Pei, Yu Li, Zhanping Zhong, Xin Gao, Yanfeng Wang, Conghui He, and Lijun Wu. 2026b. Scientific image synthesis: Benchmarking, methodologies, and downstream utility. *arXiv preprint arXiv:2601.17027*.

Weizhe Lin, Rexhina Blloshmi, Bill Byrne, Adria de Gispert, and Gonzalo Iglesias. 2023. An inner table retriever for robust table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9909–9926, Toronto, Canada. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

Zheng Liu, Hao Liang, Xijie Huang, Wentao Xiong, Qinhan Yu, Linzhuang Sun, Chong Chen, Conghui He, Bin Cui, and Wentao Zhang. 2024. Synthvlm: High-efficiency and high-quality synthetic data for vision language models. *arXiv preprint arXiv:2407.20756*, 3.

Zheng Liu, Honglin Lin, Chonghan Qin, Xiaoyang Wang, Xin Gao, Yu Li, Mengzhang Cai, Yun Zhu, Zhanping Zhong, Qizhi Pei, and 1 others. 2026. Chartverse: Scaling chart reasoning via reliable programmatic synthesis from scratch. *arXiv preprint arXiv:2601.13606*.

Zheng Liu, Mengjie Liu, Jingzhou Chen, Jingwei Xu, Bin Cui, Conghui He, and Wentao Zhang. 2025a. Fusion: Fully integration of vision-language representations for deep cross-modal understanding. *arXiv preprint arXiv:2504.09925*.

Zheng Liu, Mengjie Liu, Siwei Wen, Mengzhang Cai, Bin Cui, Conghui He, and Wentao Zhang. 2025b. From uniform to heterogeneous: Tailoring policy optimization to every token's nature. *arXiv preprint arXiv:2509.16591*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5725–5737, Mexico City, Mexico. Association for Computational Linguistics.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864, Online. Association for Computational Linguistics.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

Fei Wu, Zhenrong Zhang, Qikai Chang, Jianshu Zhang, Quan Liu, and Jun Du. 2026. Step potential advantage estimation: Harnessing intermediate confidence and correctness for efficient mathematical reasoning. *arXiv preprint arXiv:2601.03823*.

Jian Wu, Yicheng Xu, Yan Gao, Jian-Guang Lou, Börje F. Karlsson, and Manabu Okumura. 2023. TACR: A table alignment-based cell selection method for HybridQA. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6535–6549, Toronto, Canada. Association for Computational Linguistics.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, and 1 others. 2025a. Tablebench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25497–25506.

Zhenhe Wu, Zhongqiu Li, Mengxiang Li, Jie Zhang, Zhongjiang He, Jian Yang, Yu Zhao, Ruiyu Fang, Yongxiang Li, Zhoujun Li, and 1 others. 2025b. Mr-sql: multi-level retrieval enhances inference for llm in text-to-sql. In *International Conference on Database Systems for Advanced Applications*, pages 403–413. Springer.

Zhenhe Wu, Zhongqiu Li, Jie Zhang, Zhongjiang He, Jian Yang, Yu Zhao, Ruiyu Fang, Bing Wang, Hongyan Xie, Shuangyong Song, and Zhoujun Li. 2025c. UCS-SQL: Uniting content and structure for enhanced semantic bridging in text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8156–8168, Vienna, Austria. Association for Computational Linguistics.

Zhenhe Wu, Jian Yang, Jiaheng Liu, Xianjie Wu, Changzai Pan, Jie Zhang, Yu Zhao, Shuangyong Song, Yongxiang Li, and Zhoujun Li. 2025d. Table-r1: Region-based reinforcement learning for table understanding. *arXiv preprint arXiv:2505.12415*.

Zirui Wu and Yansong Feng. 2024. ProTrix: Building models for planning and reasoning over tables with sentence context. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4378–4406, Miami, Florida, USA. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhe Yang, Yichang Zhang, Yudong Wang, Ziyao Xu, Junyang Lin, and Zhifang Sui. 2025b. Confidence v.s. critique: A decomposition of self-correction capability for LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3998–4014, Vienna, Austria. Association for Computational Linguistics.

Zheyuan Yang, Lyuhao Chen, Arman Cohan, and Yilun Zhao. 2025c. Table-r1: Inference-time scaling for table reasoning tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20616–20635, Suzhou, China. Association for Computational Linguistics.

Shenghao Ye, Yu Guo, Dong Jin, Yikai Shen, Yunpeng Hou, Shuangwu Chen, Jian Yang, and Xiaofeng Jiang. 2025. When tableqa meets noise: A dual denoising framework for complex questions and large-scale tables. *arXiv preprint arXiv:2509.17680*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 174–184.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Peiying Yu, Guoxin Chen, and Jingjing Wang. 2025. Table-critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17432–17451, Vienna, Austria. Association for Computational Linguistics.

Wenyuan Zhang, Xinghua Zhang, Haiyang Yu, Shuaiyi Nie, Bingli Wu, Juwei Yue, Tingwen Liu, and Yongbin Li. 2026. Expseek: Self-triggered experience seeking for web agents. *arXiv preprint arXiv:2601.08605*.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. 2025. A survey of table reasoning with large language models. *Frontiers of Computer Science*, 19(9):199348.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840, Bangkok, Thailand. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Yixiao Zhou, Yang Li, Dongzhou Cheng, Hehe Fan, and Yu Cheng. 2026. Look inward to explore outward: Learning temperature policy from llm internal states via hierarchical rl. *arXiv preprint arXiv:2602.13035*.

Yixiao Zhou, Ziyu Zhao, Dongzhou Cheng, Zhiliang Wu, Jie Gui, Yi Yang, Fei Wu, Yu Cheng, and Hehe Fan. 2025. Dropping experts, recombining neurons: Retraining-free pruning for sparse mixture-of-experts llms. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 15169–15186.

A Additional Implementation Details

A.1 SQL Decomposition

Given a training triple $(Q, S_{\text{gold}}, T_{\text{raw}})$ from a Text-to-SQL dataset, we first parse the gold SQL query S_{gold} into an abstract syntax tree using a SQL parser. We then linearize this tree into an ordered sequence of clause-level operations

$$(c_1, c_2, \ldots, c_n),$$

which follows the logical execution order of the query. In practice, we focus on clauses that directly affect the evidential content of the table. Concretely, we restrict intermediate operations to: (1) row selection based on predicates in `WHERE` (e.g., conjunctions or disjunctions over comparison operators), and (2) column pruning that determines the relevant subset of attributes in the base table (e.g., keeping columns that appear in `SELECT`, `GROUP BY`, `HAVING`, or aggregate expressions, while dropping attributes that never influence the answer), but implemented purely as column projection without changing row cardinality.

Starting from the full table $T_0 = T_{\text{raw}}$, each clause c_t is applied as a transformation

$$T_t^+ = \text{Exec}(c_t, T_{t-1}^+),$$

where $\text{Exec}(c_t, \cdot)$ programmatically applies the clause-level operation c_t (e.g., filtering or projection) to the current sub-table to obtain the next sub-table. By construction, yielding an answer-critical data preserved pruning trajectory

$$\tau^+ = (T_0, T_1^+, \dots, T_n^+)$$

that decomposes the pruning process into a sequence of sub-table. These trajectories serve as process supervision signals for training the pruner in Sec.2.2.

A.2 SQL Modification

To train the pruner and verifier to distinguish answer-critical sub-table from incorrect ones, we construct negative samples by modifying the clause sequence obtained in Appendix A.1. Given a training triple $(Q, SQL_{\text{gold}}, T_{\text{raw}})$ and its clause sequence $(c_1, \dots, c_n)$, we generate modified clauses $\{c_t^-\}$ and obtain corresponding negative sub-tables $\{T_t^-\}$ by executing these clauses on the $\{T_{t-1}^+\}$.

Modification Space. We operate at the same granularity as our decomposition, i.e., on row-selection predicates and column-pruning decisions, so that all modifications remain executable and interpretable as pruning operations. Concretely, we use the following families of modifications:

- **Predicate-level modifications.** For a `WHERE` clause predicate of the form a `op` v, we generate variants that (1) replace v with another value from the same column domain (e.g., nearest neighbor in the sorted value set), or (2) swap `op` with a nearby operator (e.g., < vs. <=, > vs. >=). For composite predicates connected by `AND`/`OR`, we either drop one conjunct (to admit more irrelevant rows) or add a spurious conjunct that filters out some answer-critical rows.

- **Column-pruning modifications.** For the column selection induced by c_t, we create perturbed versions by (1) dropping at least one attribute that participates in `SELECT`, `GROUP BY`, `HAVING`, or aggregate expressions, or (2) keeping additional irrelevant attributes. These are implemented as column projections on the table and do not change row cardinality.

λ	WikiTQ	TableBench
0.3	76.5	58.1
0.5	78.1	60.8
0.8	78.8	61.7
1.0	79.4	61.2

Table 7: Sensitivity analysis of the correction weight λ in the pruner training objective.

All modified clauses are required to be syntactically valid and executable on T_{raw}. After execution, yielding modified sub-tables T_t^- that are structurally comparable to the gold sub-table.

Negative Samples Construction. Starting from the gold trajectory $\tau^+ = (T_0, T_1^+, \dots, T_n^+)$, we generate negative states in a step-wise manner. For each clause c_t, we sample one or more modifications from the above space to obtain c_t^- and define

$$T_t^- = \text{Exec}(c_t^-, T_{t-1}^+),$$

where Exec denotes the executor described in Appendix A.1. We discard degenerate cases where T_t^- is identical to T_t^+ or becomes trivially empty or full, and keep modifications that induce non-trivial but incorrect deviations from the on-trajectory sub-table.

A.3 Cell-index Canonicalization

To make pruned sub-tables well-defined and comparable, we convert each raw table T_{raw} into a set of indexed cells. All sub-table states T_t in TabTrim are represented as subsets of this canonicalized cell set.

Canonical Cell Representation. We treat each table as a grid with row indices and column indices. For each data row r and column c with value v_{rc}, we create a canonical cell

$$x_{rc} = (\text{row} = r,\ \text{col} = c,\ \text{val} = v_{rc}),$$

and define the canonicalized table as the cell set

$$\mathcal{C}(T_{\text{raw}}) = \{x_{rc}\}.$$

A sub-table state T_t is then represented as a subset $\mathcal{C}(T_t) \subseteq \mathcal{C}(T_{\text{raw}})$ obtained by selecting a subset of rows and/or columns (equivalently, a subset of cells).

B Additional Experimental Details

B.1 Experiment Setup

All experiments are conducted on 8 NVIDIA A100 40GB GPUs. We train all models for 2 epochs

α	WikiTQ	TableBench
1.0	77.8	58.3
1.2	78.6	59.7
1.5	79.4	61.2
1.7	78.9	61.5
2.0	78.7	60.4

Table 8: Sensitivity analysis of the recall-bias parameter α in the loss-aware verifier score.

Error Type	**%**
Sub-table errors	
Row mis-selection	5%
Column mis-selection	2%
Reasoning errors	
Arithmetic error	42%
Aggregation/Counting error	38%
Misinterpretation	7%
Logical error	2%
Others	4%

Table 9: Error taxonomy (percentage) on 100 failed examples of TabTrim-8B.

using the AdamW optimizer, employing a cosine learning rate schedule with a 3% warmup ratio. For the Pruner (Qwen3-4B and Qwen3-8B), we perform full-parameter fine-tuning with a peak learning rate of 2×10^{-5} and set the correction weight to $\lambda = 1$. In the subsequent DPO stage, we reduce the learning rate to 2×10^{-6} and set $\beta = 0.2$, following standard DPO configurations. The Verifier (Qwen3-0.6B) is trained with a peak learning rate of 5×10^{-5}. During inference, pruner generates outputs using top-p sampling of 0.95 and a temperature of 0.6, with a maximum generation length of 4,096 tokens. To ensure the stability of our results, we also repeated experiments with different random seeds. For TabTrim, we report the average accuracy over 3 runs. Across all datasets, the results were highly stable, with the standard deviation consistently within 1% absolute accuracy.

B.2 Dataset Details

WikiTQ introduces question answering over semi-structured HTML tables, aiming to test both compositional reasoning and domain generalization. It comprises 22,033 natural language questions paired with 2,108 Wikipedia tables, where the training and test tables are disjoint to ensure generalization to unseen schemas. The tables are semi-structured and heterogeneous, often containing multi-part cell values that require normalization into multiple semantic types such as numbers or dates. Questions range from simple lookups to highly compositional queries involving comparison, aggregation, arithmetic, and superlatives. Each table contains at least 8 rows and 5 columns, and the question collection was conducted with quality control through multiple annotators.

Dataset	# Tokens per Table		Comp. (%)
	Entire Table	Pruned Table	
TabFact	353	215	39.1%
WikiTQ	631	282	55.3%

Table 10: Token counts and compression rates before and after pruning across WikiTQ and TabFact datasets.

TabFact is a large-scale benchmark for table-based fact verification. Given a semi-structured Wikipedia table and a natural-language statement, the task is to predict whether the statement is ENTAILED or REFUTED by the table evidence. TabFact contains 118,275 human-annotated statements grounded in 16,573 Wikipedia tables. Each table with an average of 14 rows and 5-6 columns corresponds to 2–20 different statements, while each cell has an average of 2.1 words.

TableBench is a comprehensive benchmark specifically designed to evaluate the reasoning abilities of LLMs over tabular data. It consists of 3,681 unique tables drawn from diverse domains such as finance, sports, politics, and science, with each table containing on average 16.7 rows and 6.7 columns. The dataset emphasizes numerical reasoning, with over 65% of table cells containing numerical values. TableBench questions are organized into four major categories: fact-checking, numerical reasoning, data analysis, further divided into 18 subcategories, yielding a total of 886 carefully annotated samples. Each question typically requires 6.3 reasoning steps, making the dataset significantly more complex than prior TableQA corpora.

To ensure a fair evaluation, we de-duplicate the training corpus and exclude any training tables that overlap with the tables in our test benchmarks.

C Additional Experiments

C.1 Sensitivity to Correction Weight

We analyze the correction weight λ in the pruner training objective, which trades off progression learning on gold trajectories and recovery learning from off-trajectory states. We vary λ while keeping the same training set, base model, optimization setup, and inference budget fixed, and report EM on WikiTQ and TableBench using TabTrim-8B in Tab. 7. As λ increases from 0.3 to 1.0, performance overall improves on both benchmarks, indicating that emphasizing recovery learning is important for mitigating off-trajectory errors. Moreover, the gains become marginal beyond $\lambda \approx 0.8$, suggesting that TabTrim is not overly sensitive to precise tuning within this moderate range.

C.2 Sensitivity to Recall-bias Parameter

We study the sensitivity of TabTrim to the recall-bias parameter α in the loss-aware verifier score. We keep the training data, base model, optimization setup, and inference budget fixed, and report EM on WikiTQ and TableBench using TabTrim-8B in Tab. 8. Overall, adopting a recall-biased setting ($\alpha > 1$) consistently improves over the balanced score ($\alpha = 1$), confirming the benefit of explicitly penalizing evidence loss. Performance peaks around $\alpha \in [1.5, 1.7]$ and remains competitive for nearby values, suggesting that TabTrim only requires a moderate recall bias rather than precise tuning.

C.3 Error Analysis

We perform a fine-grained error analysis on 100 erroneous responses produced by TabTrim-8B. We manually categorize each failure into mutually exclusive error types based on whether the final pruned sub-table preserves answer-critical data, and if so, what type of reasoning mistake leads to the incorrect answer, the results are shown in Tab.9.

Overall, answer-critical selection errors are rare (7%), indicating that TabTrim largely preserves answer-critical rows and columns in the pruned sub-table. The dominant failures arise from downstream reasoning, especially numerical computation (42%) and aggregation/counting (38%), suggesting that improving the answerer's arithmetic and aggregation robustness is the primary direction for further gains.

C.4 Deployment-oriented Cost Analysis

While the raw token counts in Sec.3.7 demonstrate TabTrim's efficiency, its practical advantage is even more pronounced in self-hosted environments. TabTrim is designed on open-source models where prefix KV caching can amortize the cost of repeated prefills. Since calls to the pruner (and similarly the verifier) within the parallel search share a common prompt prefix (the query and raw table), this prefix is computed only once and reused across all branches. Besides, TabTrim's decoding overhead can be further reduced without changing the algorithm by adopting a more compact sub-table representation. Taken together, these considerations suggest that TabTrim can be even more favorable in practical self-hosted settings.

C.5 Table Pruning Effectiveness

To evaluate the effectiveness of TabTrim's table pruning capability, we randomly sample 200 correctly answered instances in WikiTQ and TabFact, and compute the average number of tokens before and after pruning. Table 10 reports the average token counts and corresponding compression rates.

D Training Data and Prompt

D.1 Example of training data for Pruner

Tab.11 presents the example of training data for the pruner.

D.2 Example of training data for Verifier

Tab.12 presents the example of training data for the verifier.

D.3 Prompt of Answerer

Tab.13 presents the prompt format used in answerer.

E Additional Related Work

LLM Reasoning. In recent years, the emergence of LLMs has established reasoning as a core capability for modern AI systems. Reasoning-based paradigms have been widely adopted across diverse domains, including vision-language modeling (Liu et al., 2024, 2025a; Lin et al., 2026a,b), chart understanding (Liu et al., 2026), mathematical problem solving (Wu et al., 2026; An et al., 2025), and web-based autonomous agents (Zhang et al., 2026). Concurrently, emerging evidence suggests that reasoning processes exhibit significant heterogeneity across different tokens, modules, modalities, and intermediate steps, calling for more adaptive reasoning, optimization, and inference strategies (Liu et al., 2025b; Zhou et al., 2025, 2026). Against this backdrop, structured reasoning tasks have attracted increasing attention, including table reasoning and closely related text-to-SQL settings, where models must jointly understand structured schemas, content, and compositional reasoning procedures over tabular evidence (Wu et al., 2025c,b,d).

```
Example of training data for pruner
### Question:
How many Engineering employees in Tokyo or Osaka were hired after 2020 and earn more than 130,000?
### Raw Table:
col: Employee | Department | City | Hire Year | Salary | Level | Manager
row 1: Akira Sato | Engineering | Tokyo | 2021 | 120000 | L4 | K. Tanaka
row 2: Mei Chen | Engineering | Osaka | 2019 | 130000 | L4 | K. Tanaka
row 3: Haruto Ito | Engineering | Osaka | 2022 | 105000 | L3 | M. Suzuki
row 4: Yuna Park | Sales | Tokyo | 2021 | 90000 | L3 | R. Lee
row 5: Kenji Watanabe | Engineering | Nagoya | 2023 | 115000 | L4 | M. Suzuki
row 6: Sara Kim | Engineering | Tokyo | 2020 | 98000 | L3 | M. Suzuki
row 7: Rina Nakamura | Engineering | Tokyo | 2024 | 140000 | L5 | K. Tanaka
row 8: Daichi Mori | HR | Osaka | 2022 | 80000 | L2 | T. Yamada
...
### Current Sub-table:
(after Step 1: filter Department = Engineering; Step 2: filter City in Tokyo, Osaka)
col: Employee | Department | City | Hire Year | Salary | Level | Manager
row 1: Akira Sato | Engineering | Tokyo | 2021 | 120000 | L4 | K. Tanaka
row 2: Mei Chen | Engineering | Osaka | 2019 | 130000 | L4 | K. Tanaka
row 3: Haruto Ito | Engineering | Osaka | 2022 | 105000 | L3 | M. Suzuki
row 6: Sara Kim | Engineering | Tokyo | 2020 | 98000 | L3 | M. Suzuki
row 7: Rina Nakamura | Engineering | Tokyo | 2024 | 140000 | L5 | K. Tanaka
...
### Next Sub-table:
(Step 3: filter Hire Year > 2020)
col: Employee | Department | City | Hire Year | Salary | Level | Manager
row 1: Akira Sato | Engineering | Tokyo | 2021 | 120000 | L4 | K. Tanaka
row 3: Haruto Ito | Engineering | Osaka | 2022 | 105000 | L3 | M. Suzuki
row 7: Rina Nakamura | Engineering | Tokyo | 2024 | 140000 | L5 | K. Tanaka
...
```

Table 11: Example of training data for the pruner.

```
Example of training data for verifier
### Question:
How many Engineering employees in Tokyo or Osaka were hired after 2020 and earn more than 130,000?
### Raw Table:
col: Employee | Department | City | Hire Year | Salary | Level | Manager
row 1: Akira Sato | Engineering | Tokyo | 2021 | 120000 | L4 | K. Tanaka
row 2: Mei Chen | Engineering | Osaka | 2019 | 130000 | L4 | K. Tanaka
row 3: Haruto Ito | Engineering | Osaka | 2022 | 105000 | L3 | M. Suzuki
row 4: Yuna Park | Sales | Tokyo | 2021 | 90000 | L3 | R. Lee
row 5: Kenji Watanabe | Engineering | Nagoya | 2023 | 115000 | L4 | M. Suzuki
row 6: Sara Kim | Engineering | Tokyo | 2020 | 98000 | L3 | M. Suzuki
row 7: Rina Nakamura | Engineering | Tokyo | 2024 | 140000 | L5 | K. Tanaka
row 8: Daichi Mori | HR | Osaka | 2022 | 80000 | L2 | T. Yamada
...
### Sub-table:
(intermediate sub-table after Step 3)
col: Employee | Department | City | Hire Year | Salary | Level | Manager
row 1: Akira Sato | Engineering | Tokyo | 2021 | 120000 | L4 | K. Tanaka
row 3: Haruto Ito | Engineering | Osaka | 2022 | 105000 | L3 | M. Suzuki
row 7: Rina Nakamura | Engineering | Tokyo | 2024 | 140000 | L5 | K. Tanaka
...
### Score:
0.67
```

Table 12: Example of training data for the verifier.

Prompt of Answerer
Instruction: You are a table question answering expert. Your task is to infer the answer to the question based on the provided table. ### Question: How many Engineering employees in Tokyo or Osaka were hired after 2020 and earn more than 130,000? ### Table: (final sub-table after Step 4: filter Salary > 130000; Step 5: keep only key columns) col: Employee | City | Hire Year | Salary row 7: Rina Nakamura | Tokyo | 2024 | 140000 ### Answer: 1

Table 13: Prompt format for the answerer.