p²-TQA: a process-based preference learning framework for self-improving table question answering models

Wei Zhou, Mohsen Mesgar, Heike Adel, Annemarie Friedrich

Angaben zur Veröffentlichung / Publication details:

Zhou, Wei, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. 2025. "p^2-TQA: a process-based preference learning framework for self-improving table question answering models." In *IJCNLP-AACL 2025: proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, volume 2: short papers, Mumbai, India, December 20-24, 2025*, edited by Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh, 217–31. Stroudsburg, PA: The Asian Federation of Natural Language Processing and The Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.ijcnlp-short.20.

p^2-TQA: A Process-based Preference Learning Framework for Self-Improving Table Question Answering Models

Wei Zhou[1,3] **Mohsen Mesgar**[1] **Heike Adel**[2] **Annemarie Friedrich**[3]
[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]Hochschule der Medien, Stuttgart, Germany [3]University of Augsburg, Germany
{wei.zhou3|mohsen.mesgar}@de.bosch.com
annemarie.friedrich@uni-a.de adel-vu@hdm-stuttgart.de

Abstract

Table question answering (TQA) focuses on answering questions based on tabular data. Developing TQA systems targets effective interaction with tabular data for tasks such as cell retrieval and data analysis. While recent work has leveraged fine-tuning to improve TQA systems, existing approaches often under-utilize available data and neglect the potential of post-training for further gains. In this work, we introduce p^2-TQA, a process-based preference learning framework for TQA post-training. p^2-TQA automatically constructs process-based preference data via a table-specific pipeline, eliminating the need for manual or costly data collection. It then optimizes models through contrastive learning on the collected data. Experiments show that p^2-TQA effectively improves TQA models by up to 5% on in-domain datasets and 2.4% on out-of-domain datasets with only 8,000 training instances. Furthermore, models enhanced with p^2-TQA achieve competitive results against larger, more complex state-of-the-art TQA systems, while maintaining up to five times higher efficiency.

1 Introduction

Table question answering (TQA) aims to generate accurate responses to queries over tables. Current TQA systems fall into two categories: fine-tuned models (Zhang et al., 2025a; Wu and Feng, 2024) and training-free frameworks (Zhou et al., 2025b; Nahid and Rafiei, 2024). The former fine-tune pre-trained small-size ($\leq$ 8B) large language models (LLMs) while the latter rely on large LLMs and involve complex designs. There is a growing interest in understanding and developing fine-tuned TQA models (Deng et al., 2025; Deng and Mihalcea, 2025) due to their promising performance and inference efficiency.

Current fine-tuning methods for TQA often augment only a subset of existing datasets with Chain-of-Thought (CoT) reasoning (Wei et al., 2023), constrained by the high cost of querying large commercial models (Wu and Feng, 2024). Models are then trained on these reasoning chains and answers via supervised fine-tuning. This workflow has two key limitations: (1) only part of the training data is used, i.e., data is under-utilized; and (2) potential performance gains from post-training are neglected. Based on the identified gaps, this study aims to answer the research question: *How can we leverage existing datasets to post-train a TQA model for further performance gains?* Our effective post-training method (shown in Figure 1) enhances models without requiring additional manual or costly data.

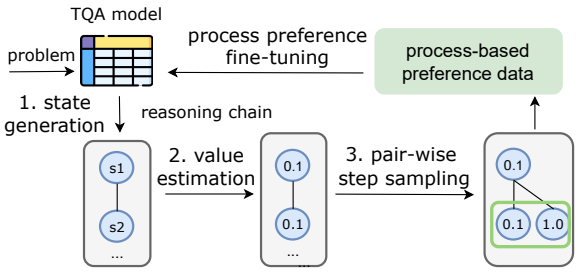


Figure 1: An overview of p^2-TQA: An existing model generates reasoning chains for a given problem. The chains are parsed into states, composed of cumulative steps. Each state is scored by a value function. We then create pairwise steps by rolling out parent states, selecting those with value differences exceeding a threshold. Lastly, contrastive learning is performed over collected data to improve the TQA model.

While post-training with self-generated data has primarily been explored in mathematics and coding (Singh et al., 2024; Zelikman et al., 2024; Xiong et al., 2024; He et al., 2024), particularly through step-wise preference learning (Tu et al., 2025; Xu et al., 2024), its application in TQA remains underexplored. Previous methods for obtaining step-wise preference pairs typically rely on (1) closed-source or large open-source models as judges to discern correct and incorrect steps (Lai et al., 2024), (2)

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 217–231
December 20-24, 2025

Monte Carlo Sampling (MCS) to estimate a step's quality (Wang et al., 2024; Xiong et al., 2024; Hwang et al., 2024), or (3) a combination of both (Zhang et al., 2025b). However, performing step-wise preference learning to TQA presents unique challenges. Compared to mathematics, it typically involves longer inputs and intermediate reasoning due to large tables. This necessitates a careful design of steps; simply using newline breaks to obtain steps, as in math, could lead to excessive computation costs, because newline breaks also denote new rows in tables. Furthermore, the structured nature of input in TQA demands a reconsideration of value functions, since LLM judges have a limited understanding of table structures (Sui et al., 2024), especially with larger tables (Zhou et al., 2024).

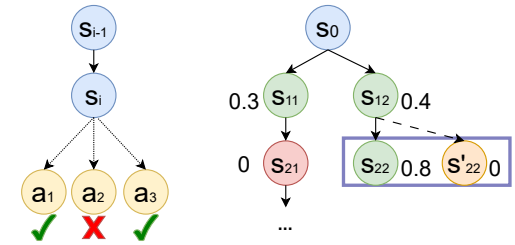
To this end, we introduce p^2-TQA, a process-based preference learning pipeline for TQA post-training: p^2-TQA operates in three stages for data collection (Figure 1): *state generation*, *state value estimation*, and *pair-wise step sampling*. The first stage collects and parses reasoning chains into states, composed of cumulative reasoning steps that are carefully designed for TQA. The last two stages construct step-wise preference pairs via MCS for state value estimation and a stringent filtering process for quality control. We apply direct preference optimization (DPO) (Rafailov et al., 2024) with the collected data to self-improve a TQA model.

Experiments show that p^2-TQA improves TQA models by up to 5% on in-domain datasets and by up to 2.4% on out-of-domain datasets with only 8k preference training pairs. It surpasses methods that require additional LLMs as judges, yet it is ten times more efficient. This underscores our contribution in establishing an effective and efficient framework for self-improving TQA models. The self-improved models outperform existing fine-tuned TQA models and achieve comparable performance to much larger and more complex frameworks on three datasets, while maintaining five times higher inference efficiency. Code is available. [1]

2 Step-wise Preference Learning for TQA

Given a table t, a question q, and a fine-tuned TQA model M_{ft} that outputs a reasoning chain r consisting of l steps: $\{k_1, k_2, ..., k_l\}$, along with a predicted answer a, our goal is to collect high-quality step-wise preference data (k_i^{good}, k_i^{bad}) from M_{ft} and perform contrastive learning to improve M_{ft}.

[1] https://github.com/boschresearch/p2-TQA



(a) Value estimation (b) Data sampling

Figure 2: Process-based preference data collection. We estimate a state value by the probability of a state leading to a correct answer. In the first example, $V(s_i) = \frac{2}{3}$. After obtaining state values, we do not consider intermediate states that have a value of 0 (s_{21}), together with their child states. We sample pair-wise states for each remaining state, e.g., s'_{22} is sampled by rolling out s_{12} and is regarded as a pair state for s_{22}.

Step Design and State Generation. Designing an effective step scope is crucial for both performance and sampling efficiency. Inspired by SQL query operations, we define a step as a basic operation, such as filtering or counting. To aid LLM reasoning in TQA, each step includes a planning component that outlines the required operation and a reasoning component that provides its result, e.g., *Count the number of gold medals received in 2004. There are 6 gold medals received in 2004.* For each TQA problem, an initial state s_0 is formed from t, q, and an instruction u. We then sample m reasoning chains, denoted as $\{r\}_{i=1}^{m}$ from M_{ft}. Prompts can be found in Appendix A.1. A new state s_i is defined as the combination of previous state s_{i-1} and a step k_i generated at the timestep i: $s_i = (s_{i-1}, k_i)$, where $s_{i-1} = \{s_0, k_1, k_2, ...k_{i-1}\}$. Problems where all reasoning traces lead to correct answers are discarded, as they are considered too easy.

State Value Estimation. A state value function V takes in a state and returns its value. We approximate a state's value by using Monte Carlo Sampling similar to Wang et al. (2024): M_{ft} takes in s_i and completes the current reasoning chain until reaching an answer. This is repeated n times. $V(s_i)$ is calculated as the probability of s_i leading to the correct answer. An example is shown in Figure 2, where $V(s_i) = \frac{2}{3}$. The continuous value allows a flexible and controlled selection of pair-wise steps described in the following paragraph.

Pair-wise Step Sampling. After obtaining state values, we filter out intermediate states s_i, where $V(s_i) = 0$, and also remove their child states $\{s_{i+1}, ...s_z\}$. This is exemplified by the red nodes

in Figure 2. We assume a state of value 0 to contain erroneous steps. Rolling out from it is likely to create bad-quality child states. For each remaining state s_i, we use the completion traces sampled when calculating $V(s_{i-1})$: $\{(s_{i,j}, ..., s_{z_j,j}, a_j)\}_{j=1}^{n}$ as rollouts, where a_j is the predicted answer and z_j is the total number of steps for the j-th finalized solution. Next, we calculate state values for each sampled $\{s_{i,j}\}_{j=1}^{n}$. This results in a set of pair-wise states: $(s_i, \{s_{i,j}\}_{j=1}^{n})$ that can be used to construct step-wise preference dataset D_{sdpo}. As $V(s_i)$ is a continuous value, a pair comprising one good state s_{good} and one bad state s_{bad} is selected if $V(s_{good}) - V(s_{bad}) \geq \tau$, where τ is a hyper-parameter. We prove later in our experiments that for TQA, this filtering mechanism greatly improves performance and efficiency on top of using MCS as the value function, while maintaining efficiency. The preference data for step DPO can be represented as $D_{sdpo} = \{(s_{i-1}, k_i^{good}, k_i^{bad})^d\}_{d=1}^{|D_{sdpo}|}$.

After collecting the preference dataset, we fine-tune M_{ft} using pairs of good and bad steps given previous steps. The loss function is defined as follows, where β is a hyper-parameter controlling the strength of incorporating the preference signal. π_{ref} and π_θ denote the original reference and updated model, respectively.

$$\mathcal{L} = -\mathbb{E}_{(s_{i-1},k_i^{good},k_i^{bad})\sim\mathcal{D}_{sdpo}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(k_i^{good}|s_{i-1})}{\pi_{ref}(k_i^{good}|s_{i-1})}\right) - \beta\log\frac{\pi_\theta(k_i^{bad}|s_{i-1})}{\pi_{ref}(k_i^{bad}|s_{i-1})}\right] \quad (1)$$

3 Experiments

We present details for TQA models, baselines, datasets, and experimental settings in this section.

TQA Models. Existing fine-tuned TQA models do not feature clear step separations. We therefore obtain a TQA model M_{ft} by fine-tuning an LLM using the step definition introduced before. Following previous work (Wu and Feng, 2024; Zhang et al., 2025a), we employ Deepseek-V3 (DeepSeek-AI et al., 2025) to generate reasoning chains. We sample 2.4k, 1.5k, and 2.3k examples from the training sets of WTQ (Pasupat and Liang, 2015), TabFact (Chen et al., 2020), and HiTab (Cheng et al., 2022), respectively. We prompt Deepseek-V3 to produce reasoning chains along with final answers, retaining only those chains that yield correct answers. This process results in 1,612 instances from WTQ, 1,425 from TabFact, and 1,277 from HiTab, for a total of 4,314 instances.

Baselines. We consider the following baselines for **self-improvement strategies**: (1) *RFT* (Yuan et al., 2023) trains a model with self-generated reasoning traces that lead to correct answers using supervised fine-tuning. (2) *FDPO* (Xu et al., 2024) trains a model with pair-wise correct and incorrect full reasoning chains using DPO.

Baselines for **value functions** include: (3) *MC with binary labels* (MC-B) (Wang et al., 2024) returns binary state values based on whether states derive final correct answers. *Mixed estimation* (MIX) (Zhang et al., 2025b) scores s_i 1 if both MC-B and an external LLM judge M_j output 1. If both judges return 0, s_i is 0. States receiving different scores from the judges are not considered for building the preference dataset. (4) SELF-EXPLORE (Hwang et al., 2024) randomly selects a preferred reasoning trace and uses full completion of it instead of a step. This results in longer preferred responses over rejected ones.

Baselines for **TQA models** include both end-to-end and training-free frameworks: (1) TableLLaMA (Zhang et al., 2023a) is an end-to-end fine-tuned model with LlaMA-2-7B (Touvron et al., 2023) as the base model. (2) Protrix (Wu and Feng, 2024) is fine-tuned with around 4k instances with reasoning chains generated from GPT-4, also using LlaMA-2-7B as the base model. (3) MACT (Zhou et al., 2025b) is a training-free framework, leveraging tools and agent collaboration. (4) TabSQLify (Nahid and Rafiei, 2024) decomposes tables into relevant sub-tables with SQL query generation and execution. Then, sub-tables and questions are passed to LLMs to obtain final answers.

Datasets. We train M_{ft} using the training sets of WTQ, TabFact, and HiTab. To obtain preference data, we sample from their validation sets. We use the test sets of these three datasets as in-domain evaluation data and incorporate three out-of-domain datasets: WikiSQL (Zhong et al., 2017), SCITAB (Lu et al., 2023), and CRT (Zhang et al., 2023b) to test models' generalisability. These datasets test our method in various degrees of complexity. Thus, we make sure our method is generally effective. Details about the datasets are presented in Appendix A.2.

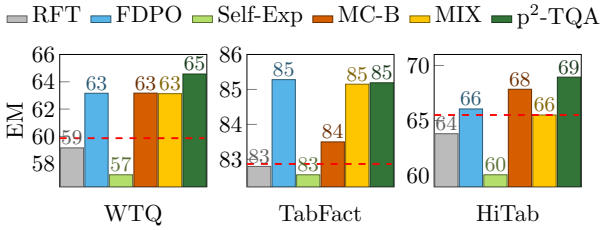


Figure 3: Comparing p^2-TQA with baselines using Exact Match. Results are averaged across models. RFT and FDPO stand for rejected sampling fine-tuning and full-chain DPO, respectively. We experiment with several value functions: SELF-EXP (Self-Exploration), MC-B (Monte Carlo with binary values), and MIX (a combination of LLM-as-a-judge and MC-B). Dashed lines show performances of fine-tuned TQA models M_{ft} before applying self-improvement methods.

Models	WTQ	TabFact	HiTab	WikiSQL	SCITAB	CRT
Protrix	56.2	71.6	-	67.4	45.0	40.2
T-LlaMA	35.0	82.6	64.7	50.5	38.6	26.9
M_{si}-Qwen	63.1	84.9	67.6	72.0	**56.9**	51.4
M_{si}-LlaMA	65.8	**85.5**	**70.3**	70.8	54.4	50.3
MACT	**70.4**	-	-	-	55.8	**57.4**
T-SQLify	64.7	80.2	-	**76.7**	50.9	42.0

Table 1: Exact Match of TQA models. M_{si} refers to self-improved models. T-LlaMA and T-SQLify refer to TableLlaMA and TabSQLify, respectively. Framework results (the last two rows) are obtained using GPT-3.5 as the backbone. State-of-the-art TQA results are obtained from previous work (Zhou et al., 2024; Zhang et al., 2023a; Wu and Feng, 2024; Nahid and Rafiei, 2024).

Experimental Settings. We choose Qwen-2.5-7B (Qwen et al., 2025) and LlaMA-3.1-8B (Grattafiori et al., 2024) as base models. During preference learning, we fix the fine-tuning dataset size to 8k for Qwen-2.5-7B and 6.7k for LlaMA-3.1-8B, as different baselines result in different sample sizes (statistics are shown in Appendix A.3). For fair comparison, we use the smallest sample size collected as the fine-tuning data size. Hyperparameters are shown in Appendix A.4. We use Qwen-2.5-72B as M_j (Prompt in Appendix A.1). The number of reasoning chains m is set to 4, and the roll-out number n is set to 8. The threshold τ is set to 0.9, and the temperature is set to 0.7 and 0 during dataset construction and inference, respectively. We use Exact Match (EM) as the evaluation metric. All experiments are conducted using 4 A100 GPUs. Training is performed with LlaMA-Factory (Zheng et al., 2024) and inference is performed with VLLM (Kwon et al., 2023).

4 Results and Discussions

Figure 3 shows the Exact Match of different methods on in-domain datasets, averaged across models. Per-model results can be found in Appendix A.5.

p^2-TQA effectively improves the performance of TQA models. As Figure 3 shows, applying p^2-TQA enhances the performance of M_{ft} by 3.5% on average on in-domain datasets. The gains are more obvious on WTQ (5%) compared to TabFact (2%). This might be attributed to dataset features: TabFact is a binary classification dataset, thus it is easier for models to achieve high performance and harder to further reach improvements. When evaluating on out-of-domain datasets, we witness an average of 2.2% performance gain after applying our framework (↑ 2.2% for WikiSQL, ↑ 2.4% for SCITAB, and ↑ 2.1% for CRT). These findings demonstrate the generalisability of our method on out-of-domain data. The performance gains on both in-domain and out-of-domain datasets are significant and can be observed with inference sampling over multiple runs. Detailed results are shown in Appendix A.5.

p^2-TQA delivers competitive results against baselines, highlighting the effectiveness of pairing a lightweight value function with stringent filtering for self-improving models. Comparing p^2-TQA with RFT and FDPO, we find that our framework leads to higher improvements. The effect is more obvious on in-domain datasets than out-of-domain datasets, as results in Appendix A.5 show. Though FDPO is generally computationally cheaper than p^2-TQA, i.e., under the same token budgets, it generates more training instances. We find that the performance of p^2-TQA improves when training example sizes increase. In contrast, more examples do not necessarily lead to better performance using FDPO, suggesting early saturation. More analysis is presented in Appendix A.6.

When comparing against methods using different value functions, we observe that p^2-TQA greatly outperforms SELF-EXPLORE. It also shows advantages over methods that simply use a binary value function (MC-B), or combining it with an LLM judge (MIX). Notably, p^2-TQA takes 10 times less time than using MIX when sampling the same amount of data. This demonstrates that our method delivers strong performance while being efficient. Interestingly, the efficiency is not compromised for

Model # instances	Retrieval 1133	Reasoning 451	Total 1584
Qwen (M_{ft})	71.32	36.36	61.36
Qwen (M_{ft}) +p^2-TQA	**78.02**	**41.24**	**67.55**
LlaMA (M_{ft})	79.44	**45.23**	69.70
LlaMA (M_{ft}) +p^2-TQA	**81.38**	42.57	**70.32**

Table 2: Exact Match of models with and without p^2-TQA, evaluated across different question types on the HiTab test set.

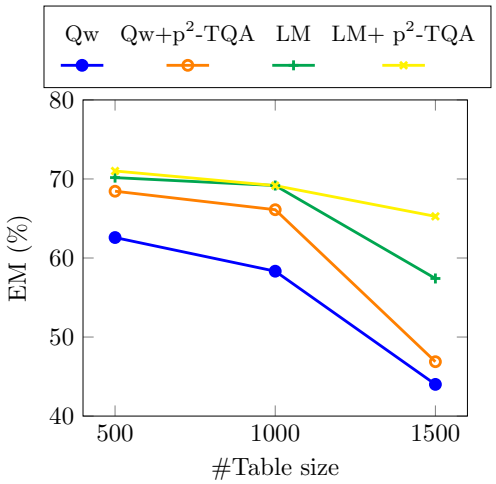


Figure 4: Exact Match of models with and without p^2-TQA, evaluated across different table sizes and averaged over in-domain datasets. Qw and LM show the performance of Qwen Mft and LlaMA Mft respectively. Instances are grouped into three bins by table token count: < 500, 500–1000, and ≥ 1000.

reasoning correctness. We evaluate models fine-tuned using data generated by p^2-TQA and MIX with regard to step correctness and find that the two methods achieve similar accuracy (95.7% vs. 94.6%). Detailed analysis can be found in Appendix A.7. We provide an analysis of threshold impact in Appendix A.8, showing the necessity of picking a relatively high threshold for data filtering.

Self-improved TQA models achieve competitive results compared to complicated state-of-the-art approaches, with five times less inference time. Table 1 shows results for current TQA models. first four rows of Table 1 compare small-size fine-tuned TQA models, and the last two rows show state-of-the-art training-free frameworks back-boned by GPT-3.5. We find that both the Qwen and LlaMA models, enhanced using p^2-TQA, outperform existing TQA models. More importantly, both self-improved models achieve competitive performance compared to larger and more complex frameworks where tools and agentic collaboration are involved. On SCITAB, M_{si} with the Qwen even achieves the best performance. Apart from the competitive task performance, we emphasize the inference efficiency of self-improved models: they require eight times less inference time than MACT and five times less than TabSQLify.

Applying p^2-TQA generally improves accuracy across question types, table sizes, and step correctness. We compare models fine-tuned with p^2-TQA against those without, from two perspectives: We take a closer look at question type and table size, inspired by Zhou et al. (2025a). For question type analysis, we categorize questions into those requiring only retrieval and those requiring reasoning in addition. We evaluate models on the HiTab dataset, which provides explicit question type annotations. As shown in Table 2, the enhanced Qwen-2.5-7B model outperforms its pre-trained counterpart in both categories, whereas the LLaMA-3.1-8B model shows notable gains primarily in retrieval questions. For table size analysis, we partition tables into three bins based on token count and compute EM accuracy for each bin. Figure 4 reports results averaged over in-domain datasets, revealing that self-improved models consistently achieve higher accuracy across all table sizes.

Finally, we examine step correctness by comparing models with (M_{si}) and without (M_{ft}) p^2-TQA. We randomly sample 50 instances from HiTab and manually examine the correctness of reasoning steps for both Qwen and LLaMA, yielding a total of 200 reasoning chains (50 instances × 2 models × 2 variants) and 245 steps for M_{ft} versus 239 steps for M_{si}. Models enhanced with p^2-TQA demonstrate higher step accuracy than those without (83% vs. 75%), with most improvements arising from reduced errors in planning and numerical reasoning.

5 Conclusions

We have introduced a self-improvement framework p^2-TQA that uses process-based preference learning. Our framework effectively improves the performance of TQA models by up to 5%. The resulting models demonstrate competitive performance compared to state-of-the-art TQA systems, which depend on huge LLMs and tool usage. Yet, models enhanced with p^2-TQA require five times less inference time.

Limitations

First, While our method effectively enhances the performance of small-sized TQA models, its impact on large TQA models remains unexplored. To the best of our knowledge, current fine-tuned TQA models only focus on small-sized LLMs. Future work can explore an efficient training strategy for large fine-tuned TQA models. Second, we limit the task in our study to only TQA, while there exist other table-related tasks, such as table summarization. Third, although our framework supports iterative self-learning, the present work only demonstrates the effectiveness of the first iteration, leaving multi-iteration evaluations for future study. As the datasets we used in this study are originally sourced from Wikipedia, scientific papers, and statistical reports, we do not observe any potential risks from the datasets.

References

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, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. Deepseek-v3 technical report.

Naihao Deng and Rada Mihalcea. 2025. Rethinking table instruction tuning.

Naihao Deng, Sheng Zhang, Henghui Zhu, Shuaichen Chang, Jiani Zhang, Alexander Hanbo Li, Chung-Wei Hang, Hideo Kobayashi, Yiqun Hu, and Patrick Ng. 2025. Towards better understanding table instruction tuning: Decoupling the effects from data versus models.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen,

Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov,

Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models.

Yifei He, Haoxiang Wang, Ziyan Jiang, Alexandros Papangelis, and Han Zhao. 2024. Semi-supervised reward modeling via iterative self-training.

Hyeonbin Hwang, Doyoung Kim, Seungone Kim, Seonghyeon Ye, and Minjoon Seo. 2024. Self-explore: Enhancing mathematical reasoning in language models with fine-grained rewards. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1444–1466, Miami, Florida, USA. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms.

Xinyuan Lu, Liangming Pan, Qian Liu, Preslav Nakov, and Min-Yen Kan. 2023. SCITAB: A challenging benchmark for compositional reasoning and claim verification on scientific tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7787–7813, Singapore. Association for Computational Linguistics.

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.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. 2024. Beyond human data: Scaling self-training for problem-solving with language models.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Niko lay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cris tian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich,

Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melissa Hall Melanie Kambadur, Sharan Narang, Aur'elien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Songjun Tu, Jiahao Lin, Xiangyu Tian, Qichao Zhang, Linjing Li, Yuqian Fu, Nan Xu, Wei He, Xiangyuan Lan, Dongmei Jiang, and Dongbin Zhao. 2025. Enhancing llm reasoning with iterative dpo: A comprehensive empirical investigation.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

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.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! LLM agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, Miami, Florida, USA. Association for Computational Linguistics.

Yifan Xu, Xiao Liu, Xinghan Liu, Zhenyu Hou, Yueyan Li, Xiaohan Zhang, Zihan Wang, Aohan Zeng, Zhengxiao Du, Zhao Wenyi, Jie Tang, and Yuxiao Dong. 2024. ChatGLM-math: Improving math problem-solving in large language models with a self-critique pipeline. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9733–9760, Miami, Florida, USA. Association for Computational Linguistics.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. Self-taught optimizer (stop): Recursively self-improving code generation.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023a. Tablellama: Towards open large generalist models for tables. In *North American Chapter of the Association for Computational Linguistics*.

Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, Jifan Yu, Shu Zhao, Juanzi Li, and Jie Tang. 2025a. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023b. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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

Wei Zhou, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. 2024. FREB-TQA: A fine-grained robustness evaluation benchmark for table question answering. 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 2479–2497, Mexico City, Mexico. Association for Computational Linguistics.

Wei Zhou, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. 2025a. Texts or images? a fine-grained analysis on the effectiveness of input representations and models for table question answering. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2307–2318, Vienna, Austria. Association for Computational Linguistics.

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025b. Efficient multi-agent collaboration with tool use for online planning in complex

table question answering. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 945–968, Albuquerque, New Mexico. Association for Computational Linguistics.

A Appendix

A.1 Prompts

Figure 6, 7, and 8 show prompts for generating a full reasoning trace, completing a reasoning trace, and LLM judge evaluation for a reasoning trace.

A.2 Datasets

Table 3 shows the number of instances and domains for the test data we used. WTQ (Pasupat and Liang, 2015), HiTab (Cheng et al., 2022) and WikiSQL (Zhong et al., 2017) are under the license of CC-BY-SA-4.0[2], BSD-3 CLAUSE[3] and C-UDA[4] respectively. TabFact (Chen et al., 2020), CRT (Zhang et al., 2023b) and SCITAB (Lu et al., 2023) are under the MIT[5] license.

Datasets	#instances	Domain
WTQ	4344	Wikipedia
TabFact	12779	Wikipedia
HiTab	1584	statistical reports
WikiSQL	15878	Wikipedia
SCITAB	1224	scientific paper
CRT	728	Wikipedia

Table 3: Test data statistics. The second column shows the number of test instances in each dataset.

A.3 Sampled Dataset Statistics

Table 4 shows the sampling size for each method. We find MC-B results in the most data while RFT the least.

A.4 Hyper-parameters

Table 5 shows the hyper-parameters used for model fine-tuning.

A.5 Additional Results

Table 6 shows different models' performance on the six investigated datasets under greedy decoding. To validate the effectiveness of applying p^2-TQA, we set the temperature to 0.8 during inference. We report the mean and standard deviation over five runs in Table 7.

Methods	HiTab	WTQ	TabFact	Total
Original	1.6k	2.8k	5k	9.4k
RFT	2.2k/1.6k	2.7k/2.4k	3k/2.7k	8k/6.7k
FDPO	3.6k/2k	4.7k/4k	4.6k/3.6K	12.9k/9.6k
SDPO				
+MC-B	25k/20.8k	46k/47.1k	37k/38.4k	109k/106k
+MIX	5.4k/3.5k	9.6k/9.7k	5k/4.9k	20k/18.1k
+MC-0.9	6.8k/1.6k	16k/4.4k	11k/2.4k	33.8k/8.4k

Table 4: Sampled dataset sizes for different methods. Results for Qwen-2.5-7B and LlaMA-3.1-8B are separated by "/". MC-B refers to using Monte Carlo sampling with binary values as the value function. MIX stands for using both MC-B and an LLM judge (Qwen-2.5-72B) as the value function. MC-0.9 stands for using Monte Carlo sampling with continuous values and setting the selection threshold as 0.9. RFT refers to rejected sampling fine-tuning. FDPO and SDPO stand for full-trace DPO and step-wise DPO.

A.6 Cost Effectiveness Analysis

Note that step-wise sampling requires higher computing budgets than full chain sampling; we conduct a cost-effectiveness analysis over FDPO and p^2-TQA. We do that by examining models' performance under different computing budgets (approximately by training instances). We first calculate the average number of tokens needed to generate an instance for different methods. This results in approximately 3k for FDPO and 10K for p^2-TQA. We fine-tune models with varying sizes of training samples. This not only allows us to compare FDPO and p^2-TQA under the same computing budgets, but also demonstrates each method's sensitivity to training size. Table 8 shows the results. We observe that for FDPO, adding more training data does not necessarily improving models' performance (69.1–>67.1). In contrast, scaling training sizes remains effective for p^2-TQA. This suggests that though FDPO generates more training instances than our method under the same budgets, the real effect of the generated data on performance is limited.

A.7 Reasoning Chains Analysis

We sample 100 reasoning chains leading to correct answers generated from models using p^2-TQA and models using MIX as the value function. We manually examine the correctness of the reasoning chains. Among the 100 instances, we exclude 8

[2] https://creativecommons.org/licenses/by-sa/4.0/

[3] https://opensource.org/license/bsd-3-clause

[4] https://github.com/microsoft/HiTab?tab=License-1-ov-file

[5] https://opensource.org/license/mit

Models	Method	Fine-tuning	Learning rate	Epoch	Batch size	LoRA rank	DPO β
Qwen-2.5-7B	supervised fine-tuning	full-parameter	5e-6	2	128	-	-
Qwen-2.5-7B	rejected sampling fine-tuning	LoRA	1e-5	1	128	64	-
Qwen-2.5-7B	full chain DPO	LoRA	1e-5	3	128	64	0.1
Qwen-2.5-7B	step-wise DPO	LoRA	1e-5	3	128	64	0.1
LlaMA-3.1-8B	supervised fine-tuning	full-parameter	5e-6	2	128	-	-
LlaMA-3.1-8B	rejected sampling fine-tuning	LoRA	1e-5	1	128	32	-
LlaMA-3.1-8B	full chain DPO	LoRA	1e-5	3	128	32	0.1
LlaMA-3.1-8B	step-wise DPO	LoRA	1e-5	3	128	32	0.1

Table 5: Hyper-parameters used for model fine-tuning.

Models	WTQ	TabFact	HiTab	WiKiSQL	SCITAB	CRT	In-domain	Out-of-domain
Qwen-2.5-7B	28.66	73.77	26.07	47.76	39.46	35.58	42.83	40.93
+ TQA training (M_{ft})	54.93	82.37	61.36	68.26	54.90	48.49	66.22	57.22
+RFT	56.26	82.44	60.29	68.15	52.29	46.15	66.33	55.53
+FDPO	61.10	84.65	63.83	**72.04**	**56.94**	52.06	69.86	**60.34**
+SELF-EXPLORE	52.70	82.30	55.62	65.20	53.67	44.23	63.54	54.37
MC-B	60.80	82.98	65.72	70.08	52.04	50.24	69.83	57.45
MIX	**63.86**	**85.32**	64.20	71.44	55.56	**52.47**	71.63	59.82
P^2-TQA ($\tau = 0.9$)	63.10	84.88	**67.55**	71.97	**56.94**	51.37	**71.84**	60.09
LlaMA-3.1-8B	30.64	63.91	26.20	31.87	43.38	32.55	40.25	35.93
+ TQA training (M_{ft})	64.80	83.36	69.63	69.48	51.63	49.04	72.60	52.76
+RFT	62.06	84.15	67.30	70.73	50.49	48.63	70.84	56.62
+FDPO	65.22	**85.91**	68.18	**71.34**	53.35	50.41	73.10	**58.37**
+SELF-EXPLORE	61.67	82.83	64.52	67.79	52.20	44.28	69.67	54.76
MC-B	65.54	84.01	69.95	70.67	50.25	49.31	73.17	56.74
MIX	62.39	84.98	66.79	68.31	52.94	**51.24**	72.05	57.50
P^2-TQA ($\tau = 0.9$)	**65.84**	85.50	**70.33**	70.08	**54.44**	50.27	**73.98**	58.26

Table 6: Exact Match accuracies of models fine-tuned with different strategies and value functions, generated with greedy decoding. We bold the best results and underline the second best results for each model type.

instances where either the answers are incorrect or the questions are ambiguous. We find similar accuracies of the reasoning chains generated from the aforementioned methods, with 95.7 and 94.6, respectively. This suggests the two methods do not differ much in terms of leading to correct reasoning chains. Nevertheless, wrong reasoning chains leading to correct answers still exist, possibly due to overly complex table inputs. An error case is shown in Figure 9.

A.8 Threshold Analysis

The threshold τ decides the state value differences when sampling a pair of (preferred and not preferred) states. We set τ to 0.9 in our study. We experiment with different values of τ to investigate its impact on the fine-tuned process-supervised models. The experimental settings are the same as described in Section 3 except that we change the values of τ. Figure 5 shows the performances of models fine-tuned with data sampled using different τ. We observe that there is a tendency for higher thresholds to lead to better performance. However,

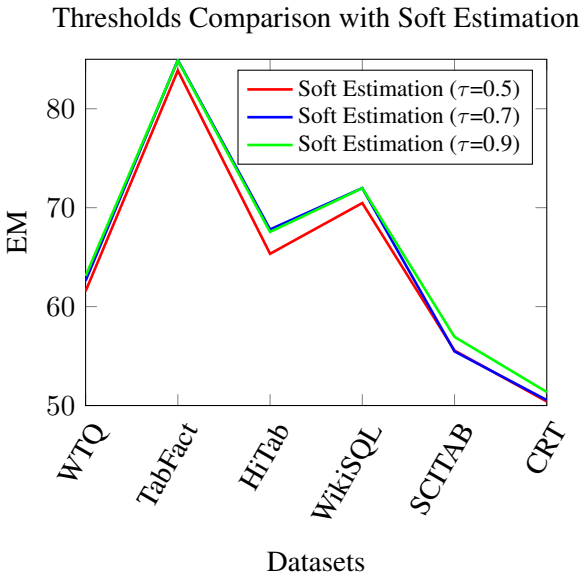


Figure 5: Thresholds comparisons with different value functions on six TQA datasets.

Models	WTQ	TabFact	HiTab	WiKiSQL	SCITAB	CRT
Qwen-2.5-7B (M_{ft})	57.00 ±0.46	81.85 ±0.22	61.76 ±0.22	67.68 ±0.22	52.16 ±1.13	46.92 ±1.15
+RFT	51.08 ±0.60	80.28 ±0.32	56.92 ±0.55	61.77 ±0.08	49.90 ±0.83	46.02 ±0.60
+FDPO	58.54 ±0.23	84.32 ±0.28	61.40 ±0.49	68.20 ±0.23	**56.98** ±0.80	49.64 ±1.40
MC-B	57.36 ±0.33	82.74 ±0.04	62.63 ±0.65	66.39 ±0.17	52.71 ±0.71	48.71 ±1.00
MIX	**60.43** ±0.57	84.79 ±0.10	62.58 ±0.41	68.59 ±0.14	51.98 ±1.05	**52.09** ±0.37
P^2-TQA	60.42 ±0.29	**84.81** ±0.21	**65.05** ±0.80	**68.90** ±0.13	55.08 ±1.04	50.99 ±0.80
LlaMA-3.1-8B (M_{ft})	57.67 ±0.38	82.82 ±0.29	63.18 ±0.80	62.17 ±0.18	49.98 ±1.34	47.86 ±0.65
+RFT	58.27 ±0.47	81.44 ±0.15	63.14 ±0.70	65.47 ±0.22	47.83 ±0.98	47.91 ±1.61
+FDPO	62.18 ±0.41	85.52 ±0.34	65.83 ±0.43	**68.12** ±0.18	52.94 ±0.39	**50.41** ±0.60
MC-B	60.92 ±0.18	83.98 ±0.17	65.86 ±0.69	65.86 ±0.30	52.25 ±1.54	49.07 ±0.83
MIX	59.57 ±0.62	84.99 ±0.07	62.11 ±0.44	63.23 ±0.19	51.42 ±0.63	50.05 ±1.09
P^2-TQA	**62.66** ±0.55	**85.71** ±0.23	**66.49** ±0.79	65.83 ±0.26	**53.76** ±0.56	49.15 ±1.20

Table 7: Exact Match accuracies of models fine-tuned with different strategies and value functions, generated with sampling. P^2-TQA significantly improve fine-tuned model. Compared to baselines, it achieves competitive performance across in-domain datasets.

Method	2k	4k	6k	8k	12k
FDPO	67.3±0.6	68.2±0.5	68.5 ±0.4	69.1±0.4	67.1±0.3
p^2-TQA	69.0 ±0.5	69.7±0.3	70.8±0.3	71.4±0.2	72.9±0.2

Table 8: Exact Match against varying training sizes. Results are obtained by averaging across three runs and three in-domain datasets using Qwen-2.5-7B.

we do not observe big differences in terms of model performances when setting τ to 0.7 or 0.9.

```
You are an expert in table question answering.
Based on the given question and table, provide a step by step solution to the question.
Start each step with 'Step x.' where x is the current step number.
Do not carry out verification in each step.
Each step should include two parts: a planning part that indicates what to do and a reasoning part that returns the results of the planning part.
Separate these two parts via the [SEP] token.
Return the result in the last line following 'Therefore, the final answer is: '.
Table: {table}
Question: {question}
```

Figure 6: Prompt to generate full reasoning trace given a TQA problem.

```
Given the following table, question and past steps to solve the question, continue to generate the steps following past steps to obtain an answer.
Each step should include two parts: a planning part that indicates what to do and a reasoning part that returns the results of the planning part.
Separate these two parts via the [SEP] token.
Start each step with 'Step x.' where x is the current step number.
Do not carry out verification in each step.
Return the result in the last line following 'Therefore, the final answer is: '.
Table: {table}
Question: {question}
Past steps: {steps}
```

Figure 7: Prompt to complete a reasoning trace given a TQA problem and past steps.

I will provide a table question answering(TQA) problem along with a step-by-step reasoning to solve the problem. They will be formatted as follows: [TQA Problem]
...(TQA problem)...
[Solution]
<step_1 >
...(step 1 of a plan)...
</step_1 >
...
<step_n >
...(step n of a plan)...
</step_n >
Your task is to review each step of the plan in sequence, analyzing, verifying, and critiquing a step in details to decide if a step is helpful or not for solving the problem.
A helpful step has the following features:
- It provides unique information about how to solve a question and does not repeat information appeared in the previous steps.
- It is relevant to solving the question.
- It is correct in terms of the reasoning.
Please provide your analyses, decisions (1 for helpful and 0 not helpful) for each step and confidence for each decision (from 0 to 1).
Higher confidence value suggests you are more certain that your decision is correct.
In contrast, lower confidence suggests you are more uncertain about your decision and your decision might be incorrect.
Please do not be over confident about your decision.
You should be honest if you are not sure about your decision.
Please return your response strictly following the format below:
<analysis_1 >
...(analysis of step 1)...
</analysis_1 >
<decision_1 >
1/0
</decision_1 >
<confidence_1 >
(a numerical value from 0 to 1)
</confidence_1 >
...
<analysis_n >
...(analysis of step n)...
</analysis_n >
<decision_n >
1/0
</decision_n >
<confidence_n >
(a numerical value from 0 to 1)
</confidence_n >

The following is a TQA problem and a step-by-step solution to solve the problem:
[TQA Problem]
{problem}
[Solution]
{solution}

Figure 8: LLM judge prompt to generate analysis, decisions and confidence for each step.

Table:
| # | Date | Location | Winner | Score\nJSU | Score\nTU | Series |
| 1 | November 27, 1924 | Jacksonville, AL | Jacksonville State | 14 | 9 | JSU 1–0 |
…
| 8 | November 11, 1938 | Jacksonville, AL | Tied | 6 | 6 | TSU 4–3–1 |
…
| 21 | October 15, 1955 | Troy, AL | Jacksonville State | 12 | 0 | Tied 10–10–1 |
…
| 28 | October 6, 1962 | Jacksonville, AL | Jacksonville State | 21 | 14 | JSU 17–10–1 |
…
| 38 | November 11, 1972 | Jacksonville, AL | Tied | 14 | 14 | JSU 22–14–2 |
…
| | Totals | | | 1086 | 1110 | JSU 32–29–2 |

Question:what is the least amount of points scored in a tie game?

Step 1. Identify all the tie games in the table by checking the "Score\nJSU" and "Score\nTU" columns for values that are the same.
#	Date	Location	Winner	Score\nJSU	Score\nTU	Series
8	November 11, 1938	Jacksonville, AL	Tied	6	6	TSU 4–3–1
21	October 15, 1955	Troy, AL	Jacksonville State	**12	0**	Tied 10–10–1
28	October 6, 1962	Jacksonville, AL	Jacksonville State	**21	14**	JSU 17–10–1
38	November 11, 1972	Jacksonville, AL	Tied	14	14	JSU 22–14–2

Step 2. Extract the scores of the tie games identified in Step 1.
Score\nJSU	Score\nTU
6	6
12	0
21	14
14	14

Step 3. Determine the least amount of points scored in a tie game by finding the minimum value among the extracted scores.
The scores are 6, 12, 21, and 14. The least amount of points scored in a tie game is 6.

Therefore, the final answer is: 6.

Figure 9: Wrong reasoning chain generated by a self-improved model (Qwen-2.5-7B) using P^2-TQA. The first wrong step is highlighted with red.