Towards Interpretable Tabular Reasoning: Enhancing LLM Reasoning on Tabular Data with Pre-Constructed Logic Graph

Lirong Gao[1,2*], Zewei Yu[1,2], Zhongrui Yin[1], Qi Zhang[1],
Yuke Zhu[2], Bo Zheng[2], Haobo Wang[1], Junbo Zhao[1], Gang Chen[1†], Sheng Guo[2†]
[1]State Key Laboratory of Blockchain and Data Security, Zhejiang University
[2]MYbank, Ant Group
{gaolirong,wanghaobo,j.zhao,cg}@zju.edu.cn,
{guosheng.guosheng}@mybank.cn

Abstract

Tabular data analysis is critical in high-stakes domains like finance and healthcare, where both accuracy and interpretability are paramount. Traditional tree-based models excel in performance but suffer from poor interpretability and reliance on rigid feature engineering. Large Language Models (LLMs), while transparent in reasoning, often fail to capture the precise statistical distributions needed for accurate tabular prediction. To bridge this gap, we propose LogGER (**Log**ic-**G**raph-**E**nhanced LLM **R**easoning), a novel framework that harmonizes the predictive power of trees with the reasoning transparency of LLMs. Unlike existing hybrids, LogGER explicitly distills statistical patterns from data priors into a human-readable *Logic Graph*, transforming numerical decision into correlation semantic chains. This graph then serves as a verifier for our Process Supervision mechanism. Specifically, we employ a process reward model that steers the LLM's intermediate reasoning steps, ensuring alignment with the established logic graph. Extensive experiments demonstrate that LogGER not only achieves state-of-the-art accuracy, consistently outperforming both tree-based models and LLM baselines, but also provides verifiable, transparent decision paths, establishing a new paradigm for interpretable tabular reasoning.

1 Introduction

Tabular data, consisting of rows as samples and columns as heterogeneous features (such as categorical and numerical attributes), is fundamental to real-world domains like finance (Assefa et al., 2020) and healthcare (Hernandez et al., 2022). Given the heterogeneity of features (Borisov et al., 2022; Gorishniy et al., 2021), tree-based models have been proven effective for tabular prediction (Breiman, 2001; Chen and Guestrin, 2016; Ke et al., 2017; Grinsztajn et al., 2022). However, their performance often hinges on labor-intensive feature engineering, which hinders their adaptability across diverse tasks. Additionally, the complex structures of tree-based models, especially ensembles, make their decision-making process difficult for humans to trust and intuitively understand, significantly limiting their interpretability.

Recently, LLMs have exhibited groundbreaking capabilities in solving complex reasoning tasks (Yang et al., 2024b; Jaech et al., 2024; Team, 2024; Guo et al., 2025), sparking growing interest in applying LLMs to structured tabular data tasks. A common line of work first serializes tabular samples into natural language descriptions and then fine-tunes LLMs on this serialized corpus (Hegselmann et al., 2023; Zhang et al., 2023, 2024; Gardner et al., 2024). These approaches typically follow an end-to-end paradigm, where models are fine-tuned or prompted to directly map serialized tabular inputs to prediction outputs. This end-to-end paradigm often suffers from limited interpretability, making it difficult to understand the model's decision process. To address this, some researchers have explored chain-of-thought (CoT) prompting (Wei et al., 2022; Xu et al., 2024) and inference-time scaling (Snell et al., 2024; Yang et al., 2025b; Jin et al., 2025) to elicit intermediate reasoning steps of LLMs. Such intermediate reasoning steps increase decision-making transparency to some extent, but they do not always align with the reasoning processes of domain experts. For example, they may overlook important features or misinterpret correlation among features, ultimately resulting in subpar performance on tabular data.

From a fundamental perspective, tabular data represents a highly structured form of information. Reasoning over such data typically follows a hierarchical and structured pathway: perform in-depth analyses of individual key input features

[*]Work done during internship at MYbank, Ant Group
[†]Corresponding Authors

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 30260–30280
July 2-7, 2026 ©2026 Association for Computational Linguistics

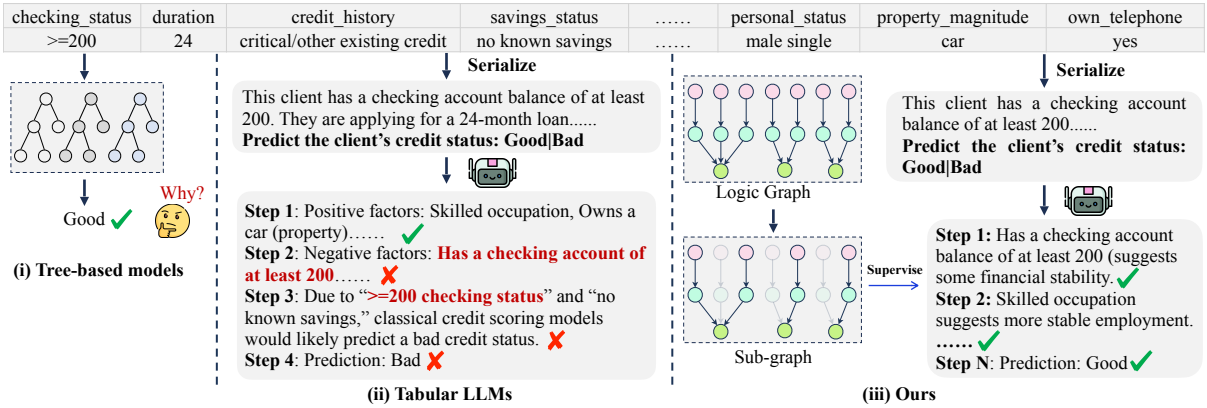

Figure 1: Comparison of tabular prediction methods: (i) Tree-based Models: Predicts targets with a hierarchical tree structure. (ii) Tabular LLMs: Serializes tables into text for LLMs, but may lead to unfaithful reasoning. (iii) Ours: Introduces a logic graph to supervise and verify LLM reasoning paths, improving accuracy and faithfulness.

to derive preliminary insights, and subsequently integrate these partial analyses to reach a final outcome. Clearly, relying solely on the text-based reasoning processes of LLMs to emulate this structured analytic process is difficult and unreliable. Some studies have tried to address this gap by organizing the LLM reasoning process in a tree-like manner, such as tree-based approaches like Tree-of-Thoughts (ToT) (Yao et al., 2023; Zhang et al., 2025; Guan et al., 2025). However, they still largely depend on the LLM's internal, unstructured reasoning capabilities, rather than explicitly utilizing the structural signals present in tabular data. Therefore, there is a pressing need to design new tabular reasoning paradigms that are compatible with both structured tabular prediction and LLM reasoning mechanisms.

To address these challenges, we propose a novel **Log**ic-**G**raph-**E**nhanced LLM **R**easoning (**LogGER**) framework for tabular prediction. Our main goal is to integrate the strengths of tree-based models and LLMs. Inspired by tree-based models, we design a directed acyclic graph, called a *logic graph*, to represent the decision logic of tabular prediction. As is shown in Figure 1, unlike traditional trees, the logic graph connects nodes defined by natural language descriptions to model correlation between features and targets, enabling straightforward translation into natural language and seamless integration with LLMs. To effectively leverage this logic graph, we further develop a **logic-graph-enhanced process supervision** method that constrains and guides the LLM's intermediate reasoning. Specifically, we design a novel process reward function based on the logic graph to evaluate intermediate reasoning steps and use large-scale reward-annotated data to train a process reward model (PRM) for more effective supervision. Guided by this reward mechanism, LogGER is able to generate high-quality reasoning trajectories, which are used to train a policy model for accurate and interpretable tabular reasoning. Extensive experiments show that LogGER significantly outperforms traditional tree-based models and mainstream LLM methods on various tabular prediction tasks, achieving superior accuracy and interpretability.

2 Related Works

2.1 Tree-based Models for Tabular Prediction

Predictive modeling on tabular data has been a longstanding research topic. Decision tree-based models (Breiman, 2001; Chen and Guestrin, 2016; Ke et al., 2017), particularly ensemble methods like Random Forests (Breiman, 2001) and gradient boosting (Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018), have become dominant due to their ability to naturally handle heterogeneous features. However, these models face two key limitations: (1) they require labor-intensive feature engineering that limits generalizability, and (2) their complex ensemble structures lack interpretability for human understanding.

2.2 LLMs for Tabular Prediction

Motivated by the success of LLMs (OpenAI, 2024; Anthropic, 2024; Meta, 2024; Yang et al., 2025a), recent research has explored applying LLMs to tabular prediction. Early efforts (Yin et al., 2020; Bertsimas et al., 2022; Dinh et al., 2022; Hegselmann et al., 2023) focused on bridging the structural gap

between structured data and text for traditional LMs. More recently, works including TableLlama (Zhang et al., 2023), TableLLM (Zhang et al., 2024; Gardner et al., 2024), and TableGPT-series (Zha et al., 2023; Su et al., 2024) have constructed serialized tabular instruction datasets for fine-tuning LLMs. Despite these advances, most tabular LLMs are fine-tuned in an end-to-end manner, and their decision process can only be revealed through prompting methods such as CoT (Wei et al., 2022), which lacks decision transparency. Recently, LLMs based on test-time scaling (Jaech et al., 2024; Guo et al., 2025; Team, 2024) have shown superior performance on various tasks, and have also inspired attempts on tabular tasks, such as Table-R1, etc (Jin et al., 2025; Yang et al., 2025b). These works effectively improve decision transparency, but there are still some hallucination issues: LLMs may generate unfaithful reasoning paths that do not align with expert experience, which limits their faithfulness and interpretability.

3 Methodology

In this section, we introduce the **Lo**gic-**G**raph-**A**ugmented LLM **R**easoning (**LogGER**) framework, which consists of two main stages: (i) constructs a logic graph through LLMs from tabular data, (ii) leverages this logic graph to enhance LLM reasoning in tabular prediction tasks.

3.1 Task Formulation

Tabular Prediction Given a tabular dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ with N samples and n features, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}] \in \mathcal{X}$ represents the input features and $y_i \in \mathcal{Y}$ denotes the class label (classification) or continuous value (regression). The goal is to learn a model $\mathbf{G} : \mathcal{X} \to \mathcal{Y}$ that minimizes the expected loss $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\mathcal{L}(\mathbf{G}(\mathbf{x}), y)]$, where $\mathcal{L}$ is a task-specific loss function (e.g., cross-entropy or mean squared error).

LLM-based Tabular Prediction To use an LLM for tabular data, the table must first be transformed into a natural text representation. Typically, given feature names $\mathbf{f}$ and feature values $\mathbf{x}$, we define a function $\text{serialize}(F, \mathbf{x})$ that serializes the structural data into natural language. After serialization, a policy LLM $\mathbf{M}$ is utilized to perform tabular prediction on the serialized text: $\hat{y} = \mathbf{M}(\text{serialize}(\mathbf{f}, \mathbf{x}))$. The goal of LLM-based tabular prediction is to minimize $\mathbb{E}[\mathcal{L}(\hat{y}, y)]$. Furthermore, for LLM with intermediate reasoning steps t, the model outputs both the reasoning trace and the prediction: $t, \hat{y} = \mathbf{M}(\text{serialize}(\mathbf{f}, \mathbf{x}))$. In this work, we primarily focus on LLMs with intermediate reasoning steps, as these LLMs provide transparent decision-making processes.

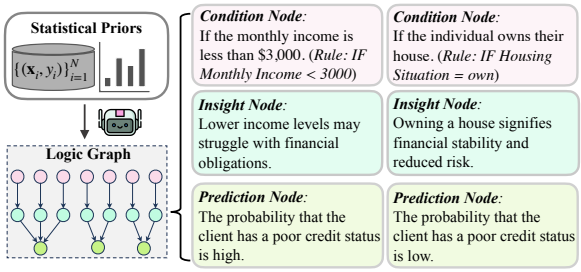

Figure 2: An illustration of *logic graph* construction. The constructed *logic graph* contains three types of nodes: *Condition*, *Insight*, and *Prediction nodes*.

3.2 Logic Graph

Our primary objective is to integrate structural signals from tree-based models into LLMs for tabular reasoning. However, a fundamental misalignment exists: tree models encode reasoning through opaque hierarchical structures and numerical feature statistics, while LLMs process human-readable natural language. This gap prevents LLMs from effectively utilizing tree-based signals. To bridge this gap, we propose a human-readable logic graph that transforms tree models' hierarchical reasoning paths into a directed graph with natural language descriptions.

Definition *A logic graph is a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$ that encodes decision paths from features to predictions in natural language, where $V = V^C \cup V^I \cup V^P$ represents logic nodes and E represents edges. The node set V comprises three types:* ***Condition nodes*** *(V^C) defining feature partitioning rules,* ***Insight nodes*** *(V^I) providing intermediate analysis for features satisfying split criteria, and* ***Prediction nodes*** *(V^P) outputting probabilistic predictions. These nodes form directed reasoning paths as illustrated in Figure 2.*

For a given input $\mathbf{x}_i$, its reasoning path is constructed by aggregating single-feature paths:

$$g(\mathbf{x}_i) = \cup_{m=1}^{n}\, g(x_{im}) = (\cup_{m=1}^{n} V_{im}, \cup_{m=1}^{n} E_{im}) \quad (1)$$

where $g(x_{im})$ denotes the reasoning path for feature x_{im} extracted from $\mathcal{G}$, with $V_{im} \subseteq V$ representing the associated nodes and $E_{im} \subseteq E$ the

connecting edges.

Construction Since the logic graph represents tree structures through natural language descriptions rather than preserving the original tree format, traditional tree-based learning methods cannot be directly applied for logic grpah construction. Instead, we leverage LLMs to automatically generate node descriptions. To ensure the logic graph captures statistical priors from the data, we first extract these priors from the dataset and then incorporate them into the LLM-based construction process:

i) Data Priors Extraction. We extract statistical priors $P_{stat}(\mathcal{D})$ from the dataset to guide logic graph construction, including: (1) distributions of categorical features, (2) key statistics of numerical features (mean, median, standard deviation, extrema), and (3) conditional probabilities $P(o \mid f)$ quantifying how feature value f influences outcome o. The conditional probability is computed as:

$$P(o \mid f) = \frac{\text{count}(x_{im} = f, y_i = o)}{\text{count}(x_{im} = f)}, \quad (2)$$

where $\text{count}(x_{im} = f)$ denotes the number of samples with feature value $x_{im} = f$, and $\text{count}(x_{im} = f, y_i = o)$ counts samples satisfying both conditions. Details on other priors are in Appendix A.

ii) Graph Construction. We construct a structured prompt by integrating input features $\mathbf{x}$, task description desc, and statistical priors $P_{stat}(\mathcal{D})$. An LLM then generates natural language descriptions for the three node types:

$$V^C, V^I, V^P = \text{LLM}(\text{prompt}(\mathbf{x}, \text{desc}) \oplus P_{stat}(\mathcal{D})). \quad (3)$$

We employ the advanced GPT-4o model to ensure logical coherence and accuracy. For *Condition nodes*, the LLM generates both a semantic description (e.g., "If monthly income is less than $3,000") and its corresponding matching rule (e.g., "IF Monthly Income < 3000"). Further details and a representative example are provided in Appendix G.2 and Figure 2, respectively.

3.3 Logic-Graph-Enhanced Process Supervision

Another key challenge is effectively leveraging the logic graph to guide LLM reasoning. Previous methods(Yao et al., 2023; Zhang et al., 2025; Guan et al., 2025) achieve suboptimal performance because they fail to incorporate the structural signals in tabular data. We address this by explicitly integrating the structured logic graph into the reasoning process. Inspired by recent advances in process supervision(Wang et al., 2024; Zhang et al., 2025; Guan et al., 2025), we propose a logic-graph-enhanced process supervision approach. As shown in Figure 3, we design a logic-graph-aided reward mechanism to generate high-quality reasoning trajectories for training both the PRM and policy model.

Process-Level Reward Design Given input features $\mathbf{x}_i$, the policy model M generates a step-by-step reasoning trajectory $t_i = \text{serialize}(\mathbf{f}, \mathbf{x}_i) \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_a$ to predict the target y_i. Each step s_j in the path is a correlation analysis over one or more input features. s_a is the outcome step. The process reward based on logic graph for each step s_j is computed through two stages:

i) Subgraph Retrieval. To evaluate each reasoning step in proper context, we retrieve the relevant subgraph from the logic graph $\mathcal{G}$ based on the features involved in that step. Specifically, we prompt the policy model to append feature names after each step using few-shot learning (Brown et al., 2020), marked as *<mentioned_key>*. These feature names are then used to extract corresponding values $\mathbf{x}_i^j$ from $\mathbf{x}_i$ and retrieve the associated logic subgraph $g(\mathbf{x}_i^j)$ from $\mathcal{G}$.

ii) Process Reward Assignment. The logic-graph-aided process reward $r_{\text{LG}}(s_j)$ measures consistency between reasoning step s_j and the retrieved subgraph $g(\mathbf{x}_i^j)$:

$$r_{\text{LG}}(s_j) = \begin{cases} 1, & \text{if } s_j \text{ is consistent with } g(\mathbf{x}_i^j), \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

Since both s_j and $g(\mathbf{x}_i^j)$ are textual, we train a logic consistency verifier to assess their alignment. We collect reasoning steps and their corresponding retrieved subgraphs, using GPT-4o (OpenAI, 2024) to generate verification labels. The trained verifier achieves over 90% accuracy (details in Appendix E.5), enabling reward computation as:

$$r_{\text{LG}}(s_j) = \text{verifier}(s_j, \text{serialize}(g(\mathbf{x}_i^j))). \quad (5)$$

Large-Scale Reward Annotation Then, we use the logic-graph-aided process reward function to supervise the standard Monte Carlo Tree Search (MCTS) to generate stepwise verified reasoning trajectories. Notably, unlike standard MCTS, which

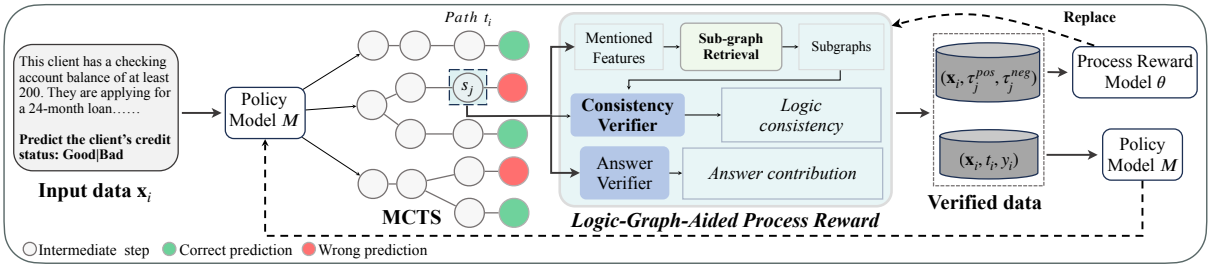

Figure 3: Overview of the proposed Logic-Graph-Enhanced Process Supervision: using the logic graph to verify LLM-generated reasoning paths step-by-step and generate training data for PRM and Policy Model training.

computes only outcome-based reward, we incorporate our proposed logic-graph-aided process reward into the final reward (also called Q-value) calculation. Let $r(s_j)^k$ denote the reward of step s_j after value back-propagation in the k-th rollout, the reward is computed as follows:

$$\begin{aligned} r(s_j)^k =& r(s_j)^{k-1} + [w \cdot r_{\text{LG}}(s_j)^k \\ &+ (1-w) \cdot r_{\text{outcome}}(s_a)^k], \end{aligned} \tag{6}$$

where the initial reward $r(s_j)^0 = 0$ in the first rollout, $r_{\text{outcome}}(s_j)^k$ denotes the outcome reward, which equals 1 if the terminal step s_a produces a correct outcome and 0 otherwise. w is the logic consistency weight. More MCTS details are provided in Appendix B.

Model Training While the process reward function evaluates logic consistency, it cannot provide fine-grained rankings among consistent steps (e.g., distinguishing best from average). To address this, we train a process preference model (Guan et al., 2025) as our PRM. We construct preference pairs $\mathcal{D}_{\text{PRM}}$ by selecting the top-2 reward steps as positives and bottom-2 as negatives. The PRM θ is trained via the Bradley-Terry model (Bradley and Terry, 1952) with a pairwise ranking loss:

$$\begin{aligned} \mathcal{L}_{\text{PRM}}(\theta) = -\frac{1}{2\times 2}\mathbb{E}_{(\mathbf{x},\tau_j^{pos},\tau_j^{neg})\sim\mathcal{D}_{\text{PRM}}} \\ [\log(\sigma(r_\theta(\mathbf{x},\tau_j^{pos}) - r_\theta(\mathbf{x},\tau_j^{neg})))], \end{aligned} \tag{7}$$

where $\tau_j^{pos} = s_1 \oplus ... \oplus s_{j-1} \oplus s_j^{pos}$, $\tau_j^{neg} = s_1 \oplus ... \oplus s_{j-1} \oplus s_j^{neg}$, and j is not final outcome step. $r_\theta(\mathbf{x}_i, \tau_j) \in [-1, 1]$ denotes the output of the PRM, where τ_j is the partial path from the first step to the j-th step. Formally, for each step s_j, the PRM predicts its reward by:

$$r(s_j) = \theta(\text{serialize}(\mathbf{f}, \mathbf{x}_i) \oplus s_1 \oplus ... \oplus s_j). \tag{8}$$

We select the top-2 reasoning paths with the highest average reward among those yielding correct predictions as SFT data $\mathcal{D}_{\text{SFT}}$. The policy model is optimized by:

$$\mathcal{L}_{\text{SFT}}(M) = -\mathbb{E}_{(\mathbf{x}_i,t_i,y_i)\sim\mathcal{D}_{\text{SFT}}}[\log M(t_i, y_i|\mathbf{x}_i)]. \tag{9}$$

LogGER employs a self-training pipeline to train the policy model and PRM iteratively, as outlined in Algorithm 1 in Appendix C.

4 Experiments

4.1 Experimental Setup

Baselines. We conduct a comparative evaluation of LogGER against a diverse variety of prominent methods in tabular prediction. Our baselines span four key categories: frontier general-purpose LLMs (e.g., OpenAI o1 (Jaech et al., 2024), DeepSeek-R1 (Guo et al., 2025)), domain-specific tabular LLMs (e.g., TableGPT2 (Su et al., 2024), TableLlama (Zhang et al., 2023)), process-oriented reasoning frameworks (including CoT SFT, GPRO (Yang et al., 2025b), and rStar (Guan et al., 2025)), and classical tree-based models (e.g., XGBoost, Random Forest). Detailed configurations for all baselines are provided in the Appendix E.3.

Datasets. We consider a variety of supervised tabular prediction tasks with heterogeneous features, including classification and regression. Specifically, the tabular datasets include: Income (IN), Heart (HE), Bank (BA), Credit (CR), Credit-g (CRG), Blood (BL) (Grinsztajn et al., 2022), California_housing (CA) (Pace and Barry, 1997), Diamond (DI) (Chen et al., 2023) and Fried (FR) (Breiman, 1996). Following previous studies (Gorishniy et al., 2021), we use Accuracy (higher is better) to evaluate classification tasks, Normalized Root Mean Squared Error (NRMSE) (lower is better) to evaluate the regression tasks. For more details, please refer to Appendix D.

LLM	Method	IN ↑	HE ↑	BA ↑	CR ↑	CRG ↑	BL ↑	Avg. Acc. ↑
Frontier LLMs	QwQ-32B	50.87	53.26	15.04	63.94	47.00	25.33	42.57
	DeepSeek-R1	78.60	66.30	63.71	68.43	42.00	25.67	57.45
	GPT-4o	60.33	72.82	67.02	47.80	59.00	40.00	57.83
	GPT-4.1	81.27	79.35	67.48	72.59	53.00	34.67	64.73
	OpenAI o1	81.50	79.35	71.40	71.91	66.00	46.67	69.47
Tabular LLMs	TableGPT2-7B	72.27	40.22	50.27	48.43	42.00	45.33	49.75
	TableLlama-7B	45.80	58.70	50.22	49.63	66.00	80.00	58.39
	Table-R1-Zero-8B	45.27	58.70	57.57	63.99	49.00	80.00	59.09
	Table-R1-Zero-7B	65.46	60.87	52.88	63.96	56.00	80.00	63.19
	TableLLM-7B	82.69	47.83	69.28	71.74	55.00	78.67	67.54
Llama-3.1-8B -Instruct	Vanilla	55.83	66.30	51.66	64.21	44.00	56.00	56.33
	Direct SFT	82.47	70.65	68.38	65.01	**68.00**	**80.00**	72.42
	CoT SFT	77.83	80.43	68.78	71.97	58.00	64.00	70.17
	GRPO	76.97	77.17	69.14	49.80	67.00	**80.00**	70.01
	rStar	84.03	77.17	**72.79**	73.02	61.00	**80.00**	74.67
	LogGER (*Ours*)	**84.57**	**81.52**	72.21	**74.42**	67.00	**80.00**	**76.62**
Qwen-2.5-7B -Instruct	Vanilla	80.92	78.26	49.98	62.51	51.00	44.00	61.11
	Direct SFT	77.46	58.70	58.83	53.87	68.00	**80.00**	66.14
	CoT SFT	76.17	80.43	62.79	68.35	54.00	78.67	70.07
	GRPO	82.57	58.70	71.04	74.50	68.00	**80.00**	72.46
	rStar	82.73	76.09	68.69	**75.56**	64.00	78.67	74.29
	LogGER (*Ours*)	**83.00**	**85.87**	**71.62**	72.19	**72.00**	78.67	**77.22**

Table 1: Performance comparison on classification tasks across frontier LLMs, specialized tabular LLMs, and various training methods. LogGER consistently outperforms baselines using the same base models (Llama-3.1-8B-Instruct and Qwen-2.5-7B-Instruct). Best and second-best results are shown in bold and underlined, respectively.

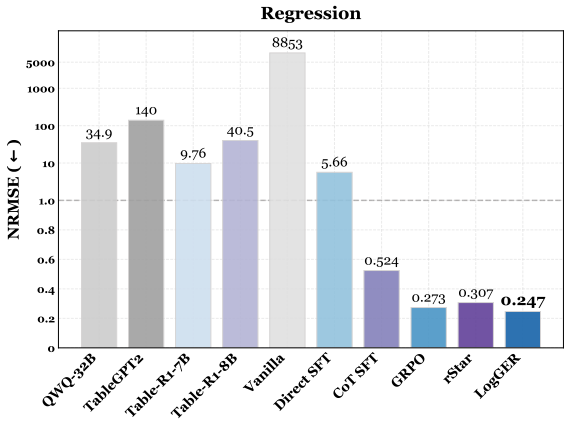

Figure 4: Performance comparison on the regression task. LogGER achieves the lowest NRMSE (0.247), outperforming other baselines.

Implementation Details. We use both Llama-3.1-8B-Instruct (Meta, 2024) and Qwen-2.5-7B-Instruct (Yang et al., 2024a) as the base policy model and PRM. For policy model training, we use a batch size of 128 and a learning rate of 5e-5 for 5 epochs. For PRM training, we set the batch size to 512, train for 1 epoch, and use a learning rate of 7e-6. More details are provided in Appendix E.

4.2 Main results

Comparable Performance with Frontier LLMs and Tabular LLMs. As shown in Table 1, results demonstrate that LogGER achieves comparable performance across all tasks. While models like DeepSeek-R1 and OpenAI o1 have shown their strong performance in mathematical reasoning (Jaech et al., 2024; Guo et al., 2025), they achieve only 57.45% and 69.47% accuracy on tabular data. And even TableLLM, trained on tabular data, struggles with similar limitations due to the lack of high-quality reasoning paths. In contrast, LogGER significantly enhances LLMs' reasoning capabilities by selecting high-quality training data using the logic graph. Despite being based on a relatively smaller 7B model, LogGER achieves a remarkable accuracy of 77.22% and NRMSE of 0.247, demonstrating superior precision in continuous value prediction.

Same Backbones, Superior Performance. Table 1 provides a comprehensive quantitative comparison of LogGER against baseline methods. In most tasks, LogGER significantly improves the reasoning performance of LLMs compared to most

Method		Classification							Regression			
		IN ↑	HE ↑	BA ↑	CR ↑	CRG ↑	BL ↑	Avg. ↑	CA ↓	DI ↓	FR ↓	Avg. ↓
Classical ML	KNN	79.78	76.08	68.79	64.54	62.00	**81.33**	72.09	**0.34**	0.37	0.17	0.29
	Decision Tree	80.16	77.60	69.21	69.11	66.00	78.66	73.46	0.38	0.40	0.18	0.32
	Random Forest	80.17	76.52	69.54	**76.68**	64.80	78.66	74.40	0.38	0.36	0.13	0.29
	XGBoost	76.99	78.26	69.29	73.84	68.00	80.00	74.40	0.43	0.85	0.24	0.51
Neural Architecture	MLP	80.15	79.78	68.12	74.83	68.00	80.00	75.15	0.64	1.42	0.38	0.82
	VIME	80.07	80.21	68.27	75.04	68.00	80.00	75.27	0.48	1.20	0.31	0.67
	DeepFM	80.26	82.60	65.55	75.47	68.00	80.00	75.31	0.43	1.43	0.91	0.92
	TabNet	80.29	77.17	69.15	73.24	68.00	**81.33**	74.86	0.50	0.88	5.88	2.42
	STG	77.30	59.78	63.58	61.39	64.20	80.00	67.71	0.42	0.87	0.17	0.49
	SAINT	79.93	81.08	67.70	75.42	68.00	80.00	75.36	0.35	1.42	0.37	0.71
	LogGER-7B	**83.00**	**85.87**	**71.62**	72.19	**72.00**	78.67	**77.22**	0.35	**0.29**	**0.10**	**0.25**

Table 2: Performance comparison of LogGER-7B with traditional ML and deep learning baselines across classification and regression benchmarks. For classification, higher accuracy (↑) is better; for regression, lower NRMSE (↓) is better. Best and second-best results are highlighted in bold and underlined.

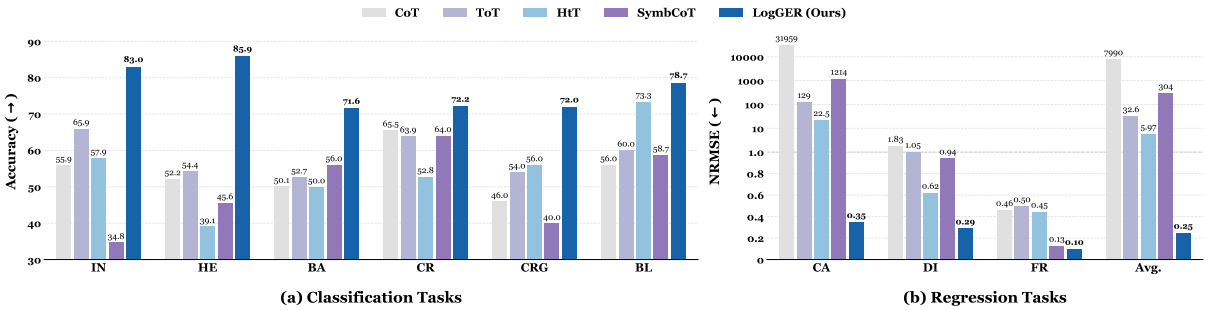

Figure 5: Comparison with Top-Tier CoT Methods. LogGER significantly outperforms existing methods.

baseline methods. Overall, LogGER improves the best baseline accuracy by an average improvement from 74.29% to 77.22% on Qwen-2.5-7B-Instruct and from 74.67% to 76.62% on Llama-3.1-8B-Instruct. Notably, in the healthcare domain, specifically on the Heart task, LogGER increases the accuracy by 5.44%. Similarly, in the financial domain, on the Credit-g task, the accuracy improves by 4.00%. This superiority is even more pronounced in regression tasks. As illustrated in Figure 4, LogGER achieves the lowest NRMSE of 0.247, drastically outperforming advanced optimization baselines such as rStar (0.307) and GRPO (0.273), proving its robustness in handling complex numerical reasoning under identical model architectures.

Superior Performance than Tree-based Models. The results in Table 2 demonstrate a clear performance improvement of LogGER over traditional tree-based models. While classic tree-based models such as Random Forest and XGBoost achieve average accuracies of around 74.40%, LogGER attain significantly higher accuracies, with LogGER-8B and LogGER-7B reaching 76.62% and 77.22%, respectively. These results indicate that LogGER successfully integrates the strengths of both tree-based methods and LLMs, resulting in performance that not only surpasses each component but also establishes a new SOTA on tabular data.

4.3 Comparison with Top-tier CoT Methods

To further validate LogGER on tabular reasoning, we compare it with top-tier CoT methods, including CoT (Wei et al., 2022), ToT (Yao et al., 2023), HtT (Zhu et al., 2023), Symbolic CoT (Symb-CoT) (Xu et al., 2024). As shown in Figure 5, these methods struggle with tabular prediction, with only ToT achieving the 58.46% average accuracy. This suggests that relying solely on the LLM's unstructured reasoning capabilities is insufficient for structured reasoning. In contrast, LogGER integrates external structural signals with the reasoning power of LLMs, effectively enabling structured reasoning and achieving superior performance.

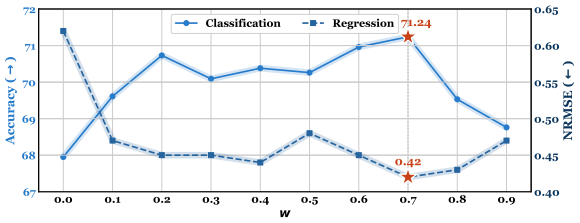

Figure 6: Average performance of LogGER under different logic consistency weights w on regression and classification tasks using Qwen-2.5-7B-Instruct. Detailed per-task results are provided in the Appendix F.2.

5 Analysis

5.1 Sensitivity Analysis of Weight w

An interesting question is how the proportion of *logic consistency* in the Q-value calculation affects the performance of LogGER. To explore this, we varied the *logic consistency* weight w and observed its impact. As shown in Figure 6, the results indicate that increasing w significantly enhances model performance, reaching a peak when $w = 0.7$. This suggests that the logic correctness of reasoning steps plays a crucial role in improving the real-world reasoning capabilities of LLMs. Accordingly, we set $w = 0.7$ as the default value.

5.2 Impact of Iterations

Figure 7 compares LogGER against rStar across four iterative rounds. For classification tasks, LogGER consistently outperforms rStar, improving from 71.2% to 77.2% accuracy by Round 4. In regression tasks, LogGER achieves superior error reduction with a final NRMSE of 0.247 versus rStar's 0.307. These results demonstrate that the logic graph effectively guides the model toward better solutions at every training stage.

5.3 Quality of Logic Graphs

As the foundation of LogGER, the quality of the logic graph is crucial for overall performance. To evaluate this, we conduct a human evaluation where three domain experts assess the logic graphs by verifying whether each sub-graph aligns with their reasoning processes. Sub-graphs that match expert judgment are marked as correct. As shown in Table 3, the logic graphs achieve an average accuracy of 93.80%, demonstrating that LLMs can effectively generate accurate and interpretable logic graphs for tabular tasks by leveraging statistical priors and our designed graph structure.

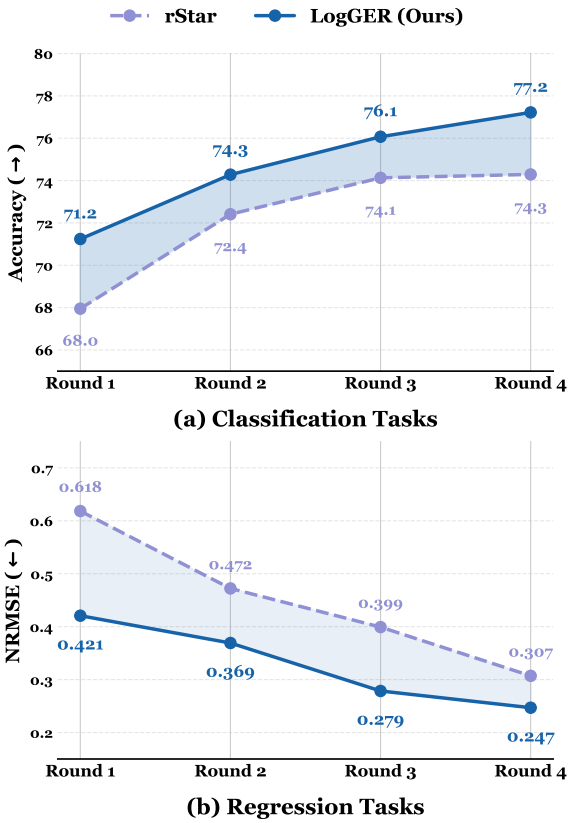

Figure 7: Average performance comparison of LogGER and rStar across different MCTS iteration numbers on regression and classification tasks. Detailed per-task results are provided in the Appendix F.3.

IN	HE	BA	CR	CRG	BL	CA	DI	FR
93.91% (108/115)	96.55% (28/29)	100.00% (19/19)	91.67% (22/24)	94.37% (67/71)	91.67% (11/12)	93.75% (30/32)	92.30% (36/39)	90.00% (18/20)

Table 3: Correctness of the constructed logic graph. The notation (/) represents (the number of sub-graphs aligning with expert thinking / total number of logic sub-graphs).

5.4 Interpretability of Reasoning Process

We employ the widely-used LLM-as-a-judge approach (Zheng et al., 2023; Feng et al., 2024; Li et al., 2025; Wang et al., 2025) to evaluate interpretability, where GPT-4o scores each reasoning path from 1 to 5 based on three criteria: (1) no factual or logical errors, (2) consistent reasoning steps that align with the outcome without contradiction, and (3) a correct final outcome. As shown in Table 4, LogGER achieves the highest interpretability scores across all tasks, demonstrating that the logic graph effectively guides the generation of coherent and reliable reasoning paths, enhancing both transparency and trustworthiness.

Method	IN	HE	BA	CR	CRG	BL	CA	DI	FR
rStar	3.25	3.58	3.64	3.56	2.72	4.08	3.89	4.39	5.00
LogGER	4.03	4.39	4.01	4.45	4.12	4.22	4.43	5.00	5.00

Table 4: Interpretability scores of reasoning paths. Higher scores indicate better interpretability.

6 Conclusion

We propose LogGER, a novel framework that integrates logic graphs with LLMs to enhance tabular data prediction. By reformulating decision trees as human-readable logic graphs and supervising LLM reasoning with these graphs, LogGER combines the predictive accuracy of tree models with the interpretability of LLMs. Experiments show that LogGER consistently outperforms both traditional tree-based models and SOTA LLM-based methods, offering superior accuracy and transparency for tabular prediction.

7 Limitations

One limitation of our work is that, due to computational constraints, we have not evaluated our approach on larger language models (e.g., models with 32B parameters or more). However, the logic graphs constructed by our method consistently show high quality and may serve as effective supervision signals for process supervision, even for moderately sized or larger models. In addition, the multi-stage MCTS and model training process in our framework introduces extra computational overhead, although it also yields clear performance gains. We leave more comprehensive validation on larger models and further efficiency improvements to future work.

Acknowledgement

This paper was supported by the National Regional Innovationand Development Joint Fund (No. U24A20254). This work was supported by MYbank, Ant Group.

References

Anthropic. 2024. Claude 3.5 sonnet.

Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. 2020. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8.

Dimitris Bertsimas, Kimberly Villalobos Carballo, Yu Ma, Liangyuan Na, Léonard Boussioux, Cynthia Zeng, Luis R Soenksen, and Ignacio Fuentes. 2022. Tabtext: a systematic approach to aggregate knowledge across tabular data structures. *arXiv preprint arXiv:2206.10381*.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2022. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 35(6):7499–7519.

Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24(2):123–140.

Leo Breiman. 2001. Random forests. *Machine learning*, 45(1):5–32.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Lucas Carrasco, Felipe Urrutia, and AndrÃŠs Abeliuk. 2025. Zero-shot decision tree construction via large language models. *arXiv preprint arXiv:2501.16247*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*.

Kuan-Yu Chen, Ping-Han Chiang, Hsin-Rung Chou, Ting-Wei Chen, and Darby Tien-Hao Chang. 2023. Trompt: towards a better deep neural network for tabular data. In *Proceedings of the 40th International Conference on Machine Learning*, pages 4392–4434.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Tuan Dinh, Yuchen Zeng, Ruisu Zhang, Ziqian Lin, Michael Gira, Shashank Rajput, Jy-yong Sohn, Dimitris Papailiopoulos, and Kangwook Lee. 2022. Lift: Language-interfaced fine-tuning for non-language machine learning tasks. *Advances in Neural Information Processing Systems*, 35:11763–11784.

Tao Feng, Yicheng Li, Li Chenglin, Hao Chen, Fei Yu, and Yin Zhang. 2024. Teaching small language models reasoning through counterfactual distillation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5831–5842.

Josh Gardner, Juan C Perdomo, and Ludwig Schmidt. 2024. Large scale transfer learning for tabular data via language modeling. *Advances in Neural Information Processing Systems*, 37:45155–45205.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34:18932–18943.

Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, pages 507–520. Curran Associates, Inc.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. 2023. Tabllm: Few-shot classification of tabular data with large language models. In *International conference on artificial intelligence and statistics*, pages 5549–5581. PMLR.

Mikel Hernandez, Gorka Epelde, Ane Alberdi, Rodrigo Cilla, and Debbie Rankin. 2022. Synthetic data generation for tabular health records: A systematic review. *Neurocomputing*, 493:28–45.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Rihui Jin, Zheyu Xin, Xing Xie, Zuoyi Li, Guilin Qi, Yongrui Chen, Xinbang Dai, Tongtong Wu, and Gholamreza Haffari. 2025. Table-r1: Self-supervised and reinforcement learning for program-based table reasoning in small language models. *arXiv preprint arXiv:2506.06137*.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 2757–2791. Association for Computational Linguistics.

Meta. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

OpenAI. 2024. Hello gpt-4o.

R Kelley Pace and Ronald Barry. 1997. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, and 1 others. 2024. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*.

Qwen Team. 2024. Qwq: Reflect deeply on the boundaries of the unknown.

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.

Victor Wang, Michael JQ Zhang, and Eunsol Choi. 2025. Improving llm-as-a-judge inference with the judgment distribution. *arXiv preprint arXiv:2503.03064*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Mangasarian-Olvi Street Nick Wolberg, William and W. Street. 1993. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024. Faithful logical reasoning via symbolic chain-of-thought. *arXiv preprint arXiv:2405.18357*.

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*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024b. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Zheyuan Yang, Lyuhao Chen, Arman Cohan, and Yilun Zhao. 2025b. Table-r1: Inference-time scaling for table reasoning. *arXiv preprint arXiv:2505.23621*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*.

Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, and 1 others. 2023. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2025. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023. Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206*.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, and 1 others. 2024. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. 2023. Large language models can learn rules. *arXiv preprint arXiv:2310.07064*.

A Statistical Priors Extraction from Training Data

In this section, we introduce estimation methods for statistical priors in addition to conditional probability.

For each categorical feature x_{ic}, we count the number of samples including x_{ic}, denoted as $\mathcal{C}(x_{ic})$, as well as the total number of samples, $\mathcal{C}$. The distribution proportion of each categorical feature $P(x_{ic})$ is then calculated as:

$$P(x_{ic}) = \frac{\mathcal{C}(x_{ic})}{\mathcal{C}}.$$

For numerical feature $x_{i\mathrm{n}}$ and its total number of samples $\mathcal{N}$, we calculate the following key statistical information:

(1) Mean:

$$\mu_{x_{i\mathrm{n}}} = \frac{1}{\mathcal{N}} \sum_{\mathrm{n}=1}^{\mathcal{N}} x_{i\mathrm{n}}.$$

(2) Minimum and Maximum:

$$\min(x_{i\mathrm{n}}) = \min\{x_{i1}, x_{i2}, ..., x_{i\mathrm{n}}, ..., x_{i\mathcal{N}}\},$$

$$\max(x_{i\mathrm{n}}) = \max\{x_{i1}, x_{i2}, ..., x_{i\mathrm{n}}, ..., x_{i\mathcal{N}}\}.$$

(3) Median:

$$\text{Median}(x_{i\mathrm{n}}) = \begin{cases} x_{i\left(\frac{\mathcal{N}+1}{2}\right)} & \text{if } \mathcal{N} \text{ is odd} \\ \frac{x_{\left(\frac{\mathcal{N}}{2}\right)} + x_{\left(\frac{\mathcal{N}}{2}+1\right)}}{2} & \text{if } \mathcal{N} \text{ is even.} \end{cases}$$

(4) Standard Deviation:

$$\sigma_{x_{i\mathrm{n}}} = \sqrt{\frac{1}{\mathcal{N}} \sum_{\mathrm{n}=1}^{\mathcal{N}} (x_{i\mathrm{n}} - \mu_{x_{i\mathrm{n}}})^2}.$$

Collecting this information helps LLM gain a comprehensive understanding of the overall data distribution, enabling it to identify more representative features and more effectively partition each feature into distinct intervals (Carrasco et al., 2025), thereby generating more precise conditional nodes in the logic graph.

Algorithm 1: Logic-Graph-Enhanced Process Supervision

Input: Process reward function r_{LG}, policy model M, PRM θ, input features $\mathbf{x}$, feature names $\mathbf{f}$, number of iterations T

1: $\mathcal{T}_0 \leftarrow M(\text{serialize}(\mathbf{f}, \mathbf{x}))$ // Perform MCTS using M to generate extensive reasoning paths

2: $g(\mathbf{x}) \leftarrow \text{Retrive}(\mathcal{T})$ // Retrieve the logic sub-graph

3: $r_0 \leftarrow r(\mathcal{T}, g(\mathbf{x}))$ // Calculate initial reward using reward function

4: **for** $t = 1$ to T **do**

5: $\quad \mathcal{D}_{\text{SFT}_t} \leftarrow \text{Generate}_{\text{SFT}}(\mathcal{T}, r_t)$ // Generate training data for the policy model

6: $\quad \mathcal{D}_{\text{PRM}_t} \leftarrow \text{Generate}_{\text{PRM}}(\mathcal{T}, r_t)$ // Generate training data for the PRM

7: $\quad M_t \leftarrow \text{Train}(M, \mathcal{D}_{\text{SFT}})$ // Train the policy model

8: $\quad \theta_t \leftarrow \text{Train}(\theta, \mathcal{D}_{\text{PRM}})$ // Train the PRM

9: $\quad \mathcal{T}_t \leftarrow M(\text{serialize}(\mathbf{f}, \mathbf{x}))$ // Perform MCTS using M_t

10: $\quad r_t \leftarrow \theta(\text{serialize}(\mathbf{f}, \mathbf{x}), \mathcal{T}_t)$ // Calculate updated reward

11: **end for**

Output: M_T

B Details of MCTS

As described in Section 3.3, at the beginning of each round, we employ MCTS to generate verified step-by-step trajectories for all training data. Specifically, this MCTS process includes four stages: *node selection*, *thought expansion*, *greedy MC rollout*, and *value backpropagation*. For input features $\mathbf{x}_i$, each path $t_i \in \mathcal{T}$ consists of a sequence of steps $t_i = \text{serialize}(\mathbf{f}, \mathbf{x}_i) \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_a$. Here, the step corresponds to the node in the Monte Carlo tree. At each step s_j, we prompt the policy model to generate multiple candidate next steps based on the partial path $x \oplus s_1 \oplus \cdots \oplus s_{j-1}$. All steps are then assigned rewards (Q-values), $r(s)$, and the optimal step is selected using the following Upper Confidence Bound for Trees (UCT) (Kocsis and Szepesvári, 2006) criterion:

$$\text{UCT}(s) = \frac{r(s)}{U(s)} + c\sqrt{\frac{\ln U(\text{parent}(s))}{U(s)}}, \quad (10)$$

where $r(s)$ is the predicted reward of step s, $U(s)$ is the number of visits to step s, $U(\text{parent}(s))$ is the number of visits to the parent step, and c is a constant balancing exploration and exploitation. The other stages are described in Section 3.3. In Implementation, the maximum tree depth is set to 16 (25 for the Credit-g task due to its richer input features), with 8 MCTS rollouts conducted per sample by default. At each step, we explore 8 candidate steps and set the constant $c = 2$ in Eq. (10) to encourage deeper exploration. In the first round, Q-values in MCTS are derived by combining logic consistency $q_1(s)$ and outcome contribution $q_2(s)$. In subsequent rounds, Q-values are produced by the PRM. Each round of MCTS is performed using the latest trained policy model and PRM.

C Logic-Graph-Enhanced Process Supervision Algorithm

Algorithm 1 outlines the training procedure for Logic-Graph-Enhanced Process Supervision. The process consists of an initialization phase followed by iterative refinement. Initially, we employ the policy model M to perform Monte Carlo Tree Search (MCTS), generating reasoning paths $\mathcal{T}_0$, and retrieve the logic sub-graph $g(\mathbf{x})$ to compute the initial rewards r_0. During the iterative loop (Lines 4-11), we construct the SFT dataset $\mathcal{D}_{\text{SFT}_t}$ and PRM dataset $\mathcal{D}_{\text{PRM}_t}$ based on the trajectories and rewards from the previous iteration. Both the policy model M and the Process Reward Model (PRM) θ are then fine-tuned. Finally, new reasoning paths $\mathcal{T}_t$ are generated using the updated policy M_t and re-evaluated by the updated PRM θ_t to guide the subsequent iteration.

D Tasks and Datasets

Details of each test dataset we used as benchmark are as follows:

Credit This task involves assessing the credit risk of borrowers and determining whether they are likely to face financial distress within the next two years based on their financial information. The dataset consists of 16,714 training samples and 3,510 test samples, with each sample comprising 10 financial attributes of the borrower and a label indicating whether they will experience financial distress.

Bank This is a bank-marketing task. Given information from direct marketing campaigns conducted by Portuguese banks via phone calls, we predict whether a client will subscribe to a term deposit. The dataset includes 7,404 samples.

Creditg Given the borrower's relevant information, the Creditg task evaluates whether their credit status is good. The dataset includes 799 training samples and 100 test samples.

Heart This task predicts heart disease risk. The Heart dataset includes 734 training records and 92 testing records with 11 diagnostic features.

Income This task predicts whether a person's annual income exceeds $50,000 based on their socioeconomic information. The dataset includes 48,840 samples with 14 demographic and socioeconomic attributes.

Blood This task predicts whether a donor will donate blood in March 2007 based on their past donation history. The dataset includes 747 samples with features like donation recency and frequency.

California_housing This task predicts the median house price for the California house block group in hundreds of thousands of US dollars based on the house features. The dataset includes 20640 instances and 9 house price features.

Diamond This task predicts the price in US dollars ($326 – $18,823) of the diamond based on the diamond features. The dataset includes 53940 diamond instances and 10 diamond features.

Fried This is an artificial data set. The cases are generated using the following method: Generate the values of 10 attributes, $X_1, \ldots, X_{10}$, independently, each of which is uniformly distributed over $[0, 1]$. Obtain the value of the target variable Y using the equation:

$$Y = 10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, 1)$. The dataset includes 17396 instances and 11 numerical features.

To reduce computational costs, for classification tasks, we sample 6,000 instances for training from datasets with over 6,000 samples. For regression tasks, we sample 2000 instances for training and 1000 instances for testing from datasets with over 3000 samples. For datasets without predefined test sets, we use an 8:1:1 train-validation-test split.

E Implementation Details

E.1 Training Details

In each round, we collect verified step-by-step trajectories to fine-tune the policy LLM and train the PRM. Following Chen et al. (2024); Guan et al. (2025), the policy model is fine-tuned from the base model in each round, rather than incrementally from the previous round. Both the policy model and PRM are trained solely on the newly collected data from the newest round to avoid learning errors from low-quality reasoning paths.

E.2 Evaluation

All evaluations are conducted using 8 NVIDIA A100 GPUs with greedy decoding. For all datasets, model performance is evaluated based on accuracy, measured by comparing the generated outputs to the targets.

E.3 Baseline Details

We compare LogGER against several strong baselines, each representing different training paradigms: (1) **Direct SFT**: The LLM is trained directly using the original dataset labels. We use 1 epoch, a learning rate of 2e-5, and a batch size of 128. (2) **CoT SFT**: The policy model generates CoT reasoning paths. Due to the base model's weaker generation ability, we set the temperature to 0.99 and sample each sample 32 times, selecting the best CoT for training. (3) **GRPO**: For GRPO baselines, we follow (Yang et al., 2025b) and design task-specific, verifiable reward functions tailored to tabular reasoning and apply the Group Relative Policy Optimization (GRPO) algorithm (Guo et al., 2025) to enable stable and scalable reinforcement learning. (4) **rStar**: In the original paper of rStar (Guan et al., 2025), step verification is performed via code execution. However, this is not applicable to our real-world tasks, where reasoning steps are non-executable. Therefore, in our comparison, we remove code execution from rStar. Further, to ensure fairness, the first-round data in rStar is also generated using the same policy model as in our method, with the same number of rollouts across all rounds. All other settings strictly follow the original paper of rStar.

E.4 MCTS Costs

We use the policy model on a single 8×80GB A100 GPU node and perform 8 MCTS rollouts in parallel. The time required to complete one round of MCTS varies depending on the dataset size. For datasets like Credit, Bank, and Income (each with

Weight w	Classification							Regression			
	IN ↑	HE ↑	BA ↑	CR ↑	CRG ↑	BL ↑	Avg. ↑	CA ↓	DI ↓	FR ↓	Avg. ↓
0.0	75.21	64.64	48.00	78.26	80.23	61.33	67.95	0.71	0.83	0.31	0.62
0.1	66.13	65.77	54.00	78.26	81.53	72.00	69.61	0.48	0.61	0.31	0.47
0.2	73.28	67.93	47.00	81.52	81.30	73.33	70.73	0.49	0.53	0.33	0.45
0.3	72.48	67.16	52.00	79.35	81.53	68.00	70.09	0.45	0.59	0.31	0.45
0.4	69.54	67.97	53.00	78.26	81.50	72.00	70.38	0.48	0.55	0.30	0.44
0.5	74.53	67.39	50.00	80.43	82.57	66.67	70.26	0.51	0.61	0.30	0.48
0.6	70.51	67.21	56.00	78.26	81.77	72.00	70.96	0.50	0.55	0.29	0.45
0.7	70.40	68.15	53.00	78.26	81.63	76.00	**71.24**	0.44	0.53	0.30	**0.42**
0.8	69.37	66.62	48.00	79.35	81.87	72.00	69.53	0.45	0.54	0.30	0.43
0.9	66.52	65.95	50.00	79.35	81.40	69.33	68.76	0.47	0.67	0.27	0.47

Table 5: First-round performance of LogGER on regression and classification tasks under different logic consistency weights w, using Qwen-2.5-7B-Instruct. We report results from the first iteration because the weight is primarily applied in this stage.

Method	Classification							Regression			
	IN ↑	HE ↑	BA ↑	CR ↑	CRG ↑	BL ↑	Avg. ↑	CA ↓	DI ↓	FR ↓	Avg. ↓
rStar Round 1	80.23	78.26	64.64	75.21	48.00	61.33	67.95	0.71	0.83	0.31	0.62
rStar Round 2	81.63	75.00	66.26	73.93	59.00	78.67	72.41	0.46	0.66	0.29	0.47
rStar Round 3	82.30	75.00	70.27	72.22	59.00	80.00	73.13	0.39	0.52	0.29	0.40
rStar Round 4	82.73	76.09	68.69	75.56	64.00	78.67	74.29	0.37	0.45	0.10	0.31
LogGER Round 1	81.63	78.26	68.15	70.40	53.00	76.00	71.24	0.44	0.53	0.30	0.42
LogGER Round 2	82.57	82.61	69.95	71.88	64.00	74.67	74.28	0.40	0.42	0.29	0.37
LogGER Round 3	82.70	82.61	71.08	72.36	69.00	78.67	76.07	0.36	0.36	0.12	0.28
LogGER Round 4	83.00	85.87	71.62	72.19	72.00	78.67	**77.22**	0.35	0.29	0.10	**0.25**

Table 6: Detailed performance comparison of LogGER across different iteration rounds on regression and classification tasks using Qwen-2.5-7B-Instruct.

6,000 training samples), data generation takes approximately 12–18 hours per dataset. For other smaller datasets, generation typically completes within 2.5–4.5 hours using all 8 GPUs. The detailed computational costs are summarized in Table 7.

LLM	CR	BA	CRG	HE	IN	BL
Llama3.1-8B-Instruct	18	18	18	4.5	4.5	3
Qwen-2.5-7B-Instruct	12	12	12	3	3	2.5

Table 7: Computational cost of LogGER during MCTS-based data generation across different datasets.

E.5 Training the Logic Consistency Verifier

One of the most direct methods for evaluating the consistency between generated steps and the logic graph is to use advanced models (such as GPT-4o). However, MCTS rollouts generate extensive reasoning paths, making it costly to rely entirely on these models for verification. To reduce this overhead, we first sample a small amount of data (30-50 samples per dataset) and then use the data generated by GPT-4o to train a smaller LLM as a consistency verifier. For this purpose, we selected Qwen-2.5-7B-Instruct as the verifier model. After training, this verifier achieved an average accuracy of over 90%, as reported in Table 8.

	CR	BA	CRG	HE	IC	BL	Avg.
Training samples	15912	9928	31072	9314	14790	2470	/
Accuracy	92.50%	90.00%	93.00%	94.50%	87.50%	84.00%	90.25%

Table 8: Performance of logic consistency verifier.

F Additional Experimental Results

F.1 Comparison with LLM Baselines on Regression

The detailed performance comparison between our proposed LogGER and tree-based models across various datasets is presented in Table 9. As shown

in the Table 9, our method LogGER, by effectively integrating data statistical priors and logic graph, enables the large language model not only to capture the statistical priors typically learned by tree-based models, but also to achieve comparable or even superior performance on most datasets.

Method	CA ↓	DI ↓	FR ↓	Avg. ↓
QwQ-32B	104.13	0.50	0.08	34.90
TableGPT2-7B	416.00	3.22	0.43	139.88
Table-R1-Zero-8B	120.50	0.92	0.23	40.55
Table-R1-Zero-7B	27.94	0.90	0.44	9.76
Vanilla	26554.00	4.55	0.47	8853.00
Direct SFT	16.68	0.20	0.09	5.66
CoT SFT	0.46	0.92	0.19	0.52
GRPO	0.41	0.34	0.07	0.27
rStar	0.37	0.45	0.10	0.31
LogGER (*Ours*)	0.35	0.29	0.10	0.25

Table 9: Detailed test performance comparison of LogGER and LLM-based baseline methods on regression tasks.

F.2 Sensitivity Analysis of Weight w

Since the logic consistency weight w is only used in the first round of MCTS to calculate the Logic-Graph-Aided Process Reward function, the rewards in the subsequent rounds are provided by the process reward model. Therefore, we compared the impact of different logic consistency weights in the first round on the proposed LogGER. The average results are shown in Figure 6, while the detailed results are reported in Table 5. As the logical consistency weight increases, the performance of LogGER also improves—from 67.95% (weight 0.0) to 71.24% (weight 0.7), an improvement of 3.29%. However, the weight should not be set too high; the optimal weight is found to be 0.7. Otherwise, performance will decrease. This is because an excessively large w causes the LLM to overemphasize the logic consistency of intermediate steps, while neglecting the correctness of the final outcome.

F.3 Impact of Iterations

We provide a detailed report in Table 6 on the impact of the number of iterations on the performance of LogGER across different datasets. Notably, as shown in Table 6, during the first round, the guidance of the logic-graph-aided process reward function leads to a significant improvement on most datasets. Moreover, with the increase in the number of iterations, LogGER is able to maintain its advantage over other method, indicating that the proposed LogGER is robust and stable across multiple rounds of optimization. These results highlight the importance of both logic guidance in early stages and continued refinement in subsequent rounds.

F.4 Impact of MCTS Rollouts

As shown in Table 10, the results demonstrate diminishing returns as MCTS rollouts increase. Performance improves most significantly from 1 to 2 rollouts (+0.94% in BA), with gradual gains continuing up to 8 rollouts (77.31% average accuracy). Beyond 8 rollouts, improvements become negligible (+0.02% at 16 rollouts), indicating performance saturation. This suggests 8 rollouts provides the optimal balance between computational cost and accuracy gains, as further increases yield diminishing marginal benefits.

MCTS Rollouts	BA ↑	IN ↑	Avg. Acc. ↑
1	70.86	82.80	76.83
2	71.80	82.20	77.00
4	71.62	82.40	77.01
8	71.62	83.00	77.31
16	71.62	83.04	77.33

Table 10: Model performance with varying MCTS rollout numbers. BA and IN are two respective benchmarks, and Avg. Acc. represents the average accuracy across both datasets.

F.5 Impact of Logic Graph Generators

To investigate how different LLMs affect logic graph construction quality, we evaluate LogGER using four advanced LLMs. The results in Table 11 reveal that logic graph quality substantially impacts overall performance. All LogGER variants consistently outperform the rStar baseline (72.44%), with improvements ranging from 1.62% to 2.45%, demonstrating the effectiveness of incorporating structured logical reasoning. Among the tested LLMs, GPT-4o achieves the best performance at 74.89% average accuracy. Notably, the performance gap between different LLMs remains relatively narrow (0.83%), suggesting that LogGER can effectively leverage structured reasoning capabilities from various foundation models.

F.6 Robustness on Noisy Datasets

As shown in Table 1, LogGER performs well on the noisy dataset IN, which contains explicit missing values and mixed-type noise. To further verify its generalizability to noisy data, we additionally conduct experiments on BCS (Breast

Model	LLM	BA ↑	IN ↑	Avg. Acc. ↑
rStar	/	64.64	80.23	72.44
LogGER	Claude Sonnet 4.5	67.52	80.60	74.06
	Gemini-2.5-pro	67.52	81.30	74.41
	DeepSeek-R1	68.02	81.03	74.53
	GPT-4o	**68.15**	**81.63**	**74.89**

Table 11: Performance comparison of LogGER using different LLMs for logic graph construction.

Cancer Wisconsin) (Wolberg and Street, 1993), a dataset commonly used for noise-robustness analysis, where we manually inject 30% feature missingness or noisy features. As shown in Table 12, LogGER achieves the best performance on both noisy datasets, indicating strong robustness to noisy data.

Method	IN ↑	BCS ↑
XGBoost	76.99	62.74
DeepFM	80.26	71.71
QWQ-32B	50.87	43.86
TableGPT2	72.27	43.86
Table-R1-Zero-7B	65.46	39.47
Vanilla	80.92	37.72
SFT	77.46	47.37
CoT SFT	76.17	42.98
GRPO	82.57	60.50
rStar	82.73	66.67
LogGER	**83.00**	**77.19**

Table 12: Performance comparison on IN and BCS.

F.7 Ablation Study on Logic Graph Node Types

To further explore the contribution of different node types in the logic graph, we conduct an ablation study by removing each node type in logic graph. As shown in Table 13, removing either Insight Nodes or Prediction Nodes leads to a clear decline in performance compared with the full LogGER model. These results indicate that both node types make important and complementary contributions to the reasoning process. Overall, the complete logic graph achieves the best performance, suggesting that a full graph structure is critical for fully unleashing the model's capability.

F.8 Scalability Analysis on Subgraph Retrieval

We report the number of nodes in the logic graph and the average reasoning path length for each task

Method	BA ↑	CA ↓
rStar	64.64	0.71
LogGER w/o Insight Nodes	66.85	0.52
LogGER w/o Prediction Nodes	67.61	0.50
LogGER	**68.15**	**0.44**

Table 13: Ablation study on different nodes of Logic Graph.

Graph Size	Reasoning Length	Vanilla	LogGER	Gain(Δ)
36 nodes, 24 edges (BL)	553.9	44.00	78.67	+34.67
72 nodes, 48 edges (CR)	1155.9	62.51	72.19	+9.68
87 nodes, 58 edges (HE)	1008.1	78.26	85.87	+7.61
213 nodes, 142 edges (CRG)	1346.4	51.00	72.00	+21.00
345 nodes, 230 edges (IN)	796.7	80.92	83.00	+2.08

Table 14: Scalability analysis on subgraph retrieval.

in Table 14. Empirically, we do not observe performance degradation as the graph grows or as reasoning paths become longer. Theoretically, our approach remains computationally scalable because retrieval does not require a global search over the entire graph. Specifically, traditional global graph search has a complexity of $O(N)$, where N is the total number of nodes in the graph. In contrast, we initiate retrieval from the activated Condition nodes (matched to the current input features) and perform a directed expansion along edges toward Insight and Prediction nodes. This directed retrieval strategy reduces the complexity to $O(k \cdot d \cdot l)$, where k is the number of activated Condition nodes (typically ≤ 5), d is the average out-degree (typically ≤ 4), and l is the average path length (typically ≤ 3). As a result, the effective per-query search space stays bounded, i.e., $k \cdot d \cdot l \ll N$, keeping the retrieval cost largely stable even when the overall logic graph contains hundreds or thousands of nodes. This complexity reduction ensures strong scalability of our method.

F.9 Robustness to the Performance of Verifier

To analyze the robustness of our method to verifier performance, we conduct experiments with five verifiers of different performance (49.96%, 65.70%, 75.00%, 80.31%, 90.25%). As shown in Table 15, our performance consistently improves as the verifier becomes stronger. Notably, the verifier does not need to be highly accurate (only above 80%) for our method to achieve strong performance. This result indicates that combining outcome rewards and process rewards can substantially improve ro-

Verifier Accuracy	BA↑	CA↓
49.95	66.31	0.65
65.70	67.30	0.51
75.00	67.39	0.47
80.31	**68.15**	**0.43**
90.25	**68.15**	0.44

Table 15: Relationship between performance and verifier accuracy.

bustness to verifier errors.

G Prompt Templates

G.1 Prompts Used in Top-tier CoT Methods

In this section, we present the prompts used by the top-tier CoT methods in our comparison, as shown in Figure 8 and 9. These prompts are carefully selected or adapted from their original implementations to ensure a fair and consistent evaluation across methods.

G.2 The Prompt Template for Logic Graph Construction

We present an example prompt for constructing the logic graph in Figure 10. The prompt is concatenated by task description, all input features, and the statistical priors.

G.3 The Prompt Template for LLM-as-Judge

We present the prompt template utilized for the interpretability evaluation in Figure 11. This prompt instructs the judge model (GPT-4o) to evaluate the generated reasoning trajectories based on dimensions such as clarity, logical coherence, and faithfulness, ensuring that the explanation effectively justifies the final prediction.

Prompt Templates Used by the Baseline Methods CoT, ToT, and HtT

CoT:
Given the following data: [...] Based on this data, is the credit payment paid or not? Please respond with '1' for 'Person experienced 90 days past due delinquency or worse' and '0' for 'Not overdue'. Do not answer anything else. Let's think step by step. The format of the output should be like [Explanation of the reasoning process |Answer] ."

ToT:
Given the following data: [...] Based on this data, please start by analyzing the data and consider how each of the following factors could impact the prediction:
1. If the person has a record of 90 days late, how likely is it that they will experience a 90-day delinquency again?
2. How does a high debt ratio influence the likelihood of overdue payments? Does it increase the risk of delinquency?
3. How might a low monthly income increase the likelihood of overdue payments?
4. Does a high revolving utilization indicate a person is at higher risk of delinquency?
5. How does the number of open credit lines and loans affect the likelihood of the person falling into delinquency?
6. If the person has a high number of dependents, could this indicate increased financial pressure and thus a higher chance of experiencing 90 days overdue?
7. What role do the previous delinquency records (30-59 days, 60-89 days) play in predicting future payment behavior?
After evaluating these factors and reasoning through their possible interactions, make a final prediction: - If the person has experienced a 90-day or worse delinquency, respond with '1'. - If the person has not experienced a 90-day or worse delinquency, respond with '0'.
The format of the output should be like [Explanation of the reasoning process | Answer]

HtT:
Task1: Induction Stage - Generate a rule base from the provided dataset.
Context: You are working with a dataset related to credit payments, aiming to determine whether the borrower will experience 90 days past due delinquency or worse ('1') or not be overdue ('0'). Your task is to generate rules that can predict the payment status of future borrowers accurately based on their financial and personal information.
Instructions:
1. Analyze the following data point and derive a rule based on its attributes.
2. The rule should aim to predict whether the borrower will be classified as '1' (experienced 90 days past due delinquency or worse) or '0' (not overdue).
3. Collect all generated rules into a rule base, ensuring their frequency and accuracy.
Data Point: [...]
Example Rule Format: If the revolving utilization of unsecured lines is below 0.2, the age is over 50, no history of being 30-59 days late in the last 2 years, the monthly income exceeds 15000, and no dependents, then the borrower is likely not to have been overdue. Please provide the derived rule and add it to the rule base.
Task2: Deduction Stage - Use the rule base to predict the outcome for a new data point.
Context: In this stage, you have access to a rule base collected during the induction stage. Your goal is to use this rule base to predict whether a new borrower will experience 90 days past due delinquency or worse ('1') or not be overdue ('0').
Instructions:
1. Before analyzing the new data point, review the provided rule base.
2. For the given new data point, identify which rule(s) apply.
3. Based on the applicable rule(s), predict the classification of the new application.
Rule Base (Include relevant rules here): [...]
New Data Point: [...]
Questions:
1. Which rule(s) from the rule base apply to this new data point?
2. Based on the applicable rule(s), what is the predicted classification for this borrower? Is the credit payment paid or not? Please respond with '1' for 'Person experienced 90 days past due delinquency or worse' and '0' for 'Not overdue'. Provide your prediction along with the reasoning behind it.

Figure 8: Prompt templates used by the baseline methods CoT, ToT, and HtT.

Prompt Template Used by the Baseline Method SymbCoT

SymbCoT:
Task1 description: Translate the premises and problem statements of natural language into symbolic format. The purpose of this step is to prepare structured inputs for the subsequent logical reasoning process, ensuring that the problem is represented in a format that is conducive to logical analysis.
Here is an example: [...]
Below is the one you need to translate, do not answer anything else:
Problem: You have a single borrower (let's call them "Borrower") with the following numerical information: [...]
Question: "True or false: Borrower is overdue (experiencing 90 days past due delinquency or worse)? Return '1' if overdue, '0' if not."

Task2 description: Decompose the original problem into smaller, more manageable subproblems. Develop a detailed, step-by-step plan that links the given premises to the problem statement, forming a blueprint for logical reasoning.
Here is an example: [...]
Below are the premises and questions you need to derive a plan to solve, please follow the instructions and example aforementioned. Do not answer anything else.
Translation: [...]
Question: "True or false: Borrower is overdue (experiencing 90 days past due delinquency or worse)? Return '1' if overdue, '0' if not."

Task3 description: Derive the answer through a series of logical reasoning steps based on the premise, problem statement, and formulated plan. This module emphasizes the use of symbolic reasoning rules for logical deduction, such as applying rules from first-order logic, such as Morgan's Law (Modus Tollens).
Here is an example: [...]
Plan: [...]
Translation: [...] Execution: Please clearly indicate whether the Borrower is overdue (experiencing 90 days past due delinquency or worse). Return '1' if overdue, '0' if not.

Task4 description: Verify the translation and reasoning process to ensure that the output of each step is correct and error-free. Validation includes checking whether the symbol translation is semantically equivalent to the original natural language context, and checking whether the reasoning steps strictly adhere to the principles of formal logic.
Below is the one you need to verify:
Original Context: [...]
Translated Context: [...]
Original Execution: [...]
Final answer:

Figure 9: Prompt template used by the baseline method SymbCoT.

The Prompt Template for Logic Graph Construction

Task description:
Analyze all variables' effect on the output: {**Input variable 1**}
Analysis the input variable on the output. Type of input variable: numeric/categorical:
If input variable 1 is numeric, analyze all values of the variable's effect on the output
Analysis of the range of the input variable:
"left": { "value": &&the lower bound of the range&&, "bound": &&whether contains the lower bound&& },
"right": { "value": &&the upper bound of the range&&, "bound": &&whether contains the upper bound&&}
"rule": [
&&the explanation of the impact of the input variable taking this optional value on a borrower's experience of financial distress in the next two years&&,
&&the probability of a person making over 50k a year. is very low/low/medium/high/very high&&
],
.....
analysis of other range of the input variable, make sure it includes all values of the variable and does not overlap, and includes more than two branches.

If input variable 1 is category, analyze all values of the variable's effect on the output.
Analysis of a list of values with similar effects:
"values": &&the values list&&,
"rule": [
&&the impact of the input variable taking this optional value on a person makes over 50k a year&&,
&&the probability of a person making over 50k a year. is very low/low/medium/high/very high&&
] ,
..... analysis of other optional values of the input variable, make sure to include all values of the variable and not overlap, and include more than two branches.
.....
analysis effect of other input variables on the output, make sure to include all variables.

Please replace the content between &&...&& when outputting. And supplement the content of ..... according to the previous example. When outputting, please ensure the analysis explicitly includes all input features, all values of the variable, and the explanation of the input on the target.

Data Statistical Priors:
The statistical characteristics of each input variable are provided below. For numerical variables, these characteristics include the minimum value, maximum value, mean, standard deviation, and quartiles. For categorical variables, the proportion of each possible value is given.
1. Duration of the credit in months (real number): In this feature for all customers, the minimum value is 4.0, the maximum value is 72.0, the mean value is 20.891113892365457, the standard deviation is 12.079183553705317, 25% of users are less than 12.0, the median value is 18.0, and 75% of users are less than 24.0.
2. Credit amount requested (real number): In this feature for all customers, the minimum value is 250.0, the maximum value is 15945.0, the mean value is 3293.474342928661, the standard deviation is 2829.3559513214723, 25% of users are less than 1357.5, the median value is 2325.0, and 75% of users are less than 4083.5.
3. Installment rate in percentage of disposable income (real number): In this feature for all customers, the minimum value is 1.0, the maximum value is 4.0, the mean value is 2.9336670838548184, the standard deviation is 1.1157812796266546, 25% of users are less than 2.0, the median value is 3.0, and 75% of users are less than 4.0.
4. Duration of residence at the current address (real number): In this feature for all customers, the minimum value is 1.0, the maximum value is 4.0, the mean value is 2.834793491864831, the standard deviation is 1.100348965452382, 25% of users are less than 2.0, the median value is 3.0, and 75% of users are less than 4.0.
.....
19. Whether the applicant owns a telephone ('none', 'yes'): In this feature for all customers, there are 2 classes, including['none', 'yes'], among this, none accounts for 60.70087609511889%, yes accounts for 39.2991239048811%.
20. Whether the applicant is a foreign worker ('yes', 'no'): In this feature for all customers, there are 2 classes, including ['yes', 'no'], among this, yes accounts for 95.99499374217773%, no accounts for 4.005006257822278%.
Before outputting, check again that the analysis explicitly includes all input variables, all values of the variable, and the explanation of the impact of the input on the target.

Logic Graph:

Figure 10: Prompt template for constructing the logic graph.

The Prompt Template for LLM-as-Judge

Please evaluate whether the given reasoning path possesses explainability based on the following five aspects:

1. Clarity: Whether the reasoning path clearly presents each step of the reasoning process.
2. Logical Consistency: Whether the reasoning path has a coherent logical structure, with clear correlation between each step.
3. Information Completeness: Whether the reasoning path contains all the critical information required to reach the conclusion.
4. Verifiability: Whether each step in the reasoning path can be verified or traced back to its source.
5. Context Relevance: Whether the reasoning path appropriately considers the context or background information, ensuring the conclusion is reasonable within the given context.

Based on these five criteria, the explainability of the reasoning path is scored on a scale of 1 to 5, where 1 indicates non-explainable and 5 indicates fully explainable. If no reasoning steps are present, the score should be directly set to 1. Please output only the final score.

Figure 11: Prompt template for constructing the logic graph.