Logic-Regularized Verifier Elicits Reasoning from LLMs

Xinyu Wang[*1,2#], **Changzhi Sun**[*2], **Lian Cheng**[1,2#], **Yuanbin Wu**[1],
Dell Zhang[2], **Xiaoling Wang**[1†], **and Xuelong Li**[2†]

[1]Department of Computer Science and Technology, East China Normal University
[2]Institute of Artificial Intelligence (TeleAI), China Telecom
{xinyu_wang@stu, ybwu@cs, xlwang@cs}.ecnu.edu.cn
{czsun}@chinatelecom.cn
{dell.z, xuelong_li}@ieee.org

Abstract

Verifiers are crucial components for enhancing modern LLMs' reasoning capability. Typical verifiers require resource-intensive supervised dataset construction, which is costly and faces limitations in data diversity. In this paper, we propose LOVER, an unsupervised verifier regularized by logical rules. LOVER treats the verifier as a binary latent variable, utilizing internal activations and enforcing three logical constraints on multiple reasoning paths: negation consistency, intra-group consistency, and inter-group consistency (grouped by the final answer). By incorporating logical rules as priors, LOVER can leverage unlabeled examples and is directly compatible with any off-the-shelf LLMs. Experiments on 10 datasets demonstrate that LOVER significantly outperforms unsupervised baselines, achieving performance comparable to the supervised verifier (reaching its 95% level on average). The source code is publicly available at https://github.com/wangxinyufighting/llm-lover.

1 Introduction

Verifiers guide LLMs by providing feedback to optimize their parameters (RL scaling) or outputs (inference scaling), which greatly enhances models' reasoning capabilities (Ouyang et al., 2022; Snell et al., 2024). Verifiers are usually trained with supervised learning (Cobbe et al., 2021; Yu et al., 2024), where they learn to classify reasoning outputs as true or false based on labeled data. It presents two challenges: 1) The verifier relies heavily on labeled data for training, which can be expensive to collect (particularly in specialized or complex domains). For example, annotating a single Olympiad-level problem typically takes a significant amount of time, and adding fine-grained step-level process annotations (Lightman et al., 2023) further increases the workload; 2) Relying on expert annotations may result in a lack of diversity in the solutions (Basile et al., 2021; Xu et al., 2024a), as annotators may favor familiar reasoning methods while overlooking equally valid but less intuitive ones. For example, when annotating geometric problems, annotators may prefer the standard coordinate method and down-vote the less obvious geometric observation. While one can improve the supervision process in various aspects (Yang et al., 2019) (e.g., more experts with diverse mathematical backgrounds and education experiences), LLMs themselves already compact large amounts of knowledge and abilities to sample diverse generations (Minaee et al., 2024; Xu et al., 2024b), a natural question is *whether we could build verifiers without the supervision process?*

To address these challenges, recent research has focused on unsupervised verifiers to uncover the intrinsic reasoning capabilities of LLMs. Typical works include: 1) CoT-Decoding (Wang and Zhou, 2024), which proposes a heuristic rule-based verifier by observing the probabilities of the outputs of LLMs. It selects the correct reasoning path based on the probability difference between the top and secondary tokens in the answer span. In experiments, we observed that CoT-Decoding is sensitive to the backbone LLM choice. For example, when using llama-7b on the GSM8K dataset, CoT-Decoding is 4.8% lower than the majority voting strategy. 2) CCS (Burns et al., 2023) introduces an unsupervised verifier, which is essentially a linear probe optimized through logical consistency loss. Unfortunately, CCS can only address Yes-No questions and struggles to scale to general reasoning tasks. A practical verifier should have fewer limitations on its target problems, enabling it to handle a broader range of reasoning scenarios and provide more flexibility in real-world applications

In this paper, we propose a principled framework

Verifier	Paradigm	Prior	Annotation	Input	Model	Scenario
Wang and Zhou (2024)	Unsup.	Heuristics	-	Probabilities	-	General
Burns et al. (2023)	Unsup.	Logic rule	-	Hidden states	Linear	Yes-No
Cobbe et al. (2021)	Sup.	-	Outcome-based	Text	Decoder-only	General
Lightman et al. (2023)	Sup.	-	Process-based	Text	Decoder-only	General
LOVER	Unsup.	Logic rule	-	Hidden states	MLP	General

Table 1: Comparison between existing verifiers. **Paradigm**: The verifier is trained using a supervised (Sup.) or unsupervised (Unsup.) learning paradigm. **Prior**: The prior knowledge used in the verifier. **Annotation**: The type of annotation data. "Outcome-based" is solution-level annotation. "Process-based" is step-level annotation. **Input**: The input data type of the verifier. **Model**: The model architecture. **Scenario**: Reasoning scenarios suitable for the verifier. "General" typically refers to reasoning problems that have a correct answer. "Yes-No" indicates that the answer to the question is either Yes or No.

LOVER, an unsupervised probabilistic verifier regularized by logical rules. For each reasoning path, we search for the implicit, internal "beliefs" or "knowledge" learned by the LLM to infer the truth value of the reasoning. LOVER begins by generating contrastive assertions through incorporating text templates. It then takes the internal activations of these assertions from the LLM as inputs and produces a binary latent variable to indicate the truth value. Furthermore, LOVER incorporates three logical constraints including negation consistency, intra-group consistency, and inter-group consistency (with multiple reasoning paths grouped by the final answer). To bridge the gap between discrete logical rules and continuous neural networks, we propose corresponding soft probabilistic objectives that support differentiable training. Our contributions are summarized as follows:

- We propose LOVER, a scalable and principled framework for verifying the truth value of reasoning paths, leveraging intrisic knowledge learned by the LLM and regularized by logical rules. Additionally, LOVER is fully compatible with any off-the-shelf LLMs.
- To combine discrete logical rules with neural networks, we propose soft probabilistic objectives that enable LOVER to be trained end-to-end, improving its scalability and performance.
- Our extensive experiments across diverse datasets, including mathematical reasoning, common sense reasoning, and various backbones, demonstrate the effectiveness of the proposed method.

2 Approach

In this section, we present the proposed LOVER, an unsupervised verifier designed to reason over the internal activations of LLMs.

Task Definition Given an LLM and an input question q, we first generate N complete solutions $\{s_i\}_{i=1}^N$, with each s_i representing a CoT path (Sec. 2.1). We then select the best solution based on a learned verifier. For each solution s_i, we define $x_i = q \oplus s_i, x_i^+ = x_i \oplus \mathtt{T}^+, x_i^- = x_i \oplus \mathtt{T}^-$, where $\oplus$ denotes the text concatenation, and $\mathtt{T}^+, \mathtt{T}^-$ are text templates. Given N reasoning paths, we group them into M sets ($M \leq N$) based on the *final answer* (extracted through rules from the answer token). $\mathcal{A}$ represents the index set from 1 to N, and $\mathcal{A}_k$ denotes the k-th group, with $\mathcal{A} = \cup_{k=1}^M \mathcal{A}_k$. The verifier models a probabilistic distribution $p_\theta(\boldsymbol{z}|x)$, where $x \in \cup_{i=1}^N \{x_i^+, x_i^-\}$ and $\boldsymbol{z} \in \{0, 1\}$ is a binary latent variable indicating whether the natural language statement x is valid. In this paper, bold letters indicate variables.

Inspired by CoT-Decoding (Wang and Zhou, 2024) and CCS (Burns et al., 2023), to find the correct answer, we first augment each reasoning path to derive its correct and incorrect assertions, and then treat the truth values of the assertions as binary latent variables. On the one hand, we leverage the internal activations of the LLM as input, enabling better utilization of the model's intrinsic knowledge. On the other hand, the logical constraints provide implicit supervision signals to update the verifier, significantly reducing the need for human supervision.

Next, we first introduce LLM decoding strategy (Sec. 2.1) and how to obtain contrastive assertions (Sec. 2.2). Then we detail the latent verifier model (Sec. 2.3) and describe the logical constraints imposed on the latent variables (Sec. 2.4). Finally, we present the training and inference procedure (Sec. 2.5). Fig. 1 shows an overview of our method.

2.1 LLM Decoding Strategy

Given an input question q and a typical decode-only LLM, there are various strategies to decode N solutions, such as beam search, nucleus sampling, and others. In this work, we follow the
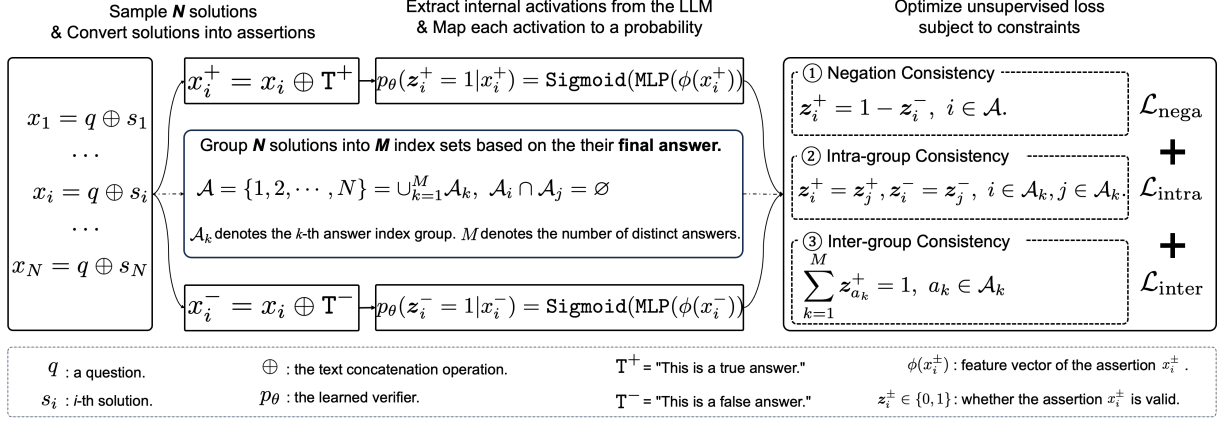
Figure 1: An illustration of our proposed LOVER. For any question q, we create x_i by combining q with the i-th solution from N solutions. We form x_i^+ and x_i^- by adding "This is a true/false answer." to x_i, respectively. Choosing the correct solution involves determining which assertion, x_i^+ or x_i^-, is correct. The hidden states of LLMs are used to represent x_i^+ and x_i^-, which are then input into LOVER to predict the correctness probability of each assertion. We extract the final answer from each solution and group assertions with identical answers together. These assertions follow three natural logical constraints that guide LOVER's unsupervised training. **Negation Consistency** ensures that only one of x_i^+ or x_i^- is correct. **Intra-group Consistency** requires that assertions in the same group have equal correctness probabilities. **Inter-group Consistency** ensures that only one group's x^+ assertion is correct across all groups.

CoT-Decoding (Wang and Zhou, 2024). Specifically, we keep the top N tokens with the highest probabilities at decoding step 0, and then continue with greedy decoding for each token, ultimately producing N solutions. Compared to other strategies, this method is more likely to produce a natural CoT reasoning path and does not rely on complex prompt engineering (Wang and Zhou, 2024). In the experiments, we also study the impact of different decoding strategies(Table 4).

2.2 Contrastive Assertions

For each $x_i = q \oplus s_i$, we construct each contrastive assertions by appending the text templates $\mathtt{T}^+$ and $\mathbb{T}^-$. Formally, this is denoted as $x_i^+ = x_i \oplus \mathtt{T}^+$ and $x_i^- = x_i \oplus \mathtt{T}^-$. In this paper, we adopt $\mathtt{T}^+ =$ "This is a true answer." and $\mathtt{T}^- =$ "This is a false answer.". Importantly, rather than directly considering each reasoning path x_i, we introduce contrastive assertions x_i^+ and x_i^-, which help elicit the internal "beliefs" or "knowledge" learned by the model (Burns et al., 2023).

2.3 Latent Verifier Model

For each natural language assertion $x \in \cup_{i=1}^N \{x_i^+, x_i^-\}$, we first compute the feature vector of x, denoted as $\phi(x)$, [1] then pass it through a

randomly initialized MLP, and finally map it to a probability value using the sigmoid function. Formally, we define the probabilistic distribution of verifier $p_\theta(\boldsymbol{z}|x)$ where $\boldsymbol{z} \in \{0, 1\}$ is a binary latent variable indicating whether the natural language statement x is valid. For simplicity, we use $p_\theta(\boldsymbol{z})$ to represent $p_\theta(\boldsymbol{z} = 1|x)$:

$$p_\theta(\boldsymbol{z}) = p_\theta(\boldsymbol{z} = 1|x) = \mathtt{Sigmoid}(\mathtt{MLP}(\phi(x)).$$

Importantly, LOVER does not modify the weights of the LLM and it does not use labels.

2.4 Logical Constraints

After introducing the binary latent variables $\cup_{i=1}^N \{\boldsymbol{z}^+, \boldsymbol{z}^+\}$, we observe that certain natural logical consistencies between them should be satisfied. Let us look at three such logical consistency requirements.

Negation Consistency Given the contrastive assertions x_i^+ and x_i^-, their corresponding binary latent variables $\boldsymbol{z}_i^+$ and $\boldsymbol{z}_i^-$ should satisfy negation consistency:

$$\boldsymbol{z}_i^+ = 1 - \boldsymbol{z}_i^-, i \in \mathcal{A}.$$

To this end, we relax the logic with soft probability (Chen et al., 2022a; Burns et al., 2023) for differentiability in training and regularization of binary latent variables. Inspired by CCS (Burns et al.,

---

[1] The default is the hidden representation of the last token in the middle layer, and we also explore other options. For details, please refer to Table 5.

2023), we aim for the contrastive assertions x_i^+ and x_i^- to satisfy the following: 1).the sum of their probabilities equals 1 (probability normalization); 2).their probabilities differ significantly (the law of excluded middle).

$$\mathcal{L}_{\text{sum}} = \sum_{i=1}^{N} \left[p_\theta(z_i^+) + p_\theta(z_i^-) - 1 \right]^2,$$

$$\mathcal{L}_{\text{diff}} = \sum_{i=1}^{N} \min \left\{ p_\theta(z_i^+), p_\theta(z_i^-) \right\}^2,$$

$$\mathcal{L}_{\text{nega}} = \mathcal{L}_{\text{sum}} + \mathcal{L}_{\text{diff}}.$$

Note that both losses are necessary; using either one alone leads to a degenerate solution (Burns et al., 2023).

Intra-group Consistency For each group $\mathcal{A}_k$ of reasoning paths, they share the same answer, though their reasoning processes may differ. Overall, we expect the corresponding binary latent variables to satisfy intra-group consistency, i.e.,

$$z_i^+ = z_j^+, z_i^- = z_j^-, \quad i \in \mathcal{A}_k, j \in \mathcal{A}_k.$$

To achieve this goal, we use a simple squared loss:

$$\mathcal{L}_{\text{intra}}^+ = \sum_{k=1}^{M} \sum_{i \in \mathcal{A}_k, j \in \mathcal{A}_k} \left[p_\theta(z_i^+) - p_\theta(z_j^+) \right]^2,$$

$$\mathcal{L}_{\text{intra}}^- = \sum_{k=1}^{M} \sum_{i \in \mathcal{A}_k, j \in \mathcal{A}_k} \left[p_\theta(z_i^-) - p_\theta(z_j^-) \right]^2,$$

$$\mathcal{L}_{\text{intra}} = \mathcal{L}_{\text{intra}}^+ + \mathcal{L}_{\text{intra}}^-.$$

Inter-group Consistency Among the N reasoning paths, there are M distinct answers. We assume that the LLM's capabilities are sufficiently strong to ensure the presence of a correct answer. We examine the GSM8k dataset and find that when $N = 10$, the $P@10$ accuracy of qwen-2.5 can reach 91.43%. This confirms the validity of the above assumption. Specifically, for each group $\mathcal{A}_k$, we randomly select an a_k and hope that its corresponding binary latent variable satisfies:

$$\sum_{k=1}^{M} z_{a_k}^+ = 1, \quad a_k \in \mathcal{A}_k. \tag{1}$$

To achieve this inter-group consistency, we propose a soft probability solution.

$$\mathcal{L}_{\text{inter}}^{\text{sum}} = \left[\sum_{k=1}^{M} p_\theta(z_{a_k}^+) - 1 \right]^2.$$

However, in the experiments, we observe that if only the loss $\mathcal{L}_{\text{inter}}^{\text{sum}}$ is used, the M probabilities $p_\theta(z_{a_k}^+)$ tend to become uniform. To address this issue, we propose an entropy regularization. We introduce a probability distribution $\hat{p}$ which defines over the M variables $\{z_{a_k}^+\}_{k=1}^M$.

$$\mathcal{L}_{\text{inter}}^{\text{h}} = \mathcal{H}\left[\hat{p}(\cdot)\right], \quad \hat{p}(z_{a_k}^+) = \frac{p_\theta(z_{a_k}^+)}{\sum_{i=1}^{M} p_\theta(z_{a_i}^+)},$$

$$\mathcal{L}_{\text{inter}} = \mathcal{L}_{\text{inter}}^{\text{sum}} + \mathcal{L}_{\text{inter}}^{\text{h}},$$

where $\mathcal{H}$ denote the entropy function. In addition, we also explore a soft logic-based solution, which is encapsulated in Appendix A.

2.5 Training and Inference

Training The final loss function is the sum of three losses mentioned above, which is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{nega}} + \mathcal{L}_{\text{intra}} + \mathcal{L}_{\text{inter}}.$$

LOVER is structured as a MLP with 2 hidden layers, and we select ReLU as the activation function. We use AdamW(Loshchilov and Hutter, 2019) as optimizer (weight_decay = 0.01), and set learning rate to $1 \times e^{-5}$.

Inference Given an input question q, we first decode N candidate solutions (Sec. 2.1) to obtain $\{x_i\}_{i=1}^N$. For each x_i, we generate contrastive assertions x_i^+ and x_i^- of s_i (Sec. 2.2) and compute corresponding probability $p_\theta(z_i^+)$ and $p_\theta(z_i^-)$ based on the latent verifier model (Sec. 2.3). Both $p_\theta(z_i^+)$ and $1 - p_\theta(z_i^-)$ should represent the probability that the x_i is correct. we consequently take the average of these (Burns et al., 2023):

$$p_\theta(z_i) = \frac{1}{2} \left[p_\theta(z_i^+) + (1 - p_\theta(z_i^-)) \right].$$

Then we group them into M sets based in the final answer (extracted through rules from answer token). For each group $\mathcal{A}_k$, we compute the group score g_k using two strategies: *max* and *sum*. The *max* strategy computes the group score by selecting the maximum p_θ within the group: $g_k = \max_{i \in \mathcal{A}_k} p_\theta(z_i)$. The *sum* strategy computes the group score by summing up all p_θ within the group: $g_k = \sum_{i \in \mathcal{A}_k} p_\theta(z_i)$. Finally, we select the answer with the highest group score g_k among the M groups. Appendix C provides PyTorch-style pseudocode for the inference procedure.

LLM	Dataset → Verifier ↓	Mathematics		Open-Domain Knowl.	
		GSM8K	iGSM	HotpotQA	MMLU-P
mistral-7b	Supervised	61.18	44.50	82.96	41.93
	Greedy	16.15	19.00	70.74	37.65
	Majority Voting	43.82	34.50	76.86	41.07
	CoT-Decoding (max)	36.62	12.75	69.87	38.04
	CoT-Decoding (sum)	47.76	36.75	76.86	41.43
	LoVer (max)	50.41	28.75	74.24	40.36
	LoVer (sum)	**53.14**	**41.25**	**77.95**	**41.82**
llama-8b	Supervised	83.24	43.75	82.96	55.86
	Greedy	36.09	35.00	77.29	45.67
	Majority Voting	75.96	39.00	81.00	50.58
	CoT-Decoding (max)	38.51	23.25	77.95	45.92
	CoT-Decoding (sum)	71.11	40.50	78.16	54.61
	LoVer (max)	70.60	39.00	81.04	52.65
	LoVer (sum)	**79.97**	**42.50**	**83.80**	**55.61**
qwen-7b	Supervised	92.11	54.25	83.62	61.13
	Greedy	61.71	46.50	78.82	54.93
	Majority Voting	89.46	51.00	**83.41**	57.00
	CoT-Decoding (max)	63.84	31.50	81.66	54.22
	CoT-Decoding (sum)	89.76	51.00	82.09	58.81
	LoVer (max)	83.47	48.25	81.41	59.28
	LoVer (sum)	**91.43**	**52.00**	82.75	**61.24**

Table 2: The overall experimental results of LoVer and other baselines on the four datasets. Accuracy is utilized to measure the performance. The best results of each setting are in bold. MMLU-P stands for "MMLU-Pro" dataset.

3 Experiments

Datasets We conduct experiments on datasets of both mathematical and open-domain knowledge reasoning. For mathematical reasoning, we use the Grade-school math problems, GSM8K (Cobbe et al., 2021) and more challenging iGSM dataset (Ye et al., 2024). For open-domain knowledge reasoning, we use HotpotQA(Yang et al., 2018) and MMLU-Pro(Wang et al., 2024). Furthermore, to evaluate the out-of-distribution (OOD) generalization of LoVer, we employ Boolean Expressions, Web of Lies, Object Counting, Navigate, Multi-Step Arithmetic and Causal Judgement from BIG-Bench Hard(Suzgun et al., 2022). The details of datasets are provided in Appendix B.

Evaluation We evaluate accuracy by *strictly matching* the final answer from the response with the ground truth answer.

Baselines. We test three open-source LLMs: llama-8b, mistral-7b and qwen with different

scales, ranging from 0.5B, 1.5B, 3B, 7B, and 32B. [2] We compare LoVer against following methods:

- "Greedy" decoding selects the most probable token at each step.
- "Majority Voting" decodes multiple outputs and select the optimal answer by voting (Lewkowycz et al., 2022; Wang et al.). [3]
- "CoT-Decoding" (Wang and Zhou, 2024) selects correct reasoning paths based on answer confidence (probability disparity between the top and secondary tokens in answer spans).
- "Supervised" is the supervised LoVer. It is trained using gold label data, with the training objective being the standard binary cross-entropy loss. Theoretically, this is the ceiling of LoVer.

3.1 Main Results

LoVer *effectively enhances reasoning abilities*

---

across models and reasoning types. As shown in Table 2, LOVER (sum) achieves the highest accuracy in all scenarios. LOVER (max) outperforms CoT-Decoding in 40% of cases and matches the average accuracy of Majority Voting.

LOVER (sum) achieves an average absolute accuracy gain of 3.3% over Majority Voting. Unlike Majority Voting, which relies solely on the frequency of answers, LOVER not only considers answer counts but also leverages the internal knowledge of LLMs. Driven by logical constraints, LOVER can more effectively utilize the LLMs' internal knowledge to score the correctness of assertions. Thus, LOVER represents an optimized and weighted voting method. Compared to CoT-Decoding, LOVER shows an average absolute accuracy gain of 2.9%. LOVER focuses on the correctness of the solution itself, rather than emphasizing the format of the solution as in CoT-Decoding. CoT-Decoding aims to elicit reasoning paths with CoT processes, leading to significant accuracy gains on weaker models (those unable to autonomously generate CoT solutions without CoT prompting) but limited improvements on stronger models. As a result, compared to CoT-Decoding, LOVER is less affected by the underlying capabilities of the LLM. LOVER (sum) consistently outperforms LOVER (max), demonstrating the effectiveness of the *sum* strategy and highlighting the importance of the frequency of answers. CoT-Decoding (max) achieves an average accuracy similar to Greedy, indicating that relying solely on the probability with answer tokens is insufficient.

3.2 Ablation Studies

3.2.1 The effect of different logic constraints

Incorporating logical constraints can significantly enhance LOVER*'s performance.* Table 3 reveals that the exclusion of the $\mathcal{L}_{\text{inter}}$ led to the most significant drop in reasoning accuracy, indicating its crucial role in enhancing model performance. Without $\mathcal{L}_{\text{inter}}$, LOVER tends to optimize towards assigning a score of 1 to all x_i^+ and 0 to all x_i^-.

In this scenario, LOVER loses the ability to discern the correctness of assertions. Removing $\mathcal{L}_{\text{nega}}$ also results in a noticeable decrease in accuracy. Without $\mathcal{L}_{\text{nega}}$, the premise that each assertion has only one correctness label cannot be satisfied. As a result, LOVER tends to optimize towards assigning identical scores to both x_i^+ and x_i^-. The removal of $\mathcal{L}_{\text{intra}}$ has a minimal impact on reasoning ac-

Loss	GSM8K		MMLU-Pro	
	sum	**max**	**sum**	**max**
LOVER	**53.14**	**50.41**	**41.82**	**40.36**
w/o $\mathcal{L}_{\text{nega}}$	51.78	34.79	40.07	37.37
w/o $\mathcal{L}_{\text{intra}}$	52.76	44.20	41.75	40.00
w/o $\mathcal{L}_{\text{inter}}$	49.96	38.13	38.15	35.44

Table 3: Accuracy of LOVER on GSM8K and MMLU-Pro using different setting of logic constraints over `mistral-7b`.

curacy. $\mathcal{L}_{\text{intra}}$ enforces consistency in correctness probabilities for solutions with the same final answer. However, solution correctness depends not only on the final answer but also on the problem-solving process, which may contain errors even if the answer is correct. Enforcing the consistency of correctness probabilities solely based on the same final answer may have limitations.

Logical constraints have a more substantial impact on LOVER (max) than on LOVER (sum). Insufficient constraints hinder LOVER (max)'s ability to accurately assess assertion validity, while LOVER (sum) mitigates this by incorporating answer frequency, reducing sensitivity to constraint variations.

3.2.2 The effect of decoding strategies

LOVER *achieves hightest reasoning accuracy combined with different decoding strategies.* Table 4 shows that all methods achieve their highest accuracy under the natural CoT-Decoding, outperforming temperature sampling and beam search sampling by an average of 19.5% and 22.3%.

Decoding Strategies	Nat	Temp	Beam
Majority Voting	43.82	27.07	24.79
CoT-Decoding	47.76	28.05	24.94
LOVER	**53.14**	**31.31**	**28.20**

Table 4: Accuracy of LOVER on GSM8K test set using different decoding strategies over `mistral-7b`. Nat stands for natural CoT decoding, which is the default decoding strategy in this paper. Temp stands for Temperature sampling (`temperature = 0.7`). Beam stands for Beam Search sampling.

LOVER consistently achieves the best performance across all decoding strategies, delivering an average absolute accuracy improvement of 5.7% compared to Majority Voting. In contrast, CoT-Decoding only achieves only a 1.69% improvement over Majority Voting. The performance of

CoT-Decoding tends to rely more heavily on the format of LLMs' output rather than its correctness. If the decoding strategy fails to elicit outputs in a specific format (e.g., those containing a CoT process), the effectiveness of CoT-Decoding is compromised. In contrast, LOVER analyzes the correctness of LLMs' outputs and is therefore less affected by changes in sampling strategies compared to CoT-Decoding.

3.2.3 The effect of hidden states from different layers

Hidden States of middle layer optimize LOVER *'s Performance.* Previous research shows that hidden state from the middle to deeper layers contain richer knowledge compared to those in the shallower layers. Table 5 shows that LOVER achieves

Layer	GSM8K		MMLU-Pro	
	sum	max	sum	max
5	51.82	36.26	41.32	38.78
15	51.91	37.28	41.34	39.46
20	**53.14**	**50.41**	**41.82**	**40.36**
25	52.41	32.16	41.25	38.07
32	52.22	38.80	41.37	38.10

Table 5: Accuracy of LOVER on GSM8K test set and MMLU-Pro using hidden states from different `mistral-7b` layers.

optimal performance when utilizing hidden states from the 15th or 20th layers, which aligns with the conclusions of prior research. The selection of hidden states' layer has a significantly greater impact on LOVER (max) compared to LOVER (sum). For instance, on GSM8K, the standard deviation of accuracy for LOVER (max) across different layers is 7.1, whereas it is only 0.6 for LOVER (sum). This is because the knowledge contained in the hidden states directly influences LOVER (max)'s judgment on the correctness of assertions. LOVER (sum) incorporates the frequency of answers, which mitigates the influence of layer selection.

3.2.4 The effect of problem difficulty

We investigate the impact of problem complexity using iGSM dataset with varying levels of difficulty. More detailed information about iGSM dataset is provided in Appendex B.4. LOVER *Effectively enhances reasoning accuracy across various levels of problem difficulty.* Table 6 demonstrates LOVER consistently achieves accuracy gains over

max_op	Majority Voting	LOVER
2	55.0	67.0 (+21.8%)
4	40.0	47.0 (+17.5%)
8	25.0	30.0 (+20.0%)
16	18.0	21.0 (+16.7%)

Table 6: Accuracy of LOVER on iGSM with different difficulty over `mistral-7b`. A higher `max_op` indicates greater difficulty. The values in parentheses indicate the percentage accuracy gain over the baseline.

the baseline across varying levels of problem difficulty, with no significant decline in performance gains as the difficulty increases. This is attributed to LOVER's effective utilization of logical rules to harness the internal knowledge of LLMs, suggests that LOVER exhibits greater robustness in handling complex reasoning problems.

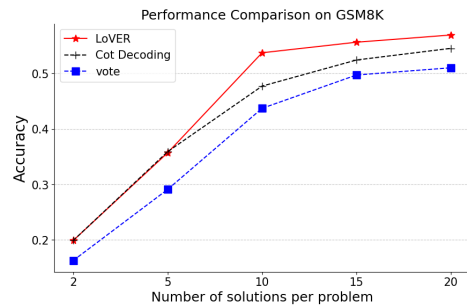### 3.2.5 The effect of numbers of solutions per question

Figure 2: An accuracy comparison of LOVER and baselines across different numbers of solutions on GSM8K over `mistral-7b`.

LOVER *maintains high reasoning accuracy regardless of* N. The reasoning accuracy of all methods increases as the number of solutions grows shown in Table 2. Across different numbers of solutions, LOVER consistently outperform Majority Voting, achieving an average absolute accuracy gain of 6.4%. As the number of solutions increases (> 5), LOVER surpasses CoT-Decoding, delivering an average absolute accuracy gain of 3.9%. Compared to CoT-Decoding, LOVER extracts richer information from hidden states, enabling more effective utilization of the diverse solutions sampled during the decoding process.

3.2.6 The effect of model scales

LOVER *enhances reasoning accuracy across model scales.* Figure 3 shows that LOVER en-
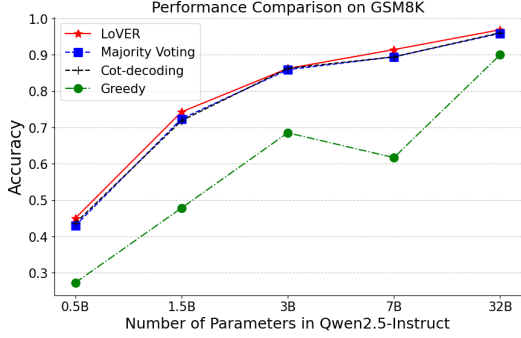
Figure 3: LOVER reliably improves reasoning performance across model scales (Qwen-2.5 family).

hances reasoning accuracy across different model scales over the Qwen-2.5 family. LOVER enables a 7B-parameter LLM to achieve reasoning accuracy comparable to that of a 32B-parameter LLM. LOVER achieves an average accuracy gains of 1.7% across five models with varying parameter sizes and consistently outperforms the baselines. In contrast, CoT-Decoding and Majority Voting exhibit comparable performance.

3.3 OOD Generalization

Datasets	Voting	CoT-D	LOVER
Boolean Exp	61.6	**72.4**	**72.4**
Web of Lies	25.6	39.6	**44.8**
Object Cnt	46.0	48.0	**48.4**
Navigate	49.6	57.2	**57.6**
Arithmetic	12.0	11.6	**12.4**
Causal Jud	45.9	**60.4**	**60.4**

Table 7: Reasoning accuracy of LOVER on six OOD datasets over mistral-7b. Voting stands for "Majority Voting"; CoT-D stands for "CoT-Decoding". Boolean Exp stands for "Boolean Expression" dataset; Object Cnt stands for "Object Counting" dataset; Arithmetic stands for "Multi-Step Arithmetic Two" dataset; Causal Jud stands for "Causal Judgment" dataset.

The improvements brought by LOVER *can be transferred to OOD problems.* We select six distinct datasets from BIG-Bench Hard as OOD datasets. As shown in table 7, LOVER achieves higher or equal accuracy than CoT-Decoding across six all datasets, and outperforms Majority Voting on all datasets, with an average accuracy improvement of 2.5%. This demonstrates the strong OOD generalization capability of LOVER. By leveraging the latent knowledge and logical constraints, LOVER learns patterns for evaluating assertion correctness, independent of the domain-specific context of the assertions.

4 Related Work

LLM Reasoning Existing research on LLM reasoning can be roughly divided into two categories: extrinsic reasoning and intrinsic reasoning. Extrinsic reasoning mainly involves complex prompt engineering (Wei et al., 2022; Yao et al., 2024), verifier based on outcome or processes (Cobbe et al., 2021; Lightman et al., 2023), and customized search algorithms (e.g., A*, MCTS) (Zhuang et al.; Wan et al., 2024). Research in this domain focuses on various reasoning tasks such as mathematical reasoning (Cobbe et al., 2021; Hendrycks et al.), logical reasoning (Liu et al., 2020), common-sense reasoning (Yang et al., 2018), and more (Liu et al., 2024; Yuan et al., 2024; Chen et al., 2022b). Intrinsic reasoning seeks to explore the model's internal knowledge, primarily through the observation and manipulation of its hidden layers or output probabilities. The probe method involves using auxiliary classifiers or probes to analyze and interpret the internal representations learned by the model, offering insights into its understanding and reasoning processes (Belinkov, 2022; Alain and Bengio, 2016). Unlike supervised probes, CCS (Burns et al., 2023) learns a linear classifier to uncover latent knowledge in an unsupervised manner, while CoT-Decoding (Wang and Zhou, 2024) assesses the truth value of candidate solutions based on answer confidence. However, the main drawbacks are that CoT-Decoding is essentially an expert-curated heuristic rule, and CCS is limited to Yes-No questions, lacking scalability. LOVER belongs to the second category and is applicable to general reasoning tasks that do not require supervised data. It can be seen as an unsupervised probe (verifier) guided by logic rule.

Neural Logical Reasoning Neural logic integrates neural networks with logical reasoning to enhance model's interpretability, consistency, and reasoning capabilities. One paradigm involves learning logical operators such as AND, OR, and NOT as differentiable neural modules, guided by self-supervised logic regularization (Shi et al., 2020). Prior studies have demonstrated its effectiveness in proof generation (Sun et al., 2021), fact checking (Chen et al., 2022a), NLI (Li et al., 2019) and recommender systems (Chen et al., 2021). Another standard method is based on the variational EM

framework (Ru et al., 2021; Qu and Tang, 2019; Zhou et al., 2020). LoVer draws inspiration from both lines of work. We represent the output of the verifier as binary latent variables, which are regularized with soft logic.

5 Conclusion

We propose LoVer, an unsupervised verifier regularized by logical rules for enhancing LLMs' reasoning capability. We design three logical rules to guide LoVer in effectively leveraging unlabeled data, achieving performance comparable to supervised methods. LoVer is compatible with any white-box LLMs and adaptable to diverse reasoning tasks. Experiments show LoVer significantly enhances the reasoning accuracy of LLMs while demonstrating strong OOD generalization capabilities.

Limitations

LoVer relies on the hidden states of LLMs, which inherently restricts its applicability to white-box LLMs. This dependency prevents LoVer from being directly utilized in black-box LLMs scenarios. Although LoVer does not rely on extracting final answers from responses and can be applied to scenarios where responses lack explicit conclusions, we do not conduct experiments to explore this aspect in this paper.

Acknowledgments

References

Guillaume Alain and Yoshua Bengio. 2016. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*.

Valerio Basile, Michael Fell, Tommaso Fornaciari, Dirk Hovy, Silviu Paun, Barbara Plank, Massimo Poesio, Alexandra Uma, et al. 2021. We need to consider disagreement in evaluation. In *Proceedings of the 1st workshop on benchmarking: past, present and future*, pages 15–21. Association for Computational Linguistics.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. 2023. Discovering latent knowledge in language models without supervision. In *The Eleventh International Conference on Learning Representations*.

Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. 2021. Neural collaborative reasoning. In *Proceedings of the Web Conference 2021*, pages 1516–1527.

Jiangjie Chen, Qiaoben Bao, Changzhi Sun, Xinbo Zhang, Jiaze Chen, Hao Zhou, Yanghua Xiao, and Lei Li. 2022a. Loren: Logic-regularized reasoning for interpretable fact verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10482–10491.

Jiangjie Chen, Rui Xu, Ziquan Fu, Wei Shi, Zhongqiao Li, Xinbo Zhang, Changzhi Sun, Lei Li, Yanghua Xiao, and Hao Zhou. 2022b. E-kar: A benchmark for rationalizing natural language analogical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3941–3955.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. 2019. A logic-driven framework for consistency of neural models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3924–3935.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.

Yufang Liu, Tao Ji, Changzhi Sun, Yuanbin Wu, and Aimin Zhou. 2024. Investigating and mitigating object hallucinations in pretrained vision-language (clip) models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18288–18301.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey, 2024. *arXiv preprint arXiv:2402.06196*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Meng Qu and Jian Tang. 2019. Probabilistic logic neural networks for reasoning. *Advances in neural information processing systems*, 32.

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. *Preprint*, arXiv:2412.15115.

Dongyu Ru, Changzhi Sun, Jiangtao Feng, Lin Qiu, Hao Zhou, Weinan Zhang, Yong Yu, and Lei Li. 2021. Learning logic rules for document-level relation extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1239–1250.

Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural logic reasoning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1365–1374.

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*.

Changzhi Sun, Xinbo Zhang, Jiangjie Chen, Chun Gan, Yuanbin Wu, Jiaze Chen, Hao Zhou, and Lei Li. 2021. Probabilistic graph reasoning for natural proof generation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3140–3151.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. In *Forty-first International Conference on Machine Learning*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jin Xu, Mariët Theune, and Daniel Braun. 2024a. Leveraging annotator disagreement for text classification. In *Proceedings of the 7th International Conference on Natural Language and Speech Processing (IC-NLSP 2024)*, pages 1–10.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024b. A survey on knowledge

distillation of large language models. *arXiv preprint arXiv:2402.13116*.

Yinfei Yang, Oshin Agarwal, Chris Tar, Byron C Wallace, and Ani Nenkova. 2019. Predicting annotation difficulty to improve task routing and model performance for biomedical information extraction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1471–1480.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. 2024. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. In *The Thirteenth International Conference on Learning Representations*.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2024. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 858–875.

Siyu Yuan, Jiangjie Chen, Changzhi Sun, Jiaqing Liang, Yanghua Xiao, and Deqing Yang. 2024. Analogykb: Unlocking analogical reasoning of language models with a million-scale knowledge base. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1249–1265.

Wangchunshu Zhou, Jinyi Hu, Hanlin Zhang, Xiaodan Liang, Maosong Sun, Chenyan Xiong, and Jian Tang. 2020. Towards interpretable natural language understanding with explanations as latent variables. *Advances in Neural Information Processing Systems*, 33:6803–6814.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*.

A Logic-based inter-group Consistency

The basic idea is to transform Eq. 1 into a logical expression and then apply product t-norms to relax the logic (Chen et al., 2022a; Li et al., 2019).

$$
\begin{aligned}
r &= (z_1^+ \wedge \neg z_2^+ \wedge \cdots \wedge \neg z_M^+) \\
&\vee (\neg z_1^+ \wedge z_2^+ \wedge \cdots \wedge \neg z_M^+) \\
&\quad \vdots \\
&\vee (\neg z_1^+ \cdots \wedge z_M^+), \\
\mathcal{L}_{\text{inter}}^{\text{logic}} &= \texttt{t-norms}(r).
\end{aligned}
$$

The accuracy comparison of LOVER using using different kinds of $\mathcal{L}_{\text{inter}}$ are provided in Table 8. LOVER using default $\mathcal{L}_{\text{inter}}$ is better.

	GSM8K	
	sum	**max**
default $\mathcal{L}_{\text{inter}}$	**53.14**	**50.41**
logic-based $\mathcal{L}_{\text{inter}}$	52.99	46.39

Table 8: Reasoning accuracy of LOVER on GSM8K test set over `mistral-7b` using different kinds of $\mathcal{L}_{\text{inter}}$.

B Datasets

B.1 HotpotQA

HotpotQA is a Wikipedia-based question-answering dataset. In the full wiki setting, HotpotQA consists of 90,447 training samples, 7,405 validation samples, and 7,405 test samples. A key feature of HotpotQA is that these questions require finding and reasoning over multiple supporting documents to answer.

Since the test set does not provide standard answers, we use the validation set as the test set. To make the extraction of final answers from response easier and more accurate, we only selected questions with "yes" or "no" as answers for our experiments. This resulted in 5,481 training samples and 458 test samples. Table 9 provides examples of questions with "yes" and "no" answer. We used the entire test set and a randomly selected subset of 3,000 training samples for the experiments.

In our experiments, we only utilized the questions and did not incorporate the related supporting texts.

B.2 MMLU pro

MMLU-Pro is an advanced benchmark for assessing language models on more extensive and challenging tasks. It consists of over 12,000 questions,

HotpotQA
Question: Were Scott Derrickson and Ed Wood of the same nationality?   **Answer**: yes
Question: Were Pavel Urysohn and Leonid Levin known for the same type of work?   **Answer**: no

Table 9: Examples of HotpotQA dataset with "yes" and "no" answer.

each furnished with ten possible answers, spanning 14 distinct domains. These domains include Biology, Business, Chemistry, Computer Science, Economics, Engineering, Health, History, Law, Math, Philosophy, Physics, Psychology, and Others.

MMLU-Pro provides over 12,000 test samples and 70 validation samples, with no training set included. Therefore, we randomly selected one-third (4 out of 14) of the domains to serve as the test set, while the remaining domains were used for training and validation. With the random seed set to 0, we obtained data from the following four domains as the test set: Computer Science, Physics, History, and Biology. The statistics of test set is shown in Tabel 10.

Datasets	#Data
Computer Science	410
Physics	1299
History	381
Biology	717
All	2807

Table 10: The statistics of the four test datasets from MMLU-Pro.

B.3 BIG-Bench Hard

We randomly selected six datasets from BIG-Bench Hard as out-of-distribution (OOD) datasets, with the specific dataset names and corresponding statistics summarized in Table 11.

B.4 iGSM

We construct a more challenging mathematics dataset called iGSM. Following iGSM(Ye et al., 2024), we control problems difficulty by setting different numbers of operations (**max_op**) in the solutions. We utilized the iGSM synthetic data

Datasets	#Data
Boolean Expression	250
Web of Lies	250
Object Counting	250
Navigate	250
Multi-Step Arithmetic Two	250
Causal Judgment	187
All	1437

Table 11: The statistics of the six OOD datasets from BIG-Bench Hard.

generator[4] to construct the dataset by setting `random_seed` to 42, `max_edge` to 12, `perm_level` to 5, `detail_level` to 0, and `max_op` to 2, 4, 8, and 16, respectively. The detailed data statistics are presented in Table 12.

max_op	#Data	
	test	training
2	100	-
4	100	-
8	100	1500
16	100	-
All	400	1500

Table 12: The statistics of the iGSM test set in our experiments.

Table 14 provides two examples of iGSM where `max_op` is 2 and 16 respectively.

---

iGSM: `max_op=2`

Question: The number of each Goldfish's Proximal Phalanx equals 8. The number of each Swordfish's Metacarpal I equals the sum of each Goldfish's Bone, each Goldfish's Proximal Phalanx and each Eel's Bone. The number of each Mahi Mahi's Radial Carpal equals 3. The number of each Goldfish's Metacarpal IV equals 12. How many Radial Carpal does Mahi Mahi have?
Solution: Define Mahi Mahi's Radial Carpal as n; so n = 3.
Answer: 3

Table 13: Examples of iGSM dataset with different `max_op`.

---

iGSM: `max_op=16`

Question: The number of each Spinal Cord's Transitional Epithelial Cells equals the difference of each Cerebellum's Hepatocytes and each Albatross's Spinal Cord. The number of each Cerebellum's Hepatocytes equals each Parrot's Organs. The number of each Albatross's Spinal Cord equals each Cerebellum's Hepatocytes. The number of each Spinal Cord's Hepatocytes equals the sum of each Cerebellum's Hepatocytes, each Albatross's Spinal Cord and each Parrot's Cerebellum. The number of each Parrot's Cerebellum equals 6. How many Cells does Albatross have?
Solution: Define Parrot's Cerebellum as c; so c = 6. Define Parrot's Organs as i; so i = c = 6. Define Cerebellum's Hepatocytes as X; so X = i = 6. Define Albatross's Spinal Cord as u; so u = X = 6. Define Spinal Cord's Hepatocytes as S; b = X + u = 6 + 6 = 12; so S = b + c = 12 + 6 = 18. Define Spinal Cord's Transitional Epithelial Cells as F; so F = X - u = 6 - 6 = 0. Define Spinal Cord's Cells as W; so W = S + F = 18 + 0 = 18. Define Albatross's Cells as o; so o = u * W = 6 * 18 = 16.,
Answer: 16

Table 14: Examples of iGSM dataset with different `max_op`.

B.5 GSM8K

GSM8K consists of 8,792 high quality grade school math problems, with 7,473 in the training set and 1,319 in the test set. GSM8K is designed to evaluate the mathematical reasoning capabilities of models. In our experiments, we select 7,000 samples randomly from the training set for training, 473 samples from the training set as the validation set, and the entire test set for testing.

---

[4]https://github.com/facebookresearch/iGSM

C   Pseudocode (PyTorch-like)

```python
"""Step 1: Sample n responses from LLM"""
def sample_responses(llm, prompt, n):
    responses = []

    for _ in range(n):
        response = model.sample(prompt)
        responses.append(response)
    return responses

"""Step 2: Create contrastive assertions"""
def create_assertions(question, responses):
    pos_assertions = []
    neg_assertions = []

    for response in responses:
        qa_pair = f"Q:{question} A:{response}"

        pos_assertion= qa_pair+"this is a true answer."
        neg_assertion= qa_pair+"this is a false answer."

        pos_assertions.append(positive_assertion)
        neg_assertions.append(negative_assertion)

    return pos_assertions, neg_assertions

"""Step 3: Get hidden states of assertions"""
def get_hidden_states(llm, assertions, layer_idx):
    representations = []

    for assertion in assertions:
        outputs = llm(assertion, output_hidden_states)
        hidden_state = outputs.hidden_states[layer_idx]
        hidden_state = hidden_state[:, −1, :]
        representations.append(hidden_state)

    return representations

"""Step 4: Compute scores using verifier"""
def compute_scores(verifier, pos_reps, neg_reps):
    pos_scores = verifier(pos_reps)
    neg_scores = verifier(neg_reps)

    final_scores = 0.5 ∗ (pos_scores+(1−neg_scores))

    return final_scores

"""Step 5a: Select final answer by Max strategy"""
def select_by_max(responses, scores):
    max_idx = argmax(scores)
    best_response = responses[max_idx]

    return extract_final_answer(best_response)

"""Step 5b: Select final answer by Sum strategy"""
def select_by_sum(responses, scores)
    answer_scores = {}

    for response, score in zip(responses, scores):
        final_answer = extract_final_answer(response)
        answer_scores[final_answer] += score

    best_answer = find_key_by_max_value(answer_scores)
    return best_answer
```

```python
if __name__ == "__main__":
    # Step 1: Generate responses
    prompt = construct_prompt(q)
    responses = sample_responses(llm, prompt, n)

    # Step 2: Create contrastive assertions
    pos_asserts, neg_asserts=create_assertions(q, responses)

    # Step 3: Get hidden representations
    layer = 20
    pos_reps = get_hidden_states(llm, pos_asserts, layer)
    neg_reps = get_hidden_states(llm, neg_asserts, layer)

    # Step 4: Compute verification scores
    scores = compute_scores(verifier, pos_reps, neg_reps)

    # Step 5: Select final answer based on strategy
    final_answer_max=select_by_max(responses, scores)
    final_answer_sum=select_by_sum(responses, scores)
```