DeduCE: Deductive Consistency as a Framework to Evaluate LLM Reasoning

Anonymous Authors[1]

Abstract

Deductive reasoning is a key component in solving complex problems, especially those involving math and logic. Formally, deductive reasoning involves two subtasks: *understanding* a set of input premises and *inferring* the conclusions that follow from them. Recent work highlights deficiencies in deductive reasoning of language models (LMs) by measuring final accuracy. Going beyond accuracy, we propose a metric to directly characterize deductive reasoning, which allows comparison of LMs' capabilities across both deductive subtasks and guides where finetuning efforts may help the most. The ideal evaluation would require access to an oracle system that can verify any candidate conclusion from an LM given input premises, which in turn requires computing the deductive closure. Instead, we propose a practical solution that requires access to only one correct solution for a problem and measures *deductive consistency* (DC) over varying *windows* of reasoning steps. By breaking up LMs' reasoning steps into dynamic windows, we can directly evaluate the two subtasks: how well do LMs understand input premises with increasing context lengths, and how well can they infer conclusions over multiple reasoning hops? Since existing benchmarks may be memorized, we also develop a pipeline to evaluate LMs' deductive consistency on novel, perturbed versions of benchmark problems. Our key result is that LMs are more robust to processing input premises of varying lengths than inferring conclusions over a longer horizon. For instance, on datasets such as GSM-8k and ProntoQA, DC of LMs stays the same regardless of the length of prefixes; the key source of error is the number of output reasoning steps. Labeling the reasoning errors reveals that a significant fraction of the errors are calculation or logical errors. Applying prevalent mitigation techniques such as fine-tuning or tool use reduces some kind of errors, but cannot fully remove the decay in DC.

1. Related Work

Deductive consistency is a metric that goes beyond accuracy to measure reasoning abilities. Hence, it is related to works which evaluate reasoning using accuracy, which propose other metrics, and which use LLMs for deductive reasoning tasks. We discuss these categories below:

Evaluating reasoning beyond memorization. To avoid confounding in experiments due to dataset memorization, novel datasets based on synthetic data have been proposed (Zhu et al., 2023). To keep real world relevance, recent work propose perturbed or novel versions of existing datasets, for math (Mirzadeh et al., 2024; Zhang et al., 2024), analogical reasoning (Lewis & Mitchell, 2024b;a), deductive reasoning Sanyal et al. (2022), and many other diverse tasks (Wu et al., 2024).

Metrics for reasoning beyond final answer's accuracy. Xu et al. (2024) propose four different types of metrics based on answer correctness and explanation correctness. Seals & Shalin (2024) test deductive competence on logical questions.Paul et al. (2024) bias LLMs to adhere to their intermediate CoT steps. Another stream of work checks language models' ability to detect errors in a solution (Zeng et al., 2024) and fix any detected errors (Singh et al., 2024). In comparison, we aim to provide a general metric for any task that involves deductive reasoning.

Deductive reasoning in LLMs. LogicBench evaluates various models on natural language problems over propositional, first order, and non-monotonic logic (Parmar et al., 2024). JustLogic is a synthetic deductive dataset Chen et al. (2025). Other examples include analyzing categorical syllogisms (Zong & Lin, 2024) and proving theorems in intuitionistic propositional logic (An et al., 2024). However, most work uses final accuracy as the metric to evaluate deductive reasoning.

---

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

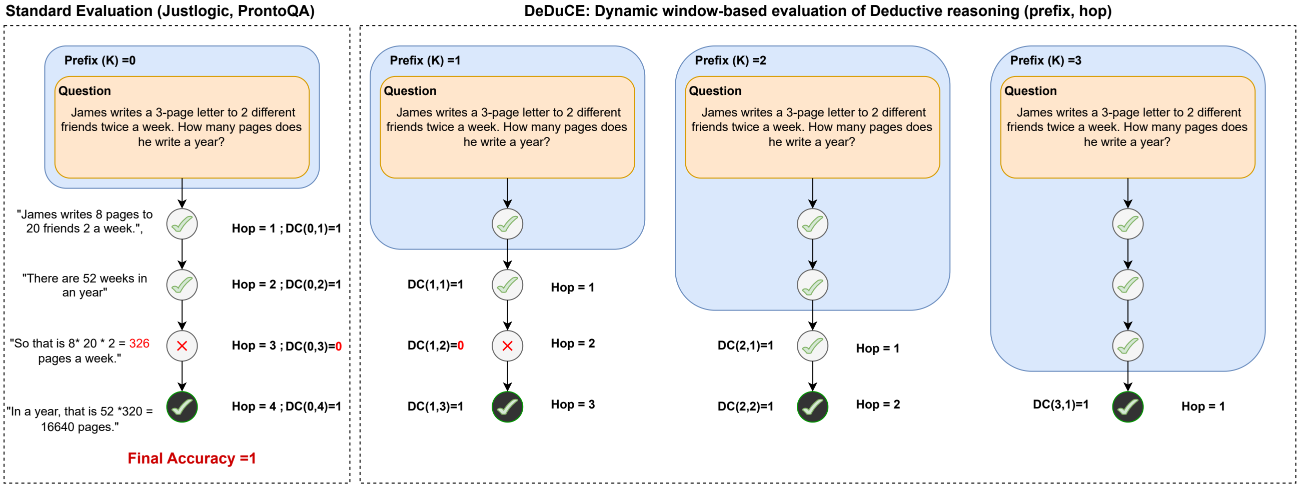

Standard Evaluation (Justlogic, ProntoQA)
DeDuCE: Dynamic window-based evaluation of Deductive reasoning (prefix, hop)
Prefix (K) =0
Question
James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?
"James writes 8 pages to 20 friends 2 a week.",
Hop = 1 ; DC(0,1)=1
"There are 52 weeks in an year"
Hop = 2 ; DC(0,2)=1
"So that is 8* 20 * 2 = 326 pages a week."
Hop = 3 ; DC(0,3)=0
"In a year, that is 52 *320 = 16640 pages."
Hop = 4 ; DC(0,4)=1
Final Accuracy =1
Prefix (K) =1
DC(1,1)=1
Hop = 1
DC(1,2)=0
Hop = 2
DC(1,3)=1
Hop = 3
Prefix (K) =2
DC(2,1)=1
Hop = 1
DC(2,2)=1
Hop = 2
Prefix (K) =3
DC(3,1)=1
Hop = 1

Figure 1: Standard accuracy-based evaluation versus Deductive Consistency that allows evaluation of step-wise computation of deductive conclusions, given a set of input premises (previous steps from a correct reference solution).

2. Defining Deductive Consistency

Given a consistent proof system $\mathcal{S} = \langle \mathcal{L}, \mathcal{R} \rangle$, where $\mathcal{L}$ is the underlying logical language and $\mathcal{R}$ is the set of inference rules, let $P \subset \mathcal{L}$ denote the set of premises and $\text{Th}(P) \subseteq \mathcal{L}$ the deductive closure of P under $\mathcal{R}$. We assume access to a dataset of problems $(P_j, t_j)_{j=1}^{M} \in \mathcal{D}$ where P_j denotes the set of *premises* and t_j the *target predicate* to be proved in each problem j. The total number of problems in the data set is M.

For example, consider a system with the language $\mathcal{L}$ of statements of the form $X \to Y$ and transitivity as a single inference rule, $W \to X, X \to Y \Rightarrow W \to Y$. A sample set of premises may be "$A \to B; B \to C; B \to D; C \to F; D \to F; E \to F; F \to G$", and a target predicate to be proved be "$A \to G$". Assume that a reasoning system $\mathcal{A}$ (e.g., an AI reasoning model) produces the (incorrect) proof, $A \to D; A \to E; E \to G$ to conclude $A \to G$. Beyond final accuracy, we define the Deductive Consistency metric to evaluate the reasoning system's proof steps on such problems. For instance, on a math word problem, we would like to check the consistency of a LM's intermediate steps with correct deductive reasoning.

2.1. Deductive consistency given a complete proof system

Consistency evaluates the extent to which a reasoning system $\mathcal{A}$ agrees with the reference proof system $\mathcal{S}$. For each problem $d \in \mathcal{D}$, where $d = (P, t)$, we generate a proof $\mathcal{A}(P, t)$ using $\mathcal{A}$. A simple way of measuring reasoning performance may be to compute per-predicate accuracy. For each $X_i \in \mathcal{A}(P, t)$: $Cons_d = \frac{\sum_i I_{X_i}}{|\mathcal{A}(P,t)|}$ where I_z is an indicator function, 1 whenever z is correctly inferred by $\mathcal{A}$ and zero otherwise. For each X_i, we use the complete proof system to verify whether it is true or not, given P and the input premises $X_k : \{k : 1, 2, ..i - 1\}$. For the example proof above, the metric will be 0.67 because the second predicate is incorrect, but final accuracy will be 1.

However, in the definition above, if a given step X_j is incorrect, evaluation of the future steps would depend on an incorrect premise. Therefore, we extend the above metric to include longer input premises than in an original problem. We do so by sampling a (correct) proof from the reference proof system and adding the first k steps of the proof to the input premises P. The input premises now become $P' = P \cup \mathcal{R}_k(P, t)$ where $\mathcal{R}_k(P, t)$ is the first k steps of the proof. Then, as k increases, we obtain a measure of how well a reasoning system can handle larger input premises. Let X'_i be the proof steps generated by $\mathcal{A}(P \cup \mathcal{R}_k(P, t), t)$. The reasoning system's goal is to complete the proof.

$$Cons_d(k) = \frac{\sum_{i=k+1}^{N} I_{X'_i}}{|\mathcal{A}(P', t)|} \tag{1}$$

Continuing our example with $k = 1$, the reference proof system may add the first step, "$A \to C$" and let the target system $\mathcal{A}$ complete the rest. Here, the system may produce a faulty proof as before, $A \to C; A \to E; A \to G$. However, with $k = 2$ and adding the first two steps "$A \to C; A \to F$", the system $\mathcal{A}$ may produce $A \to C; A \to F; A \to G$, which is a correct proof.

However, the above metric has a right censoring issue (Gijbels, 2010): this measure of deductive consistency depends trivially on the number of input premises. In general, the difficulty of a proof is associated with the number of inference rules required to complete it. If many premises are already provided, the number of inference rules to reach the target predicate decreases and the problem becomes simpler. As a result, if we see an increase in deductive consistency as the number of reference proof steps are increased (as we see for the example above), it may simply be due to the fewer steps that need to be predicted, rather than due to the reasoning system's improved consistency after access to the reference system's guidance for the first few steps. Therefore, we also introduce a *hops* parameter (l), denoting the number of inference rules (steps) at which we evaluate the reasoning system.

$$DedCons_{d_j}(k,l) = \frac{I_{X'_{k+l,j}}}{M}; \tag{2}$$

$$DedCons(k,l) = \frac{\sum_{d_j \in \mathcal{D}} I_{X'_{k+l,j}}}{M} \tag{3}$$

Compared to final accuracy, a key benefit of the our formulation is that we obtain multiple premises and evaluation sets from a single problem instance. This makes it possible to study the subtasks in deductive reasoning: 1) **Premise understanding:** vary k to see how a reasoning system's consistency changes as the number of input premises increase; 2) **Deductive inference:** vary l to see how the reasoning system's consistency changes as the number of output hops are increased. We define **base** as the deductive consistency at the first hop and **decay** as the negative slope of line that best fits deductive consistency marginalized by prefixes and hops normalized between 0 and 1, $= -\frac{Cov(\mathbb{E}_k[\text{DedCons}(k,\ell)], \frac{\ell}{\ell_{\max}})}{Var(\frac{\ell}{\ell_{\max}})}$ to refer to this change as k or l are increased. For example, decay wrt. the output hops is defined as,

$$\mu = \frac{1}{\ell_{\max}+1} \sum_{\ell=0}^{\ell_{\max}} \mathbb{E}_k\left[\text{DedCons}(k,\ell)\right] \tag{4}$$

$$\gamma = -\frac{12\ell_{\max}}{(\ell_{\max}+2)} \tag{5}$$

$$\text{Decay} = \gamma \sum_{\ell=0}^{\ell_{\max}} \left(\frac{\ell}{\ell_{\max}} - \frac{1}{2}\right) \left(\mathbb{E}_k\left[\text{DedCons}(k,\ell)\right] - \mu\right) \tag{6}$$

where $\ell_{\max}$ represents maximum number of hops in dataset. **An ideal reasoning system's deductive consistency should be a constant close to 1 (thus zero decay), independent of the number of premises k and the number of hops l.**

2.2. Deductive Consistency given a reference proof

While the above metric works for a *complete* proof system as the reference, in practice it is more common to have access to a limited reference system that can only generate a single proof $\mathcal{S}(P,t)$ given a problem. Therefore, we now assume access to a reference proof system that given a set of premises P and a target predicate t, can generate a proof involving predicates $\mathcal{S}(P,t) \subseteq \text{Th}(P)$, representing the predicates within the closure that were proved while proving the main result for t. We call such a proof as the *reference solution*. Continuing the transitivity example, it would mean that we only have access to a reference proof solution, $A \to C; A \to F; A \to G$, but cannot assess the validity of a predicate outside it such as $A \to D$.

Given a reasoning system's proof $\mathcal{A}(P,t)$, now we can only verify the predicates that are also present in $\mathcal{S}(P,t)$. We therefore orient the deductive consistency metric to focus on the verifiable predicates that are also proved by $\mathcal{A}$. For each $Z_i \in \mathcal{S}(P,t)$

$$\text{DedCons}^{single}(k,l) = \frac{\sum_{d_j \in \mathcal{D}} I_{Z_{k+l,j} \in \mathcal{A}(P'_j,t_j)}}{M} \tag{7}$$

where the numerator is an indicator function checking whether a given predicate $Z_i \in \mathcal{S}(P,t)$ is also included in the proof by $\mathcal{A}$. Note that the above metric introduces a bias because the reasoning system $\mathcal{A}$ may generate (true) predicates that are not in the reference solution (there can be multiple ways to solve the same problem). For instance, even if a reasoning system produces a valid proof, $A \to D; A \to F; A \to G$, its $\text{DedCons}^{single}(0,1)$ will be zero because the first step $A \to C$ from reference solution $\mathcal{S}(P,t)$ is not a part of the system's proof $\mathcal{A}$.

In such cases, the consistency metric above can underestimate the deductive consistency–the reasoning system may be penalized for a producing a valid solution because its steps are different than that of the reference system. Hence, we also introduce a metric for coverage. The **Coverage** metric is defined as the expected proportion of variables in $\mathcal{S}(P,t)$ inferred by $\mathcal{A}$. Let $V_{\mathcal{S}(P,t)}$ be the variables included in the reference solution. Then coverage is

$$\text{Coverage} = \frac{\sum_{d_j \in \mathcal{D}} \left| V_{\mathcal{S}(P_j,t)} \cap V_{\mathcal{A}(P_j,t)} \right|}{\sum_{d_j \in \mathcal{D}} \left| V_{\mathcal{S}(P_j,t)} \right|}. \tag{8}$$

Thus, coverage measures how verification of the set of predicates in the reference solution measures deductive consistency. Interpretation of $DedCons$ is valid only when coverage is high.

3. Evaluating Deductive Consistency of LLMs

To evaluate deductive consistency, we require at least one reference solution for each reasoning problem. Given a benchmark reasoning dataset, we propose a method to obtain such solutions. We call the model under evaluation as **SubjectLM**. It produces a chain-of-thought (CoT) solution for a question. Each question is defined by an initial set of premises Q and a target conclusion T. While standard evaluations assess accuracy only at the final step T, we evaluate reasoning quality at each intermediate step via deductive consistency. We assume that a reference solution $tCode$ is available which can be used to verify correctness of intermediate steps.

3.1. Evaluating Deductive Consistency: A Correct Solution is Known

In order to compute deductive consistency we generate windowed instances of the same problem to test SubjectLM on. We increase the window of prefix and compute value of variables at every subsequent hop using the computation graph of $tCode$ as illustrated in Figure 1. We provide the SubjectLM with the question and prefix and let it generate solution. To implement Equation 5, we check for the correctness of variables at every subsequent hop ℓ with respect to $tCode$ given prefix k. A **Variable Extraction LM** (Parser) is used to extract predicate values from natural language chain of thought, if predicate is present in the CoT for evaluation. We average these values over the entire dataset to obtain $DedCons(k, l)$ for each prefix k and hop l pair.

3.2. Evaluating Deductive Consistency: Realworld dataset requires Novel Problem Generation

The above pipeline works when a single correct solution can be obtained (e.g., from the dataset construction source code). However, for datasets sourced from real-world scenarios, that may be difficult. In addition, in popular real-world datasets such as GSM-8k, there are significant memorization concerns. To alleviate both these problems, we develop a pipeline that constructs novel versions of the benchmark problem, and uses an auxiliary LM on the original problem to obtain a reference solution. Partly, *we are exploiting LMs' memorization abilities to increase the chances that the auxiliary LM returns a correct solution.*

Obtaining Correct Solution. To generate a correct solution for a reasoning problem, we adopt a **generate-then-verify** approach. First, an expert language model produces a candidate solution in two semantically equivalent forms: a templatized chain-of-thought ($tCoT$) and an executable code snippet ($tCode$). Generating the $tCoT$ comes before code generation, ensuring that the reasoning steps are clearly outlined. Next, we perform an **internal consistency check** between these two representations. We verify that, for any given factual input, both the $tCode$ and the $tCoT$ yield identical intermediate variable values at each step. Only when this equivalence is confirmed do we accept the code as a reliable, symbolic encoding of both the problem and its correct solution.

Generating Novel Problem. For generating a novel problem, we adopt a simple perturbation: changing the values of variables in the problem statement. Other perturbations, such as changing variable names and adding irrelevant info (Mirzadeh et al., 2024) can be easily added. This is done by first generating the correct code for a novel problem and then generating the text for the problem. To create a new problem, we **sample seed premises**: random integers under 100 (avoiding large numbers and floating-point values). These sampled inputs are fed into $tCode$ to compute intermediate results (val). Then, we generate a **novel problem** by reinserting val back into the templated question (tQ), producing a new question Q', and into the $tCoT$, producing a fresh chain-of-thought ground-truth CoT'. This process lets us confirm that the method consistently generates correct solutions across different inputs.

To summarize, we use auxiliary expert LMs to help with transformation tasks. Specifically, we use **Code Generation LM** that generates executable reasoning graphs ($tCode$) from the solution. **Templatization LM** that creates templatized version of Question (tQ) and the Chain of Thought ($tCoT$). The entire pipeline is shown in Figure 4.

4. Results

4.1. Datasets

GSM8K A subset of 1000 problems from GSM8K is randomly chosen. The responses of LMs under evaluation are filtered as described in section 3. Problem instances common across the models are collected and used as final dataset that will be used to evaluate these models. This consists of 165 problem instances. For each instance we generate n(=10) counterfactual samples.

SynDeduct is our custom synthetic data-generation pipeline. It allows generation of questions with large number of steps with fine-grained control, which is not available in GSM8k. For example, a question can be orchestrated to move from the initial premise (A) to an intermediate node I (“prefix k”) and then from node I to the target node B (“hop l”), such that Distance(A, B) = k+l. This level of control over the graph's structure proves highly valuable for generating datasets with specific properties and systematically assessing deductive performance. Additional generation steps are present in Appendix (subsection A.6 and subsection A.10)

ProntoQA This is a synthetic dataset introduced to evaluate language models on logical reasoning tasks such as use of Modus Ponens in Natural Language. We generate dataset of upto 5 Hops and 100 question per hop for each of the three ontologies - True, False and Fictional. Statements in CoT are converted into Z3 (de Moura & Bjørner, 2008) FOL.

4.2. Setup

LMs in DeduCE pipeline. We use LLama-3-70B-Instruct as the Templatizer, code generation and variable extractor LLM. We obtain a failure rate (unable to generate json) close to 30%, which we filter out. The additional sanity checks ensures that we have high quality dataset for evaluation. We use Llama-3.1-8B-Instruct as Parser LLM. For ProntoQA, each predicate in the ground truth CoT and response CoT are verified for consistency using Z3 solver. GPT-4o is used for error analysis.

Models under evaluation. We evaluate the following LMs: Phi-3.5-mini-instruct, Phi-4, Qwen2.5-Math-7B-Instruct, Qwen2.5-Math-72B-Instruct, Llama-3.3-70B-Instruct, Llama-3-8B-Instruct and their R1 Distill variants. *All models are Instruct tuned. Model suffixes will be truncated in plots*. More details can be found in Figure 4. All the experiments were done on a Nvidia 4xA100 80GB GPU compute.

4.3. Results: Math reasoning on GSM8K

Premise understanding v/s deductive inference. Mean deductive consistency is computed by averaging predicate consistency across prefixes for a given hop. Only hops where the ratio of single-premise samples to those with premise length of 1 hop-1 exceeds 20% is included, ensuring sufficient data for reliable estimates.

Deductive consistency as a function of hops on the original benchmark achieves a constant value of 1 across all models. When deductive consistency is computed on perturbed problem, it is significantly lower (see Figure 2). This indicates **Memorization Effect** on the benchmark.

The key result is decay in deductive consistency as hops increase, which was masked due to memorization of the original benchmark. An ideal model must achieve zero decay and a base value of one.

Figure 2 reveals that larger models (Qwen-Math-72B-Instruct, and Llama-3.3-70B-Instruct), models trained on synthetic data (Phi-4) as well as math-specific models (Qwen-2.5-Math-72B-Instruct, and Qwen-2.5-Math-72B-Instruct) do achieve greater base values. However, even these models show significant decay in the deductive consistency as the number of hops increases. Smaller models like Llama-3-8B-Instruct and Phi-3.5-mini-instruct perform poorly with lower base values and Llama-3-8B-Instruct exhibits a high decay value compared to other models (also see Figure 2).

Importantly, deductive consistency does not vary much as the length of input premises are changed. A lower variance in mean deductive consistency as a function of prefix as seen in Figure 8 . Which implies a partial evidence of hypothesis: it depends on hops, but not on the premises. A caveat is that due to the simplicity of the GSM8K problems, the maximum premise length we could evaluate on is 7.

Other hypotheses: Based on the deductive consistency metric, we can answer some hypotheses on why final accuracy decreases for novel math problems, as reported in past work (Mirzadeh et al., 2024; Srivastava et al., 2024). We also test the hypothesis on the effects of language style on reasoning (Han et al., 2024).

H1: Novel problems induce early errors in the math computation, which propagate to lead to an incorrect solution. **H2**: Novel problems induce a style change in the CoT answers, which may lead to faulty reasoning and hence incorrect final answer.

H1: *Our findings reveal that deductive consistency remains high for the first hop, contradicting hypothesis by demonstrating that models correctly answer the first step*. While novel problems do not induce early errors, we observe frequent computational errors in model responses. These errors propagate, providing evidence for part of hypothesis. Detailed error analysis, is in Appendix A.4.

H2: To decide on this hypothesis, we create four kinds of premises that can be added. The first is the original (*eg: Yasna has 60 + 12 = 72 pages to read.*), sourced from subject LMs answers on the original benchmark problem. Others are different paraphrases of the original style. They are explained in Appendix A.2. A slight decrease in base values across models due to impact of language style. While it is expected that the original benchmark's style should have highest accuracy, the variation across paraphrases is not high. Even though, on performing t-test, p-values values were significant (at 0.05 significance level), Cohens' effects sizes were too small to consider (<0.1). We find weak evidence for **H2** referring to Figure 5 in Appendix.

Error Analysis. A higher proportion of calculation errors is observed relative to other errors. These calculation errors predominantly emerge during arithmetic operations within the chain-of-thought, and they propagate through subsequent reasoning steps. Refer to Appendix A.4 for details.

4.4. Results: Evaluation on SynDeduct

To validate the conclusions from GSM8K, we now evaluate deductive consistency on a synthetic dataset. In particular, the problems are designed such that the solutions involve a large number of hops spread across prefixes.

The general trend of decreasing deductive consistency over hops supports our results in GSM8K. Our findings in Figure 3 suggests larger models demonstrate greater resilience to increases in the number of hops, while smaller models—such as Llama-3-8B-Instruct—experience a substan-

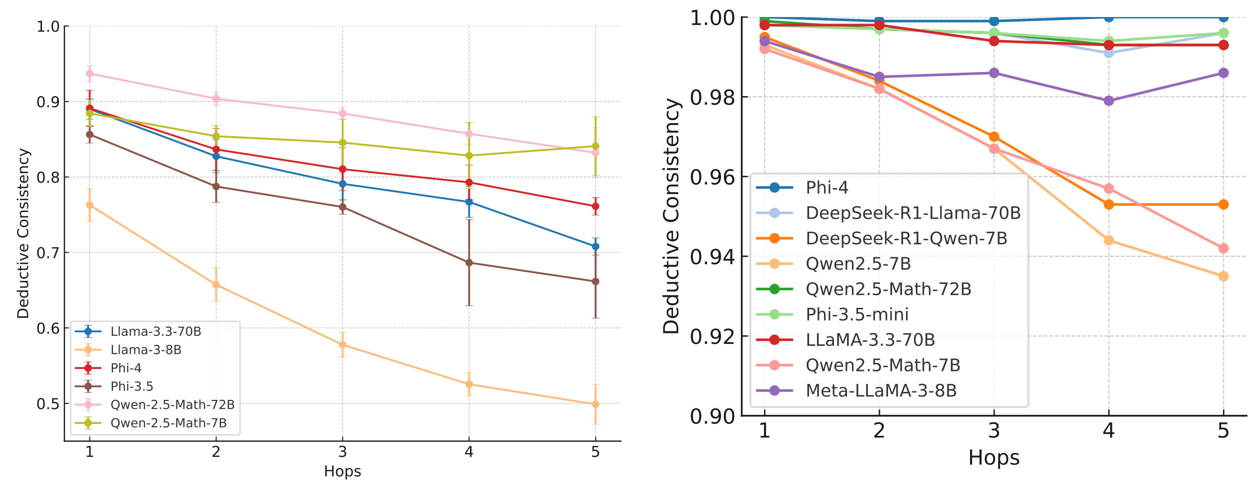


Figure 2: **Left:** Deductive Consistency vs. Reasoning Hops across models on GSM8K-Mutated. **Right:** Deductive Consistency vs. Reasoning Hops across models on ProntoQA.

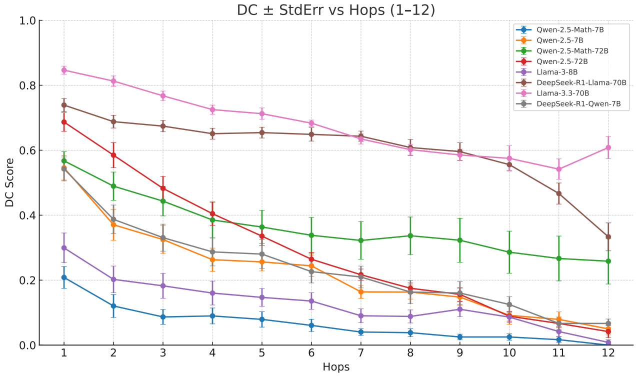


Figure 3: Deductive Consistency v/s Hops. Each Hop bin has roughly same variation of Prefixes. Y-Ticks on each data-point is Standard Error for Accuracy. For Coverage refer to Appendix Figure 12.

tial drop in performance. This trend becomes evident when examining accuracy versus prefix length across multiple hops: as prefix segments grow, the overall task accuracy declines. One contributing factor may be the larger maximum token limit, which allows these models to analyze and accommodate more extensive reasoning chains. However, barring Llama-3-8B all models have context lengths greater than maximum number of tokens required for answering the question properly (around max_length = 10k tokens).

Error Analysis: Similar to GSM8K, a higher proportion of calculation errors is observed. Interestingly, a large proportion of Garbled errors were also seen. For more details, see subsection A.7

4.5. Results: Evaluation on ProntoQA

We evaluate models on ProntoQA—a benchmark specifically designed for multi-hop factual reasoning. Figure 2(right) shows a consistent decay with respect to hops. Phi-4 performs exceptionally well with negligible decay w.r.t prefixes; plausibly due to being trained on similar synthetic data. The pattern of decay on ProntoQA is similar to that in the prior sections. For further details, refer to subsection A.8.

5. Do Prevalent Techniques Mitigate Decay ?

To better understand the impact of different popular strategies to improve deductive reasoning, we focus on two primary fine-tuning approaches: iterative fine-tuning with reinforcement learning (Ziegler et al., 2019), supervised fine-tuning (SFT) (Ouyang et al., 2022) and a calculator tool-calling approach (Schick et al., 2023).

For RL-based and iterative fine-tuning models, Qwen-2.5-Math-Instruct in both its 7B and 72B variants (Yang et al., 2024) is compared against their respective base models. Similarly, for SFT-based tuning, R1 distilled models are compared to their base counterparts (DeepSeek-AI et al., 2025), namely DeepSeek-R1-Distill-Llama-70B and DeepSeek-R1-Distill-Qwen-7B.

To improve on decay and reduce the calculation errors, an inference-time version of Calculator tool-calling approach is used. Meta llama-8B model is used as an Online Monitor that is a Reviewer and Editor. As generation of response happens step by step, a reviewer LLM monitors and checks if a calculation trace is present. It calls a calculator tool with appropriate inputs and the output from the tool is passed to editor LLM. The editor LLM changes the calculation trace with the correct one. The generation then continues after the step has been corrected for calculation errors.

Results. RL fine tuning is more effective (Table 1, Table 3) in reducing the decay of deductive consistency. RL fine-tuning shows minimal change in base deductive consistency on in-distribution datasets, and a reduction on unseen datasets such as SynDeduct. SFT after training causes a decrease in deductive consistency and worsens decay for both data sets (Table 2, Table 4). More work is required to study the extent of generalization that such post-training methods provide. In general, these findings emphasize that neither of the two post-training techniques is successful in improving deductive consistency between models and datasets.

For the tool-calling approach, we find (Table 5, subsection A.5) an improvement in base deductive consistency of up to 15% but decay did not improve. Further details are in the Appendix.

Table 1: RL post training on SynDeduct.

Model	**Base**	**Decay**
Qwen-2.5-72B-Instruct	0.6868	0.0602
Qwen-2.5-Math-72B-Instruct	0.5674	0.0273
Qwen-2.5-7B-Instruct	0.5458	0.0432
Qwen-2.5-Math-7B-Instruct	0.2083	0.0211

Table 2: SFT post training on SynDeduct.

Model	**Base**	**Decay**
Qwen-2.5-Math-7B-Instruct	0.2083	0.0211
DeepSeek-R1-Distill-Qwen-7B	0.5424	0.0381
Llama-3.3-70B-Instruct	0.8465	0.0212
DeepSeek-R1-Distill-Llama-70B	0.7389	0.0314

Table 3: RL post training on GSM8k.

Model	**Base**	**Decay**
Qwen-2.5-72B-Instruct	0.9149	0.2339
Qwen-2.5-Math-72B-Instruct	0.9164	0.1725
Qwen-2.5-7B-Instruct	0.8881	0.1618
Qwen-2.5-Math-7B-Instruct	0.8427	0.1189

Table 4: SFT post training on GSM8k.

Model	**Base**	**Decay**
Qwen-2.5-Math-7B-Instruct	0.8509	0.0613
DeepSeek-R1-Distill-Qwen-7B	0.8468	0.1957
Llama-3.3-70B-Instruct	0.8532	0.20065
DeepSeek-R1-Distill-Llama-70B	0.8366	0.24895

Table 5: Base and Decay values for LLaMA-3-8B-Instruct with and without Monitor (calculator as a tool at inference time).

LLaMA-3-8B-Instruct	**Base**	**Decay**
Without Monitor	0.7629	0.3300
With Monitor	0.9081	0.3704

6. Conclusion

We introduce a metric for assessing AI deductive reasoning based solely on text-based reasoning traces, ensuring applicability across diverse domains. Through experiments on both synthetic datasets (SynDeduct, ProntoQA) and a real-world benchmark (GSM8K), we demonstrate its impact and expose critical trade-offs. Additionally, we evaluate common reasoning-enhancement techniques designed to address the challenges identified in our study.

Limitations & Impact

Our approach depends on auxiliary language models for parsing, error analysis, and data-pipelining. Although we have taken extensive measures to mitigate errors introduced by these models, residual inaccuracies may persist. Moreover, our evaluation of deductive reasoning is limited to linear reasoning traces; extending our framework to non-linear inference chains will be an important objective for future work.

This paper presents work whose goal is to advance the reliability of AI reasoning. We expect that insights from our work can be used to improve reliability of AI reasoning, leading to a **positive societal impact** on downstream applications by avoiding reasoning bugs.

References

An, C., Chen, Z., Ye, Q., First, E., Peng, L., Zhang, J., Wang, Z., Lerner, S., and Shang, J. Learn from failure: Fine-tuning LLMs with trial-and-error data for intuitionistic propositional logic proving. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Chen, M. K., Zhang, X., and Tao, D. Justlogic: A comprehensive benchmark for evaluating deductive reasoning in large language models. 2025. URL `https://doi.org/10.48550/arXiv.2501.14851`.

de Moura, L. and Bjørner, N. Z3: An efficient smt solver. In Ramakrishnan, C. R. and Rehof, J. (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78800-3.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

Gijbels, I. Censored data. *WIREs Computational Statistics*, 2(2):178–188, 2010.

Han, Y., Xu, L., Chen, S., Zou, D., and Lu, C. Beyond surface structure: A causal assessment of llms' comprehension ability, 2024. URL `https://arxiv.org/abs/2411.19456`.

Lewis, M. and Mitchell, M. Evaluating the robustness of analogical reasoning in large language models. *arXiv preprint arXiv:2411.14215*, 2024a.

Lewis, M. and Mitchell, M. Using counterfactual tasks to evaluate the generality of analogical reasoning in large language models. *arXiv preprint arXiv:2402.08955*, 2024b.

Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL `https://arxiv.org/abs/2410.05229`.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Parmar, M., Patel, N., Varshney, N., Nakamura, M., Luo, M., Mashetty, S., Mitra, A., and Baral, C. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Paul, D., West, R., Bosselut, A., and Faltings, B. Making reasoning matter: Measuring and improving faithfulness of chain-of-thought reasoning. In Al-Onaizan, Y., Bansal, M., and Chen, Y. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, 2024.

Sanyal, S., Liao, Z., and Ren, X. RobustLR: A diagnostic benchmark for evaluating logical robustness of deductive reasoners. In Goldberg, Y., Kozareva, Z., and Zhang, Y. (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *International Conference on Learning Representations (ICLR)*, 2023.

Seals, S. and Shalin, V. Evaluating the deductive competence of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8606–8622, 2024.

Singh, J., Nambi, A., and Vineet, V. Exposing the achilles' heel: Evaluating llms ability to handle mistakes in mathematical reasoning, 2024. URL `https://arxiv.org/abs/2406.10834`.

Srivastava, S., PV, A., Menon, S., Sukumar, A., Philipose, A., Prince, S., Thomas, S., et al. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *arXiv preprint arXiv:2402.19450*, 2024.

Wu, Z., Qiu, L., Ross, A., Akyürek, E., Chen, B., Wang, B., Kim, N., Andreas, J., and Kim, Y. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1819–1862, 2024.

Xu, F., Lin, Q., Han, J., Zhao, T., Liu, J., and Cambria, E. Are large language models really good logical reasoners? a comprehensive evaluation and beyond, 2024. URL `https://arxiv.org/abs/2306.09841`.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Zeng, Z., Chen, P., Liu, S., Jiang, H., and Jia, J. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation, 2024. URL `https://arxiv.org/abs/2312.17080`.

Zhang, Z., Chen, J., and Yang, D. Darg: Dynamic evaluation of large language models via adaptive reasoning graph, 2024.

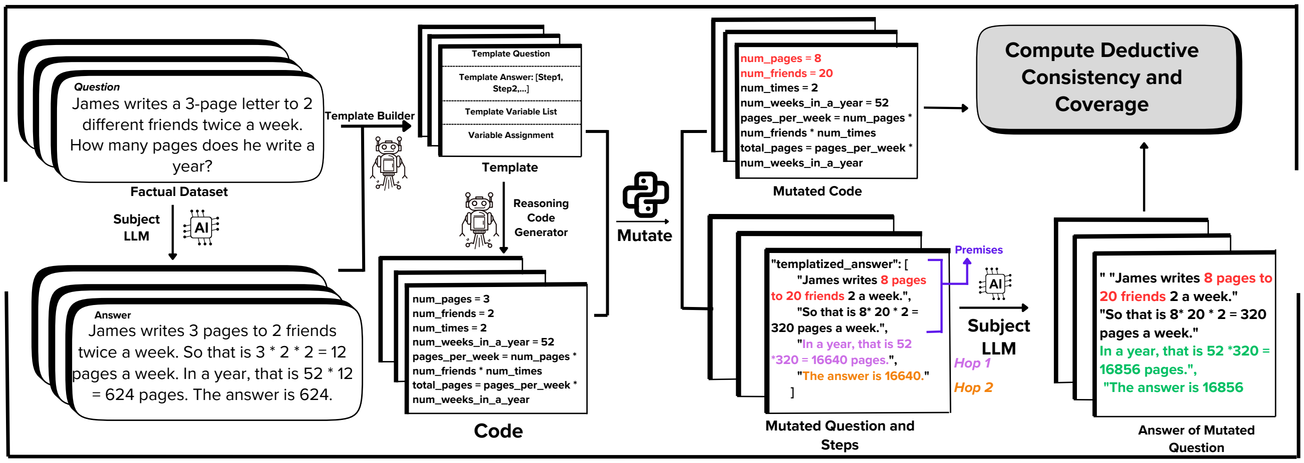
Zhu, K., Chen, J., Wang, J., Gong, N. Z., Yang, D., and Xie, X. Dyval: Dynamic evaluation of large language models for reasoning tasks. In *The Twelfth International Conference on Learning Representations*, 2023.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., and Christiano, P. F. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Zong, S. and Lin, J. Categorical syllogisms revisited: A review of the logical reasoning abilities of LLMs for analyzing categorical syllogisms. In Peled-Cohen, L., Calderon, N., Lissak, S., and Reichart, R. (eds.), *Proceedings of the 1st Workshop on NLP for Science (NLP4Science)*, 2024.

A. Appendix

A.1. Details for GSM8K pipeline


Question
James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?
Factual Dataset
Subject LLM
AI
Answer
James writes 3 pages to 2 friends twice a week. So that is 3 * 2 * 2 = 12 pages a week. In a year, that is 52 * 12 = 624 pages. The answer is 624.
Template Builder
Template Question
Template Answer: [Step1, Step2,...]
Template Variable List
Variable Assignment
Template
Reasoning Code Generator
Code
Mutate
Mutated Code
Compute Deductive Consistency and Coverage
Premises
AI
Subject LLM
Hop 1
Hop 2
Mutated Question and Steps
Answer of Mutated Question

```
num_pages = 3
num_friends = 2
num_times = 2
num_weeks_in_a_year = 52
pages_per_week = num_pages *
num_friends * num_times
total_pages = pages_per_week *
num_weeks_in_a_year
```

```
num_pages = 8
num_friends = 20
num_times = 2
num_weeks_in_a_year = 52
pages_per_week = num_pages *
num_friends * num_times
total_pages = pages_per_week *
num_weeks_in_a_year
```

```
"templatized_answer": [
    "James writes 8 pages
to 20 friends 2 a week.",
    "So that is 8* 20 * 2 =
320 pages a week.",
    "In a year, that is 52
*320 = 16640 pages.",
    "The answer is 16640."
  ]
```

```
" "James writes 8 pages to
20 friends 2 a week."
"So that is 8* 20 * 2 = 320
pages a week."
In a year, that is 52 *320 =
16856 pages.",
 "The answer is 16856
```

Figure 4: Given a benchmark problem, we use pre-trained LMs to templatize its solution and obtain an executable code solution. Then we update the variables' values in the code and generate a novel problem on which the subject LM is evaluated. For any (number of premises, number of hops) combination, we assess whether the subject LM's solution contains the correct values of the variables.

Inference on original dataset We sample a subset of GSM8K of size 1000. We prompt the LM under investigation to solve the question using the prompt template provided in subsection A.9.

Templatization and Code Generation We templatize the question and LM CoT response using Llama-3-70B as Template Builder Agent. The model is prompted (as shown in subsection A.9) to generate templatized question, templatized CoT answer (as well as chunk it into steps), explanation of variables of templates along with assignment of variables in question.

Sanity Checks We check that the code produced is an executable code, if the format of template generate is consistent with our reference template format, if the all variables in factual_assignment are present in code. The generated code is executed with factual assignment as inputs for variables in question template and the value of other variables in code are checked to be consistent with the factual assignment in template. Further we have check if the final answer in response matches the ground truth answer in original dataset. If any of these checks fail then we remove that question from pipeline. For each model we now have a reduces set of questions that has passed sanity checks. We take intersection of such questions over multiple models to get a dataset on which we can evaluate all the models under consideration. This support set depends on the set of models being used in the experiments.

Mutation Details We create mutated dataset by sampling the values of variables in question and executing the code with these newly sampled values to obtain assignment corresponding to other variables. Parameters for the sampler are (min-value,max-value,max-iter). If the factual assignment of a variable is integer, we sample from integers in the range (min-value,max-value), if factual assignment of a variable is decimal between 0 and 1, we uniformly sample from this range, else if it is any other decimal we sample a float from (min-valu,max-value). We try to make sure that all the variable assignments after positive. If not we rerun until we get a all positive assignment or we reach maximum iterations of the sampler. We substitute these values into template question and template CoT answer.We sample 10 mutated questions per question in original dataset. We create dataset with mutated question and varying length of mutated CoT answer present in LMs context. The number of steps from mutated template CoT answer is defined as prefix length. We collect the sampled variable assignments,mutated Question and Prefix into the mutated dataset.

Inference on mutated dataset We run inference of LMs on this mutated dataset. Since all LMs we evaluate are Instruction tuned, we use chat template. Mutated question is passed as user-content where as prefix is passed as assistant-content. We remove the <||eot|| >token and let the generation continue as if the model were completing the generation.

Computation of Deductive consistency The response of the model to mutated dataset is passed into a variable extraction LM which extracts value if the variables under consideration (ones in template) if present in response(see subsection A.9).

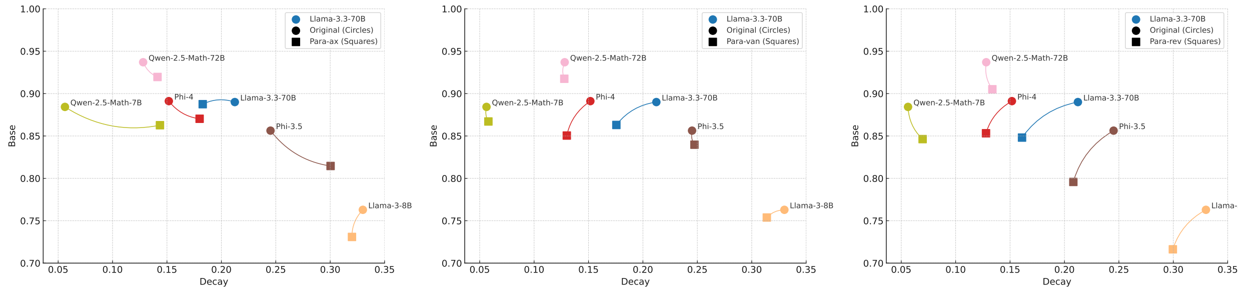


Figure 5: Comparison of Base vs. Decay Across Different Styles. Model name suffixes have been dropped for readability.

Hops are decided by the relative positions of variables under consideration in the template CoT. We check if these extracted values are consistent with code-generated values. We collect this data for every variable across mutated question and then group it by prefix and hop. We filter instances where the responses gives From this data structure we derive Deductive Consistency as function of hops or prefixes (as required).

Base DC and Decay We define Base deductive consistency as the deductive consistency at one-hop as well as decay is defined as negative slope of line fitting deductive consistency and hops (normalized between 0 and 1) [as shown in Equation 6] .

Coverage Table 7 shows that the coverage is high across all premises. Given a LM, this implies that the intermediate variables inferred in the solution for the novel problem are almost the same as the variables inferred in the solution for the original benchmark problem. Therefore the code obtained from reasoning code generator is reliable as a reference proof. So we can go ahead with interpreting the consistency results.

A.2. Para Phrasing Styles

- Vanilla (Para-van): In this approach, the text is rephrased using conventional linguistic variations without incorporating any specialized semantic constraints. For example: *Yasna's task involves 60 pages plus an additional 12 pages, which totals to 72 pages*
- Axiomatic (Para-ax): This method reformulates the statement as a set of axioms that articulate the underlying numerical relationships. An example is : *Axiom-1 (Addition): Given two numerical values x and y, the operation ADD(x, y) yields their arithmetic sum, thus ADD(60, 12) yields 72, which represents the total number of pages Yasna has to read.*
- Reverse (Para-rev): the sentence is restructured by inverting the typical cause-effect relationship—presenting the effect before providing the explanation for its cause. For instance, in the example: *Yasna has to read the 72 pages, which is the sum of 60 and 12.*

A.3. Results - GSM8K

Here we report the full results presented in the main paper.

Table 6: Model Performance and Deductive Consistency Across Hops GSM8K

Models	Hop-1	Hop-2	Hop-3	Hop-4	Hop-5
Llama-3.3-70B	0.89 ± 0.0135	0.8274 ± 0.0219	0.7909 ± 0.0215	0.7669 ± 0.0206	0.7079 ± 0.0114
Llama-3-8B	0.7629 ± 0.0217	0.6572 ± 0.0227	0.5777 ± 0.0165	0.5254 ± 0.0154	0.4988 ± 0.0264
Phi-4	0.8911 ± 0.0238	0.8365 ± 0.0278	0.8103 ± 0.0285	0.7929 ± 0.0228	0.7612 ± 0.0117
Phi-3.5	0.8563 ± 0.0114	0.7874 ± 0.0210	0.7602 ± 0.0096	0.6865 ± 0.0571	0.6616 ± 0.0488
Qwen-2.5-Math-72B	0.937 ± 0.0108	0.9037 ± 0.0085	0.8841 ± 0.0073	0.8573 ± 0.0148	0.8321 ± 0.0328
Qwen-2.5-Math-7B	0.8843 ± 0.0179	0.854 ± 0.0144	0.8456 ± 0.0307	0.8283 ± 0.044	0.8409 ± 0.039

Table 7: Coverage across different language styles for the premises.

Models	Original	Para-ax	Para-van	Para-rev
Llama-3.3-70B	0.9725	0.9639	0.9616	0.966
Llama-3-8B	0.9669	0.9431	0.9543	0.9486
Phi-4	0.9849	0.9694	0.9759	0.9747
Phi-3.5	0.9684	0.9523	0.9649	0.9624
Qwen-2.5-Math-72B	0.9888	0.9745	0.9862	0.985
Qwen-2.5-Math-7B	0.9701	0.9442	0.9656	0.9648

Table 8: Para-ax: Model Performance and Deductive Consistency Across Hops

Models	Hop-1	Hop-2	Hop-3	Hop-4	Hop-5
Llama-3.3-70B	0.8875 ± 0.0158	0.8083 ± 0.0271	0.7653 ± 0.0282	0.752 ± 0.0189	0.7328 ± 0.004
Llama-3-8B	0.7309 ± 0.0244	0.6177 ± 0.034	0.5096 ± 0.0305	0.4799 ± 0.0328	0.4798 ± 0.0173
Phi-4	0.8703 ± 0.0178	0.8245 ± 0.0177	0.8055 ± 0.0172	0.7525 ± 0.0275	0.7263 ± 0.0164
Phi-3.5	0.8146 ± 0.027	0.694 ± 0.0279	0.6106 ± 0.0426	0.5795 ± 0.0658	0.5715 ± 0.0424
Qwen-2.5-Math-72B	0.9196 ± 0.0143	0.851 ± 0.0168	0.8149 ± 0.0194	0.8038 ± 0.0154	0.8019 ± 0.026
Qwen-2.5-Math-7B	0.8627 ± 0.0192	0.8001 ± 0.0233	0.7672 ± 0.0216	0.7418 ± 0.0165	0.7483 ± 0.0148

Table 9: Para-van: Model Performance and Deductive Consistency Across Hops

Models	Hop-1	Hop-2	Hop-3	Hop-4	Hop-5
Llama-3.3-70B	0.863 ± 0.0276	0.79 ± 0.0318	0.7561 ± 0.0277	0.7242 ± 0.0323	0.7203 ± 0.0103
Llama-3-8B	0.7538 ± 0.0212	0.612 ± 0.0352	0.5503 ± 0.0207	0.4997 ± 0.0348	0.4961 ± 0.0275
Phi-4	0.8505 ± 0.0248	0.8062 ± 0.0271	0.7856 ± 0.0273	0.7655 ± 0.0356	0.7408 ± 0.0153
Phi-3.5	0.8397 ± 0.0189	0.739 ± 0.0203	0.7178 ± 0.0132	0.6679 ± 0.0366	0.6279 ± 0.0355
Qwen-2.5-Math-72B	0.9175 ± 0.0144	0.8758 ± 0.0143	0.8569 ± 0.0138	0.8427 ± 0.0182	0.8063 ± 0.0284
Qwen-2.5-Math-7B	0.8671 ± 0.0207	0.8206 ± 0.0192	0.7841 ± 0.0271	0.7854 ± 0.0246	0.8267 ± 0.0299

Table 10: Para-rev: Model Performance and Deductive Consistency Across Hops

Models	Hop-1	Hop-2	Hop-3	Hop-4	Hop-5
Llama-3.3-70B	0.8481 ± 0.0222	0.7828 ± 0.0306	0.7508 ± 0.0313	0.7148 ± 0.0334	0.7211 ± 0.0097
Llama-3-8B	0.7163 ± 0.0288	0.5509 ± 0.0504	0.4612 ± 0.0591	0.4609 ± 0.0371	0.4617 ± 0.025
Phi-4	0.8532 ± 0.0282	0.8102 ± 0.0289	0.7874 ± 0.0326	0.7724 ± 0.025	0.7442 ± 0.0122
Phi-3.5	0.7958 ± 0.0241	0.7308 ± 0.024	0.6917 ± 0.0165	0.6351 ± 0.0483	0.6355 ± 0.0431
Qwen-2.5-Math-72B	0.9051 ± 0.017	0.8636 ± 0.0166	0.8279 ± 0.0172	0.8105 ± 0.0176	0.7979 ± 0.0258
Qwen-2.5-Math-7B	0.8463 ± 0.0219	0.7914 ± 0.0227	0.7298 ± 0.0364	0.7543 ± 0.0124	0.7952 ± 0.0201

Table 11: Decay and Base Values Across Interventions

Models	Original		Para-ax		Para-van		Para-rev	
	Decay	Base	Decay	Base	Decay	Base	Decay	Base
Llama-3.3-70B	0.21235	0.89	0.18285	0.8875	0.1756	0.863	0.161	0.8481
Llama-3-8B	0.33	0.7629	0.32	0.7309	0.31385	0.7538	0.2996	0.7163
Phi-4	0.1517	0.8911	0.18	0.8703	0.13005	0.8505	0.1279	0.8532
Phi-3.5	0.24515	0.8563	0.30035	0.8146	0.24735	0.8397	0.20815	0.7958
Qwen-2.5-Math-72B	0.1281	0.937	0.1413	0.9196	0.12775	0.9175	0.13375	0.9051
Qwen-2.5-Math-7B	0.05625	0.8843	0.14355	0.8627	0.058	0.8671	0.06965	0.8463

Table 12: Deductive Consistency ± Standard Error vs. Prefix Length for Different Models

Model	**Prefix 1**	**Prefix 2**	**Prefix 3**	**Prefix 4**	**Prefix 5**
Phi-3.5	0.7664 ± 0.0304	0.7772 ± 0.0255	0.7767 ± 0.0224	0.8153 ± 0.0247	0.7707 ± 0.0215
Qwen-2.5-Math-7B	0.8549 ± 0.0268	0.8366 ± 0.0195	0.8510 ± 0.0243	0.8641 ± 0.0329	0.8922 ± 0.0383
Qwen-2.5-Math	0.8802 ± 0.0200	0.8704 ± 0.0223	0.8803 ± 0.0166	0.9067 ± 0.0117	0.8967 ± 0.0161
Llama-3-8B	0.5884 ± 0.0580	0.5826 ± 0.0523	0.5932 ± 0.0466	0.5984 ± 0.0306	0.6637 ± 0.0330
Llama-3.3-70B	0.7981 ± 0.0342	0.7929 ± 0.0322	0.7937 ± 0.0369	0.7952 ± 0.0344	0.7905 ± 0.0322
Phi-4	0.8254 ± 0.0303	0.8277 ± 0.0274	0.8309 ± 0.0311	0.8199 ± 0.0440	0.7996 ± 0.0444

Table 13: Ablation 1: Deductive Consistency vs. Hops

Hops	1	2	3	4	5
Qwen-2.5-72B	0.9149 ± 0.0123	0.861 ± 0.022	0.8078 ± 0.011	0.7656 ± 0.0203	0.7287 ± 0.02
Qwen-2.5-Math-72B	0.9164 ± 0.0091	0.8739 ± 0.0138	0.8305 ± 0.0272	0.7895 ± 0.0402	0.7861 ± 0.0584

Table 14: Ablation 2: Deductive Consistency vs. Hops

Model	Hop 1	Hop 2	Hop 3	Hop 4	Hop 5
Qwen-2.5-7B	0.8881 ± 0.0176	0.8453 ± 0.0246	0.8101 ± 0.0222	0.7738 ± 0.0281	0.7620 ± 0.0352
Qwen-2.5-Math-7B	0.8427 ± 0.0156	0.8021 ± 0.0131	0.7739 ± 0.0142	0.7499 ± 0.0251	0.7499 ± 0.0306

Table 15: Ablation 3: Deductive Consistency vs. Hops

Hops	1	2	3	4	5
Qwen-2.5-Math-7B	0.8509 ± 0.018	0.8093 ± 0.0168	0.8002 ± 0.0198	0.7957 ± 0.0382	0.7964 ± 0.0458
DeepSeek-R1-Distill-Qwen-7B	0.8468 ± 0.0271	0.7989 ± 0.0391	0.7451 ± 0.0414	0.7309 ± 0.0566	0.6851 ± 0.0531

Table 16: Ablation 4: Deductive Consistency vs. Hops

Hops	1	2	3	4	5
Llama-3.3-70B	0.8532 ± 0.0134	0.7876 ± 0.0158	0.7515 ± 0.0127	0.7075 ± 0.0107	0.6926 ± 0.0076
DeepSeek-R1-Distill-Llama-70B	0.8366 ± 0.0238	0.7726 ± 0.0333	0.7093 ± 0.0389	0.6741 ± 0.0385	0.6369 ± 0.0307

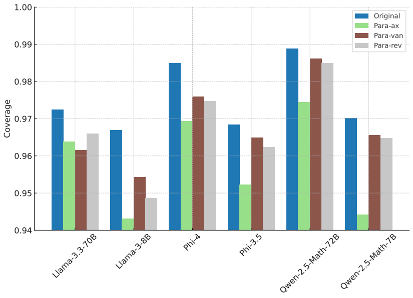


Figure 6: Coverage Comparison Across Interventions (Adjusted Y-Limits)

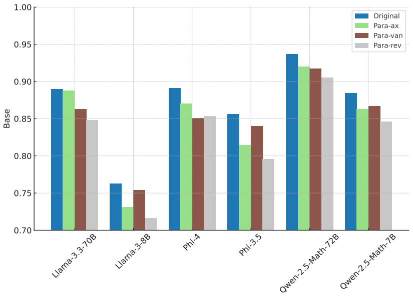


Figure 7: Base Comparison Across Interventions

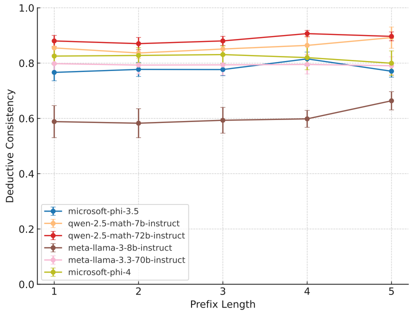
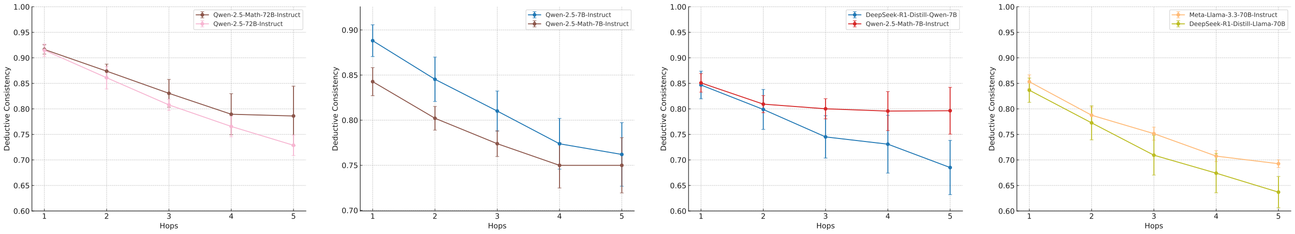
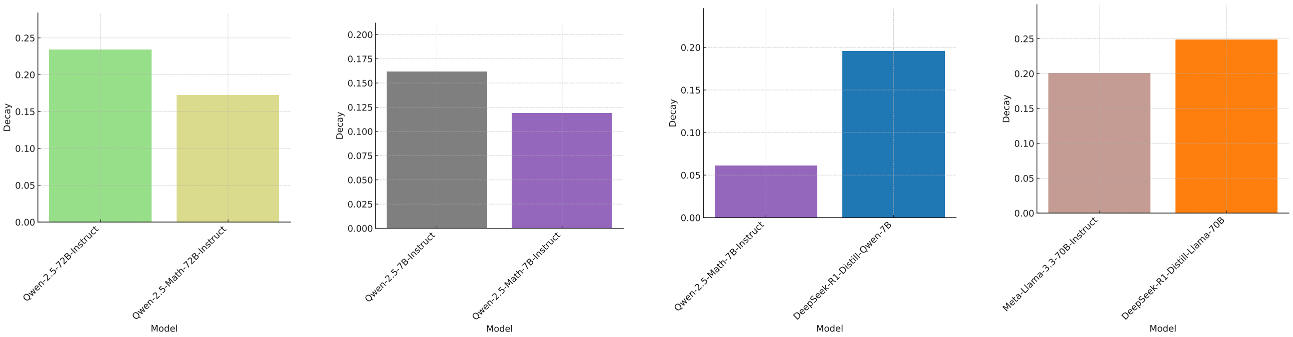


Figure 8: Deductive Consistency vs. Prefix Length for Different Models



(a) Ablation 1 (b) Ablation 2 (c) Ablation 3 (d) Ablation 4

Figure 9: Deductive Consistency vs. Hops for all Ablations



(a) Ablation 1 (b) Ablation 2 (c) Ablation 3 (d) Ablation 4

Figure 10: Comparison of Decay across Models for Ablations 1–4

A.4. Error Analysis on GSM8K

In this section, we perform a qualitative and quantitative analysis of errors that arise in the response models. Model responses are grouped into five groups based on final accuracy on the mutated dataset. Accuracy here is computed over the set of mutated problems for each problem in the original benchmark. The groups are; *Group-1 : Accuracy* $=$ *1 ; Group-2 : 1* $<$ *Accuracy* $\leq$ *0.7 ; Group-3 : 0.7* $<$ *Accuracy* $\leq$ *0.4 ; Group-4 : 0.4* $<$ *Accuracy* $<$ *0 ; Group-5 : Accuracy* $=$ *0*

We use GPT-4o as an evaluator. Calculation errors like arithmetic mistakes, as well as errors in rounding, along with error propagation Logic errors are wrong application of logic/rule/formula. Understanding errors are wrong assumption or contradiction of a given fact. These errors are seen in cases where the problem mentions scenarios that are far from real world such as there being 97 days in a week. In Table 17 we report the frequency of error normalized by number of error responses in that group.

Furthermore, models exhibit (pre-training) bias. They reproduce the original reasoning graph from the vanilla solution. This shows weak robustness from changes in reasoning structure. Logical errors stem from ambiguous natural language. For instance, the sentence "My brother is twice more older than me" should ideally be represented as: $myBrotherAge = myAge + myAge * 2$. However, models typically interpret it as: $myBrotherAge = 2 * myAge$ which correctly corresponds to the unambiguous phrasing "My brother is twice as old as me.".

A.4.1. QUALITATIVE ANALYSIS OF ERROR

Logical Error

Question

Two vampires have come to the sleepy village of Willowton, population 33. Every night, each vampire turns 95 more people into vampires. How many vampires are there after two nights?

Answer (Qwen-2.5-Math-72B)

Step-by-step solution:
Initial Condition: - There are initially 2 vampires in Willowton.
Night 1: - Each vampire turns 95 people into vampires. - Since there are 2 vampires, the total number of new vampires created is:

$$2 \times 95 = 190$$

- Total number of vampires after the first night:

$$2 + 190 = 192$$

Night 2: - Now there are 192 vampires. - Each vampire turns 95 people into vampires. - Since there are 192 vampires, the total number of new vampires created is:

$$192 \times 95 = 18240$$

- Total number of vampires after the second night:

$$192 + 18240 = 18432$$

Final Answer:

$$\boxed{18432}$$

Evaluation (GPT-4o)

Errors:

- **Wrong Assumption:** The response assumes there are enough people to turn into vampires. **Justification:** The village only has 33 people initially, so the vampires cannot turn more people into vampires than the village's population allows. The response fails to consider this limitation.
- **Wrong Logic:** The response calculates the number of new vampires incorrectly on the second night. **Justification:** On the second night, the vampires cannot turn more than the remaining population into vampires. Given the initial population, they cannot create 18,240 new vampires. [This is logic error !!]

Understanding Error

Question

James writes a 36-page letter to 71 different friends 32 times a week. How many pages does he write in a year?

Answer (Phi-4)

Step-by-step solution:
1. **Calculate the total number of pages written per week:** - James writes a 36-page letter to each of his 71 friends.
- Total pages per week:

$$\text{Total pages per week} = 36 \times 71 = 2,556 \text{ pages}$$

2. **Calculate the total number of pages written in a year:** - There are 52 weeks in a year.

$$\text{Total pages per year} = 2,556 \times 52 = 132,912 \text{ pages}$$

Final Answer:

$$\boxed{132,912} \text{ pages}$$

Evaluation (GPT-4o)

Errors:

- **Wrong Assumption:** The response assumes James writes a 36-page letter to each friend only once per week. **Justification:** The problem states he writes 32 times per week, so the calculation of total pages per week is incorrect.
- **Calculation Error:** The response calculates the total number of pages per year using an incorrect assumption. **Justification:** Since the number of times he writes per week is not factored in, the yearly calculation does not reflect the correct total.

Calculation Error

Question

Max likes to collect model trains. He asks for 53 for every birthday of his, and asks for 18 each Christmas. Max always gets the gifts he asks for, and asks for these same gifts every year for 16 years. At the end of the 16 years, his parents give him double the number of trains he already has. How many trains does Max have now?

Answer (Llama-3-8B)

Step-by-step solution:

1. Max gets 53 trains for each birthday, and he has 16 birthdays:

$$53 \times 16 = 848 \text{ trains}$$

2. Max gets 18 trains for each Christmas, and he has 16 Christmases:

$$18 \times 16 = 288 \text{ trains}$$

3. In total, Max receives:

$$848 + 288 = 1,136 \text{ trains}$$

4. At the end of the 16 years, his parents give him double the number of trains he already has:

$$1,136 \times 2 = 2,268 \text{ trains}$$

5. In total, Max now has:

$$1,136 + 2,268 = 3,404 \text{ trains}$$

Final Answer:

$$\boxed{3,404} \text{ trains}$$

Evaluation (GPT-4o)

Errors:

- **Calculation Error:** The response states that at the end of the 16 years, Max's parents give him double the number of trains he already has, calculated as $1,136 \times 2 = 2,268$. **Justification:** The correct calculation should be:

$$1,136 \times 2 = 2,272$$

- **Propagated Error Only:** The response states that the total number of trains Max has is $1,136 + 2,268 = 3,404$. **Justification:** Since the previous multiplication was incorrect, the final sum was also incorrect. The correct final total should be:

$$1,136 + 2,272 = 3,408$$

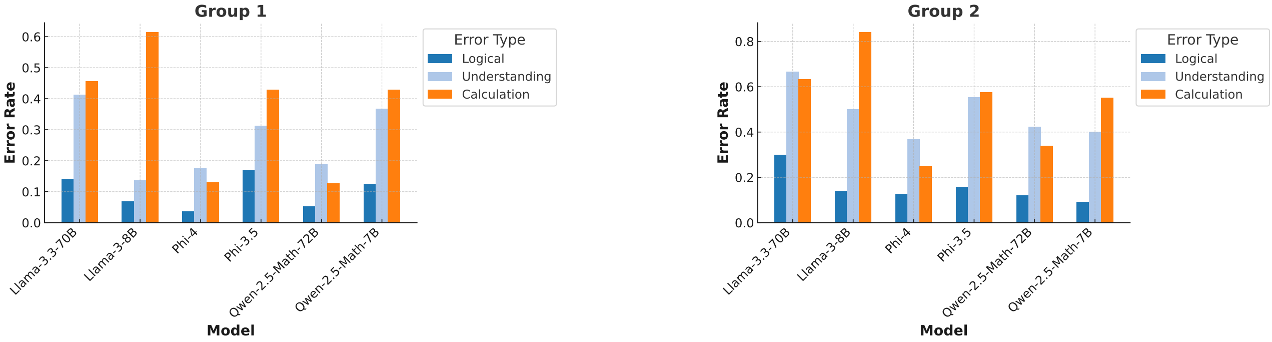
A.4.2. QUANTITATIVE ANALYSIS OF ERROR

Models	Group-1			Group-2			Group-3			Group-4			Group-5		
	Logical	Understanding	Calculation	Logical	Understanding	Calculation	Logical	Understanding	Calculation	Logical	Understanding	Calculation	Logical	Understanding	Calculation
Llama-3.3-70B	0.141	0.413	0.457	0.300	0.667	0.633	0.182	0.601	0.790	0.287	0.943	0.780	0.356	0.578	0.856
Llama-3-8B	0.068	0.136	0.614	0.140	0.500	0.840	0.198	0.548	0.853	0.279	0.624	0.886	0.344	0.672	0.822
Phi-4	0.036	0.175	0.130	0.128	0.368	0.248	0.229	0.702	0.550	0.261	0.513	0.704	0.369	0.946	0.754
Phi-3.5	0.169	0.312	0.429	0.158	0.554	0.576	0.152	0.488	0.784	0.227	0.553	0.827	0.429	0.659	0.865
Qwen-2.5-Math-72B	0.052	0.188	0.127	0.121	0.423	0.340	0.245	0.669	0.619	0.365	0.794	0.518	0.300	0.583	0.883
Qwen-2.5-Math-7B	0.125	0.368	0.429	0.091	0.400	0.551	0.263	0.563	0.721	0.304	0.562	0.788	0.255	0.391	0.818

Table 17: Error metrics for different models across dataset groups. Each cell reports Logical, Understanding, and Calculation errors separately.

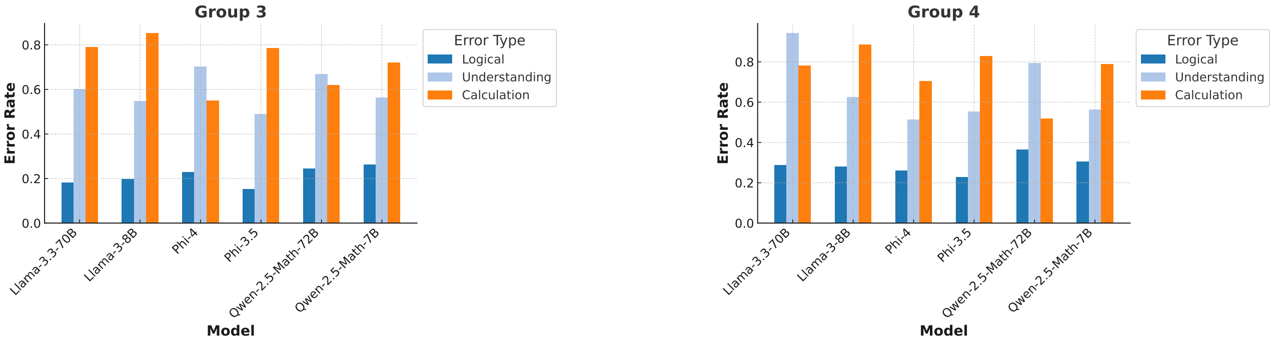
A.5. Calculation Monitor Experiment (Tool Calling)

To improve on decay and reduce the calculation errors, we try a online calculation monitor approach. We have an meta llama-8-b model as an online monitor that is a reviewer and editor. As the generator generates a step, reviewer LLM of the monitor checks if calculation is present, and if so calls a calculator with appropriate inputs and the output from calclator is passed into editor LLM which os responsible to edit that step and replace it in the original solution. The generation then



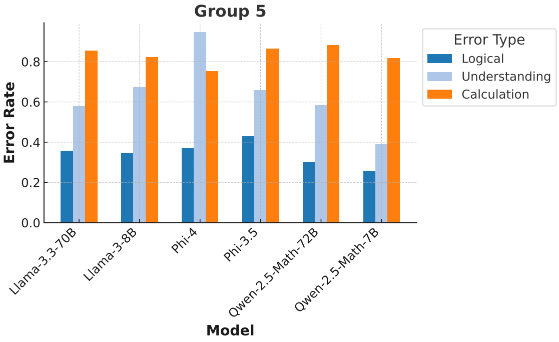
(a) Group 1

(b) Group 2



(c) Group 3

(d) Group 4



(e) Group 5

Figure 11: Error rate plots for all groups

Models	**Group 1**	**Group 2**	**Group 3**	**Group 4**	**Group 5**
Llama-3-8B	44	50	252	412	180
Llama-3.3-70B	92	90	143	209	90
Qwen-2.5-Math-72B	916	338	139	170	60
Qwen-2.5-Math-7B	560	450	240	240	110
Phi-4	553	117	131	199	130
Phi-3.5	77	139	125	255	170

Table 18: Number of instances for different models across dataset groups.

continues after the step has been corrected for calculation errors. We find an improvement in base deductive consistency of up to 15% but decay did not improve.

A.5.1. ERROR ANALYSIS

Despite the streamlined review–and–edit loop, the calculator–tool intervention yields improvement upto 15 % in base deductive consistency (Deductive consistency at 1-Hop)(Table 20, Table 19) and fails to curb the decay that accumulates over longer reasoning chains. This approach targets arithmetic slip-ups alone; it leaves intact other error modes—misconceptions, faulty inferences, and hallucinated premises, that cannot be caught or corrected by a simple tool call. Error Analysis with Monitor (Table 21) and without Monitor (Table 17) further reveals that, even after driving calculation errors to a minimum, we continue to observe substantial understanding errors and logical inconsistencies. Taken together, these findings show that tool-based approaches which merely supply a calculator do not resolve the broader problem of deductive decay.

Hop	**LLaMA-3-8B-Instruct**		**Phi-4**	
	With Monitor	**Without Monitor**	**With Monitor**	**Without Monitor**
1	0.9081 ± 0.0139	0.7629 ± 0.0217	0.9421 ± 0.0318	0.7629 ± 0.0238
2	0.8098 ± 0.0205	0.6572 ± 0.0227	0.8762 ± 0.0761	0.6572 ± 0.0278
3	0.7356 ± 0.0264	0.5777 ± 0.0165	0.8489 ± 0.0848	0.5777 ± 0.0285
4	0.6651 ± 0.0391	0.5254 ± 0.0154	0.8641 ± 0.0545	0.5254 ± 0.0228
5	0.6100 ± 0.0470	0.4988 ± 0.0264	0.7715 ± 0.1081	0.4988 ± 0.0117

Table 19: Deductive consistency with and without monitor across hops for LLaMA-3-8B-Instruct and Phi-4.

	LLaMA-3-8B-Instruct		**Phi-4**	
Monitor	**Base**	**Decay**	**Base**	**Decay**
Without Monitor	0.7629	0.3300	0.8911	0.1517
With Monitor	0.9081	0.3704	0.9421	0.1750

Table 20: Base and decay values for LLaMA-3-8B-Instruct and Phi-4 with and without monitor.

A.6. SynDeduct

Why Use Synthetic Dataset? Synthetic datasets offer a controlled framework for evaluating deductive accuracy, as all data points are generated according to predefined rules with precisely derived ground truth. This setup enables meticulous regulation of the underlying computation graph, allowing us to specify the total number of reasoning steps (i.e., edges in the graph) needed to arrive at the final answer.

In constructing of SynDeduct dataset, we begin by sampling a set of DAGs according to parameters that define constants, variable distributions, and arithmetic operators. We then extract paths from each DAG as programmatically computed reasoning traces. The resulting ground-truth derivations are converted into Chain of Thought representations by applying a set of verbalization templates, yielding readable textual explanations. Unlike in GSM8K, our approach does not require code generation or templated Chains of Thought, as the underlying computation graph is already available.

We quantify a path's difficulty by counting the number of reasoning steps (graph edges) it takes to move from the initial

Model	Calculation	Logic	Understanding
Phi-4	0.1058	0.0445	0.1058
Llama-3-8B	0.1577	0.0611	0.282

Table 21: Error Rates of Models w/Monitor normalized by total data-points

(base) node to the final (target) node. Accordingly, we generate N sets of questions, where the n-th set contains questions that require n steps.

To accommodate varying input-premise lengths, we create additional questions by progressively appending segments of the ground-truth reasoning chain to converge on the same target premise. We then place these questions into bins based on how many hops are needed, intermixing different prefix lengths within each bin. This organization yields n bins, each focused on questions requiring n hops but differing in the prefixed portion of the chain. Such binning enables robust averaging of model performance for varying prefix lengths within the same number of steps.

To reduce unnecessary increase in calculation errors we tried to keep the operators limited to only addition and subtraction.

Table 22: DC ± Standard Error for Hops 1-6

Model	**Hop 1**	**Hop 2**	**Hop 3**	**Hop 4**	**Hop 5**	**Hop 6**
Qwen-2.5-Math-7B	0.2083 ± 0.0336	0.1205 ± 0.0354	0.0867 ± 0.0228	0.0898 ± 0.0245	0.0792 ± 0.0238	0.0607 ± 0.0189
Qwen-2.5-7B	0.5458 ± 0.0376	0.3705 ± 0.0474	0.3250 ± 0.0430	0.2630 ± 0.0357	0.2562 ± 0.0279	0.2440 ± 0.0334
Qwen-2.5-Math-72B	0.5674 ± 0.0285	0.4894 ± 0.0434	0.4433 ± 0.0456	0.3852 ± 0.0553	0.3635 ± 0.0513	0.3381 ± 0.0549
Qwen-2.5-72B	0.6868 ± 0.0287	0.5848 ± 0.0389	0.4825 ± 0.0368	0.4046 ± 0.0358	0.3354 ± 0.0294	0.2643 ± 0.0215
Llama-3-8B	0.2993 ± 0.0458	0.2023 ± 0.0411	0.1825 ± 0.0382	0.1602 ± 0.0368	0.1469 ± 0.0275	0.1357 ± 0.0257
DeepSeek-R1-Llama-70B	0.7389 ± 0.0202	0.6879 ± 0.0196	0.6742 ± 0.0175	0.6509 ± 0.0169	0.6542 ± 0.0167	0.6488 ± 0.0206
Llama-3.3-70B	0.8465 ± 0.0124	0.8129 ± 0.0158	0.7675 ± 0.0147	0.7250 ± 0.0140	0.7125 ± 0.0177	0.6833 ± 0.0096
DeepSeek-R1-Qwen-7B	0.5424 ± 0.0369	0.3871 ± 0.0442	0.3308 ± 0.0416	0.2870 ± 0.0432	0.2802 ± 0.0441	0.2262 ± 0.0348

Table 23: DC ± Standard Error for Hops 7-12

Model	**Hop 7**	**Hop 8**	**Hop 9**	**Hop 10**	**Hop 11**	**Hop 12**
Qwen-2.5-Math-7B	0.0403 ± 0.0105	0.0383 ± 0.0124	0.0250 ± 0.0088	0.0250 ± 0.0097	0.0167 ± 0.0090	0.0000 ± 0.0000
Qwen-2.5-7B	0.1639 ± 0.0196	0.1633 ± 0.0217	0.1479 ± 0.0256	0.0917 ± 0.0273	0.0792 ± 0.0232	0.0500 ± 0.0186
Qwen-2.5-Math-72B	0.3222 ± 0.0582	0.3367 ± 0.0578	0.3229 ± 0.0678	0.2861 ± 0.0648	0.2667 ± 0.0691	0.2583 ± 0.0702
Qwen-2.5-72B	0.2167 ± 0.0176	0.1750 ± 0.0164	0.1562 ± 0.0209	0.0889 ± 0.0149	0.0667 ± 0.0136	0.0417 ± 0.0185
Llama-3-8B	0.0903 ± 0.0215	0.0883 ± 0.0201	0.1104 ± 0.0229	0.0861 ± 0.0169	0.0417 ± 0.0154	0.0083 ± 0.0080
DeepSeek-R1-Llama-70B	0.6431 ± 0.0159	0.6083 ± 0.0248	0.5958 ± 0.0273	0.5556 ± 0.0186	0.4667 ± 0.0325	0.3333 ± 0.0430
Llama-3.3-70B	0.6347 ± 0.0156	0.6017 ± 0.0169	0.5854 ± 0.0178	0.5750 ± 0.0391	0.5417 ± 0.0316	0.6083 ± 0.0343
DeepSeek-R1-Qwen-7B	0.2097 ± 0.0336	0.1633 ± 0.0360	0.1604 ± 0.0353	0.1250 ± 0.0243	0.0667 ± 0.0198	0.0667 ± 0.0136

Table 24: DC ± Standard Error for Prefix 1-6

Model	**Prefix 1**	**Prefix 2**	**Prefix 3**	**Prefix**	**Prefix 5**	**Prefix 6**
Qwen-2.5-Math-7B	0.2113 ± 0.0452	0.1118 ± 0.0257	0.0967 ± 0.0218	0.0878 ± 0.0231	0.0663 ± 0.0214	0.0734 ± 0.0132
Qwen-2.5-7B	0.3719 ± 0.0678	0.3715 ± 0.0483	0.2947 ± 0.0559	0.2951 ± 0.0452	0.2289 ± 0.0479	0.2401 ± 0.0421
Qwen-2.5-Math-72B	0.7968 ± 0.0140	0.6433 ± 0.0187	0.4900 ± 0.0273	0.3836 ± 0.0338	0.3812 ± 0.0313	0.3475 ± 0.0289
Qwen-2.5-72B	0.3674 ± 0.0737	0.3526 ± 0.0773	0.3674 ± 0.0714	0.3522 ± 0.0777	0.3039 ± 0.0670	0.3396 ± 0.0545
Llama-3-8B	0.3011 ± 0.0451	0.2580 ± 0.0341	0.2371 ± 0.0430	0.1494 ± 0.0285	0.1169 ± 0.0277	0.0981 ± 0.0191
DeepSeek-R1-Llama-70B	0.5970 ± 0.0546	0.6546 ± 0.0570	0.6828 ± 0.0401	0.6461 ± 0.0569	0.6287 ± 0.0563	0.6304 ± 0.0474
Llama-3.3-70B	0.6452 ± 0.0204	0.7562 ± 0.0186	0.6720 ± 0.0298	0.6691 ± 0.0292	0.6730 ± 0.0181	0.6094 ± 0.0301
DeepSeek-R1-Qwen-7B	0.4534 ± 0.0581	0.3817 ± 0.0550	0.3252 ± 0.0552	0.2780 ± 0.0440	0.3136 ± 0.0376	0.1923 ± 0.0406

Table 25: DC ± Standard Error for Prefix 7-12

Model	**Prefix 7**	**Prefix 8**	**Prefix**	**Prefix 10**	**Prefix 11**	**Prefix 12**
Qwen-2.5-Math-7B	0.0383 ± 0.0157	0.0407 ± 0.0106	0.0191 ± 0.0080	0.0151 ± 0.0061	0.0176 ± 0.0100	0.0124 ± 0.0061
Qwen-2.5-7B	0.1651 ± 0.0386	0.1544 ± 0.0383	0.1918 ± 0.0329	0.1376 ± 0.0362	0.1277 ± 0.0238	0.1217 ± 0.0227
Qwen-2.5-Math-72B	0.2979 ± 0.0410	0.3003 ± 0.0274	0.2063 ± 0.0336	0.1619 ± 0.0301	0.1676 ± 0.0401	0.2034 ± 0.0359
Qwen-2.5-72B	0.2892 ± 0.0583	0.2505 ± 0.0558	0.2376 ± 0.0490	0.2286 ± 0.0419	0.2007 ± 0.0436	0.2138 ± 0.0478
Llama-3-8B	0.0775 ± 0.0240	0.0903 ± 0.0147	0.0821 ± 0.0163	0.0362 ± 0.0085	0.0599 ± 0.0091	0.0452 ± 0.0108
DeepSeek-R1-Llama-70B	0.6311 ± 0.0498	0.6519 ± 0.0377	0.6133 ± 0.0316	0.6845 ± 0.0149	0.6875 ± 0.0352	0.6426 ± 0.0473
Llama-3.3-70B	0.6164 ± 0.0176	0.5876 ± 0.0226	0.6538 ± 0.0195	0.5721 ± 0.0317	0.5625 ± 0.0206	0.5844 ± 0.0122
DeepSeek-R1-Qwen-7B	0.1470 ± 0.0350	0.1923 ± 0.0339	0.1927 ± 0.0377	0.1070 ± 0.0292	0.1302 ± 0.0344	0.1321 ± 0.0300

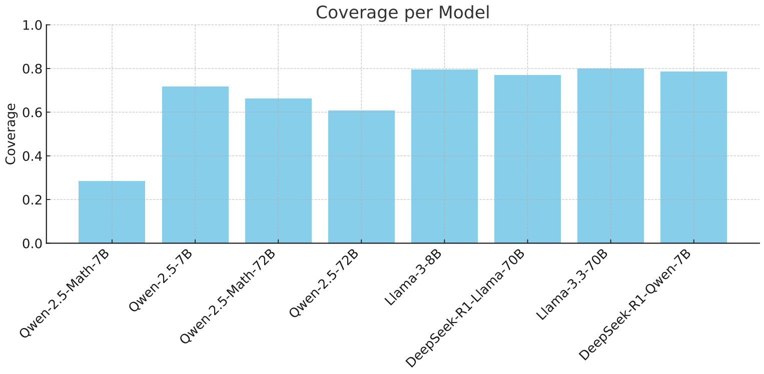
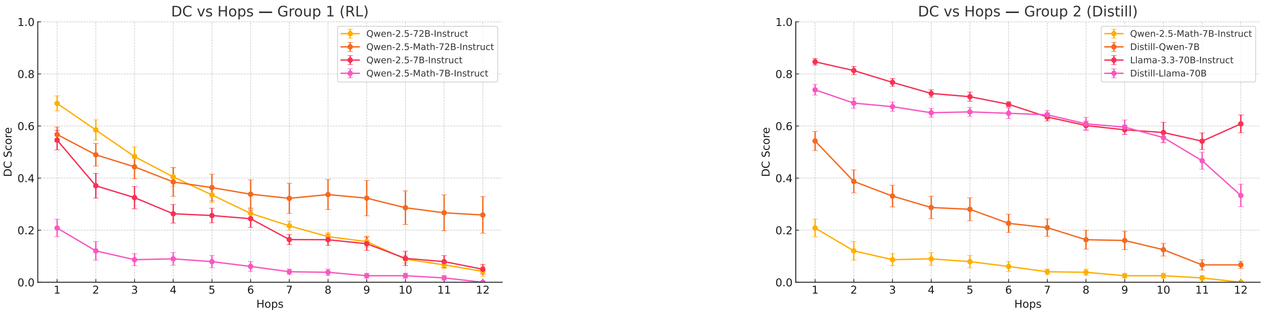


Figure 12: Coverage Metric in SynDeDeduct for Different Models



(a) DC vs. Hops for Group 1

(b) DC vs. Hops for Group 2

Figure 13: Comparison of DC vs. Hops for Different Model Groups

A.7. Error Analysis on SynDeduct

A.7.1. QUALITATIVE ANALYSIS

We have the following error categories: Calculation: correct operands & operator, but wrong arithmetic.

Logic: correct operands but wrong operator. Understanding: wrong operands or misread definition.

Garbled: incoherent or bad attempt at correct solution or repeated backtracking.

Exit: The case where response from error analysis llm could not be parsed.

Calculation Error

Inputs:
- X (value = 7)
- Y (value = 7)
- Z is the difference between X and Y.
- What is the value of Z?

Golden Answer Reasoning:
- Z is the difference between X and Y. Z = 7 - 7 = 0

Model Final Answer:
- Z is the difference between X and Y. Z = 7 - 7 = -3

Explanation:
Operands: X=7, Y=7; Operation: -; Computed: 7-7=-3 vs 0.

Understanding Error

Inputs:
- X (value = 7)
- Y (value = 7)
- T (value = 2)
- Z is the difference between X and Y.
- What is the value of Z?

Golden Answer Reasoning:
- Z = 7 - 7 = 0

Model Final Answer:
- Z is the difference between X and T. Z = X - T = 5

Explanation:
Picked Operands X and T instead of X and Y

Logic Error

Inputs:
- X (value = 7)
- Y (value = 7)
- Z is the difference between X and Y.
- What is the value of Z?

Golden Answer Reasoning:
- Z = 7 - 7 = 0

Model Final Answer:
- Z is the sum between X and Y. Z = X + Y = 14

Explanation:
Picked Operands X and Y (correct) but used "+" operations instead of "-"

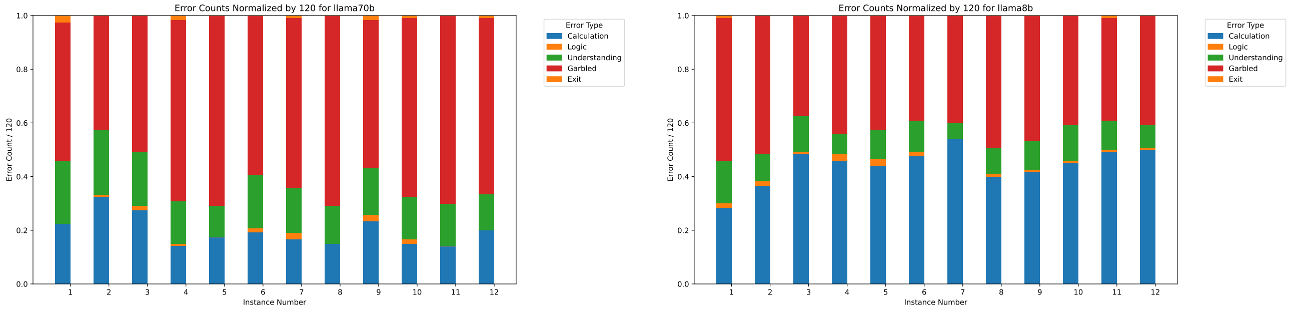
A.7.2. QUANTITATIVE ANALYSIS

Calculation errors (Figure 14) are major source of performance degradation and dominate through all prefix/hop levels. Next, Understanding errors are present and stay constant thought out and Logic errors are sparse to find.

Garbled Errors are a major source of concern. tried our best to ensure that the experimental setup is sound. We used the recommended (and other) sampling settings, prompt settings for the model and did not see major improvement in garbled errors. We also used different LLM hosting packages and their default settings (vLLM, transformers etc and see no change). Exit failure code denotes where our error analysis LLM failed to produce parse-able results and such errors are few.

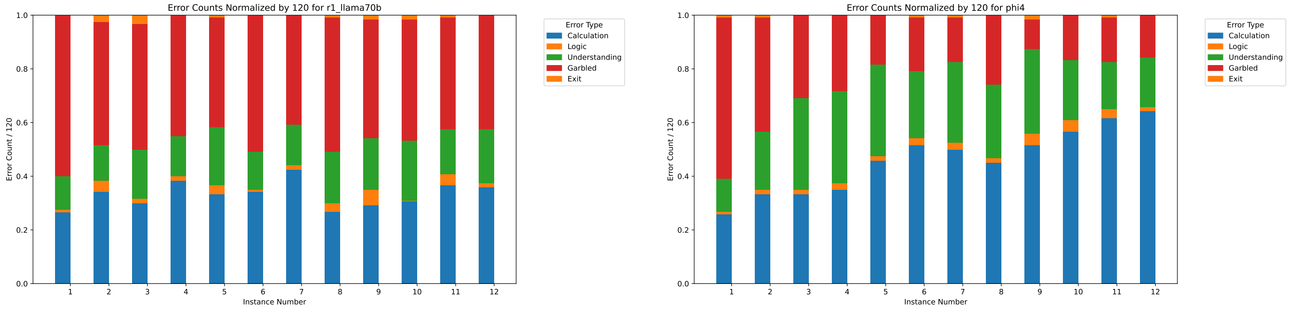
One could argue that large context length required for synDeduct may cause issues, however it doesnt hold well. As the maximum number of tokens for the task never exceeds 10k, which is well below for the maximum context size for these models. And the task is hand is very simple.

Also, models like Phi-4 and Qwen 7B, Qwen 7B R1 perform way better in the same experimental settings. This failure mode deep dive analysis is left for future work.



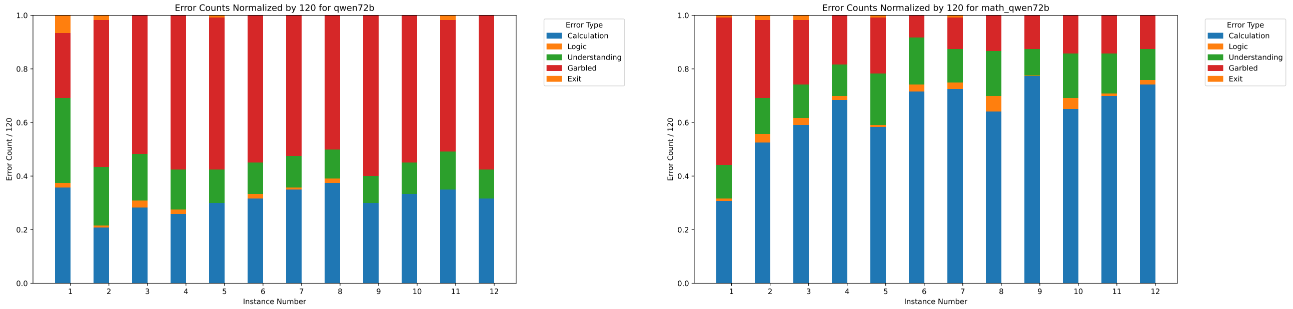
(a) Llama 3.3 70B

(b) Llama 3 8B



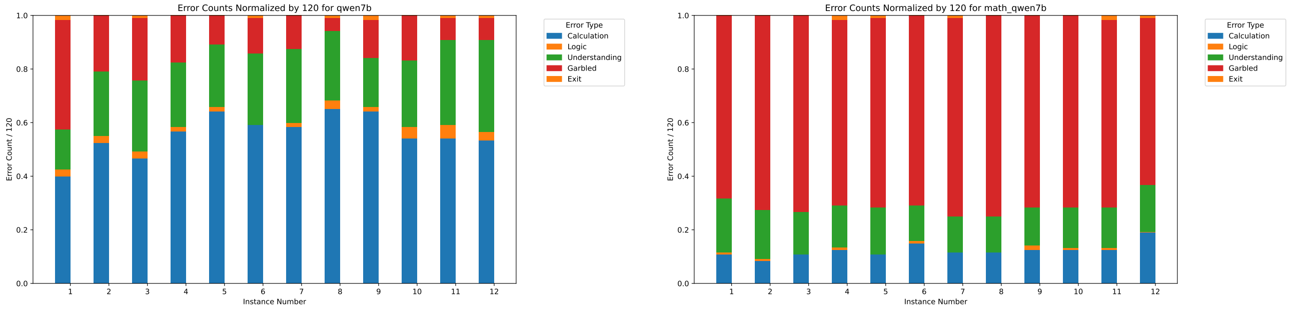
(c) Llama 3.3 70B R1

(d) Phi 4



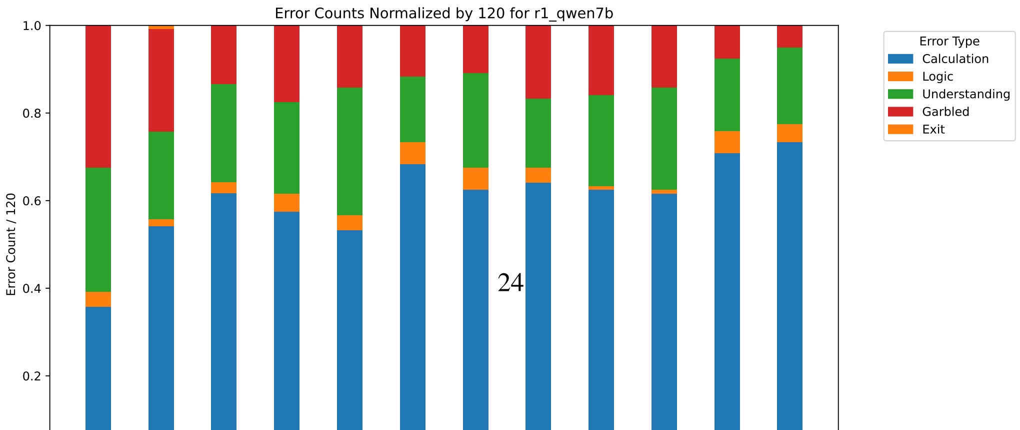
(e) Qwen 72B

(f) Qwen Math 72B



(g) Qwen 7B

(h) Qwen Math 7B



A.8. ProntoQA

Here are the results on ProntoQA.

Model Name	Coverage
Phi-4	0.9151
DeepSeek-R1-Llama-70B	0.9105
DeepSeek-R1-Qwen-7B	0.9096
Qwen2.5-7B	0.9060
Qwen2.5-Math-72B	0.8902
Phi-3.5-mini	0.8706
LLaMA-3.3-70B	0.8466
Qwen2.5-Math-7B	0.8341
Meta-LLaMA-3-8B	0.7738

Model Name	Hop@1	Hop@2	Hop@3	Hop@4	Hop@5
Phi-4	1.000	0.999	0.999	1.000	1.000
DeepSeek-R1-Llama-70B	0.998	0.997	0.996	0.991	0.996
DeepSeek-R1-Qwen-7B	0.995	0.984	0.970	0.953	0.953
Qwen2.5-7B	0.993	0.982	0.967	0.944	0.935
Qwen2.5-Math-72B	0.999	0.997	0.996	0.993	0.993
Phi-3.5-mini	0.998	0.997	0.996	0.994	0.996
LLaMA-3.3-70B	0.998	0.998	0.994	0.993	0.993
Qwen2.5-Math-7B	0.992	0.982	0.967	0.957	0.942
Meta-LLaMA-3-8B	0.994	0.985	0.986	0.979	0.986

Model Name	P@0	P@1	P@2	P@3	P@4
Phi-4	1.000	1.000	0.998	1.000	1.000
DeepSeek-R1-Llama-70B	0.999	0.973	0.966	0.994	0.937
DeepSeek-R1-Qwen-7B	0.981	0.954	0.983	0.991	0.996
Qwen2.5-7B	0.953	0.978	0.987	0.982	0.996
Qwen2.5-Math-72B	0.997	0.997	0.997	0.997	0.998
Phi-3.5-mini	0.999	0.995	0.997	0.995	0.939
LLaMA-3.3-70B	0.997	0.997	0.996	0.997	0.996
Qwen2.5-Math-7B	0.960	0.976	0.981	0.982	0.994
Meta-LLaMA-3-8B	0.991	0.986	0.984	0.994	0.996

A.9. LM Prompts for GSM8K

LLM Prompt

System prompt: "You are an expert in math. You must answer this question. You give the step-by-step solution followed by four # symbols followed by the answer."
User: "Question: <question>\nAnswer:"

Example Question:
User: "Question: Yasna has two books. One book is 60 pages long, and the other book is 12 pages long. If Yasna wants to finish both of the books in 6 weeks, how many pages will Yasna need to read every day, if she reads an equal number of pages each day? \nAnswer:"

Example Assistant Partial Answer:
Assistant:

Yasna has $60 + 12 = 72$ pages to read.

She has $6 \times 7 = 42$ days to read them.

Template Builder Agent Prompt

System Prompt:
You are a templatizing agent. Your task is to process questions and answers, templatize them by replacing specific numerical values with placeholders, and create a structured JSON output. The JSON output must contain the following keys:
1. **templatized_question**: A version of the question where specific numerical values, object names, or other unique entities are replaced by placeholders.
2. **templatized_answer**: A step-by-step reasoning answer where specific numerical values or entities are replaced by placeholders. Each step should remain logically consistent with the original answer.
3. **factual_assignment**: A dictionary mapping placeholders to their original factual values, ensuring the templatized versions can reconstruct the original question and answer. MUST only contain NUMERICAL values.
Make sure the templatized answer and question ALIGN PERFECTLY with the original answer structure.
—
ICL Examples
Example 1
Input Question:
A train travels 60 kilometers in 2 hours. What is its average speed?
Input Answer:
The train travels a distance of 60 kilometers in 2 hours. Average speed is calculated as distance divided by time.
Average speed = 60 / 2 = 30 kilometers per hour.
Output JSON:

```
{
  "templatized_question": "A train travels {distance} kilometers in {time} hours.
      What is its average speed?",
  "templatized_answer": [
    "The train travels a distance of {distance} kilometers in {time} hours.",
    "Average speed is calculated as distance divided by time.",
    "Average speed = {distance} / {time} = {average_speed} kilometers per hour."
  ],
  "factual_assignment": {
    "distance": 60,
    "time": 2,
    "average_speed": 30
  },

  "node_explanation":{
    "distance": "The distance traveled by the train",
    "time": "time taken by the train to travel the distance",
    "average_speed": "The average speed of the train"
  }
}
```

—

Example 2 **Input Question:**
Mary buys 3 books for $15 each. How much does Mary spend in total?
Input Answer:
Mary buys 3 books, each costing $15. Total cost is calculated as number of books multiplied by the cost per book. Total cost = 3 * 15 = $45.
Output JSON:

```
{
  "templatized_question": "Mary buys {quantity} books for ${cost_per_book} each. How
      much does she spend in total?",
  "templatized_answer": [
    "Mary buys {quantity} books, each costing {cost_per_book}.",
    "Total cost is calculated as number of books multiplied by the cost per book.",
    "Total cost = {quantity} * {cost_per_book} = ${total_cost}."
  ],
  "factual_assignment": {
    "quantity": 3,
    "cost_per_book": 15,
    "total_cost": 45
  },
  "node_explanation":{
    "quantity": "The number of books bought by Mary",
    "cost_per_book": "The cost of each book",
    "total_cost": "The total amount spent by Mary"
  }
}
```

—

User Prompt:
f"Question: <question>\nAnswer: <answer>\n\nProvide the templatized version as per the example above."

Code Generation LLM Prompt

System Prompt:
Generate Python code that solves the following problem step by step:
User Prompt:

```
Question:
<question>

Answer:
<CoT Answer>. The code must follow the variable names similar to ones in <
    templatized_answer>

Python Code:
```

Variable Extractor LLM Prompt

Instructions:
You are an **expert in comprehension and variable extraction**. Your task is to analyze a **question**, a **step-by-step solution**, and a **dictionary of variables** and return a JSON object that adheres to the following rules:

—

Guidelines:

1. **Inputs:**
   - **Question**: The problem description.
   - **Step-by-step solution**: The solution text, where variables may be explicitly stated or calculated.
   - **Dictionary of variables**: Contains variable names and their descriptions. Not all variables may appear in the question or solution.
2. **Output Format:**
   - Return a **JSON object** with:
     - **Keys**: Variable names from the dictionary.
     - **Values**: Numeric values extracted from the solution or question.
     - If a value is explicitly mentioned in the **step-by-step solution**, extract it without recalculating.
     - If the variable is not present in the solution or question, return `"None"`.
     - Values must **preserve their original format** (e.g., fractions, decimals, or expressions).
3. **Output Structure:**
   - Enclose the output JSON object within `<JSON>` and `</JSON>` tags.
   - All numeric values must be **string representations** (e.g., `"3/2"`, `"25.5"`, or `"12+8"`).
4. **Restrictions:**
   - **Do not solve** the problem yourself or calculate missing values.
   - Extract only the values as they appear in the solution.

Example 1:

```
Question:
At a flea market, Hillary sells handmade crafts for 15 dollars per craft.
Today, Hillary sells 6 crafts and is given an extra 5 dollars from an appreciative
    customer.
Later on, Hillary deposits 12 dollars from today's profits into her bank account.
How many dollars is Hillary left with after making the deposit?

Step-by-step solution:
Hillary earns \(15 \times 6 = 90\) dollars from selling crafts.
Adding the extra 5 dollars, she has 90 + 5 = 95 dollars.
After depositing 12 dollars, she has 95 - 12 = 83 dollars left.

Dictionary of variables:
{
   "price_per_craft": "The price of each craft",
   "number_of_crafts": "The number of crafts sold",
   "extra_dollars": "The extra amount given by the customer",
   "deposit_amount": "The amount deposited into the bank account",
   "total_earnings": "The total amount earned from selling crafts",
   "total_amount": "The total amount after receiving the extra dollars",
   "amount_left": "The amount left after depositing"
}

Output:
```

```
<JSON> {
   "price_per_craft": "15",
   "number_of_crafts": "6",
   "extra_dollars": "5",
   "deposit_amount": "12",
   "total_earnings": "90",
   "total_amount": "95",
   "amount_left": "83"
} </JSON>
```

Example 2:

```
Question:
In a truck, there are 5 pink hard hats, 16 green hard hats, and 15 yellow hard hats.
Carl takes away 10 pink hard hats.
John takes away 7 pink hard hats and twice as many green hard hats as the number of
    pink hard hats he removed.
Calculate the total number of hard hats that remained in the truck.

Step-by-step solution:
The total number of hats is 5 + 16 + 15 = 36.
Carl removes 10 pink hats, leaving 36 - 10 = 26.
John removes 7 pink hats, leaving 26 - 7 = 19.
John also removes \(7 \times 2 = 14\) green hats, leaving \(19 - 14 = 5\) hats in
    total.


Dictionary of variables:
{
   "pink": "The number of pink hard hats",
   "green": "The number of green hard hats",
   "yellow": "The number of yellow hard hats",
   "carl_pink": "The number of pink hard hats taken by Carl",
   "john_pink": "The number of pink hard hats taken by John",
   "total_initial": "The total number of hats initially",
   "total_after_carl": "The total number of hats after Carl’s removal",
   "total_after_john_pink": "The total number of hats after John’s pink hat removal",

   "john_green": "The number of green hats taken by John",
   "total_final": "The total number of hats remaining"
}

Output:
<JSON> {
   "pink": "5",
   "green": "16",
   "yellow": "15",
   "carl_pink": "10",
   "john_pink": "7",
   "total_initial": "36",
   "total_after_carl": "26",
   "total_after_john_pink": "19",
   "john_green": "14",
   "total_final": "5"
} </JSON>
```

User Prompt:

```
Here is the question and the step-by-step solution to the problem:

Question: {generation[6]}
```

```
Step-by-step solution: {generation[1]}

Dictionary of variables: {generation[5]}

For more detailed explanation of variables you can see how they were used in this
    template: {generation[2]}
```

A.10. Artifacts SynDeduct

Rule Set for SynDeduct

```
{
  "add": {
    "function": "lambda x, y: x + y",
    "verbalization": "{child} is the sum of {parent1} and {parent2}."
  },
  "subtract": {
    "function": "lambda x, y: x - y",
    "verbalization": "{child} is the difference between {parent1} and {parent2}."
  }
}
```

Data Generation Steps

Initially, Directed Acyclic Graphs are generated. A computation graph is then constructed by selecting a fixed-length path within each DAG and randomly assigning values and operators to its nodes. A rule set, in conjunction with a predetermined collection of nouns, is employed to generate verbalization.

Additionally, Chain-of-Thought solutions along with final answer are produced and later used to create prefixes. It is important to note that the questions generated in this process consist exclusively of "n" hops and do not include any prefix.

```
python3 init.py --num_graphs 99000 --m 60 --unary_ratio 0.0 --logic_mode bodmas --
    naming_mode noun --nouns.json --operators_file ruleset.json --output_file output.
    json --max_hops 24 --max_graphs 4000

Steps kept and undersampled to 4000:
Hop 1: 4000
Hop 2: 4000
Hop 3: 4000
Hop 4: 4000
Hop 5: 4000
Hop 6: 4000
Hop 7: 4000
Hop 8: 4000
Hop 9: 4000
Hop 10: 4000
Hop 11: 4000
Hop 12: 4000
Hop 13: 4000
Hop 14: 4000
Hop 15: 4000
Hop 16: 4000
Hop 17: 4000
Hop 18: 4000
Hop 19: 4000
Hop 20: 4000
Hop 21: 4000
Hop 22: 4000
Hop 23: 4000
Hop 24: 4000
```

The chain-of-solution is now appended to the question, resulting in questions that incorporate a specified number of prefixes alongside n hops. To create a balanced dataset, the maximum number of hops is limited to 12, half the total hops, and the total number of items is capped at 120.
For instance, in the case of Hop12, there are 10 questions featuring a 12-hop prefix. This configuration implies that each such question originated from a 24-hop question, wherein the first 12 hops, serving as the prefix of the chain-of-thought, are provided, and the language model is required to resolve the remaining 12 hops.

```
python transformer.py --max_hops 12 --max_items 120 --max_prefixes 12 --
    max_prefix_length 10 output.json


Prefix Length Distribution Per Hop Category (After Undersampling):

Hop 1: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 2: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 3: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 4: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 5: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 6: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 7: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 8: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 9: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 10: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 11: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10
Hop 12: Prefix1: 10, Prefix2: 10, Prefix3: 10, Prefix4: 10, Prefix5: 10, Prefix6: 10,
     Prefix7: 10, Prefix8: 10, Prefix9: 10, Prefix10: 10, Prefix11: 10, Prefix12: 10

Total Prefix Length Distribution Across Hops:

Prefix1: 120
Prefix2: 120
Prefix3: 120
Prefix4: 120
Prefix5: 120
Prefix6: 120
Prefix7: 120
Prefix8: 120
Prefix9: 120
Prefix10: 120
Prefix11: 120
Prefix12: 120

Number Of Items per Hop

Hop: 1 - 120
Hop: 2 - 120
Hop: 3 - 120
```

```
Hop: 4 - 120
Hop: 5 - 120
Hop: 6 - 120
Hop: 7 - 120
Hop: 8 - 120
Hop: 9 - 120
Hop: 10 - 120
Hop: 11 - 120
Hop: 12 - 120
Total entries in transformed JSON: 1440
```

A single Data-point of SynDeduct

Prompt Part A: Graph Structure and Question (will be given as user) *The graph structure encompasses the complete verbalization of the entire graph, whereas the question is derived solely from a specific path within that graph. Consequently, a considerable amount of the information contained in the graph structure is not necessary for generating a solution. This design serves to assess the model's capability to extract and utilize only the relevant information from a broader context.*

```
=== Graph Structure ===
Inputs:
  - Masako (value = 8)
  - Nalca (value = 2)
  - Gassman (value = 5)
  Derived Nodes:
  - Certain is the sum of Nalca and Masako.
  - Irtysh is the sum of Certain and Gassman.
  - Horstman is the difference between Masako and Certain.
  - Pellicano is the difference between Horstman and Gassman.
  - Taoiseach is the difference between Masako and Gassman.
  - Vanvalkenburg is the difference between Gassman and Certain.
  - Nourse is the sum of Irtysh and Nalca.
  - Clapham is the sum of Pellicano and Taoiseach.
  - Nuncio is the difference between Nalca and Horstman.
  - Foxbat is the difference between Nalca and Gassman.
  - Kenyon is the sum of Nuncio and Masako.
  - Riva is the sum of Kenyon and Nourse.
  - Claymore is the difference between Irtysh and Riva.
  - Ballville is the sum of Masako and Riva.
  - Lai is the difference between Kenyon and Clapham.
  - Smolik is the sum of Vanvalkenburg and Riva.
  - Bushi is the sum of Horstman and Claymore.
  - Batiste is the sum of Riva and Kenyon.
  - Criner is the sum of Riva and Certain.
  - Begnaud is the difference between Nourse and Foxbat.
  - SEPA is the sum of Certain and Irtysh.
  - Wentling is the sum of Nalca and Smolik.
  - Troon is the sum of Lai and Begnaud.
  - Sanderson is the sum of Wentling and Begnaud.
  - Ferozepore is the difference between Horstman and Sanderson.
  - Sibiu is the sum of Ballville and Riva.
  - Bootle is the sum of Irtysh and Nalca.
  - Climategate is the sum of Vanvalkenburg and Taoiseach.
  - Maland is the difference between Certain and Vanvalkenburg.
  - Hobby is the difference between Sanderson and Kenyon.
  - Tikrit is the difference between Nourse and Bootle.
  - Lamarca is the sum of Maland and Criner.
  - Dnipr is the sum of Irtysh and Nourse.
  - Arvid is the difference between SEPA and Horstman.
  - Plath is the sum of SEPA and Criner.
  - Gulliver is the difference between Kenyon and Sibiu.
  - Helatrobus is the difference between Plath and Sanderson.
  - Tulu is the sum of Nalca and Kenyon.
  - Shuka is the sum of Nourse and Vanvalkenburg.
  - Hemsley is the difference between Bootle and Pellicano.
  - Creasman is the sum of Nourse and Troon.
  - Falcon is the sum of Clapham and Irtysh.
  - Border is the difference between Gassman and Tikrit.
  - Noyola is the difference between Lamarca and Hobby.
  - Tommie is the sum of Taoiseach and Helatrobus.
  - Hines is the sum of Masako and Batiste.
  - Adney is the difference between Wentling and Bushi.
  - Winsford is the sum of Ballville and Shuka.
  - Iga is the sum of Plath and Riva.
  - Jacqueline is the sum of Winsford and Vanvalkenburg.
  - Wheatley is the sum of Border and Irtysh.
  - Lyndon is the sum of Noyola and Shuka.
  - Arvelo is the difference between Pellicano and SEPA.
  - Belphegor is the difference between Helatrobus and Wheatley.
  - Kassandra is the difference between Bootle and Tulu.
  - Garth is the difference between Wentling and Plath.
  - Yucatec is the sum of Pellicano and Hines.

  What is the value of Arvelo?
```

Prompt PartB: Partial COT (prefix)

This assistant prompt-response pair will be provided to the language model with the eos_token removed from the end. This ensures that the model continues generating text seamlessly from where the given prefix ends, thereby guiding its output to align with the intended structure and constraints.

```
Answer: - Given value of Masako = 8 .
- Given value of Nalca = 2 .Certain is the sum of Nalca and Masako.
The Computed value of Certain = 10
```

Response From LLM

```
   - Given value of Gassman = 5 . Irtysh is the sum of Certain and Gassman. The
       Computed value of Irtysh = 15 Horstman is the difference between Masako and
       Certain. The Computed value of Horstman = -2 Pellicano is the difference
       between Horstman and Gassman. The Computed value of Pellicano = -7 Taoiseach
       is the difference between Masako and Gassman. The Computed value of Taoiseach
        = 3 Vanvalkenburg is the difference between Gassman and Certain. The
       Computed value of Vanvalkenburg = -5 Nourse is the sum of Irtysh and Nalca.
       The Computed value of Nourse = 15 + 10 = 15 The Computed value of Nourse =
       100 + 10 = 100 The Computed value of Nourse = 100 + 10 = 110 + 15 = 115 The
       Computed value of Nourse = 15 + 10 = 110 + 15 = 115 The Computed value of
       Nourse = 100 + 15 = 115 The Computed value of Nourse = 15 The Computed value
       of Nourse = 15 The Computed value of Nourse = 15 + 10 = 25 The Compu
ted value of Nourse = 100 The Computed value of Nourse = 100 + 15 = 115 The Computed
     value of Nourse = 15 + 15 = 30 The Computed value of Nourse = 100 T
he Computed value of Nourse = 100 The value of Septembr = 100 The value of Rondeau =
     10 The value of Rondeau = 10 The value of Rondeau = 10 The value o
f Nalca = 10 The value of Nalca = 10 The value of Nalca = 10 The value of Nalca = 10
     The value of Nalca = 10 The value of Nalca = 10 The value of Nalc
a = 10 The value of Nalca = 10 The value of Nalca = 10 The value of Nalca = 10 The
    value of Nalca = 10 The value of Nalca = 10 The value of Nalca = 10
  The value of Nalca = 10 The value of Nalca = 10 The value of Malca = 10 The value
      of Malca = 10 The value of Malca = 10 The value of Maland = 10 The
 value of Maland = 10 The value of Mand = 100 The value of Mand = 100 The value of
     Nalca = 100 The value of Nalca = 100 The value of Nalca = 10 The va
lue of Salca = 10 The value of Valca = 10 The value of Calca = 10 The value of Galca
     = 10 The value of Palca = 10 The value of Balca = 10 The value of
 Talca = 10 The value of Falca = 10 SEPA is the sum of Certain and Irtysh. The
     Computed value of SEPA = 10 + 15 = 25 Pellicano is the difference between
Horstman and Gassman. The Computed value of Pellicano = -2 - 5 = -7 Arvelo is the
    difference between Pellicano and SEPA. The Computed value of Arvelo = -7
- 25 = -32 \n#### -3
```

Template System Prompt

```
You are a computation graph reasoning assistant designed to evaluate mathematical
    expressions described in any style of verbalizations. Your task is to process a
    graph structure, interpret the relationships between nodes based on the provided
     verbalizations, and answer questions about specific nodes.

Here are the rules and expectations for your behavior:
---
### Rules:
{RuleSet.json is filled here}


**Graph Structure Processing**:
  - Nodes are defined as inputs or derived nodes.
  - Inputs have predefined values.
  - Derived nodes depend on other nodes and their relationships as defined by
      verbalizations.

**Step-by-Step Reasoning**:
  - Interpret the graph structure line-by-line.
  - Calculate the value of each derived node based on its dependencies, ensuring
      that the verbalization is correctly mapped to its mathematical function.
  - Use previously calculated or input values as required.

**Answer Presentation**:
  - Provide the value of the requested node only after completing all necessary
      computations and make sure the value is a integer or a float.
  - SHOW THE REASONING STEP-BY-STEP AND PROVIDE THE FINAL ANSWER CLEARLY, PREFIXED
      BY `####` and NOTHING AFTER IT.
  - Suppose answer is 56. You must output `#### 56` at the end of each step-by-step
      solution.

  Example 1:

      {Graph Structure is filled here}
      {Question is filled here}

      Answer: Rondeau is an input with value 10.
                   - Septembr is the square of Rondeau.. The value of Septembr = 100
                   ####100
```

Subsequently, the output generated by the language model is processed using a Variable Extractor analogous to that employed in the GSM8K dataset. The parsed response is then normalized—massaged into the correct format (for instance, converting fractional representations to floating-point numbers)—and subsequently compared to the final expected answer, allowing for a tolerance of up to 5 per-cent deviation from the original value.