DEDUCE: DEDUCTIVE CONSISTENCY AS A FRAMEWORK TO EVALUATE LLM REASONING

Atharva Pandey*, **Kshitij Dubey***, **Rahul Sharma & Amit Sharma**
Microsoft Research India
{t-atpandey, t-ksdubey, rahsha, amshar}@microsoft.com

ABSTRACT

Despite great performance on Olympiad-level reasoning problems, frontier large language models can still struggle on high school math. We study the nature of language models' (LM) reasoning by analyzing their chain-of-thought traces. To avoid memorization issues, we present a framework that can evaluate reasoning of LMs over novel, perturbed versions of benchmark problems. Formally, we compare LMs to ideal deductive reasoners that given a set of premises, can provide valid conclusions over any number of reasoning hops. To assess reasoning performance beyond final accuracy, we introduce *deductive consistency*, a metric that evaluates the correctness of system's reasoning across varying input premise lengths and the number of solution hops. Using this metric, we examine potential explanations for language models' failures on novel problems. Through experiments on GSM8K and a synthetic dataset, we find that the failure is not primarily due to shifts in language style or the propagation of early errors. Instead, it stems from a fundamental limitation: as the number of reasoning hops increases, language models exhibit a decline in deductive consistency, which was masked by memorization for existing benchmark problems. Our analysis provides a new view to characterize LM reasoning—as computations over a window of input premises and reasoning hops—that can provide unified evaluation across problem domains.

1 INTRODUCTION

Chain-of-thought prompting, the practice of instructing a language model (LM) to output its intermediate steps before the final answer, has led to significant gains on reasoning tasks such as math Wei et al. (2024), logic Saparov & He (2023); Saparov et al. (2023); Parmar et al. (2024b), and language tasks Suzgun et al. (2022). Recent work shows that models can also solve Olympiad-level problems Gao et al. (2024).

However, a parallel stream of work shows that LLMs are sensitive to simple perturbations of the original question, such as changing the numeric values occurring in grade school word problems Mirzadeh et al. (2024); Srivastava et al. (2024). Importantly, these perturbation do not change the difficulty level of a problem, yet accuracy of frontier LLMs such as GPT-4 significantly reduces. Other studies show a similar *reasoning gap* between original and perturbed problems covering math, logic Wu et al. (2024) and syllogisms Lewis & Mitchell (2024b), but why this gap happens is less explored.

In this work, we analyze the reasoning capabilities of language models (LMs) by assessing their consistency with an ideal deductive reasoner. To this end, we introduce *deductive consistency*, a metric that evaluates an AI system's reasoning validity based on its intermediate steps. We conceptualize a reasoning problem as comprising a set of premises and a target predicate or variable. Given a set of inference rules, the goal of a reasoning system is to determine whether the target predicate can be derived from the input premises (or infer the value of the target variable). An ideal deductive reasoner operates by systematically deriving the correct reasoning steps from *any* given set of premises, eventually reaching the target predicate if it is logically deducible. Our key insight is to approximate this ideal behavior by intervening on partial chain-of-thought traces of an LM. Specifically, for a given reasoning problem, we provide the LM with the initial premises and append the first few correct

*Equal Contribution

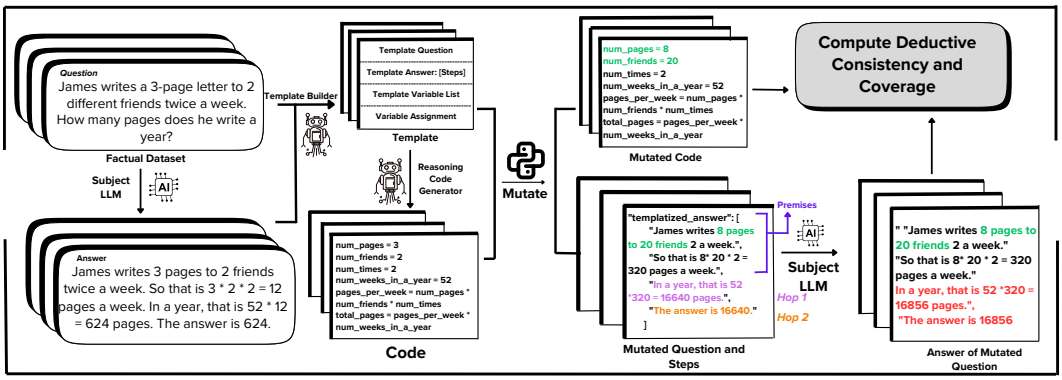

Figure 1: Pipeline for Deductive Consistency Evaluation. Our method requires access to a single valid solution to compute deductive consistency across multiple premises and reasoning hops. Given a benchmark problem, we use pre-trained LMs to templatize its solution and obtain an executable code solution. 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.

reasoning steps before allowing it to continue its generation. This approach enables us to measure consistency with an ideal deductive reasoner across different input premises and reasoning hops, offering a more fine-grained evaluation of LM reasoning beyond final accuracy.

Importantly, our metric requires only a single valid solution for a novel problem. To investigate the reasoning gap, we construct *novel* problems by modifying existing benchmark problems, such as altering variable values. We introduce a pipeline leveraging pre-trained language models (LMs) to generate correct solution steps for these novel problems. As shown in Figure 1, the pipeline first produces a chain-of-thought solution for the original benchmark problem, then translates it into a symbolic representation (Python code). We apply multiple correctness checks to filter out original problems that lack a corresponding code representation. For each novel problem, we update the Python code to reflect the new variable values, then convert the modified code back into natural language to generate the correct solution steps, which in turn can be input as additional premises to a target LM. We refer to this framework as DeduCE—Deductive Consistency Evaluation of LMs.

We apply the proposed framework to evaluate multiple LMs on grade school math problems, using GSM8K and a synthetic dataset. As shown by past work, there is a significant drop in final accuracy on GSM8K between original and mutated problems. Our main findings are:

- Deductive consistency of LMs is robust to the length of the input premises for GSM8K. It is sensitive to the number of reasoning hops. As the number of reasoning hops increases from 1 to 5, deductive consistency falls by 15-30% (Figure 2) . Note that this effect was likely masked in original benchmark problems by memorization.
- Post-training methods like supervised fine-tuning and reinforcement learning enhance task-specific patterns rather than general deductive reasoning, even for the same underlying reasoning task.
- Other hypotheses, such as novel problems inducing a different language style in the solution or early errors that propagate, are unable to explain the difference in final accuracy between original and mutated problems. The maximum deviation seen is nearly 5% (Figure 6)

2 Related Work

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), and many other diverse tasks Wu et al. (2024).

Metrics for reasoning beyond final answer's accuracy. Xu et al. (2024) four different types of metrics based on answer correctness and explanation correctness. Seals & Shalin (2024) test deductive reasoning based on logical questions. We aim to provide a general metric for any reasoning task. 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).

Deductive reasoning in LLMs. LogicBench evaluates various models on natural language problems over propositional, first order, and non-monotonic logic Parmar et al. (2024a). Other examples include analyzing categorical syllogisms Zong & Lin (2024) and proving theorems in intuitionistic propositional logic An et al. (2024).

3 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 $\boldsymbol{P} \subset \mathcal{L}$ denote the set of premises and $\text{Th}(\boldsymbol{P}) \subseteq \mathcal{L}$ the deductive closure of $\boldsymbol{P}$ under $\mathcal{R}$. We assume access to a dataset of problems $(\boldsymbol{P}_j, t_j)_{j=1}^{M} \sim \mathcal{D}$ where P_j denotes the set of premises and t_j the target predicate to be proved in each problem j. Total number of problems in the dataset being M.

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

3.1 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 $d \sim \mathcal{D}$, where $d = (\boldsymbol{P}, t)$, we generate a proof $\mathcal{A}(\boldsymbol{P}, t)$ using $\mathcal{A}$. A simple way of measuring reasoning performance may be to compute per-predicate accuracy. For each $X_i \in \mathcal{A}(\boldsymbol{P}, t)$: $Cons(d_j) = \frac{\sum_i I_{X_i}}{|\mathcal{A}(\boldsymbol{P}_j, t_j)|}$ 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 $\boldsymbol{P}$ and $\boldsymbol{X}_k : \{k : 1, 2, ..i-1\}$. For the example proof above, the metric will be 0.67 because the second predicate is incorrect.

A key part of deductive reasoning is to process multiple input premises and determine the next correct predicates. 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 $\boldsymbol{P}$. The input premises now become $\boldsymbol{P}' = \boldsymbol{P} \cup \mathcal{S}_k(\boldsymbol{P}, t)$ where $\mathcal{S}_k(\boldsymbol{P}, t)$ 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 $\mathcal{A}(\boldsymbol{P} \cup \mathcal{S}_k(\boldsymbol{P}, t), t)$. The reasoning system's goal is to complete the proof.

$$Cons(d_j, k) = \frac{\sum_{i=k+1}^{N} I_{X'_i}}{|\mathcal{A}(\boldsymbol{P}'_j, t_j)|}$$

Continuing our example with $k = 1$, the reference proof system may add the first step, "$A \rightarrow C$" and let the target system $\mathcal{A}$ complete the rest. Here, the system may produce a faulty proof as before, $A \rightarrow C; A \rightarrow E; A \rightarrow G$. However, with $k = 2$ and adding the first two steps "$A \rightarrow C; A \rightarrow F$", the system $\mathcal{A}$ may produce $A \rightarrow C; A \rightarrow F; A \rightarrow G$, 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, denoting the number of inference rules (steps) until which we evaluate the reasoning system.

$$DedCons(k, l) = \frac{\sum_{d_j \sim \mathcal{D}} I_{X'_{k+l+1,j}}}{M}$$

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 allows us to test a diverse set of deductive tasks even from a small number of problem instances.

3.2 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 $\mathcal{S}(\boldsymbol{P}, t)$ given a problem. Therefore, we now assume access to a reference proof system that given a set of premises $\boldsymbol{P}$ and a target predicate t, can generate a proof involving predicates $\mathcal{S}(\boldsymbol{P}, t) \subseteq \text{Th}(\boldsymbol{P})$, 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, $A \rightarrow C; A \rightarrow F; A \rightarrow G$, but cannot assess the validity of a predicate outside it such as $A \rightarrow D$.

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

$$DedCons(k, l) = \frac{\sum_{d_j \sim \mathcal{D}} I_{Z'_{k+l+1} \in \mathcal{A}(\boldsymbol{P}'_j, t_j)}}{M}$$

where the numerator is an indicator function checking whether a given predicate $Z_i \in \mathcal{S}(\boldsymbol{P}, t)$ 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, if a reasoning system produces a valid proof, $A \rightarrow D; A \rightarrow F; A \rightarrow G$, it will not have consistency=1 because the first step $A \rightarrow D$ is not a part of the reference solution $\mathcal{S}(\boldsymbol{P}, t)$.

In such cases, the consistency metric above can under-estimate 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 $\mathcal{S}(\boldsymbol{P}, t)$ inferred by $\mathcal{A}$. Let $\boldsymbol{V}_{\mathcal{S}(\boldsymbol{P}, t)}$ be the variables included in the reference solution. Then coverage is

$$Coverage = \frac{\sum_{d_j \sim \mathcal{D}} |\boldsymbol{V}_{\mathcal{S}(\boldsymbol{P_j}, t)} \cap \boldsymbol{V}_{\mathcal{A}(\boldsymbol{P_j}, t)}|}{\sum_{d_j \sim \mathcal{D}} |\boldsymbol{V}_{\mathcal{A}(\boldsymbol{P_j}, t)}|}$$

Thus, coverage measures how reliably verification of the set of predicates V measures consistency. When the coverage is high, **an ideal reasoning system's deductive consistency should be a constant close to 1, independent of the number of premises k and the number of hops l.**

4 Evaluating Deductive Consistency for LLMs

As noted above, we need at least one reference solution for a reasoning problem to evaluate deductive consistency. Given a benchmark reasoning dataset, we now provide a method to obtain such solutions and evaluate deductive consistency.

We use auxiliary expert LMs to help with transformation tasks. Specifically, we use **Code Generation LM** that generates executable reasoning graphs (Python Code), a **Templatization LM** that defines variable templates, and a **Variable Extraction LM** (Parser) that extracts predicate values for evaluation. The **Subject LM** is the model under evaluation. The entire pipeline is shown in Figure 1.

4.1 GENERATING CORRECT SOLUTION FOR A BENCHMARK PROBLEM

While generating the correct solution for a reasoning problem is hard, an expert LM may be able to generate candidate solutions. In particular, its capability for generating good candidate solutions may be higher for the original benchmark problem. We therefore adopt a *generate-then-verify* approach to select problems on which we can obtain a correct solution. We employ specific LMs to produce Code and templatized COT. The templatized COT is generated first, followed by the code. To build confidence in our candidate solution, we rely on the concept of internal consistency. We represent the solution in semantically equivalent forms, such as code and a templatetized version of the CoT, and perform multiple sanity checks to ensure its correctness.

One key sanity check involves verifying the equivalence between the code representation and the templatized Chain of Thought (tCoT). The variables in the code and the placeholders in the COT are equivalent. We validate this by ensuring that, given the same factual input, both representations yield identical variable values at each step. Once confirmed consistent with each other, the code serves as a symbolic representation of both the problem and its correct solution.

4.2 GENERATING A NOVEL PROBLEM BY PERTURBING THE BENCHMARK PROBLEM

Given the reasoning gap between existing benchmark problems and novel problems created by perturbation, we evaluate deductive consistency on novel problems only. This avoids any memorization concerns. In the current work, 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.

Using the template generated earlier, we sample new values for the variables in the template. These sampled values serve as inputs to the reasoning code, which computes the corresponding solution for the novel problem. We substitute the computed values back into the template Chain of Thought (tCoT) and template Question (tQ) to produce modified questions (Q') and their reasoning steps (COT'). COT' provides a detailed, step-by-step chain of reasoning for evaluation.

4.3 EVALUATING DEDUCTIVE CONSISTENCY

We evaluate deductive consistency by generating a perturbed question (Q') by modifying the seed premises and deriving its reasoning steps (COT) using the templatized Chain of Thought (tCoT). We then substitute intermediate premises into placeholders in tCoT by evaluating the Code with the same premises that generated the perturbed questions.

Through this process, we traverse a computation graph that serves as a proof Directed Acyclic Graph (DAG). Each link in the DAG represents a reasoning step, analogous to the transitivity of premises in a formal proof. For example, the sequence A $\rightarrow$ C; A $\rightarrow$ F; A $\rightarrow$ G encompasses the intermediate steps required to reach the target premise or the variable of interest.

We define a "hop" as the number of edges needed to progress from a specific premise to the target premise. Conversely, we define a "prefix" as the number of premises we provide to the model in advance (k). Using the template, we supply the LLM with k steps worth of premises and then prompt it to solve the remaining steps—l hops—needed to arrive at the final conclusion. By varying k and l, we can systematically evaluate the model's performance across different levels of partial information and reasoning depth.

4.4 REASONING EVALUATION ON SYNTHETIC DATASETS

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.

Furthermore, it becomes straightforward to craft questions that traverse designated nodes in a prescribed order. For example, a path can be orchestrated to move from the initial premise to an intermediate node A ("prefix k") and then from node A to the target node B ("hop l"), such that

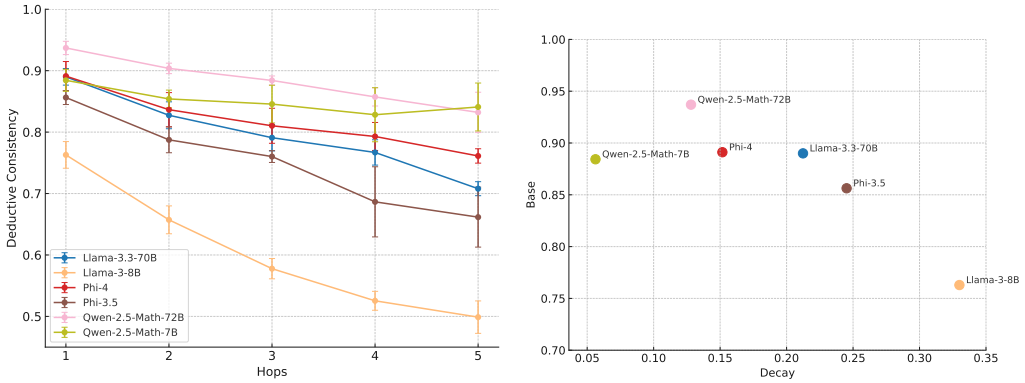

Figure 2: **Left:** Deductive Consistency vs. Reasoning Hops across models. **Right:**Base deductive consistency vs. Decay.The premises are set using the Original paraphrase setting.

Distance(seed premise, 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.

Dataset Generation 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 (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. Details are present in Appendix subsubsection A.5.2.

5 Results: Math reasoning on GSM8K

5.1 Evaluation Setup

Dataset Statistics. A subset of 1000 questions from GSM8K is randomly chosen. The responses of LMs under evaluation are filtered as described in section 4. 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.

LMs in DeduCE pipeline. We use LLama-3-70B-Instruct LM as the templatizer, code generation and variable extractor. We find that LMs such as LLama-3-70B-Instruct are reasonably capable at templatization, obtaining 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.

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. All models used in this paper are Instruct tuned. Model suffixes will be truncated in plots and tables to maintain readability.

5.2 HYPOTHESES

Based on the deductive consistency metric, we can formulate some hypotheses on why final accuracy decreases for novel math questions, as reported in past work Mirzadeh et al. (2024); Srivastava 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. **H3**: LMs have a significant decay in reasoning ability as the number of premises or hops increase. Novel problems expose this property since they are not affected by memorization.

To decide between hypotheses, 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 question. Other are different paraphrases of the original style.

- 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.*

5.3 DEDUCTIVE REASONING DECAYS WITH NUMBER OF HOPS

Coverage Table 4 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.

Deductive consistency as a function of Hops Mean deductive consistency is computed by averaging predicate consistency across prefixes for a given hop, considering only premises with sample counts above a threshold. We include only hops where the ratio of single-premise samples for the given hop to those with premise-length of 1 hop-1 exceeds 20%, ensuring sufficient data for reliable estimates. Our findings reveal that consistency remains high for the first hop, contradicting **H1** by demonstrating that models correctly answer the first step.

Decay in Deductive Consistency The second key result is that consistency decreases as hops increase. We characterize this decay as negative of slope of the best fit line in Figure 2 with hops normalized between 0 and 1 as well as base refers to the 1-Hop deductive consistency of the reasoning model. An ideal model must achieve zero decay and base value of one. We find 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 deductive consistency as the number of hops is increased. 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 high decay value when compared to other models (also see Figure 2).

Importantly, deductive consistency does not vary much as the length of input premises are changed. We observe lower variance in mean deductive consistency as a function of prefix as seen in Figure 8 . So we have partial evidence of **H3**: 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.

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 **H1**. For detailed error analysis refer subsection A.4

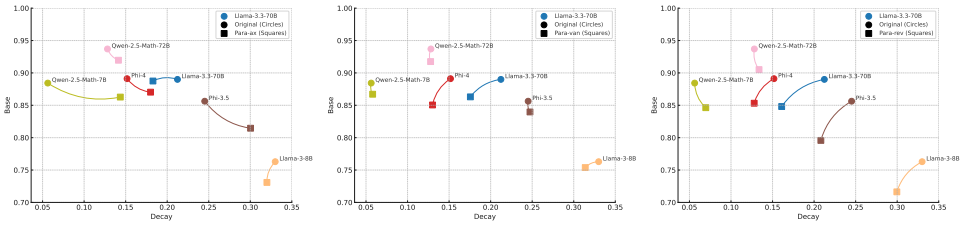

Figure 3: Comparison of Base vs. Decay Across Different Styles. Model name suffixes have been dropped for readability

5.4 IMPACT OF LANGUAGE STYLE

We observe slight decrease in base values across models due to paraphrasing in Figure 3. 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).

6 RESULTS: EVALUATION ON A SYNTHETIC DATASET

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. All models are Instruct tuned.

6.1 RESULTS ON SYNDEDUCT

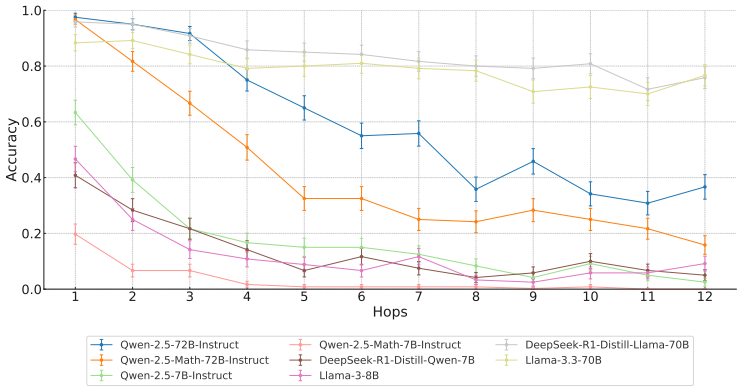

Figure 4: Accuracy v/s Hops. Each Hop bin has roughly same variation of Prefixes. Y-Ticks on each data-point is Standard Error for Accuracy. Every model is Instruct tuned.

The general trend of decreasing accuracy over hops supports our results in GSM8K. Our findings in Figure 4 suggests larger models demonstrate greater resilience to increases in the number of hops, while smaller models—such as Llama-3-8B-Instruct—experience a substantial drop in performance. One contributing factor may be the larger maximum token limit, which allows these models to accommodate more extensive reasoning chains. However, for queries with increasingly longer chains, the input size expands significantly, risking the approach of token length limits and thereby degrading performance. This trend becomes evident when examining accuracy versus prefix length across multiple hops: as prefix segments grow, the overall task accuracy declines. (Table 18, Table 19)

7 ABLATIONS

7.1 DISTILLATION AND RL TUNED MODELS

To better understand the impact of different fine-tuning strategies on deductive reasoning, we conduct an ablation study comparing base models and their post-tuned counterparts across synthetic (SynDeduct) and real-world (GSM8K) benchmarks. We focus on two primary fine-tuning approaches: iterative fine-tuning with reinforcement learning (RL-training) and supervised fine-tuning (SFT). By evaluating reasoning accuracy and decay patterns, we aim to disentangle the effects of these strategies on deductive performance across different reasoning tasks.

For RL-based and iterative fine-tuning models, we analyze Qwen-2.5-Math-Instruct in both its 7B and 72B variants Yang et al. (2024), comparing them against their respective base models (Qwen-2.5). Similarly, for SFT-based tuning, we assess models trained on derivational traces from a reasoning model R1 (DeepSeek-AI et al., 2025), including Meta Llama 3.3-70B-Instruct and Qwen-2.5-Math-7B-Instruct as base models, with their distilled counterparts being R1-distill-Llama and R1-distill-Qwen. Our evaluations span SynDeduct (Figure 5), which measures deductive accuracy at each reasoning step, and GSM8K, where we analyze "reasoning decay" to assess robustness in real-world mathematical reasoning.

Our findings highlight contrasting effects of these post-training strategies. SFT-based models exhibit improved performance on SynDeduct, enhancing deductive consistency over their base models. However, they perform worse on GSM8K (Table 2), as their reasoning decay increases despite maintaining base deductive accuracy. Conversely, RL-trained models (Table 1) improve reasoning decay in GSM8K but suffer significant performance drops on SynDeduct, suggesting that RL enhances real-world robustness at the cost of structured deductive reasoning. These results indicate that neither strategy universally improves reasoning; rather, they specialize models for distinct distributions of problems. Given that both SynDeduct and GSM8K primarily involve basic arithmetic, this further suggests that post-training primarily aligns models with task-specific patterns rather than strengthening general deductive reasoning.

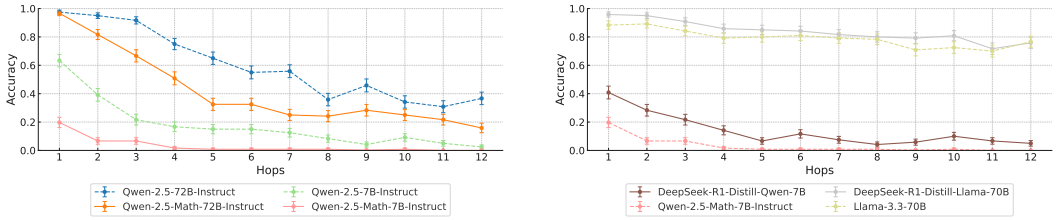

Figure 5: Results for SynDeduct. **Left:** Accuracy vs. Hops for RL-tuned math models versus their base models, **Right:** Accuracy vs. Hops for distilled math models versus their base models. The dashed lines represent the models base models and solid lines represent models after post-training technique. All models are Instruct Tuned

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	0.16185
Qwen-2.5-Math-7B-Instruct	0.8427	0.1189

Table 1: Ablation for RL 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 2: Ablation for Distillation post training on gsm8k

8 CONCLUSION

We propose a metric for evaluating AI deductive reasoning using only text-based reasoning traces, enabling broad domain applicability.Our experiments reveal contrasting fine-tuning effects on synthetic (SynDeduct) and real-world (GSM8K) benchmarks, highlighting key trade-offs.

IMPACT STATEMENT

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 impact on downstream applications by avoiding reasoning bugs.

REFERENCES

Chenyang An, Zhibo Chen, Qihao Ye, Emily First, Letian Peng, Jiayun Zhang, Zihan Wang, Sorin Lerner, and Jingbo Shang. Learn from failure: Fine-tuning LLMs with trial-and-error data for intuitionistic propositional logic proving. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024. URL https://arxiv.org/abs/2410.07985.

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

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

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

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024a.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Towards systematic evaluation of logical reasoning ability of large language models. *arXiv preprint arXiv:2404.15522*, 2024b.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=qFVVBzXxR2V.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Seyed Mehran Kazemi, Najoung Kim, and He He. Testing the general deductive reasoning capacity of large language models using OOD examples. *CoRR*, abs/2305.15269, 2023. doi: 10.48550/arXiv.2305.15269. URL https://doi.org/10.48550/arXiv.2305.15269.

S Seals and Valerie Shalin. 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.

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

Saurabh Srivastava, Anto PV, Shashank Menon, Ajay Sukumar, Alan Philipose, Stevin Prince, Sooraj Thomas, et al. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *arXiv preprint arXiv:2402.19450*, 2024.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 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.

Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are large language models really good logical reasoners? a comprehensive evaluation and beyond, 2024. URL https://arxiv.org/abs/2306.09841.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Zhongshen Zeng, Pengguang Chen, Shu Liu, Haiyun Jiang, and Jiaya Jia. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation, 2024. URL https://arxiv.org/abs/2312.17080.

Zhehao Zhang, Jiaao Chen, and Diyi Yang. Darg: Dynamic evaluation of large language models via adaptive reasoning graph, 2024.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. Dyval: Dynamic evaluation of large language models for reasoning tasks. In *The Twelfth International Conference on Learning Representations*, 2023.

Shi Zong and Jimmy Lin. Categorical syllogisms revisited: A review of the logical reasoning abilities of LLMs for analyzing categorical syllogisms. In Lotem Peled-Cohen, Nitay Calderon, Shir Lissak, and Roi Reichart (eds.), *Proceedings of the 1st Workshop on NLP for Science (NLP4Science)*, 2024.

A APPENDIX

A.1 DETAILS FOR GSM8K PIPELINE

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.2.

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.2) 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.2). 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).

A.2 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  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  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.3 RESULTS - GSM8K

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

Table 3: 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

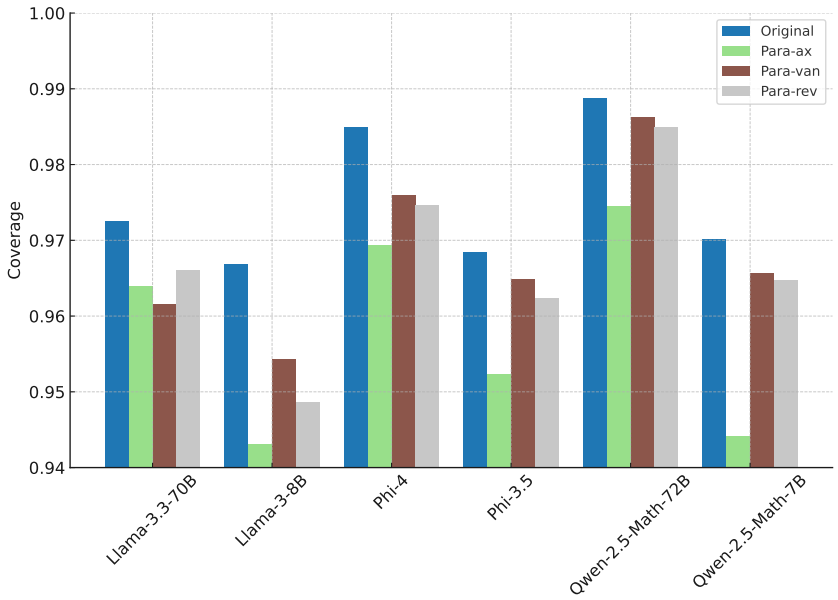

Figure 6: Coverage Comparison Across Interventions (Adjusted Y-Limits)

Table 4: 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 5: 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 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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 12: 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 13: 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

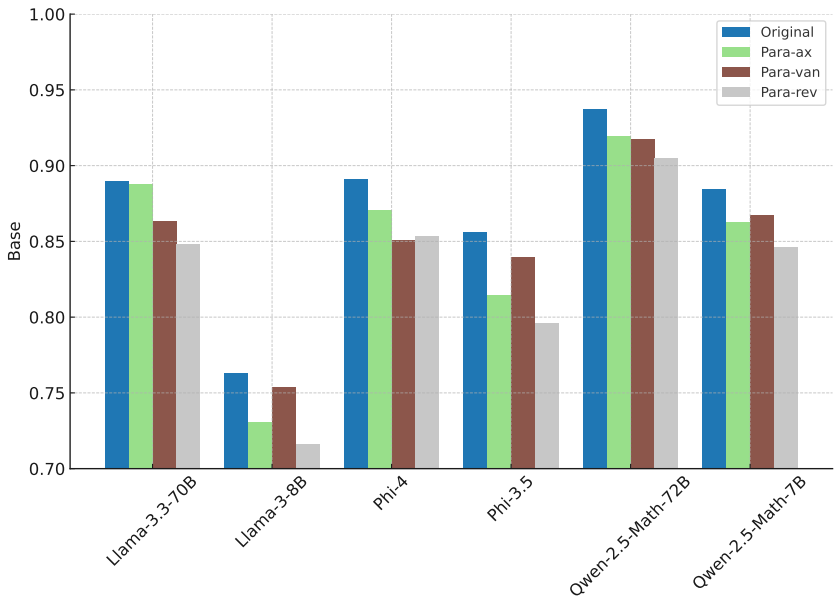

Figure 7: Base Comparison Across Interventions

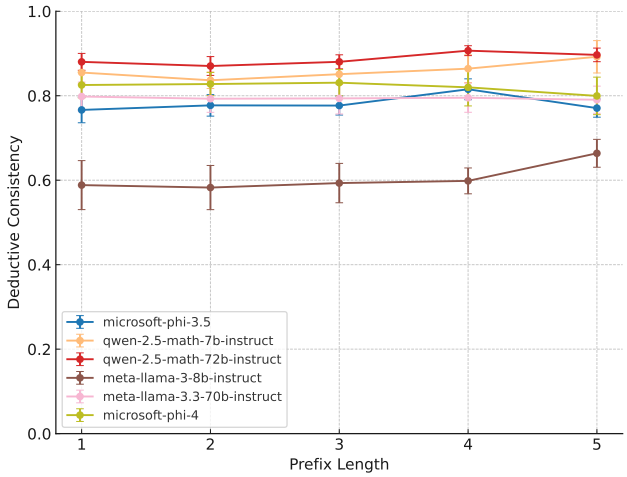

Figure 8: Deductive Consistency vs. Prefix Length for Different Models

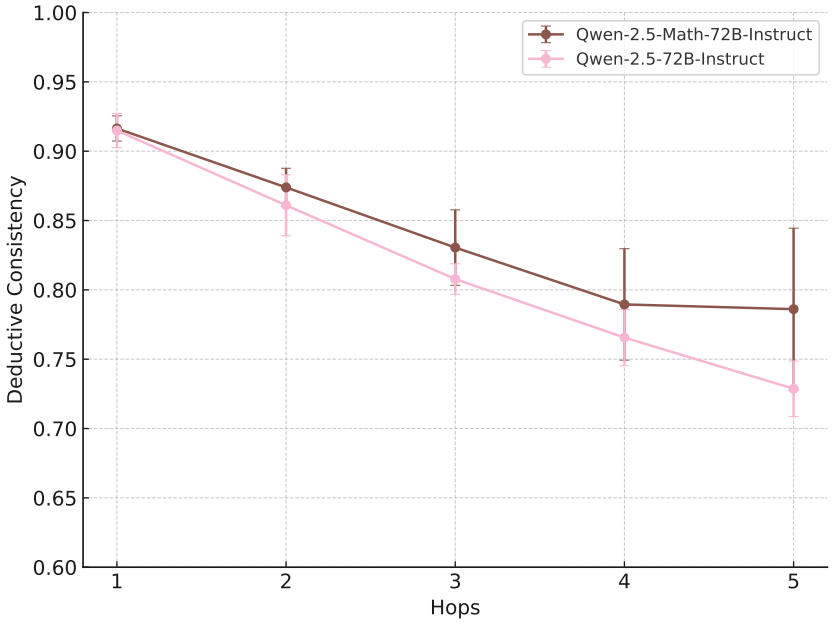

Figure 9: Deductive Consistency vs. Hops for Ablation 1

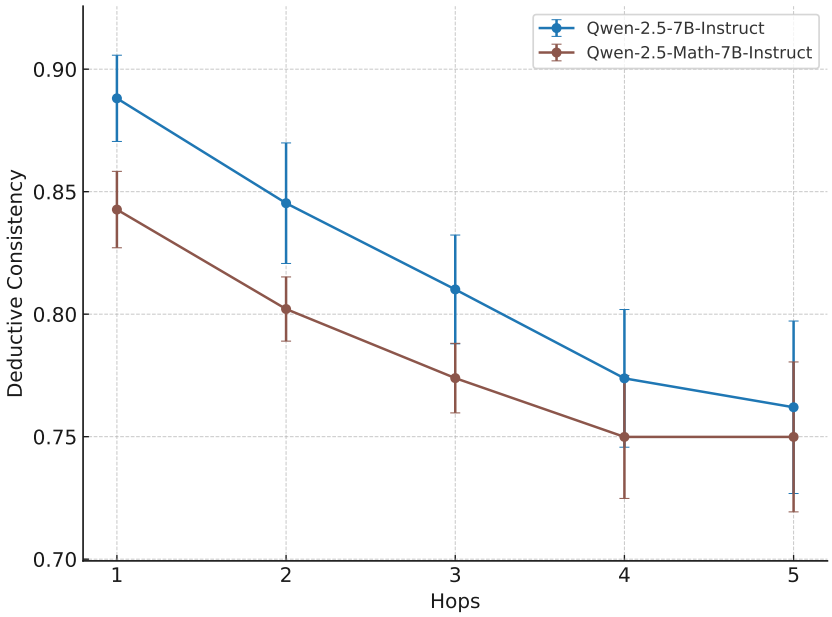

Figure 10: Deductive Consistency vs. Hops for Ablation 2

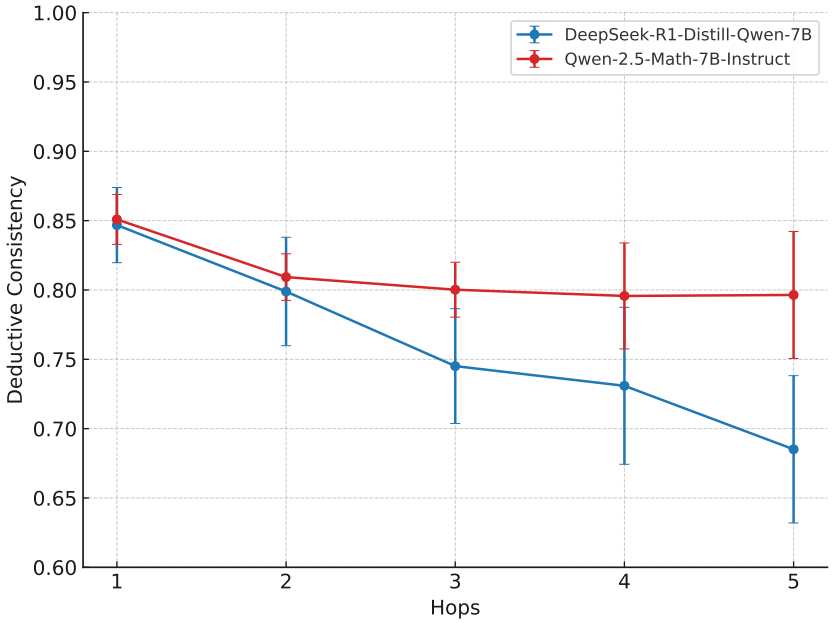

Figure 11: Deductive Consistency vs. Hops for Ablation 3

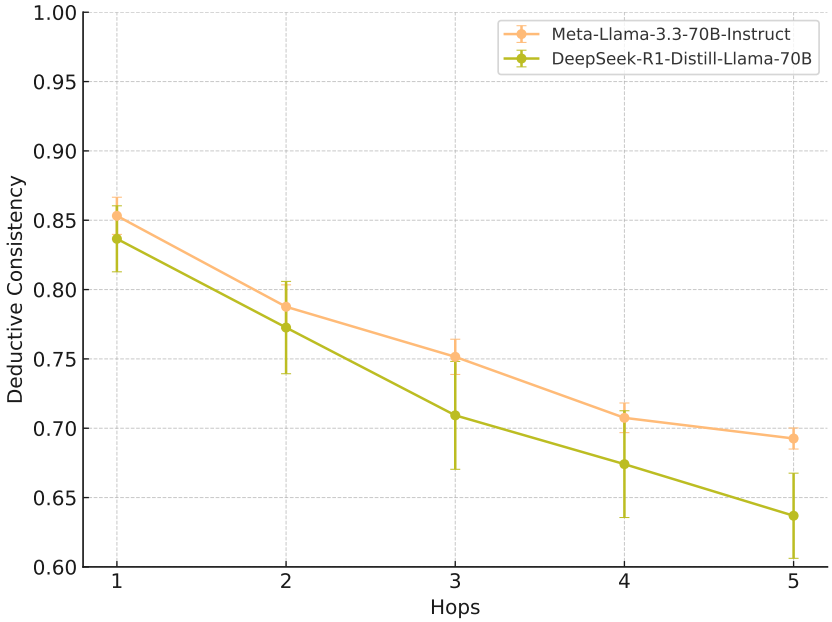

Figure 12: Deductive Consistency vs. Hops for Ablation 4

A.4 ERROR ANALYSIS ON GSM8K

In this section, we perform a qualitative and quantitative analysis of errors that arise in the response models.

Evaluation method. Model responses are grouped into five groups based on final accuracy on the mutated GSM8K dataset. Accuracy here is computed over the set of mutated questions for each question in the original benchmark. The groups are;

- Group-1 : Accuracy $= 1$
- Group-2 : $1 <$ Accuracy ≤ 0.7
- Group-3 : $0.7 <$ Accuracy ≤ 0.4
- Group-4 : $0.4 <$ Accuracy < 0
- Group-5 : Accuracy $= 0$

Error Categories. We use GPT-4o as an evaluator to classify the errors as calculation errors, Logical errors and Understanding errors. Calculation errors refer to errors due to calculation mistakes such as wrongly calculating a product of two numbers, as well as errors in rounding, along with error propagated due to calculation mistakes (to make sure we don't over count the calculation error). Logic error refers to errors in which model applies wrong logic or rule or formula. Understanding errors refers to errors in which model wrongly assumes a fact or contradicts a fact mentioned in the question. These errors are seen in cases where the question mentions scenarios that are far from real world such as cars having 100 wheels or there being 97 days in a week. In 15 we report error rate, defined as the frequency of error normalized by number of error responses in that group.

Observation and Findings. A higher proportion of calculation errors is observed relative to logical and comprehension errors. These calculation errors predominantly emerge during multiplication operations within the chain-of-thought, and they propagate through subsequent reasoning steps.

Furthermore, models exhibit a bias toward reproducing the original reasoning graph from the benchmark solution (hinting at pre-training bias). This show weak robustness to changes in reasoning structure. Additionally, logical errors frequently stem from ambiguities in the phrasing of questions. 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
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 14: Number of instances for different models across dataset groups.

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

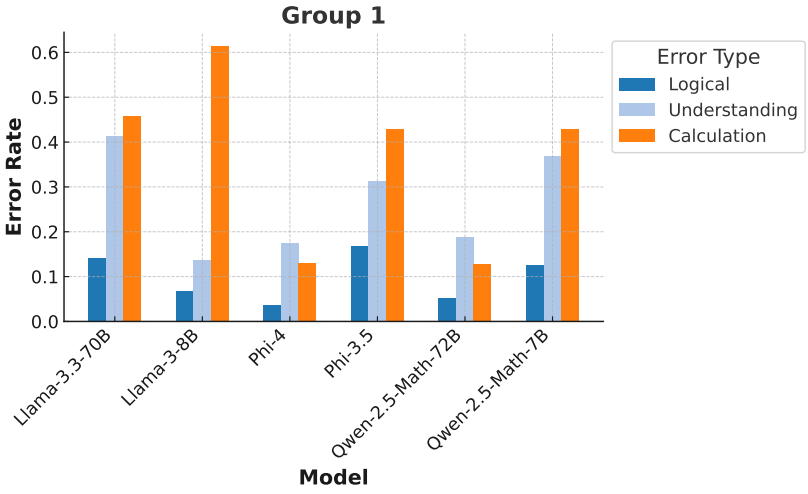

Figure 15: Error rate plot for Group 1

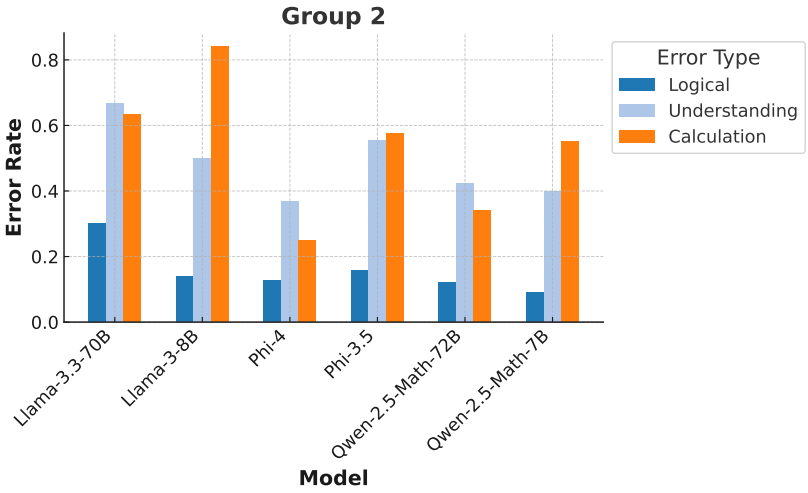

Figure 16: Error rate plot for Group 2

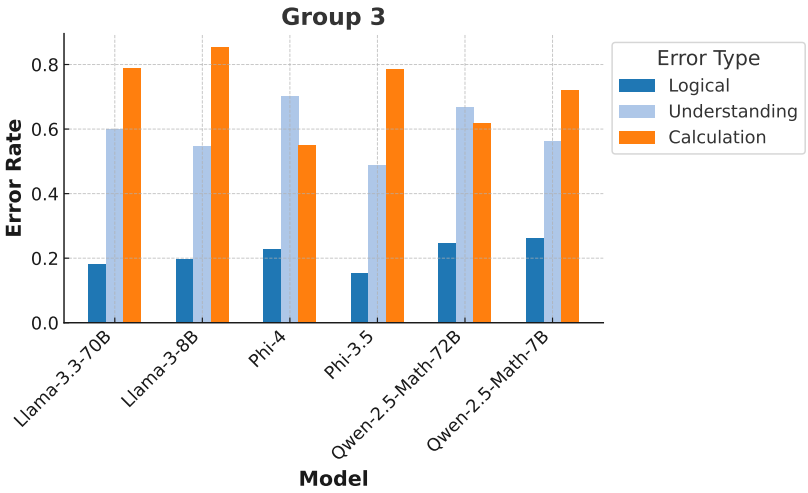

Figure 17: Error rate plot for Group 3

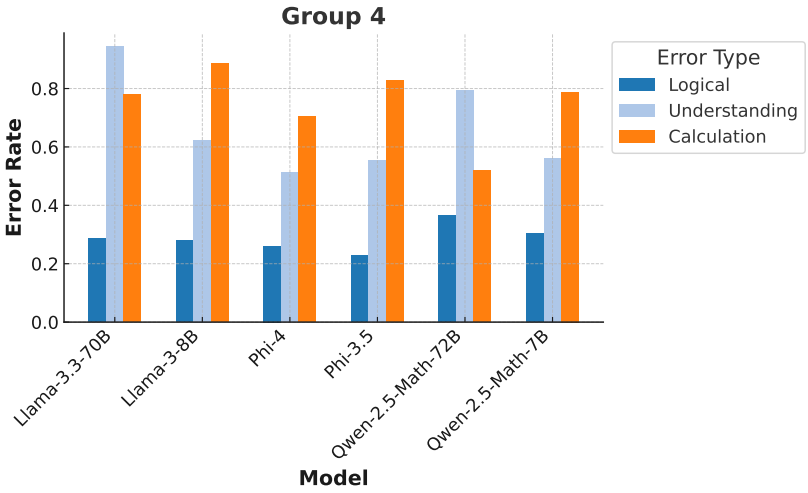

Figure 18: Error rate plot for Group 4

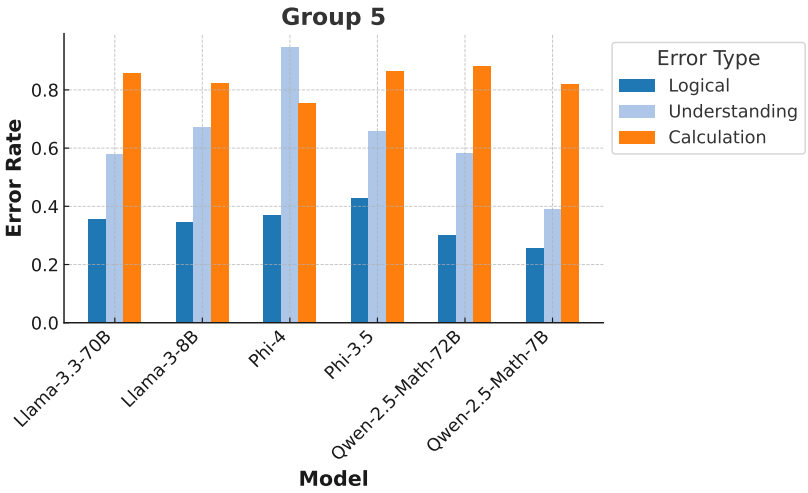

Figure 19: Error rate plot for Group 5

A.5 SYNDEDUCT

A.5.1 SYNDEDUCT RESULTS

Table 16: Accuracy ± Standard Error for Hops 1-6

Model	Hop 1	Hop 2	Hop 3	Hop 4	Hop 5	Hop 6
Qwen-2.5-72B	0.975 ± 0.0143	0.950 ± 0.0199	0.9167 ± 0.0252	0.750 ± 0.0395	0.650 ± 0.0435	0.550 ± 0.0454
Qwen-2.5-Math-72B	0.9667 ± 0.0164	0.8167 ± 0.0353	0.6667 ± 0.0430	0.5083 ± 0.0456	0.325 ± 0.0428	0.325 ± 0.0428
Qwen-2.5-7B	0.6333 ± 0.0440	0.3917 ± 0.0446	0.2167 ± 0.0376	0.1667 ± 0.0340	0.150 ± 0.0326	0.150 ± 0.0326
Qwen-2.5-Math-7B	0.1971 ± 0.0363	0.0667 ± 0.0228	0.0667 ± 0.0228	0.0167 ± 0.0117	0.0083 ± 0.0083	0.0083 ± 0.0083
DeepSeek-R1-Distill-Qwen-7B	0.4083 ± 0.0449	0.2833 ± 0.0411	0.2167 ± 0.0376	0.1417 ± 0.0318	0.0667 ± 0.0228	0.1167 ± 0.0293
Llama-3-8B	0.4667 ± 0.0455	0.2500 ± 0.0395	0.1417 ± 0.0318	0.1083 ± 0.0284	0.0883 ± 0.0259	0.0667 ± 0.0228
DeepSeek-R1-Distill-Llama-70B	0.9583 ± 0.0182	0.9500 ± 0.0199	0.9083 ± 0.0263	0.8583 ± 0.0318	0.8500 ± 0.0326	0.8417 ± 0.0333
Llama-3.3-70B	0.8833 ± 0.0293	0.8917 ± 0.0284	0.8417 ± 0.0333	0.7917 ± 0.0371	0.8000 ± 0.0365	0.8100 ± 0.0358

Table 17: Accuracy ± Standard Error for Hops 7-12

Model	Hop 7	Hop 8	Hop 9	Hop 10	Hop 11	Hop 12
Qwen-2.5-72B	0.5583 ± 0.0453	0.3583 ± 0.0438	0.4583 ± 0.0455	0.3417 ± 0.0433	0.3083 ± 0.0422	0.3667 ± 0.0440
Qwen-2.5-Math-72B	0.250 ± 0.0395	0.2417 ± 0.0391	0.2833 ± 0.0411	0.250 ± 0.0395	0.2167 ± 0.0376	0.1583 ± 0.0333
Qwen-2.5-7B	0.125 ± 0.0302	0.0833 ± 0.0252	0.0417 ± 0.0182	0.0917 ± 0.0263	0.050 ± 0.0199	0.025 ± 0.0143
Qwen-2.5-Math-7B	0.0083 ± 0.0083	0.0083 ± 0.0083	0.0025 ± 0.0046	0.0083 ± 0.0083	0.000 ± 0.0000	0.000 ± 0.0000
DeepSeek-R1-Distill-Qwen-7B	0.0750 ± 0.0240	0.0417 ± 0.0182	0.0583 ± 0.0214	0.100 ± 0.0274	0.0670 ± 0.0228	0.0500 ± 0.0199
Llama-3-8B	0.1167 ± 0.0293	0.0330 ± 0.0163	0.0250 ± 0.0143	0.0583 ± 0.0214	0.0583 ± 0.0214	0.0917 ± 0.0263
DeepSeek-R1-Distill-Llama-70B	0.8167 ± 0.0353	0.8000 ± 0.0365	0.7917 ± 0.0371	0.8083 ± 0.0359	0.7167 ± 0.0411	0.7583 ± 0.0391
Llama-3.3-70B	0.7917 ± 0.0371	0.7833 ± 0.0376	0.7083 ± 0.0415	0.7250 ± 0.0408	0.7000 ± 0.0418	0.7667 ± 0.0386

Table 18: Accuracy ± Standard Error vs. Prefix (1-6)

Model	Prefix 1	Prefix 2	Prefix 3	Prefix 4	Prefix 5	Prefix 6
Qwen-2.5-72B	0.933 ± 0.0228	0.7916 ± 0.0371	0.7833 ± 0.0376	0.7 ± 0.0418	0.625 ± 0.0442	0.5583 ± 0.0453
Qwen-2.5-Math-72B	0.7083 ± 0.0415	0.5833 ± 0.0450	0.525 ± 0.0456	0.433 ± 0.0452	0.425 ± 0.0451	0.4 ± 0.0447
Qwen-2.5-7B	0.358 ± 0.0438	0.233 ± 0.0386	0.2833 ± 0.0411	0.216 ± 0.0376	0.116 ± 0.0292	0.141 ± 0.0318
Qwen-2.5-Math-7B	0.1 ± 0.0274	0.075 ± 0.0240	0.0583 ± 0.0214	0.016 ± 0.0115	0.016 ± 0.0115	0.033 ± 0.0163
DeepSeek-R1-Distill-Qwen-7B	0.3833 ± 0.0444	0.2083 ± 0.0371	0.1583 ± 0.0333	0.125 ± 0.0302	0.133 ± 0.0310	0.0916 ± 0.0263
Llama-3-8B	0.266 ± 0.0403	0.225 ± 0.0381	0.16 ± 0.0335	0.083 ± 0.0252	0.1083 ± 0.0284	0.075 ± 0.0240
DeepSeek-R1-Distill-Llama-70B	0.908 ± 0.0264	0.941 ± 0.0215	0.883 ± 0.0293	0.808 ± 0.0360	0.868 ± 0.0309	0.825 ± 0.0347
Llama-3.3-70B	0.95 ± 0.0199	0.8583 ± 0.0318	0.8 ± 0.0365	0.78 ± 0.0378	0.79 ± 0.0372	0.66 ± 0.0432

Table 19: Accuracy ± Standard Error vs. Prefix (7-12)

Model	Prefix 7	Prefix 8	Prefix 9	Prefix 10	Prefix 11	Prefix 12
Qwen-2.5-72B	0.5 ± 0.0456	0.45 ± 0.0454	0.475 ± 0.0456	0.483 ± 0.0456	0.466 ± 0.0455	0.416 ± 0.0450
Qwen-2.5-Math-72B	0.375 ± 0.0442	0.3166 ± 0.0425	0.3166 ± 0.0425	0.3 ± 0.0418	0.3 ± 0.0418	0.291 ± 0.0415
Qwen-2.5-7B	0.183 ± 0.0353	0.103 ± 0.0277	0.133 ± 0.0310	0.116 ± 0.0292	0.141 ± 0.0318	0.091 ± 0.0263
Qwen-2.5-Math-7B	0.016 ± 0.0115	0.016 ± 0.0115	0.016 ± 0.0115	0.033 ± 0.0163	0.008 ± 0.0081	0.016 ± 0.0115
DeepSeek-R1-Distill-Qwen-7B	0.075 ± 0.0240	0.1 ± 0.0274	0.1 ± 0.0274	0.0108 ± 0.0094	0.075 ± 0.0240	0.066 ± 0.0227
Llama-3-8B	0.116 ± 0.0292	0.116 ± 0.0292	0.083 ± 0.0252	0.1 ± 0.0274	0.1 ± 0.0274	0.075 ± 0.0240
DeepSeek-R1-Distill-Llama-70B	0.841 ± 0.0334	0.85 ± 0.0326	0.81 ± 0.0358	0.8 ± 0.0365	0.76 ± 0.0390	0.75 ± 0.0395
Llama-3.3-70B	0.775 ± 0.0381	0.8416 ± 0.0333	0.733 ± 0.0404	0.7416 ± 0.0400	0.7833 ± 0.0376	0.775 ± 0.0381

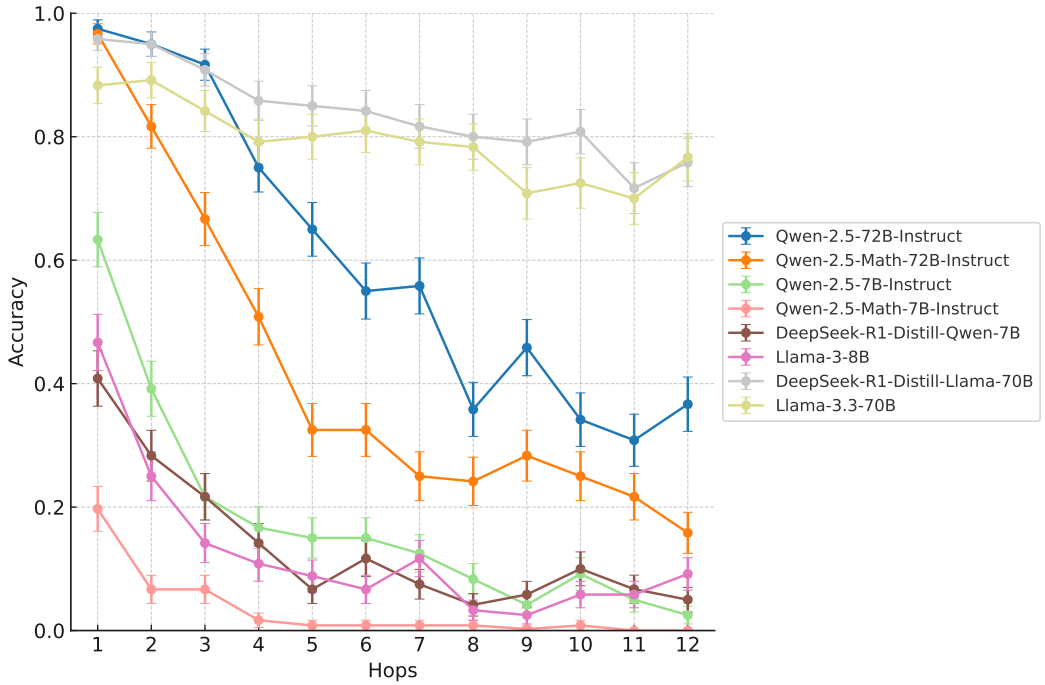

Figure 21: Accuracy vs. Hops for All Models

A.5.2 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}."
  },
  "multiply": {
    "function": "lambda x, y: x * y",
    "verbalization": "{child} is the product of {parent1} and {
        parent2}."
  },
  "divide": {
    "function": "lambda x, y: x / y if y != 0 else None",
    "verbalization": "{child} is the quotient of {parent1} divided
        by {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:
  - Mecham (value = 1)
  - Pat (value = 4)
  - Highmore (value = 6)
  Derived Nodes:
  - Sedan is the difference between Mecham and Highmore.
  - Stackpole is the sum of Sedan and Highmore.
  - Ticino is the difference between Stackpole and Pat.
  - Delle is the quotient of Sedan divided by Highmore.
  - Camino is the difference between Ticino and Stackpole.
  - Yarwood is the difference between Delle and Highmore.
  - Split is the quotient of Stackpole divided by Yarwood.
  - Leduc is the product of Delle and Pat.
  - Earth is the quotient of Delle divided by Leduc.
  - Stolly is the product of Highmore and Leduc.
  - Bachert is the product of Split and Camino.
  - Noelle is the quotient of Pat divided by Split.
  - Masterson is the quotient of Noelle divided by Mecham.
  - Balcer is the product of Noelle and Earth.
  - Ermont is the difference between Sedan and Split.
  - Ruddick is the quotient of Ticino divided by Mecham.
  - Angave is the sum of Yarwood and Balcer.
```

```
- Skolnik is the difference between Pat and Bachert.
- Padua is the product of Sedan and Mecham.
- Paet is the sum of Angave and Yarwood.
- Atkins is the sum of Ruddick and Paet.
- Worrell is the product of Ermont and Skolnik.
- Tartar is the quotient of Split divided by Angave.
- Pasquini is the product of Ermont and Worrell.
- Friesland is the product of Ermont and Split.
- Arreguin is the quotient of Tartar divided by Padua.
- Wyne is the difference between Highmore and Skolnik.
- Battin is the difference between Padua and Stolly.
- Craigmont is the difference between Yarwood and Pasquini.
- Chakravakam is the sum of Mecham and Atkins.
- Susanne is the quotient of Yarwood divided by Wyne.
- Shrum is the sum of Battin and Pat.
- Ogun is the sum of Wyne and Paet.
- GCSB is the product of Wyne and Delle.
- Buda is the difference between Leduc and Earth.
- Pinsk is the quotient of Mecham divided by Stackpole.
- Plant is the difference between Noelle and Pasquini.
- Ivan is the sum of Wyne and Split.
- Banjul is the difference between Highmore and Camino.
- Starbucks is the product of Arreguin and Ticino.
- Timmverse is the difference between Balcer and Pat.
- Barbero is the product of Banjul and Camino.
- Millay is the sum of Highmore and Skolnik.
- Placerville is the difference between Buda and Split.
- Pepper is the product of Banjul and Chakravakam.
- Yuri is the product of Pinsk and Barbero.
- Anding is the quotient of Mecham divided by Pinsk.
- Tawney is the quotient of Ticino divided by Stackpole.
- Hibbitt is the quotient of Shrum divided by Skolnik.
- Bouse is the quotient of Yarwood divided by Masterson.
- Chaldea is the difference between Balcer and Millay.
- Kaaba is the quotient of Chaldea divided by Stackpole.
- Abdulle is the quotient of Ermont divided by Yarwood.
- Sumo is the difference between Friesland and GCSB.
- Elsagate is the product of Stackpole and Ivan.
- Euramerica is the product of Skolnik and Yuri.
- Fowler is the difference between Ermont and Barbero.
- Thyone is the difference between Craigmont and Plant.
- Advincula is the difference between Wyne and Starbucks.
- Cressey is the product of Earth and Shrum.

What is the value of Atkins ?
```

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:
- Mecham is an input with value 1.
- Highmore is an input with value 6. Sedan is the difference
    between Mecham and Highmore. The value of Sedan = -5
- Stackpole is the sum of Sedan and Highmore. The value of
    Stackpole = 1
```

Response From LLM

```
   1- Pat is an input with value 4. Ticino is the difference
       between Stackpole and Pat. The value of Ticino = -3- Delle is
```

```
 the quotient of Sedan divided by Highmore. The value of
Delle = -5/6- Camino is the difference between Ticino and
Stackpole. The value of Camino = -4- Yarwood is the
difference between Delle and Highmore. The value of Yarwood =
 -5/6 - 6 = -41/6- Split is the quotient of Stackpole divided
 by Yarwood. The value of Split = 1 / (-41/6) = -6/41- Leduc
is the product of Delle and Pat. The value of Leduc = (-5/6)
* 4 = -10/3- Earth is the quotient of Delle divided by Leduc.
 The value of Earth = (-5/6) / (-10/3) = 1/4- Stolly is the
product of Highmore and Leduc. The value of Stolly = 6 *
(-10/3) = -20- Bachert is the product of Split and Camino.
The value of Bachert = (-6/41) * (-4) = 24/41- Noelle is the
quotient of Pat divided by Split. The value of Noelle = 4 /
(-6/41) = -164/6 = -82/3- Masterson is the quotient of Noelle
 divided by Mecham. The value of Masterson = (-82/3) / 1 =
-82/3- Balcer is the product of Noelle and Earth. The value
of Balcer = (-82/3) * (1/4) = -82/12 = -41/6- Ermont is the
difference between Sedan and Split. The value of Ermont = -5
- (-6/41) = -5 + 6/41 = (-205 + 6)/41 = -199/41- Ruddick is
the quotient of Ticino divided by Mecham. The value of
Ruddick = -3 / 1 = -3- Angave is the sum of Yarwood and
Balcer. The value of Angave = (-41/6) + (-41/6) = -82/6 =
-41/3- Skolnik is the difference between Pat and Bachert. The
 value of Skolnik = 4 - (24/41) = (164 - 24)/41 = 140/41-
Padua is the product of Sedan and Mecham. The value of Padua
= -5 * 1 = -5- Paet is the sum of Angave and Yarwood. The
value of Paet = (-41/3) + (-41/6) = (-82 - 41)/6 = -123/6 =
-41/2- Atkins is the sum of Ruddick and Paet. The value of
Atkins = -3 + (-41/2) = (-6 - 41)/2 = -47/2\n\n#### -23.5
```

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.