ADAPTIVE DEPENDENCY GRAPH VERIFICATION: FINE-GRAINED VERIFICATION OF LLM REASONING VIA ADAPTIVE DEPENDENCY RECONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Process-level verification of large language model reasoning requires sufficient context to judge each step, yet determining which preceding reasoning is relevant remains challenging in long reasoning traces. Existing structured verification methods typically rely on predefined reasoning units or fixed context construction strategies, which may fail to capture instance-specific dependencies. We introduce Adaptive Dependency Graph Verification (ADGV), a training-free framework that adaptively reconstructs verification contexts according to the dependency structure of each reasoning trace. ADGV first decomposes a solution into atomic reasoning units and constructs a directed graph capturing their prerequisite relations. For each target unit, it recursively traverses the dependency graph to reconstruct an ancestor-based context containing the relevant antecedent reasoning. This procedure preserves dependency-relevant information while excluding unrelated portions of the trace. We evaluate ADGV on ProcessBench across diverse mathematical reasoning tasks and multiple verifier scales. ADGV consistently improves first-error localization and step-level error attribution over existing graph-based verification, demonstrating that adaptive dependency reconstruction provides more effective verification contexts than predefined reasoning structures. These results establish ADGV as a practical approach to fine-grained reasoning verification without additional training or parameter updates.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities in mathematical reasoning and complex problem solving (Achiam et al., 2023; Shao et al., 2024). However, correct final answers do not necessarily imply reliable reasoning processes, as intermediate traces may contain incorrect or unsupported steps (Turpin et al., 2023). Fine-grained process-level verification is therefore essential for identifying and localizing reasoning errors.

A useful perspective is to view reasoning as a sequence of interdependent decisions. Rather than evaluating a complete trajectory uniformly, verification can focus on individual reasoning units together with the dependencies required to justify them. This naturally motivates a Directed Acyclic Graph (DAG), where nodes represent atomic reasoning units and edges encode prerequisite relationships (Wavering, 1989).

Recent work has investigated how reasoning granularity and contextual scope affect process verification. GRACE (Krasniqi et al., 2026) shows that verification timing and granularity substantially affect performance under a fixed computational budget. Holistic verification, however, may expose verifiers to substantial irrelevant context, while premise-selection methods such as PARC (Mukherjee et al., 2025) attempt to reduce this redundancy by retrieving supporting premises. GoV (Fang et al., 2026) further studies the interaction between verification granularity and contextual scope, but relies on fixed decomposition strategies rather than adapting the verification structure to the underlying dependencies.

These observations expose two key challenges in step-level reasoning verification: **(1) context redundancy**, where irrelevant historical reasoning can obscure the evidence needed to verify a target

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

step; and **(2) fixed verification structure**, where predefined decomposition and context-selection strategies may fail to adapt to the actual dependency structure of a reasoning trajectory.

To address these challenges, we propose **Adaptive Dependency Graph Verification (ADGV)**, a training-free framework that dynamically reconstructs reasoning dependencies and performs dependency-aware verification. ADGV incrementally decomposes reasoning trajectories into atomic units, constructs a dependency graph, and retrieves ancestor-based contexts for verifying each target step. This preserves necessary logical dependencies while removing irrelevant historical information, providing a verification context that balances completeness and efficiency.

Our contributions are summarized as follows:

- **Adaptive Dependency Graph Verification.** We propose ADGV, a training-free framework that incrementally constructs dependency graphs over reasoning trajectories and performs ancestor-based context reconstruction for adaptive verification.
- **Comprehensive evaluation.** We evaluate ADGV on ProcessBench with verification models of different scales, demonstrating improved first-error localization while reducing unnecessary verifier context consumption.

2 Related Work

2.1 Reasoning Structures in LLMs

Chain-of-Thought (CoT) prompting (Wei et al., 2022) improves multi-step reasoning by explicitly generating intermediate reasoning steps. Subsequent work has explored richer structures, including Tree-of-Thoughts (ToT) (Yao et al., 2023) and Graph-of-Thoughts (GoT) (Besta et al., 2024), which represent branching and non-linear dependencies among reasoning states. Our work adopts the structured view of reasoning but focuses on its verification rather than generation.

2.2 Verification of LLM Reasoning

Reliable verification of intermediate reasoning remains challenging despite advances in reasoning generation (Stechly et al., 2025). Existing approaches can be broadly categorized into holistic verification, training-based process verifiers, and decomposition-based verification.

Training-based process verifiers. Process Reward Models (PRMs) (Lightman et al., 2024; Wang et al., 2024; Zhang et al., 2025) learn to evaluate intermediate reasoning steps using process-level supervision. Recent work such as PRISM (Agrawal et al., 2026) and unsupervised PRMs (Gadetsky et al., 2026) investigates reward calibration and supervision-efficient verification. These approaches, however, generally require additional training or supervision.

Decomposition-based verification. Deductive Verification (Ling et al., 2023) decomposes reasoning into smaller units to facilitate intermediate verification. GoV (Fang et al., 2026) further investigates the role of reasoning granularity and contextual scope by modeling dependencies among reasoning steps with graph structures. These methods demonstrate the importance of structuring both the reasoning units and their verification context. However, predefined decomposition and verification structures may not fully reflect the dependency patterns of individual reasoning trajectories. In particular, different reasoning steps may require different subsets of preceding information, making a fixed verification structure insufficient for consistently identifying the evidence required for local verification. ADGV addresses this limitation by dynamically reconstructing dependencies and adapting the verification context to each target reasoning unit.

3 Adaptive Dependency Graph Verification

ADGV aims to decouple reasoning verification from the original linear organization and coarse segmentation of model-generated reasoning trajectories. Instead of verifying fixed reasoning blocks or relying on predefined premise selection, ADGV reconstructs the latent dependency structure among atomic reasoning units and dynamically builds verification contexts according to logical dependencies.

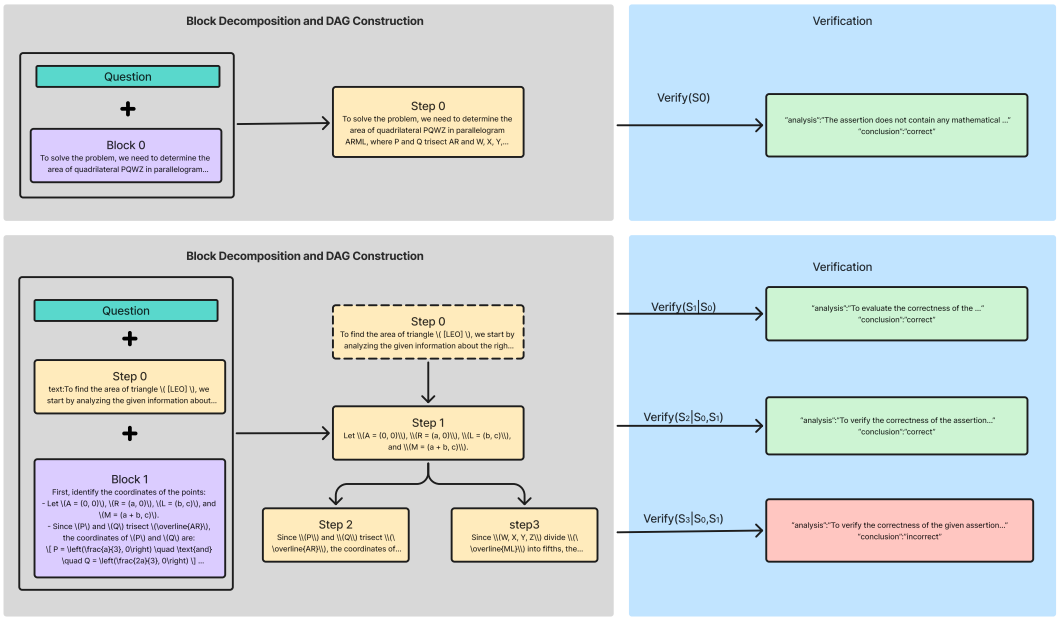


Figure 1: Overview of ADGV. Given a model-generated reasoning trajectory, ADGV incrementally decomposes reasoning blocks into atomic units, constructs a dependency graph, and retrieves ancestor-based contexts for step-level verification.

Given a model-generated reasoning trajectory $\mathcal{R} = \{B_1, B_2, \ldots, B_m\}$, ADGV consists of two main stages: (1) incremental atomic decomposition with dependency graph construction, and (2) dependency-aware context reconstruction through recursive ancestor retrieval.

3.1 Reasoning Graph Representation

ADGV represents a reasoning trajectory as a directed acyclic graph (DAG):

$$\mathcal{G} = (V, E), \tag{1}$$

where $V = \{v_1, v_2, \ldots, v_n\}$ denotes atomic reasoning units extracted from the original trajectory, and E represents logical dependency relations. A directed edge $(v_i, v_j) \in E$ indicates that v_i provides necessary information for deriving v_j.

Since reasoning units preserve their original generation order, the graph naturally satisfies the topological constraint $(v_i, v_j) \in E \Rightarrow i < j$.

For a target reasoning unit v, ADGV verifies its correctness based on the dependency-induced subgraph containing the target node and its ancestors. Specifically, the verification graph is defined as $\mathcal{G}_v = (V_v, E \cap (V_v \times V_v))$, where $V_v = \{v\} \cup Anc(v)$ and $Anc(v)$ denotes all ancestor nodes required to derive v.

Therefore, ADGV transforms global trajectory verification into localized dependency-aware verification while preserving the logical evidence required for evaluating each reasoning unit.

3.2 Incremental Dependency Graph Construction

Building upon the insights of GOV and PARC, ADGV performs incremental dependency graph reconstruction in the original reasoning order, avoiding the need to pre-decompose the entire reasoning trajectory before verification.

Before processing reasoning block B_j, ADGV maintains the current graph $\mathcal{G}_{j-1} = (V_{j-1}, E_{j-1})$, where V_{j-1} contains previously extracted atomic reasoning units. The decomposition module receives the current block together with previous graph information and produces newly identified atomic units: $(\mathcal{G}_{j-1}, B_j) \rightarrow A_j$.

The graph is then updated by incorporating new nodes and dependency relations:

$$V_j = V_{j-1} \cup A_j, \tag{2}$$
$$E_j = E_{j-1} \cup E(A_j, V_j). \tag{3}$$

Through this incremental construction process, ADGV gradually recovers latent dependencies across reasoning units while avoiding unstable global segmentation.

To preserve the faithfulness of model-generated reasoning, ADGV enforces a verbatim extraction constraint. Each atomic node must correspond to an exact span of its original reasoning block, i.e., $Text(v_i) \subseteq_{\text{span}} B_j$. Semantic rewriting or error correction is prohibited, ensuring that verification is performed on the original reasoning states rather than an interpreted reconstruction.

3.3 Dependency-aware Context Reconstruction

After constructing the dependency graph, ADGV generates verification contexts through dependency-based ancestor retrieval instead of providing the complete reasoning history.

For a target node c, its direct predecessors are defined as $Pred(c) = \{u|(u, c) \in E\}$. When explicit dependency relations are unavailable, ADGV adopts a sequential fallback strategy: $Pred(c_k) = \{c_{k-1}\}$ if $Pred(c_k) = \emptyset$, otherwise the original dependency set is retained.

The complete dependency context is obtained through recursive ancestor tracing:

$$Anc(c) = \{u \in V \mid u \rightarrow^* c\}. \tag{4}$$

The verifier input context is constructed as:

$$C(c) = \bigoplus_{u \in Anc(c)} \text{Text}(u). \tag{5}$$

where $\bigoplus$ denotes ordered concatenation of ancestor reasoning units.

Compared with full-history verification, ADGV only provides dependency relevant evidence for each target step. This reduces redundant contextual information and mitigates attention interference while maintaining the logical dependencies required for accurate verification.

4 Experiments

To comprehensively evaluate the proposed ADGV framework, we conduct systematic experiments on ProcessBench (Zheng et al. 2024), a benchmark designed for identifying process-level errors in mathematical reasoning trajectories.

4.1 Experimental Setup

ProcessBench provides model-generated reasoning trajectories together with human annotations of the first incorrect reasoning step. Following the original evaluation protocol, the goal of the verifier is to identify the earliest erroneous step in a given reasoning trajectory.

All methods are evaluated under the same setting: each verifier receives the identical problem statement and the same original chain-of-thought (CoT) trajectory from ProcessBench. Therefore, the only difference among methods lies in their verification strategies and context construction mechanisms.

We compare ADGV with representative process verification approaches, including PARC and Graph of Verification (GoV). For PARC, we follow the original verification prompt and premise extraction strategy described in its paper. For ADGV, the verification prompt template is kept consistent with GoV to ensure a fair comparison. The difference between GoV and ADGV mainly lies in the construction of verification contexts: GoV relies on its original graph-based decomposition strategy, while ADGV reconstructs verification contexts through adaptive dependency tracing over the reasoning graph.

All methods are evaluated in a training-free inference setting without additional parameter optimization.For all experiments, the same reasoning trajectories and benchmark annotations are used across different verification methods. The decomposition and dependency reconstruction modules of ADGV are executed only during inference.

The verification stage of ADGV follows the GoV verification prompt design, while the adaptive dependency reconstruction module dynamically determines the required verification context for each target step.

The experiments cover four mathematical reasoning datasets from ProcessBench: GSM8K, MATH, OlympiadBench, and OmniMath. These datasets represent different levels of reasoning complexity, ranging from elementary arithmetic problems to competition-level mathematical reasoning tasks.

To investigate the scalability and generalization ability of ADGV, we employ Qwen2.5-Instruct models with three different parameter sizes as verifiers, including Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and Qwen2.5-32B-Instruct.

Experiments were conducted on two NVIDIA A40 GPUs (48GB each).During inference, we use greedy decoding with temperature set to 0 for all models to ensure reproducibility and reduce randomness.

4.2 Evaluation Metrics

Following previous process verification studies, we evaluate models from two perspectives: error localization accuracy and rationale reliability.

Correct Accuracy (CA). Correct Accuracy measures the ability of a verifier to correctly accept fully correct reasoning trajectories. It is defined as:

$$CA = \frac{N_{\text{correctly accepted}}}{N_{\text{correct traces}}}. \tag{6}$$

Error Accuracy (EA). Error Accuracy evaluates whether the verifier can correctly identify the first incorrect step in erroneous reasoning trajectories. A prediction is considered correct only when the predicted error step index exactly matches the human annotation provided by ProcessBench:

$$EA = \frac{N_{\text{correct first error localization}}}{N_{\text{error traces}}}. \tag{7}$$

The overall verification performance is measured using the harmonic mean of CA and EA:

$$F_1 = \frac{2 \times CA \times EA}{CA + EA}. \tag{8}$$

4.3 Rationale Reliability Audit

Although ProcessBench provides reliable annotations for error locations, its original evaluation protocol mainly measures whether the predicted error step matches the human annotation, without assessing whether the verifier provides a faithful rationale. A verifier may therefore achieve correct localization while producing an unsupported explanation. Hong et al. (Hong et al., 2024) similarly argue that final task performance alone is insufficient to assess self-verification ability, as it does not directly evaluate whether LLMs can identify logical fallacies in intermediate reasoning.

Table 1: First-error localization results on ProcessBench. CA, EA, and F1 denote Correct Accuracy, Error Accuracy, and their harmonic mean, respectively. Results marked with † are reported from Fang et al. (2026); GoV and ADGV are evaluated in our experiments.

Model	Method	GSM8K			MATH			OlympiadBench			OmniMath		
		CA	EA	F1	CA	EA	F1	CA	EA	F1	CA	EA	F1
Qwen2.5-7B -Instruct	Holistic†	66.3	36.7	47.3	63.8	23.7	34.6	46.0	25.4	32.7	-	-	-
	GoV	88.60	48.31	62.53	78.57	45.96	58.00	70.80	27.69	39.80	64.32	26.48	37.52
	ADGV	90.16	49.76	**64.13**	78.08	54.71	**64.34**	58.11	37.22	**45.37**	65.98	35.18	**45.89**
Qwen2.5-14B -Instruct	Holistic†	93.8	47.8	63.3	86.9	40.4	55.2	76.4	30.9	44.0	-	-	-
	GoV	94.82	64.25	76.60	89.41	55.89	68.78	82.89	42.66	56.33	80.50	36.23	49.97
	ADGV	93.26	66.67	**77.75**	81.77	62.63	**70.93**	58.11	56.58	**57.34**	59.34	46.11	**51.90**
Qwen2.5-32B -Instruct	Holistic†	97.9	43.0	59.8	95.6	33.3	49.4	90.0	22.4	35.9	-	-	-
	GoV	95.34	68.60	79.79	89.66	63.97	**74.67**	80.24	52.19	63.25	75.93	51.65	**61.48**
	ADGV	94.30	74.40	**83.17**	82.02	68.35	74.56	66.96	60.06	**63.32**	60.17	54.55	57.22

To obtain a stricter evaluation, we perform a post-hoc rationale audit using GPT-4o and DeepSeek as external evaluators. For each prediction that localizes the first error, the evaluator judges whether the associated rationale correctly explains the identified reasoning flaw and supports the prediction. A prediction is considered rationale-correct only when both its error location and rationale are correct. We therefore define rationale-aware Error Accuracy as

$$EA_{\text{rat}} = \frac{N_{\text{correct localization with valid rationale}}}{N_{\text{error traces}}}. \tag{9}$$

The corresponding rationale-aware F_1 is

$$F_1^{\text{rat}} = \frac{2\,CA\,EA_{\text{rat}}}{CA + EA_{\text{rat}}}. \tag{10}$$

Rationale auditing is applied only to Error Accuracy, since Correct Accuracy concerns the acceptance of fully correct reasoning trajectories and does not involve error-localization rationales. The external evaluators are used only for post-hoc assessment and do not participate in ADGV or baseline inference.

We further introduce *First-Error Localization Precision* (P_{LOC}) to assess rationale-aware first-error localization. Let g_i and $\hat{g}_i$ denote the ground-truth and predicted first-error positions, respectively, and let $r_i \in 0, 1$ indicate rationale correctness. We define

$$\mathcal{V} = \{i \mid \text{a ground-truth first error exists, a predicted error location exists, } \hat{g}_i \le g_i\},$$

and

$$P_{\text{LOC}} = \frac{\sum_{i \in \mathcal{V}} \mathbf{1}\{\hat{g}_i = g_i \wedge r_i = 1\}}{|\mathcal{V}|}.$$

Thus, predictions earlier than the true first error remain in the denominator but are not counted as correct, whereas later predictions and error-free instances are excluded.

4.4 Performance Analysis

Table 1 presents the overall comparison between ADGV and existing verification approaches on ProcessBench under the original first-error localization protocol. Across different datasets and model scales, ADGV improves first-error localization performance in a substantial fraction of the evaluated settings, demonstrating the effectiveness of adaptive dependency-aware context reconstruction.

Improved first-error localization through adaptive context reconstruction. Compared with PARC and GoV, ADGV achieves higher Error Accuracy and F_1 across most benchmarks. For

Table 2: DeepSeek-audited results on ProcessBench under the first-error localization protocol.

Model	Method	GSM8K			MATH			OlympiadBench			OmniMath		
		EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}
Qwen2.5-7B -Instruct	GoV	33.82	48.95	46.36	28.11	41.41	33.60	14.98	24.72	19.84	11.46	19.46	15.21
	ADGV (Ours)	35.27	**50.70**	**50.69**	38.22	**51.31**	**43.52**	21.79	**31.95**	**26.52**	16.34	**26.19**	**23.75**
Qwen2.5-14B -Instruct	GoV	53.62	68.50	**71.15**	40.24	55.50	**54.20**	27.99	41.85	**43.53**	22.13	34.72	**39.62**
	ADGV (Ours)	57.00	**70.76**	69.41	47.14	**59.80**	53.03	36.46	**44.81**	36.19	32.15	**41.70**	37.89
Qwen2.5-32B -Instruct	GoV	59.42	73.21	72.35	49.66	63.92	**57.28**	36.16	49.85	**42.60**	32.28	45.30	**41.60**
	ADGV (Ours)	64.25	**76.43**	**74.30**	56.40	**66.84**	57.46	41.15	**50.97**	40.24	40.97	**48.75**	36.55

Table 3: GPT-4o audited results on ProcessBench under the first-error localization protocol.

Model	Method	GSM8K			MATH			OlympiadBench			OmniMath		
		EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}	EA_{rat}	F_1^{rat}	P_{LOC}
Qwen2.5-7B -Instruct	GoV	28.99	43.68	39.74	30.47	43.91	36.42	15.89	25.95	21.04	14.62	23.83	19.41
	ADGV (Ours)	31.88	**47.11**	**45.83**	39.73	**52.66**	**45.65**	22.39	**32.59**	**27.26**	21.34	**32.25**	**31.03**
Qwen2.5-14B -Instruct	GoV	41.06	57.31	**54.49**	42.26	57.39	**56.92**	29.05	43.02	**45.18**	22.00	34.56	**39.39**
	ADGV (Ours)	43.96	**59.76**	53.53	47.64	**60.21**	53.60	38.88	**46.59**	38.59	32.94	**42.36**	38.82
Qwen2.5-32B -Instruct	GoV	44.93	61.07	54.71	47.47	62.08	**54.76**	38.12	**51.69**	**44.92**	36.76	49.54	**47.37**
	ADGV (Ours)	50.24	**65.56**	**58.10**	53.70	**64.91**	54.72	40.24	50.27	39.35	43.61	**50.57**	44.97

example, with Qwen2.5-32B-Instruct, ADGV improves Error Accuracy over GoV from 68.60% to 74.40% on GSM8K and from 63.97% to 68.35% on MATH. On Qwen2.5-7B-Instruct, ADGV achieves higher F_1 than GoV on all four evaluated datasets. These results suggest that reconstructing verification context according to reasoning dependencies can provide more effective evidence for identifying the first incorrect step than relying on fixed verification contexts.

Effectiveness across verifier scales. The effectiveness of ADGV varies across verifier scales and datasets. The method provides consistent improvements over GoV at the 7B scale, while the relative gains become more dataset-dependent for larger verifiers. This pattern suggests that dependency-aware context reconstruction is complementary to verifier capacity, but its benefit is not determined by model scale alone. In particular, selectively retrieving dependency-related evidence can reduce irrelevant context and provide useful verification signals for smaller verifiers, whereas larger models may already possess greater capacity to reason over the available context.

Error detection without excessive rejection. To examine whether the gains in Error Accuracy could be attributed primarily to more conservative error prediction, we further evaluate rationale-aware first-error localization using P_{LOC}. Unlike Error Accuracy, P_{LOC} requires the predicted location to match the actual first error and the associated rationale to be valid, while including premature error predictions and false rejection of otherwise correct reasoning in its denominator. Thus, P_{LOC} provides a stricter measure of first-error localization that is sensitive to excessive error rejection. The results show that ADGV maintains non-trivial P_{LOC} across datasets and verifier scales, indicating that its improved Error Accuracy cannot be attributed solely to conservatively rejecting reasoning as incorrect.

4.5 Ablation Study: Effect of Dependency Reconstruction

To evaluate the contribution of dependency reconstruction, we compare ADGV with a Split-only variant. Split-only preserves the same atomic decomposition and verification pipeline as ADGV, but removes dependency graph construction and directly follows the original sequential segmentation for context selection. Therefore, the comparison isolates the effect of adaptive dependency-based context reconstruction.

Table 4 presents the rationale-aware F_1 results under GPT-4o evaluation. ADGV achieves comparable or higher verification performance than Split-only across the evaluated datasets and model scales. For Qwen2.5-7B-Instruct, ADGV improves F_1^{rat} on all four benchmarks, with absolute gains of 0.07, 0.70, 0.89, and 2.07 percentage points on GSM8K, MATH, OlympiadBench, and OmniMath, respectively. For Qwen2.5-14B-Instruct, ADGV improves F_1^{rat} on GSM8K, OlympiadBench, and OmniMath, while showing only a small decrease of 0.26 percentage points on MATH. These results

Table 4: Ablation study of ADGV on rationale-aware verification performance.

Model	Method	GSM8K	MATH	OlympiadBench	OmniMath
Qwen2.5-7B -Instruct	Split-only	47.04	51.96	31.70	30.18
	ADGV	**47.11**	**52.66**	**32.59**	**32.25**
Qwen2.5-14B -Instruct	Split-only	54.34	**60.47**	45.60	41.65
	ADGV	**59.76**	60.21	**46.59**	**42.36**

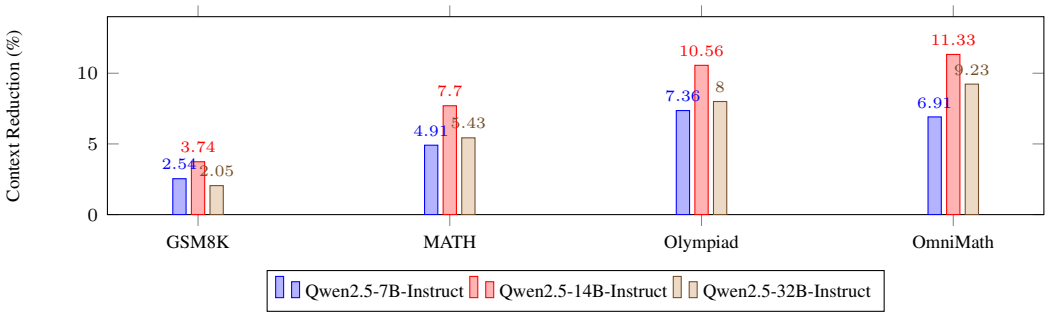


Figure 2: Verifier context reduction achieved by ADGV compared with Split-only. ADGV consistently reduces verifier input context across different models and benchmarks.

indicate that dependency reconstruction can provide additional verification benefits beyond atomic decomposition, although the effect is dataset-dependent.

Beyond verification performance, we further analyze the impact of dependency reconstruction on verifier context consumption. As shown in Figure 2, ADGV consistently reduces the number of verification context tokens compared with Split-only across all evaluated models and benchmarks. The reduction is more pronounced on the more challenging mathematical benchmarks, reaching 11.33% on OmniMath with Qwen2.5-14B-Instruct. This shows that dependency-aware context reconstruction can retain the information required for verification while avoiding part of the redundant historical reasoning context.

Overall, the ablation results suggest that dependency reconstruction serves primarily to improve the relevance and efficiency of verification context. While its effect on rationale-aware F_1 is not uniform across all settings, ADGV maintains comparable verification performance while consistently reducing context consumption. This provides evidence that adaptive dependency-aware context selection offers a more efficient alternative to sequential context selection without requiring additional verifier training.

5 Discussion and Limitations

Although ADGV provides an effective training-free framework for structured reasoning verification, its design also introduces several practical considerations. Since ADGV improves verification through contextual reconstruction rather than parameter optimization, its effectiveness depends on both the quality of dependency structures and the reasoning capability of the underlying verifier.

5.1 Verifier Dependence in Parameter-Free Contextual Engineering

A major advantage of ADGV is that it separates verification context construction from model parameter optimization. Unlike training-based process reward models (PRMs), ADGV does not require additional process supervision data or verification-specific fine-tuning. Therefore, it can be directly applied to different verifier models and adapted to evolving backbone models without retraining.

However, this parameter-free design also introduces dependence on the intrinsic capability and behavioral patterns of the verifier model. Although ADGV provides a more relevant dependency-aware

context, the final correctness judgment is still generated by the underlying LLM. Consequently, verifier-specific biases may affect the reliability of process evaluation.

One potential failure mode is independent re-solving, where the verifier ignores the provided reasoning trajectory and attempts to solve the original problem from scratch. Such behavior changes verification into answer generation and may lead to incorrect judgments for valid reasoning paths that differ from the verifier's preferred solution strategy. Another challenge is efficiency bias, where verifiers may incorrectly penalize logically valid but redundant reasoning steps because they deviate from shorter or more elegant solution patterns.

These observations suggest that future verification systems should not only optimize context selection, but also improve verifier alignment through verification-oriented objectives, explicit auditing instructions, or lightweight calibration strategies.

5.2 Limitations of Dependency Reconstruction

Although ADGV improves verification by recovering dependency-aware contexts, the quality of the reconstructed graph remains an important factor. Since LLM reasoning trajectories are expressed in free-form natural language, dependency relations are not always explicitly stated.

First, implicit dependencies may be difficult to identify. A reasoning step can depend on unstated assumptions, previously introduced concepts, or intuitive transformations without explicit references. Missing such dependencies may result in incomplete verification contexts and reduce the effectiveness of ancestor-based retrieval.

Second, some reasoning processes contain complex non-linear structures, including temporary assumptions, branching exploration, and proof by contradiction. While DAGs naturally represent directional logical dependencies, recovering high-level proof structures from unconstrained natural language remains challenging.

Third, incremental graph construction may introduce error propagation. Since later dependency reconstruction relies on previously extracted reasoning units, incorrectly identified dependencies may introduce irrelevant evidence or omit necessary premises during context reconstruction.

Future work could investigate uncertainty-aware dependency extraction, joint optimization of dependency identification and correctness estimation, and adaptive graph structures for more complex reasoning patterns. These directions may further improve the robustness of dependency-based reasoning verification while preserving the advantages of training-free contextual engineering.

6 Conclusion

We propose Adaptive Dependency Graph Verification (ADGV), a training-free framework for fine-grained verification of LLM reasoning trajectories. ADGV reconstructs the latent dependency structure of reasoning trajectories through incremental atomic decomposition and dependency graph construction, and retrieves dependency-relevant contexts through recursive ancestor traversal. This enables step-level verification using sufficient and logically connected evidence while reducing contextual redundancy and semantic fragmentation.

Experiments on ProcessBench show that ADGV improves first-error localization across multiple mathematical reasoning benchmarks and verifier scales. Ablation studies further show that dependency reconstruction improves verification effectiveness while reducing unnecessary verifier context consumption. ADGV thus provides a complementary direction to parameter-based verifier optimization by improving the organization of reasoning evidence rather than modifying model parameters.

AI use statement

In this work, we used generative AI tools for research ideation and execution, including discussing research directions, methodological alternatives, experimental design, ablation settings, evaluation metrics, and the interpretation and presentation of experimental results.

We have not used generative AI tools for generating synthetic datasets, proving mathematical claims, or other required disclosure tasks not applicable to this work. Additionally, we used generative AI

tools to aid or polish the writing of the paper and to draft and revise parts of the manuscript, including portions of the introduction, methodology, experiments, and conclusion.

We have reviewed and verified all AI-assisted work. Research decisions, experimental implementations, experimental results, analyses, and final claims were independently checked by the authors. AI-assisted text was reviewed and revised by the authors for accuracy, consistency, and conformity with the actual methods and results. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Aakriti Agrawal, Souradip Chakraborty, Armin Saghafian, Nihal Sharma, Rizal Fathony, Nam H Nguyen, C Bayan Bruss, Amrit Singh Bedi, and Furong Huang. The hidden bias of process reward models: Prism for rewarding the right reasoning. *arXiv preprint arXiv:2606.09078*, 2026.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 17682–17690, 2024.

Jiwei Fang, Bin Zhang, Changwei Wang, Jin Wan, and Zhiwei Xu. Graph of verification: Structured verification of llm reasoning with directed acyclic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 30665–30672, 2026.

Artyom Gadetsky, Maxim Kodryan, Siba Smarak Panigrahi, Hang Guo, and Maria Brbic. Unsupervised process reward models. *arXiv preprint arXiv:2605.10158*, 2026.

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. A closer look at the self-verification abilities of large language models in logical reasoning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 900–925, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.52. URL https://aclanthology.org/2024.naacl-long.52/.

Ardit Krasniqi, Luan Vejsiu, and Elira Dervishi. Granularity-regulated adaptive computational efficiency for optimal verification in test-time scaling. 2026. URL https://api.semanticscholar.org/CorpusID:289624055.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, pp. 39578–39601, 2024.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, pp. 36407–36433, 2023.

Sagnik Mukherjee, Abhinav Chinta, Takyoung Kim, Tarun Anoop Sharma, and Dilek Hakkani-Tür. Premise-augmented reasoning chains improve error identification in math reasoning with llms. *arXiv preprint arXiv:2502.02362*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International conference on learning representations*, pp. 98190–98243, 2025.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, pp. 74952–74965, 2023.

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

Michael J Wavering. Logical reasoning necessary to make line graphs. *Journal of Research in Science Teaching*, pp. 373–379, 1989.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, pp. 11809–11822, 2023.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025.

A ADGV ALGORITHM

The overall procedure of Adaptive Dependency Graph Verification (ADGV) is summarized in Algorithm 1. ADGV first decomposes the reasoning trajectory into atomic steps and constructs a dependency graph among them. Then, it verifies each step using its dependency-aware context consisting of the current step and its verified ancestors. The verification process terminates once the first incorrect step is identified.

Algorithm 1 Adaptive Dependency Graph Verification (ADGV)

Require: Reasoning trajectory R, problem statement P
Ensure: First incorrect step index e

Split R into sequential reasoning blocks: $B = \{b_1, b_2, \ldots, b_m\}$
Initialize atomic step set $S \leftarrow \emptyset$ and dependency graph $G \leftarrow \emptyset$
Initialize verified step set $V \leftarrow \emptyset$
for each block $b_i \in B$ **do**
  Extract atomic steps: $A_i \leftarrow \textsc{Decompose}(P, b_i, V)$
  for each atomic step $s_j \in A_i$ **do**
    Assign a unique identifier to s_j
    Analyze dependencies: $D_j \leftarrow \textsc{Dependency}(s_j, S)$
    Add node s_j and edges D_j into G
    Retrieve ancestor nodes: $C_j \leftarrow \textsc{Ancestor}(s_j, G)$
    Construct verification context: $X_j = C_j \cup \{s_j\}$
    Verify current step: $r_j \leftarrow \textsc{Verify}(P, X_j)$
    if $r_j = \textsc{Incorrect}$ **then**
      return s_j
    end if
    Add s_j into verified set V
  end for
end for
return $\emptyset$

B PROMPT TEMPLATES

B.1 ATOMIC STEP DECOMPOSITION AND DEPENDENCY ANALYSIS PROMPT

ADGV uses a decomposition module to transform continuous reasoning trajectories into atomic verification units. The decomposer is required to preserve the original reasoning text, avoid unnecessary fragmentation, and identify logical dependencies among extracted steps.

The prompt is shown below.

```
You are a rigorous mathematical logic deconstructor.
Your task is to decompose the current reasoning block into atomic
verification steps represented by structured XML tags.
[Original Question]
{question}
[Previously Verified Steps]
{history_context}
[Current Block]
{current_block_text}
STRICT RULES:
1.  TARGET BLOCK ONLY
Extract atomic steps only from the current block.
Do not re-extract previously verified steps.
2.  COPY-MACHINE MODE
Copy the extracted text exactly as written.
Do not paraphrase, summarize, or correct errors.
Do not rewrite mathematical expressions.
3.  ATOMIC STEP REQUIREMENTS
Each atomic step must contain a complete verifiable reasoning unit.
The step should include a mathematical fact, condition, equation,
or logical transition.
Do not split a reasoning unit if doing so destroys its logical
meaning.
4.  PRESERVE HYPOTHESIS AND REFUTATION CHAINS
Do not separate temporary assumptions, trial cases, or hypotheses
from their subsequent verification or refutation.
If an assumption and its conclusion form one logical chain, extract
them as a single atomic step.
5.  CONTINUOUS IDENTIFIER ASSIGNMENT
Assign an integer step identifier to every new step.
Start exactly from:
{start_step_id}
Increase the identifier by one for every step.
6.  DEPENDENCY ANALYSIS
For each extracted step, identify all previous steps that are
logically required.
Do not assume that only the immediately preceding step is relevant.
Return dependency IDs as a list.
For independent steps, return an empty list.
```

B.2 XML OUTPUT FORMAT

The decomposition module outputs atomic steps in the following format.

```
<atomic_steps>
<step>
<step_id>ID</step_id>
<step_text> EXACT_TEXT_FROM_SOURCE </step_text>
<dependencies> [ID1, ID2, ...]  </dependencies>
</step>
</atomic_steps>
```

The dependency field stores the identifiers of previous atomic steps required to derive the current step.

B.3 VERIFICATION PROMPT

After constructing the dependency graph, ADGV verifies each atomic step using its dependency-aware context.

```
The following is a math problem, the verified preceding steps of
its solution, and the current step to be evaluated.
[Global Background]
{problem}
[Previously Verified State]
{preceding_premises if preceding_premises else "No previous state."}
[Current Step]
"{current_atomic_step}"
Your task is to judge whether the [Current Step] is correct.  Once
you identify an error in the [Current Step], return "incorrect".
Otherwise, return "correct."
Please think step by step, and put your final conclusion (i.e.,
'correct' or 'incorrect') in \\boxed{{}}
```

B.4 RATIONALE RELIABILITY CALIBRATION PROMPT

You are evaluating the quality of a mathematical reasoning verifier.
The verifier claims that the target step contains a mathematical error.
Your task is NOT to solve the problem again and NOT to judge whether the step is correct by itself.
Your task is to determine whether the verifier's analysis provides a correct and sufficient explanation for its error judgment.
You should evaluate whether the analysis:
- correctly understands the target step,
- identifies the actual mathematical issue,
- uses valid mathematical reasoning,
- and provides a rationale that supports the verifier's final conclusion.
Output correct=true ONLY if:
1. The analysis identifies the real mathematical problem in the step.
2. The mathematical reasoning in the explanation is correct.
3. The explanation is logically connected to the step and problem context.
4. The stated rationale is sufficient to justify the verifier's judgment.
Output correct=false if ANY of the following applies:
1. The analysis identifies the wrong error or misses the actual issue.
2. The analysis misunderstands the meaning of the step.
3. The mathematical reasoning in the explanation is incorrect.
4. The explanation is irrelevant, incomplete, or does not support the conclusion.
5. The verifier's final judgment is correct, but the provided explanation is mathematically invalid or unjustified.
Original problem:
problem
Full solution:
{model_response}
Target step:
{step_text}
Verifier analysis:
{analysis}
Strictly output JSON only.
Output format:

{ ”reason”: ”Briefly explain whether the verifier analysis is correct and why.”, ”correct”: true }
or
{ ”reason”: ”Briefly explain why the verifier analysis is incorrect or insufficient.”, ”correct”: false }

B.5 Dependency-Aware Verification Example

After decomposing the solution into atomic steps, ADGV constructs a directed acyclic graph (DAG), where each node represents an atomic reasoning step and each edge indicates a dependency relation.

Reasoning Structure:

- S_1: Identify the centers and radii of the two circles: $C_1 = (10, 0), r_1 = 6$ and $C_2 = (-15, 0), r_2 = 9$.
- S_2: Compute the distance between the two centers:
$$d = 10 - (-15) = 25.$$
- S_3: Introduce an auxiliary tangent construction and derive the tangent distances.
- S_4: Correct the negative tangent distance by taking its absolute value.
- S_5: Compute the tangent segment length PQ.
- S_6: Output the final answer:
$$|PQ| = 13.$$

Dependencies:

$$S_1 \to S_2$$

$$S_1 \to S_3$$

$$S_3 \to S_4$$

$$S_2, S_4 \to S_5$$

$$S_5 \to S_6$$

For verifying an atomic step S_i, ADGV first retrieves all ancestor nodes of S_i in the dependency graph. The retrieved ancestor states provide the minimal sufficient context required for verification.

Ancestor-aware Verification:

- Verify S_1 using the original problem statement.
- Verify S_2 using its ancestor set:
$$Anc(S_2) = \{S_1\}.$$
- Verify S_3 using:
$$Anc(S_3) = \{S_1\}.$$
- Verify S_4 using:
$$Anc(S_4) = \{S_1, S_3\}.$$
- Verify S_5 using:
$$Anc(S_5) = \{S_1, S_2, S_3, S_4\}.$$
- Verify S_6 using:
$$Anc(S_6) = \{S_1, S_2, S_3, S_4, S_5\}.$$

By retrieving ancestor states rather than only immediate dependencies, ADGV preserves necessary reasoning history while avoiding the unnecessary context expansion caused by using the full solution trajectory.

C Dataset Details

C.1 OmniMath

OmniMath is a large-scale mathematical reasoning benchmark containing diverse mathematical problems with model-generated reasoning trajectories. Each instance consists of a problem statement, the generating model, and a sequence of intermediate reasoning steps. The reasoning trajectory is stored as a list of sequential steps, where each step corresponds to a portion of the generated solution.

An example instance from OmniMath is shown below.

Model
Qwen2.5-Math-7B-Instruct
Problem
Compute the smallest positive integer such that, no matter how you rearrange its digits (in base ten), the resulting number is a multiple of 63.
(Paragraph 0)To determine the smallest positive integer such that any rearrangement of its digits is a multiple of 63, we need to consider the properties of numbers divisible by 63. A number is divisible by 63 if and only if it is divisible by both 7 and 9.
(Paragraph 1)A number is divisible by 9 if the sum of its digits is divisible by 9.
(Paragraph 2)A number is divisible by 7 if the number itself is divisible by 7.
(Paragraph 3)We need to find the smallest number whose digits can be rearranged in any way, and the resulting number is always divisible by both 7 and 9.
(Paragraph 4)Let's start by considering the smallest possible sums of digits that are divisible by 9. The smallest sum is 9 itself.
(Paragraph 5)We need to find a number with digits that sum to 9 and check if any permutation of these digits is divisible by 7.
(Paragraph 6)First, consider one-digit numbers: 9 (not divisible by 7).
(Paragraph 7)Next, consider two-digit numbers. The possible pairs of digits are (1,8), (2,7), (3,6), and (4,5). None of these pairs form a number divisible by 7. *Error: The numbers (3, 6) together form 63, which is divisible by 7.*
(Paragraph 8)Now, consider three-digit numbers. The possible sets of digits are (1,0,8), (1,1,7), (1,2,6), and other combinations whose digit sum equals 9.
(Paragraph 9)We check each set to see if any permutation forms a number divisible by 7.
For example:
$108 \div 7 \approx 15.4286$ (not divisible)
$117 \div 7 \approx 16.7143$ (not divisible)
$126 \div 7 = 18$ (divisible)
(Paragraph 10)Since 126 is divisible by 7 and the sum of its digits is 9, it meets both criteria.
(Paragraph 11)The smallest positive integer such that any rearrangement of its digits is a multiple of 63 is $\boxed{126}$.