ATOMGRAPH: REASONING ISN'T LINEAR, WHY SHOULD VERIFICATION BE?

Anonymous authors
Paper under double-blind review

ABSTRACT

Verifying the correctness and consistency of multi-step reasoning chains generated by large language models to solve reasoning problems remains a challenge. Current verification strategies treat reasoning as strictly linear chains(Chain-of-Thought) or rigid trees (Tree-of-Thought). These approaches fail to capture complex dependencies in mathematical reasoning. AtomGraph is a graph neural network approach that represents reasoning chains as directed acyclic graphs where nodes act as atomic reasoning steps and edges encode logical dependencies. AtomGraph also combines semantic embeddings with structural features then applies multi head graph attention to learn the dependencies which matter the most. On the GSM8k dataset, AtomGraph achieves 75.9% F1 score outperforming CoT, ToT baselines by 386%. Further analysis reveals that AtomGraph learns to assign significantly higher attention to computation nodes than reasoning nodes ($\mu = 0.70$ vs $\mu = 0.478$). Skip connections receive higher attention and demonstrate that reasoning verification benefits from explicit modeling of multi-hop dependencies rather than treating reasoning as linear.

1 INTRODUCTION

LLMs have transformed natural language processing, with many models solving complex problems via step-by-step reasoning. Chain of ThoughtWei et al. (2022) prompting and its variants are some of the techniques for eliciting structural reasoning from LLMs by breaking complicated problems into steps. However, how do we verify that the generated reasoning is correct?

Consider a simple arithmetic problem : "Sarah has 12 apples. She gives 3 to John and 2 to Mary. Then she buys 5 more apples. How many does she have?". CoT produces sequential steps but the final calculation depends on multiple earlier values and previous calculations. The reasoning graphs capture skip connections, parallel computations and reconvergent paths. Linear models cannot capture this structure.

1.1 GRAPH BASED REASONING VERIFICATION

Mathematical reasoning can be represented as directed acyclic graphs (DAGs) where:

- Nodes represent atomic steps
- Edges encode logical dependencies across steps
- Graph structure captures parallel reasoning paths and skip connection.

This representation is more expressive, interpretable, robust and learnable

1.2 ATOMGRAPH

AtomGraph is a graph neural network approach for verifying reasoning chains. This consists of four key components

1. **Reasoning Decomposition:** Solutions are parsed into structured DAGs by extracting reasoning steps and identifying numerical dependencies.

2. **Rich Node Representations:** Each node is represented by semantic structural and graph based features.
3. **Multi-Head Graph Attention:** Graph attention networks are applied to 4 attention heads to learn which edges are most important for verificationVeličković et al. (2017).
4. **Contrastive Consistency Learning:** A combined loss function trains consistent reasoning graphs to cluster together in embedding space while keeps the inconsistent ones apartGao et al. (2021).

1.3 Key Contributions

Our work makes the following contributions:

- **A**tomGraph is the first graph-based approach for mathematical reasoning verification that is different from linear chains or rigid trees.
- **A**tomGraph achieves 75.9% F1 on GSM8K verification, outperforming Chain-of-Thought baselines by 386% and strong machine learning baselines by 16.8%.
- **A**cross 5,330 edges from 600 test graphs, we demonstrate that learned attention weights correlate with logical importance.

2 Literature Review

2.1 Language models for reasoning

Recent work shows that large language models solve mathematical problems through prompting strategies. Chain-of-Thought(CoT) prompting Wei et al. (2022) shows that few-shot examples with intermediate steps improve performance on arithmetic and reasoning tasks. Tree-of-Thought Yao et al. (2023) demonstrated this to explore multiple reasoning paths by treating problem solving patterns as a tree search. Most recently, Atoms of Thoughts Teng et al. (2025) decompressed reasoning into even finer-grained atomic steps for better generation.

These approaches focus on generation rather than verification. Self consistency Wang et al. (2022) verifies by sampling multiple reasoning paths and taking the majority vote, however this requires expensive inference and does'nt fully explain why reasoning fails. AtomGraph complements these by providing verification of reasoning chains.

2.2 Reasoning Verification

Prior work on reasoning verification has taken various different approaches. Cobbe et al.Cobbe et al. (2021) trains verifiers to judge the correctness of solutions to grade-school-math-problems. This approach treats verification as a classification task over flat text representations. Uesato et al. Uesato et al. (2022) leverage process based feedback to train step-level verifiers. Lightman et al. Lightman et al. (2023) extend this approach by scaling step level verification using human feedback at every reasoning step.

These methods treat reasoning as sequences and chain, these lack explicit structural modeling.

2.3 Graph Neural Networks

Graph Neural Networks are effective models to learn structured data. Graph Convulational Networks(GCNs)Kipf (2016) average the neighborhood features by spectral convolutions. Graph Attentional Transformers (GATs)Veličković et al. (2017) demonstrated attention mechanisms to weight edges by learned importance. GNN has been applied to NLP tasks including semantic parsingLi et al. (2020) and knowledge graph reasoning Zhang & Yao (2022) AtomGraph applies GNNs for reasoning verification where the graph structure emerges from logical dependencies rather than being pre defined. The attention weights provide information on what dependencies matter the most.

2.4 CONTRASTIVE LEARNING

Contrastive learning learns representation by representations by putting similar examples together and dissimilar ones apart. SimCLR Chen et al. (2020) applied this to vision whereas Gao.et.al applied contrastive learning for NLP Gao et al. (2021). AtomGraph uses contrastive loss for reasoning verification and it ensures consistent reasoning graphs to cluster.

2.5 ATOMGRAPH

Prior verification methods use flat reasoning sequences or rigid trees whereas AtomGraph uses flexible DAG with learned dependencies that capture skip connections. AtomGraph learns interpretable patterns as computation nodes receive 46% higher attention , showing that graph structure enables meaningful verification beyond simple sequence modeling.

3 METHOD

3.1 PROBLEM FORMULATION

Given a reasoning problem q and it's reported solution s having reasoning steps. Our main goal is to classify whether s is a correct reasoning chain. This is formulated as a binary classification task. Each reasoning chain is represented as a directed acyclic graph containing nodes and edges.

- Nodes $v_i \in V$ represent reasoning steps.
- Edges $(v_i, v_j) \in E$ represent logical dependencies between steps.

3.2 REASONING DECOMPOSITION

3.2.1 STEP EXTRACTION

A certain pattern is used in the GSM8K dataset. This is represented as `<<computation>>reasoning text<<computation>>...`. We split this on delimiters and we get computational steps and reasoning steps.

3.2.2 CONTEXTUAL NUMBER EXTRACTION

For each step, numbers and surrounding context is captured. This gives us triples as shown below: `(prefix, number, suffix)`, e.g., `''has 25 apples''` $\rightarrow$ `(''has'', 25.0, ''apples'')`.

3.2.3 DEPENDENCY CONSTRUCTION

Edges are based on multiple dependent types. These dependencies are : sequential, numerical, question grounding and terminal dependencies.

3.3 FEATURE REPRESENTATION

Each node v_i is represented by concatenating semantic and structural features.

3.3.1 SEMANTIC FEATURES

Sentence-BERT(all-MiniLM-L6-v2)is used to encode node text into embeddings. Through this we capture the semantic meaning of reasoning steps enabling the model to understand mathematical operations, logical relationships and contextual understanding between numbers. all-MiniLM-L6-v2 is chosen for it's compact 384 dimensional embeddings and it's computational efficiency.

3.3.2 STRUCTURAL FEATURES

While semantic features captures what a reasoning step says, structural features encode where the reasoning step sits in the logical flow and what role it plays. A 12 dimensional structural feature vector is constructed for each node. The 12 feature vectors are listed below :

- Type Encoding(4 dim): Checks is this node a question, reasoning step, computation or final answer?
- Position Features (3 dim): Checks how deep the node in the reasoning chain is. Is it the root or leaf?
- Content Features (4 dim) : Checks if the step contains numbers and the length of the text.
- Placeholder for future extensions.

These features provide the model with signals about the reasoning structure that complement the semantic content.

3.3.3 GRAPH-LEVEL FEATURES

To capture all the global reasoning patterns of the entire graph G we calculate the structural properties. Graph features enables the model to reason about complexity, balance and structure.

3.4 NEGATIVE EXAMPLE GENERATION

Positive and negative examples are needed to train a reasoning verifier. Using random corruptions to capture detectable errors does not train the model to capture subtle mistakes. GS8Mk contains only correct solutions, without negative examples the model would simply learn to output 'consistent' for everything. A good negative dataset should be created that is subtle(should not be obviously wrong,2+2=5), realistic(should mimic human errors) and localised(only 1-2 errors per example).

3.4.1 NEGATIVE GENERATION STRATERGY

Negative examples are created by taking a correct reasoning graph and introducing one of two types of errors:

Type 1: Wrong Computation This simulates arithmetic errors where the calculation is wrong. We start with a correct graph and find a random computation node. Then we pick a number in that computation and correct that number by scaling(multiply by 0.8, 0.9, 1.1, or 1.2) or by shifting(add -5, -3, +3, or +5) it. Lastly we replace the original computation with the corrupted one. This is represented by an example below:

Original (Correct):

- Node 3: "She started with 12 apples"
- Node 4: "$\langle\langle 12 - 3 = 9 \rangle\rangle$" [COMPUTATION]
- Node 5: "She has 9 apples left"

After Corruption:

- Node 3: "She started with 12 apples"
- Node 4: "$\langle\langle 12 - 3 = 10 \rangle\rangle$" [COMPUTATION - WRONG!]
- Node 5: "She has 9 apples left" [Now inconsistent with node 4]

The error is subtle, the text makes sense and the graph structure remains the same.

Type 2: Broken Dependency (50% of negatives) The error is simulated here by logical gaps where the conclusion does not follow its premises.

In this case, we start with a correct graph and identify and pick a skip connection edge(edges that jump over steps) and deleted that edge. We verify that the graph is still a valid DAG after deletion.

Original (Correct):

Nodes: [0:Question, 1:Step1, 2:Step2, 3:Step3, 4:Answer]

Edges:

$(0, 1)$,	# Question $\rightarrow$ Step1
$(1, 2)$,	# Step1 $\rightarrow$ Step2 [sequential]
$(1, 3)$,	# Step1 $\rightarrow$ Step3 [SKIP CONNECTION]
$(2, 3)$,	# Step2 $\rightarrow$ Step3 [sequential]
$(3, 4)$	# Step3 $\rightarrow$ Answer

Imagine Step 3 is: "Using the 12 from Step 1 and adding the 5 from Step 2, we get 17".

After Corruption (remove edge $(1, 3)$):

Edges:

$(0, 1)$,	# Question $\rightarrow$ Step1
$(1, 2)$,	# Step1 $\rightarrow$ Step2
$(2, 3)$,	# Step2 $\rightarrow$ Step3
$(3, 4)$	# Step3 $\rightarrow$ Answer

Edge $(1, 3)$ removed - Step 3 claims to use "12 from Step 1" but has no direct edge to it. The graph is still a DAG and connected and step 3's text is unchanged but now there is a logical gap for the model to learn.

3.4.2 Ensuring Subtlety

To prevent obviously broken examples we impose constraints.

For Wrong Computation:

- Error magnitude is small.
- Only one number in one node is changed.
- Graph structure remains identical.

For Broken Dependency:

- Only the skip connections are removed.
- The graph is never fully disconnected.
- Verify the result is still a valid DAG.

78% of the negative examples pass structural validation making them hard to detect without semantic context.

3.5 Model Architecture

The model architecture consists of: graph encoding, attention based refinement, graph pooling and consistency learning.

3.5.1 Graph Convolutional Encoding

A GCN layer performs local aggregation, this allows each node to incorporate information through it's dependencies and capture logical relationships Kipf (2016).

3.5.2 MULTI-HEAD GRAPH ATTENTION

Two GAT layers are applied to learn edge importance with $K = 4$ attention heads Veličković et al. (2017).

Interpretation of Attention weights

- Attention asssigns higher weight to computation nodes ($\mu = 0.70$ vs 0.478, $p < 0.001$).
- Elevated attention on skip attention (depth $+2$, $+3$;($\mu = 0.76, 0.81$)) support non local dependency modelling.
- Multi-head structure captures various reasoning patterns.

3.5.3 GRAPH POOLING

Mean and Max Pooling are combined for robust graph representation. Through this method, overall reasoning pattern and most salient features are captured. Graph structural features are captured to leading the final dimension to be 266(128+128+10). A combined approach is robust to graph size variations and captures critical steps.

3.5.4 CLASSIFICATION HEAD

Final prediction uses a two-layer MLP with residual connections. Table-1 shows the entire architecture.

Architecture Summary:

Table 1: Model Architecture Summary

Layer	**Transformation**
Input	Graph G with node features $X \in \mathbb{R}^{\|V\| \times 396}$, edges E, graph features $g \in \mathbb{R}^{10}$
GCN Layer	$396 \rightarrow 128$ dimensions
GAT Layer 1	$128 \rightarrow 512$ dimensions (4 heads $\times$ 128)
GAT Layer 2	$512 \rightarrow 512$ dimensions (4 heads $\times$ 128)
GCN Refinement	$512 \rightarrow 128$ dimensions
Graph Pooling	$128 + 128 + 10 \rightarrow 266$ dimensions
MLP Layer 1	$266 \rightarrow 128$ dimensions
MLP Layer 2	$128 \rightarrow 64$ dimensions (embedding)
Output Layer	$64 \rightarrow 2$ dimensions (binary classification)

3.6 LOSS FUNCTION

A combined loss function is used to capture the information better.

3.6.1 CONTRASTIVE LOSS FORMULATION

A contrastive loss improves quality and separation and encourages consistent graphs to cluster together while inconsistent graphs to be apart Gao et al. (2021).

3.6.2 COMBINED LOSS FUNCTION

The combined loss is given by

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{contrastive}}$$

- $\mathcal{L}_{\text{CE}}$ is the Weighted cross-entropy loss

$$\mathcal{L}_{\text{CE}} = -\sum_{i} w_{y_i} \log(\hat{y}_i)$$

  with class weights $w_0 = 1.0$, $w_1 = 1.0$ (balanced after data generation)
- $\lambda = 0.1$: Contrastive loss weight

4 RESULTS

4.1 EXPERIMENTAL SETUP AND BASELINES

We sample 1,500 examples from GSM8K's 7,473 training problems, ensuring diversity across difficulty levels. An equal number of negative examples with subtle errors are generated, yielding 3,000 total examples split 70/10/20 for train/validation/test.

4.1.1 IMPLEMENTATION DETAILS

Sentence-BERT(all-MiniLM-L6-v2) embeddings with 384 dimensional embeddings are used with 12 structural concatenated features per node. The model uses a hidden dimension of 128, 4 attention heads, dropout of 0.3 and is trained with AdamW with early stopping. Code is implemented on NVIDIA T4X2 GPUs.

4.1.2 BASELINES

A thorough comparison is done against: Chain-of-Thought(CoT): Structural verification checking for linear chains. Tree-of-Thought(ToT): Verification based on tree structure properties. Random Forest(RF): Ensemble method on flat graph features. Gradient Boosting(GB): Boosted trees. Vanilla GNN: 3-layer GCN without attention. GNN+Attention: GCN with multi-head GAT layers.

4.2 MAIN RESULTS

Table 2: Performance Comparison on GSM8K Reasoning Verification

Method	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**
CoT Baseline	44.2%	32.0%	10.3%	15.6%	—
ToT Baseline	49.3%	47.0%	10.3%	16.9%	—
Random Forest	55.7%	55.4%	58.3%	56.8%	54.8%
Gradient Boosting	62.5%	60.9%	69.7%	65.0%	62.8%
Vanilla GNN	53.0%	52.3%	68.3%	59.2%	49.7%
GNN + Attention	68.3%	62.6%	91.3%	74.2%	63.2%
Full Model (Ours)	**70.7%**	**64.4%**	**92.3%**	**75.9%**	**65.4%**

Key Findings:

- CoT and ToT fail at verification, yielding only 15.6% and 16.9% F1 scores.
- AtomGraph achieves 75.9% F1 score, achieving a 386% improvement over CoT and 348% improvement over ToT.
- Attention mechanisms are useful as they contribute over 15 points of F1 score over vanilla GNN.

4.3 ATTENTION ANALYSIS

Attention patterns are analyzed to understand what exactly the model focuses during verification.

Table 3: Attention Weight Statistics

Edge Property	**Mean Attention**	**Std Dev**	**Count**	**Statistical Test**
Source = Computation	0.700	0.298	1,850	t=22.8, p<0.001
Source = Reasoning	0.478	0.357	2,114	baseline
Sequential (depth +1)	0.535	0.374	4,266	—
Skip Connection (depth +2)	0.756	0.247	98	—
Skip Connection (depth +3)	0.814	—	38	—

The model learns to focus on critical mathematical operations as computational nodes receive 46% higher attention than reasoning nodes ($p<1$).

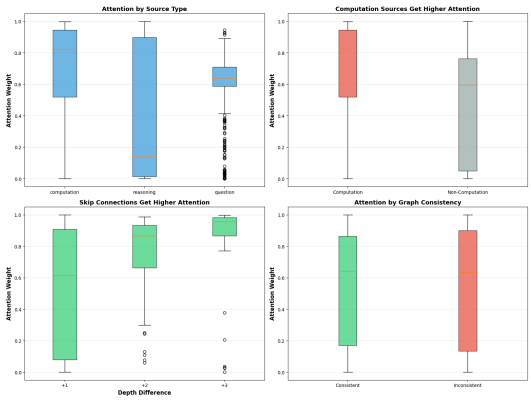

Figure 1: Distribution of attention weights by edge properties. (Top-left) Computation nodes receive higher attention than reasoning or question nodes. (Top-right) The computation vs. non-computation gap is statistically significant. (Bottom-left) Skip connections (depth +2, +3) receive higher attention than sequential edges. (Bottom-right) Attention is similar for consistent and inconsistent graphs, indicating structure-focused behavior.

AtomGraph identifies non-local dependencies too as skip connections receive 41% higher attention than sequential edges (0.756 vs 0.535).

4.4 Embedding Space Quality

Table 4: Learned Representation Analysis

Metric	Value	Interpretation
Silhouette Score	0.255	Moderate cluster separation
NN Purity (k=10)	67.8%	Good local consistency
Calibration (ECE)	0.132	Well-calibrated predictions

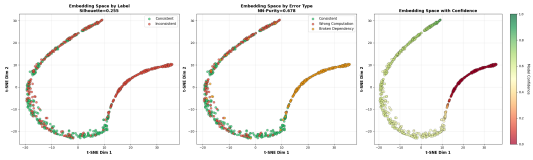

Figure 2: t-SNE visualization of the 64-dimensional learned embeddings.

t-SNE visualization shows that consistent and inconsistent reasoning graphs form separate clusters with Silhouette scores of 0.255 which shows meaningful separation. Low ECE score indicates the model's confidence matches with the actual accuracy.

5 Conclusion

AtomGraph is a graph neural network for verifying mathematical reasoning.By representing reasoning chains as DAG, AtomGraph achieves 75.9% F1 score, a 386% improvement over Chain-of-Thought baselines. The key insight from this paper is that reasoning verification requires understanding multi-hop sequences and not sequential steps. This paper shows that graph based approaches learn meaningful patterns aligned with mathematical reasoning. A limitation is that AtomGraph requires pre-parsed reasoning graphs. Future work should explore error aware architectures and an extension to formal theorem proving. As language models evolve in reasoning, strong verification mechanisms are essential for safe deployment in education and scientific computing.

REFERENCES

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PmLR, 2020.

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

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. *arXiv preprint arXiv:2004.13781*, 2020.

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 *The Twelfth International Conference on Learning Representations*, 2023.

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. Atom of thoughts for markov llm test-time scaling. *arXiv preprint arXiv:2502.12018*, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

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*, 35:24824–24837, 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*, 36:11809–11822, 2023.

Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pp. 912–924, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390965. doi: 10.1145/3485447.3512008. URL https://doi.org/10.1145/3485447.3512008.

A ADDITIONAL ANALYSIS

The resulting DAGs exhibit the following empirical properties:

Table 5: Graph Properties

Property	**Value**
Average nodes	6.2 ± 2.3 steps
Average edges	7.8 ± 3.1 dependencies
Average degree	1.8 edges per node
Maximum depth	6.2 steps on average

Table 6: Node Type Distribution

Node Type	**Percentage**
Questions	14.2%
Reasoning	38.5%
Computation	35.7%
Answer	11.6%

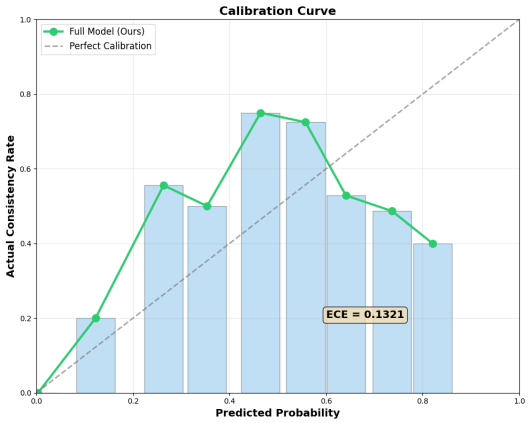

Figure 3: Calibration curve showing predicted probability vs actual consistency rate. The model's confidence (green line) roughly tracks actual performance (gray diagonal = perfect calibration), with an Expected Calibration Error (ECE) of 0.132.