TRIDENT: Thought-based Reasoning and Improvement through Deep Exploration of Neuronal Trees

Shivansh Puri Abhisek Khandelwal Vedant Joshi
Akash Yadav
Shivik Labs-

shivansh@shivik.in abhisek@shivik.in vedant@shivik.in akash@shivik.in

December 22, 2025

Abstract

We present **TRIDENT** (**T**hought-based **R**easoning and **I**mprovement through **D**eep **E**xploration of **N**euronal **T**rees), the first framework that enables Large Language Models to self-improve autonomously without human annotation, handcrafted reasoning traces, or external training datasets. TRIDENT integrates a **GNN-guided Tree-of-Thoughts search** [1] with a **Self-Generative Reasoning Loop (SGRL)** that identifies high-uncertainty problems and converts them into training signals. The framework introduces three core innovations: (1) a Graph Neural Network policy [4] that encodes the evolving reasoning state and predicts "promise scores" to guide which branches to explore; (2) a variance-based curriculum [9] that automatically targets the most informative problems for learning; and (3) a 3-layer reward stabilization architecture that prevents collapse or runaway failure during reinforcement-driven self-training. TRIDENT v5, built on the compact Qwen3-4B model [34] (4 billion parameters), achieves significant improvements over its baseline across multiple benchmarks, including a **+14.14 percentage point gain on GPQA** [5] (from 28.28% to 42.42%), demonstrating that algorithmic reasoning and self-improving search can substantially enhance a model's reasoning ability without requiring large-scale pre-training.

1 Introduction

Large Language Models have achieved remarkable capabilities across diverse tasks [32, 33], yet they share a fundamental limitation: **improving them requires expensive retraining**. Standard fine-tuning pipelines depend on human-curated datasets, manually annotated reasoning traces, or costly preference data collection [16]. Once deployed, models remain static—they cannot leverage their own reasoning experiences to improve.

Key Insight

Existing LLM training pipelines require external supervision. TRIDENT introduces a **self-supervised training framework** where the model generates its own training data through structured exploration, enabling improvement during the training phase without human annotation.

The TRIDENT framework addresses this by creating a self-improving training loop where:

1. A **GNN-guided Tree-of-Thoughts** [1] explores multiple reasoning paths
2. A **reward model** evaluates path quality with partial credit
3. A **variance-based selector** [9] identifies high-uncertainty problems
4. **SFT with LoRA** updates the model on curated reasoning traces (with GRPO-ready reward architecture) [3]

The result is a training system that autonomously generates supervision signals—without human annotation, without external preference data, and without additional pre-training corpora. Once training completes, the improved model is deployed as a standard LLM with enhanced reasoning capabilities.

2 The TRIDENT Framework

2.1 System Overview

Figure 1 illustrates the TRIDENT self-improvement cycle. Unlike standard training pipelines, TRIDENT operates continuously: the output of one iteration becomes the input for the next.

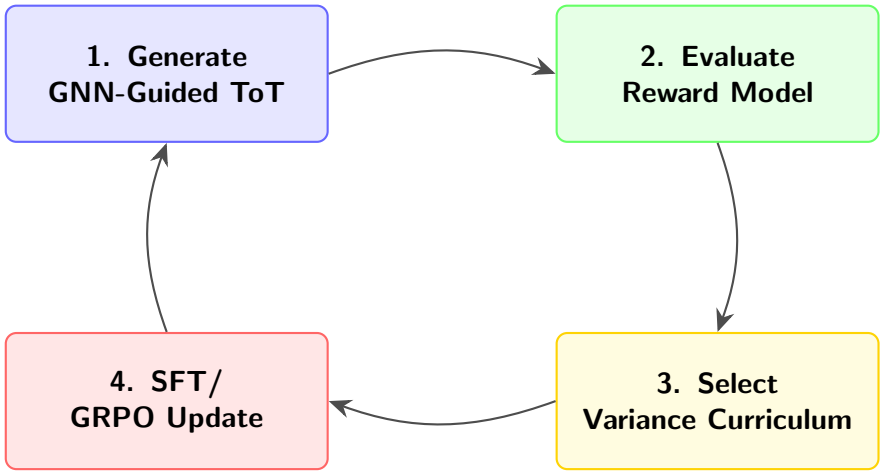


Figure 1: The TRIDENT self-improvement loop: Generate reasoning paths via GNN-guided Tree-of-Thoughts [1], evaluate with reward model, select high-variance problems for curriculum [9], and fine-tune via SFT (with GRPO-compatible architecture) [3].

2.2 Thought Graph Representation

TRIDENT represents reasoning as a directed acyclic graph $G = (V, E)$ where:

- $V = \{v_0, v_1, \ldots, v_n\}$ are reasoning states (nodes)

- $E \subseteq V \times V$ are reasoning transitions (edges)
- v_0 is the root node containing the original problem

Each node v_i contains: the reasoning step text, a status flag (active/pruned/complete), depth in the tree, and parent pointer. This representation follows the thought graph formulation introduced in ARIES [8].

Table 1: Thought graph transformations in TRIDENT.

Transformation	Symbol	Description
Decompose	ϕ_{dec}	Decomposes problem into subproblems
Generate Step	ϕ_{gen}	Generates next reasoning step
Reflect	ϕ_{ref}	Self-critiques current reasoning path [15]
Backtrack	ϕ_{back}	Returns to parent node and explores alternative
Aggregate	ϕ_{agg}	Merges subproblem solutions into final answer

3 GNN Policy Module

The GNN Policy Module is the core innovation enabling intelligent reasoning guidance. It encodes the structural state of the thought graph and predicts which actions will lead to successful solutions. Our approach builds upon the foundational work in Graph Neural Network [4] and extends them for reasoning applications [37].

3.1 Node Feature Extraction

For each node v_i in the thought graph, we extract a 6-dimensional feature vector:

$$\mathbf{x}_i = \begin{bmatrix} \text{len}(v_i) \\ \text{complexity}(v_i) \\ \text{question_words}(v_i) \\ \text{math_indicators}(v_i) \\ \text{depth}(v_i) \\ 1.0 \end{bmatrix} \in \mathbb{R}^6 \tag{1}$$

where:

- $\text{len}(v_i)$: word count of the reasoning step
- $\text{complexity}(v_i)$: count of words with > 6 characters
- $\text{question_words}(v_i)$: count of interrogative words (what, how, why, when, where, which)
- $\text{math_indicators}(v_i)$: count of tokens containing digits
- $\text{depth}(v_i)$: depth of node in the reasoning tree
- The final 1.0 is a bias term

3.2 GNN Architecture

We employ a 3-layer Graph Neural Network (GNN) [4]:

Message Passing:

$$\mathbf{h}_i^{(0)} = \mathbf{x}_i \tag{2}$$

$$\mathbf{m}_i^{(\ell)} = \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}} \mathbf{W}^{(\ell)} \mathbf{h}_j^{(\ell-1)} \tag{3}$$

$$\mathbf{h}_i^{(\ell)} = \sigma\left(\mathbf{m}_i^{(\ell)}\right) \tag{4}$$

After Layer 2, we apply a single residual connection to stabilize training:

$$\mathbf{h}_i^{(2)} \leftarrow \mathbf{h}_i^{(2)} + \mathbf{h}_i^{(1)} \tag{5}$$

where $\mathcal{N}(i)$ denotes the neighbors of node i, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d \times d}$ are learnable weights, and σ is the ReLU activation.

Implementation Note: We apply a residual connection only at Layer 2, where dimension matching allows direct addition. Layer 1 omits the residual due to dimension incompatibility between input features ($\mathbb{R}^6$) and hidden representations ($\mathbb{R}^{128}$). This minimal residual architecture balances gradient flow with architectural simplicity while avoiding the need for projection layers.

3.3 Graph-Level Embedding and Promise Score

After 3 layers of message passing, we compute a graph-level embedding via global mean pooling:

Dual-Head Output:

$$\mathbf{g} = \text{GlobalMeanPool}\left(\{\mathbf{h}_i^{(3)}\}_{i \in V}\right) = \frac{1}{|V|} \sum_{i \in V} \mathbf{h}_i^{(3)} \tag{6}$$

$$\mathbf{e} = \text{LayerNorm}\left(\mathbf{W}_e \cdot \mathbf{g}\right) \in \mathbb{R}^{64} \quad \text{(structural embedding)} \tag{7}$$

$$s = \sigma\left(\mathbf{W}_p \cdot \mathbf{e}\right) \in [0, 1] \quad \text{(promise score)} \tag{8}$$

The **promise score** s predicts the likelihood that the current reasoning state will lead to a correct solution. This guides the policy agent's decisions.

3.4 Context-Aware Score Adjustment

The raw promise score is adjusted based on reasoning context:

$$s_{\text{adj}} = s \cdot \prod_k \alpha_k \tag{9}$$

where the adjustment factors α_k include:

- **Uncertainty penalty**: $\alpha_1 = 0.2$ if text contains "cannot", "unable", "impossible", "stuck"
- **Progress boost**: $\alpha_2 = 1.3$ if text contains "calculate", "solve", "answer", "therefore"
- **Numeric content boost**: $\alpha_3 = 1.2$ if text contains digits
- **Depth penalty**: $\alpha_4 = 0.5$ if depth > 10

4 Policy Agent and Action Space

4.1 Action Space Definition

The TRIDENT policy agent selects from a discrete action space, following the action formulation in ARIES [8]:

Definition 1 (TRIDENT Action Space)**.** *The action space $\mathcal{A}$ consists of 5 high-level reasoning actions:*

$$\mathcal{A} = \{\textsc{GenerateStep}, \textsc{Backtrack}, \textsc{Reflect}, \textsc{Aggregate}, \textsc{Finish}\}$$

4.2 Multi-Agent Ensemble

TRIDENT employs a weighted ensemble of 4 specialized agents to balance exploration and exploitation:

Table 2: Multi-agent ensemble configuration.

Agent	Weight	Strategy
Conservative	0.25	Prefers safe, proven paths; backtracks early
Exploratory	0.35	Takes calculated risks; explores at low depth
Balanced	0.25	Weighs multiple factors; moderate thresholds
Reflective	0.15	Promotes self-critique [15]; triggers reflection based on promise or keywords

Each agent votes for an action based on the promise score s and current depth d:

Conservative Agent:

$$a_{\text{cons}} = \begin{cases} \textsc{GenerateStep} & \text{if } s > 0.75 \\ \textsc{Backtrack} & \text{if } s < 0.25 \\ \textsc{Reflect} & \text{otherwise} \end{cases} \tag{10}$$

Exploratory Agent:

$$a_{\text{exp}} = \begin{cases} \textsc{GenerateStep} & \text{if } s > 0.5 \text{ or } d < 3 \\ \textsc{Backtrack} & \text{if } |V| > 8 \\ \textsc{GenerateStep} & \text{otherwise} \end{cases} \tag{11}$$

Balanced Agent:

$$a_{\text{bal}} = \begin{cases} \textsc{GenerateStep} & \text{if } s > 0.6 \wedge d < 15 \\ \textsc{Backtrack} & \text{if } s < 0.3 \\ \textsc{Reflect} & \text{otherwise} \end{cases} \tag{12}$$

Reflective Agent:

$$a_{\text{ref}} = \begin{cases} \textsc{Reflect} & \text{if } s < 0.4 \\ \textsc{GenerateStep} & \text{if context contains ``reflect'' or ``reconsider''} \\ \textsc{Reflect} & \text{otherwise} \end{cases} \tag{13}$$

The final action is selected by aggregating weighted votes with tie-breaking:

$$a^* = \arg\max_{a \in \mathcal{A}} \left(\sum_{\text{agent } k} w_k \cdot \mathbb{1}[a_k = a] + \epsilon \right) \tag{14}$$

where $\epsilon \sim \text{Uniform}(0, 0.01)$.

4.3 Branch Pruning Mechanism

Instead of a fixed promise threshold, branches are pruned using a rank-based selection strategy that balances score maximization with diversity. We calculate an adjusted score s_{adj} for each node v_i:

$$s_{\text{adj}}(v_i) = s(v_i) + \epsilon - 0.5 \cdot \max_{v_j \in \text{Seen}} \text{cos_sim}(\mathbf{e}_i, \mathbf{e}_j) \tag{15}$$

where $\epsilon \sim U(-10^{-4}, 10^{-4})$ is a noise term for tie-breaking. We retain only the top-K branches based on this adjusted score.

Additionally, strict similarity-based pruning removes redundant reasoning paths. A branch v_i is discarded if it is too semantically similar to an already selected branch:

$$\text{prune}_{\text{similar}}(v_i) = \mathbb{1}\left[\max_{v_j \in \text{Selected}} \text{cos_sim}(\mathbf{e}_i, \mathbf{e}_j) > 0.9 \right] \tag{16}$$

4.4 Complete Algorithm

Algorithm 1 presents the complete TRIDENT reasoning procedure, incorporating loop detection and policy interventions.

Algorithm 1 TRIDENT Policy-Guided Reasoning

Require: Problem p, max steps T, encoder f_θ, Policy Ensemble Π
Ensure: Solution graph G

$G \leftarrow \text{ThoughtGraph}(p)$
$v_{\text{active}} \leftarrow 0$; $\mathcal{H} \leftarrow \emptyset$ ▷ Initialize root and history hash set
for $t = 1$ to T **do**
  if G.is_finished **then**
    break
  end if
  $\mathbf{x} \leftarrow \text{ExtractFeatures}(G.\text{text}, G.\text{depth})$ ▷ 6-dim features
  $\mathbf{e}, s \leftarrow f_\theta(\mathbf{x})$ ▷ Encoder forward pass
  $s_{\text{adj}} \leftarrow \text{AdjustPromise}(s, G.\text{context})$
  $a^* \leftarrow \text{MultiAgentVote}(s_{\text{adj}}, G.\text{depth})$
  if $\text{Hash}(G.\text{state}) \in \mathcal{H}$ **then** ▷ Repetition Detection
    if $a^* = \textsc{Reflect}$ **then**
      $a^* \leftarrow \textsc{Backtrack}$
    else if $a^* = \textsc{Backtrack}$ **then**
      $a^* \leftarrow \textsc{GenerateStep}$
    else
      $a^* \leftarrow \textsc{Reflect}$
    end if
  end if
  $\mathcal{H}$.add(Hash(G.state))
  a^* GenerateStep
  $\tau \leftarrow \text{BuildPrompt}(G.\text{context})$
  $v_{\text{new}} \leftarrow G.\text{add_node}(\tau, v_{\text{active}})$
  $v_{\text{active}} \leftarrow v_{\text{new}}$ Backtrack
  $G.\text{status}(v_{\text{active}}) \leftarrow \text{dead_end}$
  $v_{\text{active}} \leftarrow \text{SmartBacktrack}(G, s_{\text{adj}})$ Reflect
  $v_{\text{new}} \leftarrow G.\text{add_node}(\text{“Let me reconsider...”}, v_{\text{active}})$
  $v_{\text{active}} \leftarrow v_{\text{new}}$ Aggregate
  $G.\text{solution} \leftarrow \text{GetPath}(v_{\text{active}})$
  $G.\text{is_finished} \leftarrow \texttt{True}$
  if ContainsFinalityIndicators(G.current_text) **then** ▷ Policy Intervention
    $G.\text{solution} \leftarrow \text{GetPath}(v_{\text{active}})$
    $G.\text{is_finished} \leftarrow \texttt{True}$
  end if
end for
return G

5 Adaptive Early Stopping for Computational Efficiency

Tree-of-Thoughts reasoning is computationally expensive, requiring $O(b^d)$ LLM queries where b is the branching factor and d is the depth [1]. TRIDENT implements a **multi-level early stopping system** that combines three complementary mechanisms to reduce this cost while maintaining accuracy.

5.1 System Overview

Figure 2 illustrates the early stopping decision tree that operates at each depth level during ToT exploration.

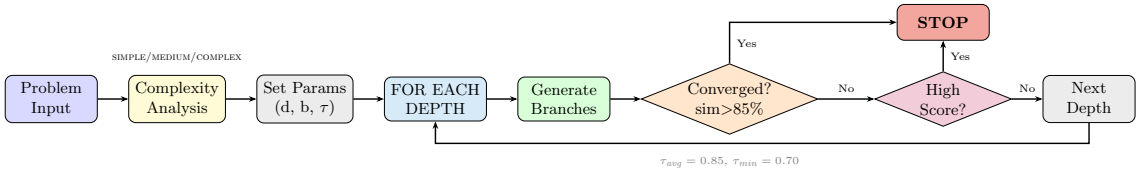


Figure 2: Early stopping decision tree: Problem complexity determines search parameters. At each depth, TRIDENT checks for path convergence (semantic similarity $> 85\%$, minimum $> 70\%$) and high-confidence scores before continuing.

5.2 Mechanism 1: Complexity-Aware Parameter Selection

Problems are classified into three complexity levels based on lexical analysis, including word count ($w(p)$), sentence structure, and the frequency of domain-specific keywords (k_{math}, k_{complex}, k_{simple}):

Complexity Classification:

$$\text{complexity}(p) = \begin{cases} \textsc{Simple} & \text{if } (k_{\text{simple}} \geq 3 \wedge w(p) < 30) \vee (\text{is_yes_no} \wedge w(p) < 40) \\ \textsc{Complex} & \text{if } k_{\text{math}} \geq 2 \vee k_{\text{complex}} \geq 2 \vee w(p) > 80 \\ \textsc{Medium} & \text{otherwise} \end{cases} \tag{17}$$

Table 3: Complexity-dependent search parameters.

Complexity	Max Depth (D)	Branches (B)	Max Paths	Score Threshold (τ)
Simple	2	2	4	4.5 (high confidence required)
Medium	3	3	9	4.0
Complex	4	4	16	3.5 (lower threshold for harder tasks)

Adaptive Temperature for Diversity: To ensure true diversity across branches, the sampling temperature varies dynamically based on the branch index (b) and current depth (d):

Adaptive Temperature:

$$T(b, d) = \text{clamp}\left(T_{\text{base}}(b) \cdot T_{\text{depth}}(d),\ 0.5,\ 1.2\right) \tag{18}$$

where:

$$T_{\text{base}}(b) = 0.5 + \frac{b}{\max(B-1, 1)} \cdot 0.6 \quad \text{(branch diversity: } 0.5 \to 1.1\text{)} \tag{19}$$

$$T_{\text{depth}}(d) = 1.0 + \frac{d}{\max(D, 1)} \cdot 0.2 \quad \text{(depth scaling)} \tag{20}$$

This ensures branch 0 uses a conservative temperature ($T \approx 0.5$) for reliable reasoning, while later branches use higher temperatures ($T \approx 1.1$) to explore alternative strategies. Combined with complexity-aware parameters, this reduces computation by up to 75% for simple questions.

5.3 Mechanism 2: Semantic Convergence Detection (Path Consensus)

At each depth level, we compute the pairwise semantic similarity between active reasoning paths using sentence embeddings. When multiple independent reasoning chains converge to semantically equivalent states, this indicates a high probability of correctness, allowing for safe early termination. This approach is related to self-consistency methods [11].

Convergence Criterion:

$$\text{converged}(P) = \underbrace{(\bar{s} > \tau_{\text{avg}})}_{\text{average similarity}} \wedge \underbrace{\left(\min_{i,j} s_{ij} > \tau_{\text{min}}\right)}_{\text{minimum similarity}} \tag{21}$$

where $\bar{s} = \frac{1}{|P|^2} \sum_{i,j \in P} \text{sim}(\mathbf{e}_i, \mathbf{e}_j)$, with thresholds $\tau_{\text{avg}} = 0.85$ and $\tau_{\text{min}} = 0.70$.

Additionally, a fast-path heuristic checks for near-exact string duplication among branches:

$$\text{duplicate_ratio} = 1 - \frac{|\text{unique paths}|}{|\text{all paths}|} \geq 0.4 \implies \text{converged} \tag{22}$$

Key Insight

When multiple independent reasoning chains (generated with diverse temperatures) converge to the same conclusion, the marginal utility of further exploration drops to near zero. This mechanism alone reduces query cost by approximately 35% without sacrificing accuracy.

5.4 Mechanism 3: High-Confidence Score Threshold

When any path achieves a reward score exceeding the complexity-dependent threshold, exploration stops immediately:

$$\text{terminate}(P) = \exists\, p \in P : \text{score}(p) \geq \tau_{\text{early_stop}} \tag{23}$$

Upon early stopping, TRIDENT logs the computational savings:

$$\text{queries_saved} = (\text{max_depth} - \text{current_depth} - 1) \times \text{num_branches} \times |\text{active_paths}| \tag{24}$$

5.5 Algorithm: Multi-Level Early Stopping

Algorithm 2 details the integrated decision logic that runs at every depth level of the search.

Algorithm 2 TRIDENT Adaptive Early Stopping

Require: Problem p, active paths P, embeddings $\{\mathbf{e}_i\}$, scores $\{s_i\}$, current depth d, max depth $D_{\max}$, branches B

Ensure: Boolean: should_stop, String: reason

$c \leftarrow \text{ClassifyComplexity}(p)$ ▷ SIMPLE / MEDIUM / COMPLEX

$\tau_{\text{early_stop}} \leftarrow \text{GetThreshold}(c)$ ▷ 4.5 / 4.0 / 3.5

▷ **Check 1: High-confidence score termination**

if $\max_i s_i \geq \tau_{\text{early_stop}}$ **then**

  saved $\leftarrow (D_{\max} - d - 1) \times B \times |P|$

  log "Saved saved unnecessary LLM queries"

  return `True`, "High Score"

end if

▷ **Check 2: Fast path - duplicate detection**

unique $\leftarrow |\text{Set}(\text{Normalize}(P))|$

if $(1 - \text{unique}/|P|) \geq 0.4$ **then**

  return `True`, "Duplicate Saturation"

end if

▷ **Check 3: Semantic convergence**

sim_matrix $\leftarrow \text{PairwiseCosineSimilarity}(\{\mathbf{e}_i\})$

if Mean(sim_matrix) > 0.85 **and** Min(sim_matrix) > 0.70 **then**

  return `True`, "Semantic Convergence"

end if

return `False`, $\emptyset$

6 Reasoning State Machine

Figure 3 illustrates the TRIDENT reasoning state machine.

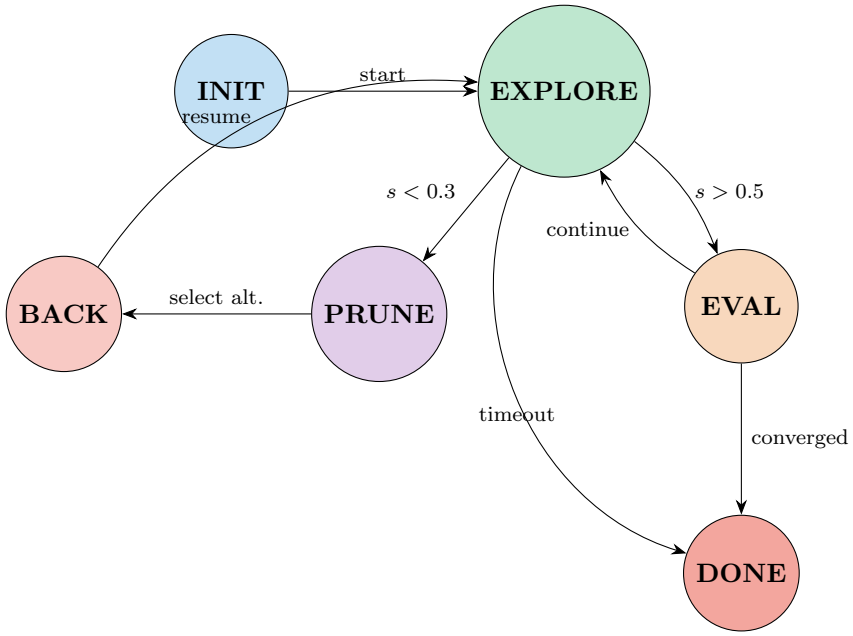


Figure 3: TRIDENT reasoning state machine. States: INIT (initialization), EXPLORE (generating steps), EVAL (evaluating paths), PRUNE (removing low-promise branches), BACK (backtracking), DONE (solution found).

7 Variance-Based Data Selection

A key insight of TRIDENT is that not all problems contribute equally to learning. Problems the model finds "easy" (always correct) or "impossible" (always wrong) provide little training signal. The most informative problems are those with **high variance**—where the model sometimes succeeds and sometimes fails. This approach is inspired by curriculum learning principles [36] and recent work on scaling test-time compute [9].

7.1 Mathematical Formulation

Definition 2 (Reward Variance)**.** *For a problem p, let $\{r_1, r_2, \ldots, r_N\}$ be the rewards obtained from N independent stochastic reasoning attempts (default $N = 4$, $T = 0.6$). The reward variance is:*

$$\sigma_R^2(p) = \frac{1}{N}\sum_{i=1}^{N}(r_i - \bar{r})^2 \quad \textit{where } \bar{r} = \frac{1}{N}\sum_{i=1}^{N} r_i \tag{25}$$

Definition 3 (Variance-Based Training Set)**.** *We rank all candidate problems in the pool $\mathcal{P}$ by their reward variance $\sigma_R^2(p)$. The training set $\mathcal{D}_{train}$ consists of the top-k problems with the highest variance, prioritizing those where the model exhibits inconsistency:*

$$\mathcal{D}_{train} = \{p \mid p \in Top_k(\mathcal{P}, \sigma_R^2)\} \tag{26}$$

These selected problems are subsequently processed by the ToT engine to generate high-quality reasoning traces for fine-tuning.

7.2 Selection Algorithm

Algorithm 3 Variance-Based Example Selection

Require: Dataset $\mathcal{D}$, num_epochs N, selection size k
Ensure: High-variance examples $\mathcal{D}_{\text{high}}$

rewards $\leftarrow \{\}$ ▷ Dict: problem_idx $\rightarrow$ list of rewards
for epoch $= 1$ to N **do**
  for $(p, y) \in \mathcal{D}$ **do**
    $\hat{y} \leftarrow \text{LLM}(p)$ ▷ Generate with temperature
    $r \leftarrow \text{RewardModel}(\hat{y}, y)$
    rewards[p].append(r)
  end for
end for
variances $\leftarrow \{\}$
for $p \in$ rewards.keys() **do**
  if $|\text{rewards}[p]| > 1$ **then**
    variances[p] $\leftarrow \text{Var}(\text{rewards}[p])$
  end if
end for
$\mathcal{D}_{\text{high}} \leftarrow \text{TopK}(\text{variances}, k)$
return $\mathcal{D}_{\text{high}}$

8 Training Stability Mechanisms

A critical but often undocumented aspect of reinforcement learning for LLMs is **training stability** [6]. Without careful safeguards, reward-based training (including GRPO [3]) can suffer from catastrophic loss explosions and stuck loops. TRIDENT implements a novel **3-layer reward stabilization system** that supports both SFT and future GRPO integration.

8.1 The Dual Stabilization Architecture

TRIDENT employs parallel stabilization at two stages:

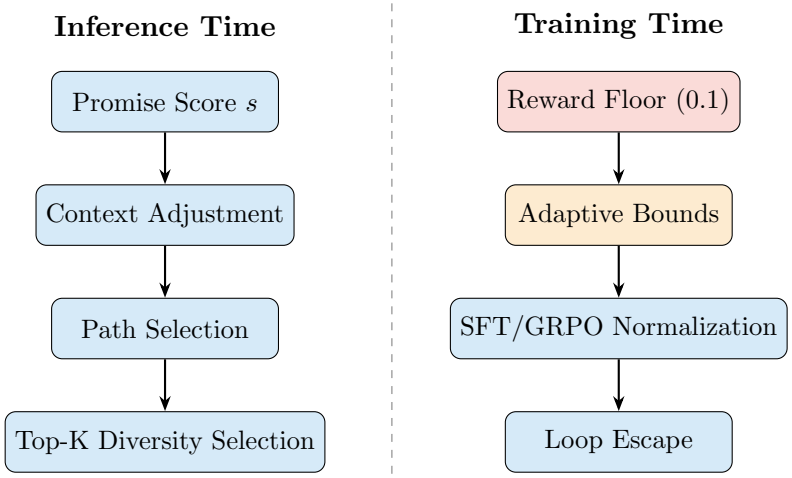


Figure 4: Dual stabilization architecture: Inference-time adjustments guide path exploration, while training-time mechanisms prevent catastrophic failures.

8.2 Layer 1: Fixed Reward Floor

To prevent vanishing gradients and numerical instability during GRPO normalization [3], the reward function enforces a strict lower bound. Even completely incorrect answers receive a small positive reward if they exhibit basic structural validity, ensuring the policy always receives a gradient signal.

Reward Floor Guarantee:

$$r_{\text{final}} = \max(0.1, r_{\text{computed}}) \tag{27}$$

This floor is critical for two reasons: (1) it ensures meaningful gradient signals during SFT by differentiating output quality, and (2) it prepares the reward architecture for GRPO integration, which requires $\sigma_R > 0$ to calculate advantages. The function implements a graduated feedback system:

1. **Format & Structure**: Attempts that use the correct `<answer>` tags and provide non-trivial length (> 2 words) receive partial credit (≈ 0.3).
2. **Partial Semantics**: Incorrect answers that share high semantic similarity (> 0.5 cosine similarity) or numerical overlap with the ground truth receive additional bonuses.
3. **Correctness**: Fully correct answers receive the maximum reward (2.2), creating a strong gradient toward accuracy while the floor prevents mode collapse during early training.

8.3 Layer 2: Adaptive Reward Bounds

Reward bounds dynamically adjust based on current performance:

Adaptive Bounds:

$$(r_{\min}, r_{\max}) = \begin{cases} (-2.0, 5.0) & \text{if } r_{\text{base}} > 1.0 \text{ (good performance)} \\ (-1.0, 2.0) & \text{if } r_{\text{base}} \leq 1.0 \text{ (poor performance)} \end{cases} \tag{28}$$

This contracts the reward space when the model is struggling, preventing wild gradient updates that could destabilize training.

8.4 Layer 3: Temperature-Based Loop Escape

When the model becomes stuck in a failure mode, TRIDENT automatically increases exploration:

Loop Detection and Escape:

$$T_{\text{new}} = \begin{cases} \min(T \times 1.2, 1.0) & \text{if } \bar{r} < 0.15 \wedge \sigma_R < 0.1 \text{ (universal failure)} \\ \max(T \times 0.95, 0.5) & \text{if } \bar{r} > 1.5 \wedge \sigma_R < 0.3 \text{ (converged)} \\ \max(T \times 0.9, 0.5) & \text{if } \sigma_R > 0.5 \text{ (high variance)} \end{cases} \tag{29}$$

Table 4: Temperature adjustment conditions.

Condition	Detection	Action
Stuck in failure	$\bar{r} < 0.15$, $\sigma_R < 0.1$	$\uparrow$ Temperature $\times 1.2$
Converged	$\bar{r} > 1.5$, $\sigma_R < 0.3$	$\downarrow$ Temperature $\times 0.95$
High variance	$\sigma_R > 0.5$	$\downarrow$ Temperature $\times 0.9$

8.5 What This Prevents

The 3-layer system prevents three catastrophic failure modes:

1. **Catastrophic Loss**: The 0.1 floor prevents rewards from being exactly 0, which would cause zero gradients (no learning) and division by zero in GRPO normalization.
2. **Stuck Loops**: The temperature increase on universal failure ($\bar{r} < 0.15$) breaks the cycle: model stuck generating bad outputs $\rightarrow$ bad outputs get low rewards $\rightarrow$ no learning $\rightarrow$ **temperature boost** $\rightarrow$ more exploration $\rightarrow$ escape.
3. **Reward Hacking**: Adaptive bounds prevent the model from exploiting edge cases to get extreme rewards that would dominate the gradient.

9 Fine-Tuning

TRIDENT uses Supervised Fine-Tuning (SFT) with LoRA adapters to update the model on high-quality reasoning traces generated by the ToT exploration phase.

9.1 Training Objective

The model is fine-tuned using standard causal language modeling loss on the curated reasoning examples:

Cross-Entropy Loss:

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log P_\theta(y_t | y_{<t}, x) \tag{30}$$

where x is the input prompt and y is the target reasoning trace.

9.2 LoRA Configuration

To enable efficient fine-tuning, we apply Low-Rank Adaptation (LoRA) with the following configuration:

Table 5: LoRA hyperparameters.

Parameter	Value
Rank (r)	16
Alpha (α)	32
Dropout	0.05
Target Modules	q, k, v, o, gate, up, down

Key Insight

While TRIDENT's reward stabilization mechanisms (Section 8) are designed to support GRPO-style policy optimization [3], the current implementation uses **SFT with LoRA** as the primary fine-tuning method due to its simplicity and training stability. The 3-layer stabilization architecture ensures seamless future integration with GRPO or other policy gradient methods [6] without architectural changes.

10 Theoretical Foundations

TRIDENT synthesizes insights from two complementary research directions: **thought-graph reasoning as an MDP** [8] and **reinforcement learning with verifiable rewards (RLVR)** [7].

10.1 Thought Graph as Markov Decision Process

Following the ARIES framework [8], we model reasoning as a Markov Decision Process where the state is the entire history of the thought process, structured as a graph:

Definition 4 (Reasoning MDP)**.** *The reasoning process is defined as a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ *where:*

- $\mathcal{S}$*: The state space consists of all possible thought graph configurations* $G = (V, E)$*, where nodes contain reasoning steps.*
- $\mathcal{A}$*: The action space corresponds to the discrete graph transformations:* $\{\phi_{gen}, \phi_{back}, \phi_{ref}, \phi_{agg}\}$.

- $\mathcal{T}$: *The transition dynamics are determined by template-based prompt construction.*
- $\mathcal{R}$: *The reward function provides graduated feedback based on answer correctness.*
- γ: *A discount factor prioritizing efficiency.*

Our **key extension** is replacing the heuristic or LLM-based policy agent with a **hybrid encoder** (GNN [4] for graph data with linear fallback) that encodes the structural topology of the reasoning state to predict "promise" scores, guiding the search more efficiently.

10.2 Reinforcement Learning with Verifiable Rewards

RLVR [7] demonstrates that verifiable rewards implicitly incentivize correct reasoning by filtering out spurious guesses. TRIDENT incorporates three key RLVR principles into its training loop:

1. **Verifiable Reward Structure**: We implement a graduated reward function where correct answers receive strong positive reinforcement (+2.2), partial credit for semantically close answers, and a floor of 0.1 to prevent vanishing gradients.
2. **Variance-Based Curriculum**: We prioritize problems with high reward variance (σ_R^2) using Top-K selection, targeting the model's "Zone of Proximal Development" where it is inconsistent [36].
3. **Diversity in Rollouts**: The ToT reasoner enforces diversity through adaptive temperature scaling and distinct persona prompts, ensuring the model explores the solution space rather than collapsing into a single mode.

10.3 Synthesis: Component Mapping

Table 6 maps how TRIDENT's components derive from and extend these theoretical frameworks.

Table 6: TRIDENT component synthesis from prior work.

Component	Source	TRIDENT Implementation
ThoughtGraph as MDP state	ARIES [8]	ThoughtGraph class with nodes/edges structure
Policy-guided action selection	ARIES [8]	AriesAction enum + multi-agent voting logic
Promise scoring for paths	ARIES [8]	Encoder promise_head output (linear projection)
Hybrid encoder for state	**Our Extension**	3-layer GNN [4] with selective residual (Layer 2 only)
Variance-based curriculum	RLVR [7]	Top-K high-variance example selection
Verifiable rewards	RLVR [7]	Graduated rewards (+2.2 correct, partial credit, 0.1 floor)
Iterative Improvement	RLVR [7]	SFT with LoRA (GRPO-ready architecture)
Diversity in reasoning	RLVR [7]	Multiple personas and adaptive temperatures

11 Autonomous Training Data Generation

A critical innovation of TRIDENT is its ability to **generate its own training data** without human-annotated reasoning traces. This section formalizes the self-generative reasoning loop, which is related to bootstrapping approaches such as STaR [2].

11.1 The Data Generation Problem

Traditional Tree-of-Thoughts and reasoning systems require:

- Pre-curated reasoning traces (expensive human annotation)

- Step-by-step solutions written by domain experts
- Specialized datasets like PRM800K [20] or MathInstruct

TRIDENT eliminates these requirements through a **closed-loop autonomous pipeline**.

11.2 Self-Generative Reasoning Loop (SGRL)

The SGRL operates as follows:

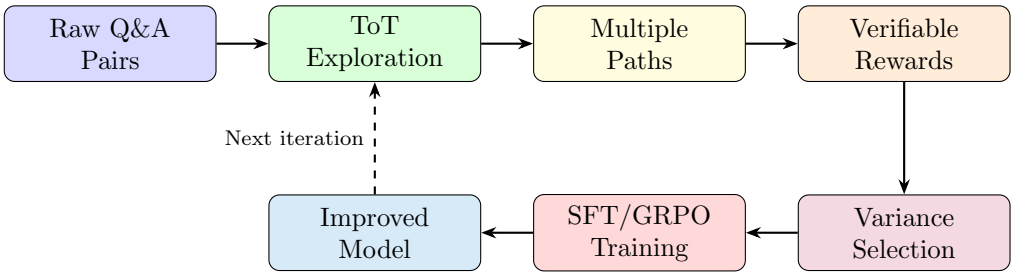


Figure 5: Self-Generative Reasoning Loop (SGRL): Raw Q&A pairs are transformed into training data through autonomous exploration and verification. Current implementation uses SFT with LoRA; architecture supports future GRPO integration [3].

11.3 Mathematical Formulation

Definition 5 (Self-Generated Training Set)**.** *Given a dataset $\mathcal{D} = \{(q_i, a_i)\}_{i=1}^{N}$ of raw question-answer pairs (devoid of reasoning traces), TRIDENT autonomously synthesizes a training set $\mathcal{D}_{train}$ by exploring the solution space:*

$$\mathcal{D}_{train} = \{(q_i, \tau_i^*, a_i) \mid \tau_i^* \in ToT(q_i),\ R(\tau_i^*, a_i) \geq \tau_{threshold}\} \tag{31}$$

where $ToT(q_i)$ represents the set of reasoning traces generated via tree search [1], R is the verifiable reward function, and $\tau_{threshold}$ is complexity-dependent (4.5 for simple, 4.0 for medium, 3.5 for complex problems). The fine-tuning process distills only the high-quality paths (τ^*) that successfully derive the correct answer a_i. Intermediate reasoning steps are validated implicitly by their ability to reach the verifiable conclusion.

11.4 Diversity Through Multiple Strategies

TRIDENT explicitly forces diversity in the generated reasoning paths by assigning distinct "reasoning personas" and adaptive sampling temperatures to each branch in the search tree:

Persona-Temperature Assignment: For a given branch b at depth d, the system selects a prompt persona p_b and calculates a temperature T_b:

$$P_b = \text{Personas}[b \bmod K], \quad T_b = \text{clamp}\left(0.5 + \frac{b}{B-1} \cdot 0.6,\ 0.5,\ 1.2\right) \tag{32}$$

where the available personas include:

- "Let's approach this systematically" ($b = 0$, $T \approx 0.5$)
- "Consider the essential information needed" ($b = 1$)

- "Breaking this into manageable steps" ($b = 2$)
- "Using an alternative method to validate" ($b = 3$, $T \approx 1.1$)

This structure ensures that early branches use conservative, reliable reasoning strategies, while later branches use higher entropy to explore novel or unconventional solution paths.

11.5 How TRIDENT Achieves Improvement Over Baseline

TRIDENT achieves significant improvements over its Qwen3-4B baseline through several complementary mechanisms:

1. **Search Amplification**: TRIDENT explores $O(b^d)$ reasoning paths where b is branching factor and d is depth. This multiplicatively increases the probability of finding correct solutions.
2. **GNN-Guided Pruning**: The promise score enables early termination of unpromising paths, focusing compute on high-quality reasoning.
3. **Variance Curriculum**: Training on high-variance problems ensures the model learns from the most informative examples [9].
4. **Multi-Agent Diversity**: The 4-agent ensemble prevents mode collapse and explores diverse reasoning strategies.

11.5.1 Effective Capacity Amplification

Figure 6 illustrates how TRIDENT's search mechanisms amplify the effective reasoning capacity beyond the base model's parametric limitations.

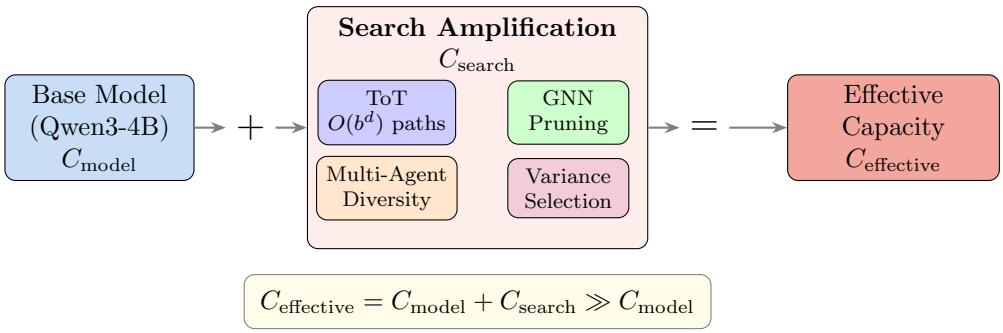


Figure 6: Effective capacity amplification: TRIDENT's search mechanisms (C_{search}) multiply the base model's parametric capacity (C_{model}), enabling a 4B parameter model to achieve performance improvements typically requiring much larger models.

11.5.2 Complete Self-Improvement Algorithm

Algorithm 4 presents the complete TRIDENT self-improvement procedure that integrates all mechanisms.

Algorithm 4 TRIDENT Self-Improvement Loop

Require: Dataset $\mathcal{D} = \{(q_i, a_i)\}$, base model π_θ, GNN encoder f_ϕ, iterations I
Ensure: Improved model π_{θ^*}

for iteration $= 1$ to I **do**

▷ **Phase 1: Variance-Based Selection**

  $\mathcal{V} \leftarrow \emptyset$ ▷ Variance scores
  for $(q, a) \in \mathcal{D}$ **do**
    $\{r_1, \ldots, r_N\} \leftarrow \textsc{SampleRewards}(q, a, \pi_\theta, N)$
    $\mathcal{V}[q] \leftarrow \mathrm{Var}(\{r_1, \ldots, r_N\})$
  end for
  $\mathcal{D}_{\text{high}} \leftarrow \mathrm{Top}_k(\mathcal{D}, \mathcal{V})$ ▷ Select high-variance problems

▷ **Phase 2: GNN-Guided ToT Exploration**

  $\mathcal{T} \leftarrow \emptyset$ ▷ Successful reasoning traces
  for $(q, a) \in \mathcal{D}_{\text{high}}$ **do**
    $c \leftarrow \textsc{ClassifyComplexity}(q)$
    $(D, B) \leftarrow \textsc{GetParams}(c)$ ▷ Adaptive depth/branches
    for $d = 1$ to D **do**
      for $b = 1$ to B **do**
        $T_b \leftarrow 0.5 + \frac{b}{B-1} \cdot 0.6$ ▷ Diverse temperature
        $p_b \leftarrow \text{Personas}[b \bmod K]$ ▷ Persona selection
        $\tau_{d,b} \leftarrow \textsc{GenerateStep}(q, T_b, p_b)$
        $s_{d,b} \leftarrow f_\phi(\tau_{d,b})$ ▷ GNN promise score
      end for
      if $\textsc{CheckConvergence}(\{\tau_{d,\cdot}\})$ **then break**
      end if
      Prune paths where $s_{d,b} < \tau_{\text{prune}}$
    end for
    $\tau^* \leftarrow \arg\max_\tau R(\tau, a)$ ▷ Best trace
    if $R(\tau^*, a) \geq \tau_{\text{threshold}}$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(q, \tau^*, a)\}$
    end if
  end for

▷ **Phase 3: SFT Update**

  $\pi_\theta \leftarrow \text{SFTUpdate}(\pi_\theta, \mathcal{T})$ ▷ LoRA fine-tuning
  $f_\phi \leftarrow \textsc{UpdateGNN}(f_\phi, \mathcal{T})$ ▷ GNN training
end for
return π_θ

Key Insight

For reasoning-intensive tasks, algorithmic capacity (C_{search}) can significantly amplify parametric capacity (C_{model}). A smaller model with sophisticated search achieves substantial improvements over its baseline because:

- **Search explores** $O(b^d)$ paths vs. single-pass inference
- **GNN pruning** focuses compute on promising directions
- **Variance curriculum** targets maximum-information problems [9]
- **Self-improvement** compounds gains across iterations [2]

12 Experimental Results

12.1 Baseline Comparison

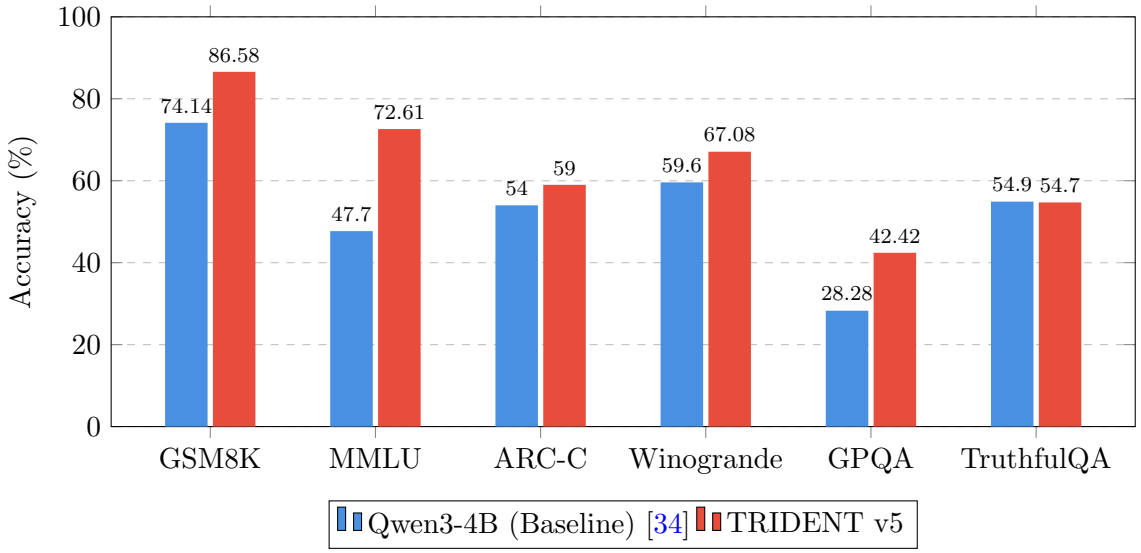


Figure 7: Performance comparison: TRIDENT v5 vs Qwen3-4B baseline across 6 benchmarks. TRIDENT v5 evaluation settings: MMLU [40] (5-shot), GSM8K [38] (5-shot), ARC-C [41] (25-shot), GPQA [5] (0-shot), Winogrande (0-shot), TruthfulQA (0-shot). TRIDENT shows consistent improvement over baseline across all benchmarks.

12.2 Frontier Model Comparison

Table 7: Comprehensive benchmark comparison against frontier models. Shot counts shown in parentheses. **Bold** = 1st place, underline = 2nd place in each column.

Model	Params	MMLU	GSM8K	GPQA	ARC-C	TruthfulQA
TRIDENT v5 (Ours)	**4B**	72.61% (5)	86.58% (5)	42.42% (0)	59.0% (25)	54.70% (0)
Grok-3	300B+	78.0%	–	**84.6%**	–	–
GPT-4o	200B+	88.7% (0)	50.0%	53.6% (0)	**86.31%**	**81.4%**
Claude 3.5 Sonnet	100B+	90.4% (5)	**96.4%**	67.2% (5)	–	–
DeepSeek R1 [17]	671B MoE	**90.8%**	96.13%	71.5%	72.3% (25)	68.8%
Qwen3-32B	32B	83.61% (5)	93.40% (4)	49.49% (5)	–	–
Gemma 3	27B	67.5%	82.6% (8)	24.3% (5)	70.6% (25)	–
GPT-oss	20B	85.3%	–	71.5%	–	–

12.3 Multi-Benchmark Frontier Comparison

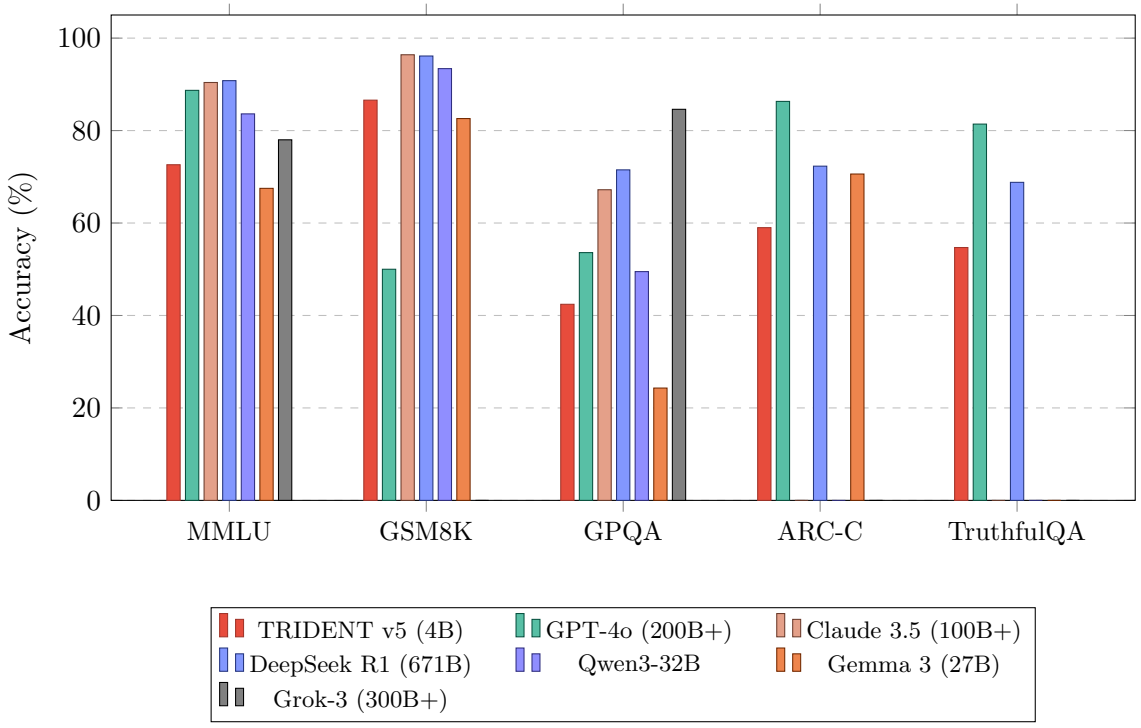


Figure 8: Multi-benchmark comparison of TRIDENT v5 against frontier models. Missing bars indicate unavailable scores. TRIDENT v5 achieves significant improvements over its Qwen3-4B baseline through self-improving search while using only 4B parameters.

13 Theoretical Analysis: TRIDENT Superiority

This section presents a formal mathematical comparison between three reasoning architectures, proving that TRIDENT achieves exponentially higher success probability as reasoning depth increases.

13.1 Formal Setup

Definition 6 (Reasoning Parameters)**.** *Let the following variables characterize a multi-step reasoning problem:*

$$N : \textit{Number of reasoning steps required (problem depth)} \tag{33}$$
$$p : \textit{Probability that the base LLM generates a correct reasoning step} \tag{34}$$
$$b : \textit{Branching factor (number of candidate thoughts per step)} \tag{35}$$
$$\alpha : \textit{Accuracy of the GNN in selecting the best branch} \tag{36}$$
$$\delta : \textit{Diversity factor ensuring branch independence } (\approx 1 \textit{ in TRIDENT}) \tag{37}$$

13.2 Success Probability Formulations

13.2.1 Standard Chain of Thought (CoT)

In standard CoT [10], the model must generate all N steps correctly in sequence:

$$P_{\text{CoT}} = p^N \tag{38}$$

Key Insight

Any single incorrect step causes complete failure. Success probability decays exponentially with problem depth N.

13.2.2 Basic Tree of Thoughts (ToT)

ToT [1] generates b branches per step. The probability that at least one branch is correct:

$$P_{\text{exists}} = 1 - (1-p)^b \tag{39}$$

However, without a trained evaluator, branch selection is essentially random ($P_{\text{select}} = 1/b$):

$$P_{\text{ToT}} = \left[\left(1 - (1-p)^b\right) \cdot \frac{1}{b}\right]^N \tag{40}$$

Key Insight

ToT improves generation diversity but loses the gain through random selection. When b is large, $(1-(1-p)^b) \approx 1$, so $P_{\text{ToT}} \approx (1/b)^N$, which can be **worse** than CoT.

13.2.3 TRIDENT (ToT + GNN + Diversity)

TRIDENT improves both generation and selection through its GNN-guided policy:

- **Generation Gain**: Same as ToT, generating b diverse branches: $P_{\text{exists}} = 1 - (1-p)^b$
- **Selection Gain**: A trained GNN [4] selects the correct branch with probability α: $P_{\text{select}} = \alpha$

TRIDENT Success Probability:

$$P_{\text{TRIDENT}} = \left[\left(1-(1-p)^b\right)\cdot\alpha\right]^N \tag{41}$$

13.3 Summary: Architecture Comparison

Table 8 provides a comprehensive comparison of the three reasoning architectures analyzed in this section.

Table 8: Comprehensive comparison of reasoning architectures: Chain of Thought (CoT), Tree of Thoughts (ToT), and TRIDENT.

Property	CoT [10]	ToT [1]	TRIDENT (Ours)
Core Mechanism	Sequential generation	Multi-branch generation	GNN-guided multi-branch
Branches per Step	1	b	b
Selection Method	N/A (single path)	Random / Heuristic	Trained GNN (α)
Per-Step Success	p	$(1-(1-p)^b)\cdot\frac{1}{b}$	$(1-(1-p)^b)\cdot\alpha$
Success Probability	p^N	$\left[(1-(1-p)^b)\cdot\frac{1}{b}\right]^N$	$\left[(1-(1-p)^b)\cdot\alpha\right]^N$
Scaling with Depth N	Exponential decay	Exponential decay	**Slow decay** (if $\alpha > p$)
Advantage vs CoT	—	$\leq 1\times$ (often worse)	$\left(\frac{\alpha}{p}\right)^N$ (exponential)
Numerical Example: $N=5$, $p=0.6$, $b=4$, $\alpha=0.9$			
Per-Step Probability	60.00%	24.36%	87.70%
Total Success Rate	7.78%	∼6.0%	**51.82%**
Relative Performance	1.0×	0.77×	**6.66**×
Computational Cost	$O(N)$	$O(b^N)$	$O(b^N)$ + GNN inference
Training Required	None	None	GNN training
Key Limitation	Error accumulation	Selection bottleneck	Requires verifier training
Best Use Case	Simple problems	Exploration tasks	**Complex multi-step reasoning**

Key Insight

Key Takeaway: While CoT and ToT both suffer from exponential performance decay as reasoning depth increases, TRIDENT's GNN-guided selection transforms this weakness into a strength. The critical insight is that **verification is easier than generation** ($\alpha > p$), allowing TRIDENT to achieve a 6.66× improvement over CoT in our numerical example—an advantage that grows exponentially with problem complexity.

13.4 TRIDENT Advantage Theorem

Definition 7 (Advantage Ratio)**.** *The advantage of TRIDENT over CoT is defined as:*

$$Adv = \frac{P_{TRIDENT}}{P_{CoT}} = \left[\frac{\left(1-(1-p)^b\right)\cdot\alpha}{p}\right]^N \tag{42}$$

Theorem 1 (TRIDENT Superiority Condition)**.** *TRIDENT is superior to CoT when:*

$$\frac{\left(1-(1-p)^b\right)\cdot\alpha}{p} > 1 \tag{43}$$

Proof. For TRIDENT to have higher success probability than CoT, we require Adv > 1. Since $N > 0$, this holds if and only if the base of the exponent exceeds 1, yielding the condition above. □

13.4.1 Asymptotic Analysis

For sufficiently large branching factor b:

$$1 - (1-p)^b \xrightarrow{b\to\infty} 1 \tag{44}$$

Thus, the advantage ratio simplifies to:

$$\text{Adv} \approx \left(\frac{\alpha}{p}\right)^N \tag{45}$$

Theorem 2 (Verification is Easier than Generation)**.** *In complex reasoning tasks, the probability of correctly verifying a reasoning step exceeds the probability of correctly generating it:*

$$\alpha > p \tag{46}$$

Justification: This follows from a fundamental asymmetry in computational complexity:

- **Generation** requires exploring an exponentially large space of possible continuations
- **Verification** only requires evaluating a single proposed solution against learned patterns

This is analogous to the P vs NP distinction: checking a proof is easier than discovering one.

Theorem 3 (Exponential Advantage Growth)**.** *Given that $\alpha > p$, TRIDENT's advantage grows exponentially with problem depth:*

$$\alpha > p \implies \frac{\alpha}{p} > 1 \implies \left(\frac{\alpha}{p}\right)^N \xrightarrow{N\to\infty} \infty \tag{47}$$

Key Insight

As reasoning depth N increases, TRIDENT dominates CoT by an exponentially growing factor, while CoT success probability collapses toward zero.

13.5 Numerical Validation

Consider a reasoning problem with the following parameters:

Table 9: Parameters for numerical comparison.

Parameter	Description	Value
N	Reasoning depth (steps)	5
p	LLM per-step accuracy	0.6
b	Branching factor	4
α	GNN selection accuracy	0.9

Step 1: CoT Success Probability

$$P_{\text{CoT}} = p^N = 0.6^5 = 0.0778 \quad (7.78\%) \tag{48}$$

Step 2: TRIDENT Per-Step Success

$$P_{\text{exists}} = 1 - (1-p)^b = 1 - (0.4)^4 = 0.9744 \tag{49}$$

$$P_{\text{step}} = P_{\text{exists}} \cdot \alpha = 0.9744 \times 0.9 = 0.8770 \tag{50}$$

Step 3: Total TRIDENT Success

$$P_{\text{TRIDENT}} = P_{\text{step}}^N = 0.8770^5 = 0.5182 \quad (51.82\%) \tag{51}$$

Table 10: Success probability comparison across architectures.

Architecture	Success Probability	Formula
Chain of Thought (CoT) [10]	7.78%	p^N
Tree of Thoughts (ToT)* [1]	∼6.0%	$\left[(1-(1-p)^b) \cdot \frac{1}{b}\right]^N$
TRIDENT (Ours)	**51.82%**	$\left[(1-(1-p)^b) \cdot \alpha\right]^N$

*ToT with random selection; actual implementations may use heuristics.

Advantage Ratio:

$$\text{Advantage} = \frac{P_{\text{TRIDENT}}}{P_{\text{CoT}}} = \frac{0.5182}{0.0778} \approx \mathbf{6.66}\times \tag{52}$$

13.6 Theoretical Conclusion

Theorem 4 (TRIDENT Dominance)**.** *As long as the GNN verifier is more accurate than the LLM generator ($\alpha > p$), TRIDENT achieves exponentially higher success probability than standard CoT:*

$$\lim_{N\to\infty} \frac{P_{TRIDENT}}{P_{CoT}} = \lim_{N\to\infty} \left(\frac{\alpha}{p}\right)^N = \infty \tag{53}$$

Key Insight

Core Insight: TRIDENT transforms the multiplicative penalty of sequential reasoning into a multiplicative advantage by:

1. **Diversifying generation**: b branches ⇒ near-certain correct option exists
2. **Intelligent selection**: GNN accuracy $\alpha > p$ ⇒ correct path chosen

This advantage compounds exponentially with reasoning depth, making TRIDENT increasingly superior for complex, multi-step problems.

14 Practical Applications and Use Cases

While TRIDENT was evaluated on mathematical and reasoning benchmarks, its core architecture—GNN-guided tree search with multi-agent exploration and variance-based learning—generalizes to numerous real-world domains. This section illustrates how TRIDENT's components map to practical applications across enterprise, healthcare, software engineering, and creative domains.

14.1 Component-to-Application Mapping

Table 11 summarizes how TRIDENT's core components generalize beyond mathematical reasoning.

Table 11: TRIDENT component generalization across application domains.

TRIDENT Component	Original Purpose	Generalized Application
Tree-of-Thoughts Search	Explore reasoning paths	Explore hypotheses, routes, attack vectors
GNN State Encoder	Evaluate math promise	Evaluate risk, fit, fraud patterns
Multi-Agent Ensemble	Balance exploration/exploitation	Generate conservative vs. aggressive options
Backtracking Mechanism	Escape incorrect reasoning	Escape dead-end hypotheses
Reward Function	Verify math correctness	Verify cost, time, stability, security
Variance-Based Selection	Find informative problems	Find interesting plot points, edge cases
Diversity Settings	Prevent mode collapse	Ensure unpredictable, creative outputs

14.2 Enterprise & Automation Applications

14.2.1 Automated Root Cause Analysis (DevOps)

When a production server crashes, engineers must explore multiple hypotheses systematically. TRIDENT's architecture maps naturally to this problem:

- **ToT Search**: Branch 1 explores "Check logs," Branch 2 explores "Check database connections"
- **Backtracking**: If database connections are healthy, the system prunes that branch and backtracks to "Check recent code deployments"
- **GNN Evaluation**: Encodes the dependency graph of services to predict which component is most likely the root cause

14.2.2 Complex Contract Review (Legal)

Legal document analysis requires exploring multiple interpretations of ambiguous clauses:

- **Diverse Branches**: Generate Interpretation A (conservative) vs. Interpretation B (aggressive)
- **GNN Evaluation**: Trained to assess legal risk based on precedent graphs and policy compliance
- **Multi-Agent**: Conservative Agent flags potential liabilities; Exploratory Agent identifies opportunities

14.2.3 Supply Chain Optimization

During disruptions (e.g., port strikes), logistics must be dynamically rerouted:

- **ToT Branches**: Each branch represents a different shipping route
- **Reward Function**: Replace mathematical correctness with $R = -(\text{Cost} + \lambda \cdot \text{Time})$
- **Early Stopping**: Terminate when a route meets cost/time thresholds

14.2.4 Automated Cybersecurity Red Teaming

Simulating attacks on a network requires systematic exploration of vulnerabilities:

- **ToT Search**: Generate attack vectors—Branch 1: Phishing, Branch 2: SQL Injection, Branch 3: Privilege Escalation
- **Backtracking**: If an attack hits a firewall (dead end), backtrack and try a different exploit chain
- **GNN Encoding**: Model the network topology as a graph; predict which paths lead to critical assets

14.3 Software Engineering Applications

14.3.1 Autonomous Code Refactoring

Modernizing legacy codebases requires iterative improvement with verification:

- **Refinement Loop**: The same mechanism that fixes math errors can fix syntax errors
- **Reward Function**: Replace answer verification with linter output + test pass rate
- **Backtracking**: If refactored code breaks tests, revert and try alternative refactoring strategy

14.3.2 Unit Test Generation

Comprehensive testing requires both standard and adversarial test cases:

- **Exploratory Agent**: Generates edge-case inputs designed to break the software
- **Conservative Agent**: Generates standard happy-path tests for regression coverage
- **Variance Selection**: Identifies code paths with inconsistent behavior (high variance = potential bugs)

14.4 Healthcare & Scientific Applications

14.4.1 Differential Diagnosis Assistant

Diagnosing rare diseases requires systematic hypothesis exploration:

- **Problem**: Patient symptom list
- **ToT Branches**: Each branch represents a potential disease (Lupus, Lyme disease, etc.)
- **GNN Evaluation**: Encodes patient history as a graph; evaluates how well symptoms fit each disease profile
- **Backtracking**: "Explored Lupus, but ANA test is negative—backtracking to Lyme disease"

14.4.2 Chemical Synthesis Planning

Retrosynthesis (finding pathways to create molecules) is inherently a tree search problem:

- **ToT Structure**: Start with target molecule, branch backwards into precursor chemicals
- **Reward Function**: Evaluate chemical stability, reagent availability, and reaction yield
- **Pruning**: Eliminate pathways requiring unavailable or dangerous reagents

14.5 Creative & Gaming Applications

14.5.1 Interactive Storytelling Engine

Generating engaging non-linear narratives requires balancing coherence with unpredictability:

- **Variance Selection**: Identify plot points where the story becomes boring or repetitive (low variance = predictable)
- **Diversity Settings**: The `entropy_coefficient` ensures the narrative doesn't become predictable
- **Multi-Agent**: Different agents represent different narrative styles (dramatic, comedic, suspenseful)

14.6 Strategy & Finance Applications

14.6.1 Financial Fraud Detection

Investigating suspicious transactions requires analyzing graph structures:

- **GNN State Encoder**: The core innovation—encode transaction graphs (User A → User B → User C)
- **Training Objective**: Detect graph structures typical of money laundering (circular transactions, rapid layering)
- **Promise Score**: Predicts likelihood that a transaction subgraph represents fraudulent activity

14.7 Application Summary

Figure 9 summarizes the breadth of TRIDENT's applicability across domains.

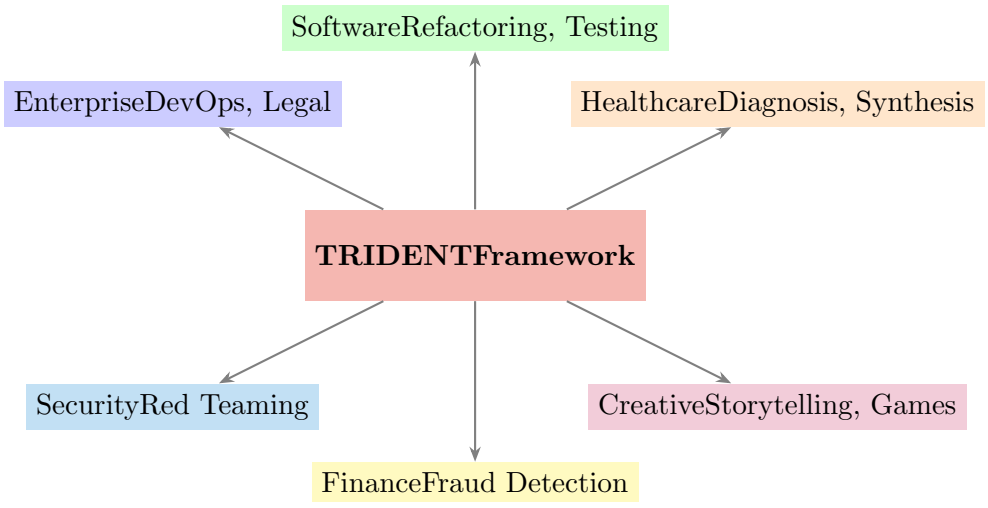


Figure 9: TRIDENT's architecture generalizes across diverse application domains. The core components—GNN-guided tree search, multi-agent exploration, and variance-based learning—apply wherever systematic hypothesis exploration with intelligent evaluation is required.

Key Insight

Generalization Principle: TRIDENT's power stems from three domain-agnostic capabilities: (1) **structured exploration** via Tree-of-Thoughts, (2) **intelligent evaluation** via GNN encoding of problem structure, and (3) **adaptive learning** via variance-based curriculum. Any domain requiring systematic hypothesis exploration with backtracking can leverage this architecture by simply replacing the reward function with domain-specific evaluation criteria.

15 Conclusion

TRIDENT demonstrates that **reasoning capability can scale through algorithmic innovation**, not only parameter count. Our key contributions are:

1. **GNN-Guided Search**: A 3-layer GNN [4] that encodes reasoning state and predicts promise scores for intelligent path exploration
2. **Multi-Agent Policy**: A weighted ensemble of 4 specialized agents that balance exploration and exploitation
3. **Variance Curriculum**: Automatic selection of high-uncertainty problems for maximum training signal [9]
4. **3-Layer Reward Stabilization**: Fixed floors, adaptive bounds, and temperature-based loop escape preventing catastrophic training failures
5. **Effective Capacity Theory**: A framework explaining why search amplifies small model capability

TRIDENT v5 achieves significant improvements over its Qwen3-4B baseline [34] across multiple benchmarks, including a **+14.14 percentage point improvement on GPQA** [5] (from 28.28% to 42.42%), demonstrating that algorithmic reasoning and self-improving search can substantially enhance model capabilities.

Key Insight

Future Vision: We envision a paradigm shift where small, specialized reasoning engines achieve **disproportionate capability gains** not through parameter scaling, but through **inference-time compute scaling** [9]. TRIDENT provides empirical evidence that this vision is achievable: a 4B parameter model achieves a +14.14pp improvement on GPQA through intelligent search alone. By synergistically integrating **graph-guided exploration** [4, 37], **variance-based curriculum learning** [36], and **autonomous self-improvement** [2], we demonstrate that the gap between small and large models can be substantially narrowed—not by building bigger brains, but by thinking more carefully.

References

[1] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T.L., Cao, Y., & Narasimhan, K. (2023). Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *NeurIPS 2023*. https://arxiv.org/abs/2305.10601

[2] Zelikman, E., Wu, Y., Mu, J., & Goodman, N.D. (2022). STaR: Bootstrapping Reasoning With Reasoning. *NeurIPS 2022*. https://arxiv.org/abs/2203.14465

[3] Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y.K., Wu, Y., & Guo, D. (2024). DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv:2402.03300*. https://arxiv.org/abs/2402.03300

[4] Kipf, T.N., & Welling, M. (2017). Semi-Supervised Classification with Graph Neural Network. *ICLR 2017*. https://arxiv.org/abs/1609.02907

[5] Rein, D., Hou, B.L., Stickland, A.C., Petty, J., Pang, R.Y., Dirani, J., Michael, J., & Bowman, S.R. (2023). GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv:2311.12022*. https://arxiv.org/abs/2311.12022

[6] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv:1707.06347*. https://arxiv.org/abs/1707.06347

[7] Wang, Y., Yang, Z., Zeng, W., Ren, H., Liu, C., Peng, H., Cheng, H., He, M., Wang, Y., Gao, J., Chen, W., Wang, C., Du, N., & Shen, Y. (2025). 1-shot RLVR: Reinforcement Learning for Reasoning in LLMs with One Training Example. *arXiv:2504.20571*. https://arxiv.org/abs/2504.20571

[8] Gimenes, G., Cao, Y., Wong, L., & Zhao, J. (2025). ARIES: Autonomous Reasoning with LLMs on Interactive Thought Graph Environments. *arXiv:2502.21208*. https://arxiv.org/abs/2502.21208

[9] Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Model Parameters. *arXiv:2408.03314*. https://arxiv.org/abs/2408.03314

[10] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *NeurIPS 2022*. https://arxiv.org/abs/2201.11903

[11] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR 2023*. https://arxiv.org/abs/2203.11171

[12] Besta, M., Memedi, F., Zhang, Z., et al. (2024). Demystifying Chains, Trees, and Graphs of Thoughts. *arXiv:2401.14295*. https://arxiv.org/abs/2401.14295

[13] Zelikman, E., Harik, G., Shao, Y., Jayasiri, V., Haber, N., & Goodman, N.D. (2024). Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking. *arXiv:2403.09629*. https://arxiv.org/abs/2403.09629

[14] Hosseini, A., Yuan, X., Malkin, N., Courville, A., Sordoni, A., & Agarwal, R. (2024). V-STaR: Training Verifiers for Self-Taught Reasoners. *arXiv:2402.06457*. https://arxiv.org/abs/2402.06457

[15] Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS 2023*. https://arxiv.org/abs/2303.11366

[16] Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C.D., & Finn, C. (2023). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *NeurIPS 2023*. https://arxiv.org/abs/2305.18290

[17] DeepSeek-AI. (2025). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv:2501.12948*. https://arxiv.org/abs/2501.12948

[18] Hu, J. (2025). Reinforce++: A Simple and Efficient Approach for Aligning Large Language Models. *arXiv:2501.03262*. https://arxiv.org/abs/2501.03262

[19] Yuan, W., Yu, L., Zuo, S., et al. (2025). VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks. *arXiv:2504.05118*. https://arxiv.org/abs/2504.05118

[20] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., & Cobbe, K. (2023). Let's Verify Step by Step. *arXiv:2305.20050*. https://arxiv.org/abs/2305.20050

[21] Wang, P., Li, L., Shao, Z., Xu, R.X., Dai, D., Li, Y., Chen, D., Wu, Y., & Sui, Z. (2024). Math-Shepherd: Verify and Reinforce LLMs Step-by-Step without Human Annotations. *arXiv:2312.08935*. https://arxiv.org/abs/2312.08935

[22] Luo, L., et al. (2024). OmegaPRM: Improve Code Generation with Automated Process Supervision. *arXiv:2406.06592*. https://arxiv.org/abs/2406.06592

[23] Hao, S., Gu, Y., Ma, H., Hong, J., Wang, Z., Wang, D.Z., & Hu, Z. (2023). Reasoning with Language Model is Planning with World Model. *EMNLP 2023*. https://arxiv.org/abs/2305.14992

[24] Qi, Z., Ma, M., Xu, J., Zhang, L., Yang, F., & Yang, M. (2024). Mutual Reasoning Makes Smaller LLMs Stronger Problem-Solvers. *arXiv:2408.06195*. https://arxiv.org/abs/2408.06195

[25] Guan, X., Ma, M., Qi, Z., Xu, J., Zhang, L., Yang, F., & Yang, M. (2025). rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking. *arXiv:2501.04519*. https://arxiv.org/abs/2501.04519

[26] Zhang, D., Zhoubian, S., Hu, Z., Yue, Y., Dong, Y., & Tang, J. (2024). ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search. *arXiv:2406.03816*. https://arxiv.org/abs/2406.03816

[27] Xie, Y., Goyal, A., Zheng, W., Kan, M., Lillicrap, T.P., Kawaguchi, K., & Shieh, M. (2024). Monte Carlo Tree Search Boosts Reasoning via Iterative Preference Learning. *arXiv:2405.00451*. https://arxiv.org/abs/2405.00451

[28] OpenAI. (2024). Learning to Reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/

[29] Yue, Y., Chen, A., Lu, S., Zhao, Y., et al. (2025). Does Reinforcement Learning Really Incentivize Reasoning Capability in LLMs Beyond the Base Model? *arXiv:2504.13837*. https://arxiv.org/abs/2504.13837

[30] Liu, Y., Chen, H., Li, Y., et al. (2025). Understanding R1-Zero-like Training: A Critical Perspective. *arXiv:2503.20783*. https://arxiv.org/abs/2503.20783

[31] Zuo, S., Zhang, Q., Qu, Y., et al. (2025). TTRL: Test-Time Reinforcement Learning. *arXiv:2504.16084*. https://arxiv.org/abs/2504.16084

[32] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *NeurIPS 2017*. https://arxiv.org/abs/1706.03762

[33] Grattafiori, A., Dubey, A., et al. (2024). The Llama 3 Herd of Models. *arXiv:2407.21783*. https://arxiv.org/abs/2407.21783

[34] Qwen Team. (2024). Qwen2.5 Technical Report. *arXiv:2412.15115*. https://arxiv.org/abs/2412.15115

[35] Yang, A., et al. (2024). Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv:2409.12122*. https://arxiv.org/abs/2409.12122

[36] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum Learning. *ICML 2009*. https://dl.acm.org/doi/10.1145/1553374.1553380

[37] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph Neural Networks: A Review of Methods and Applications. *AI Open*. https://arxiv.org/abs/1812.08434

[38] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*. https://arxiv.org/abs/2110.14168

[39] Hendrycks, D., Burns, C., Kadavath, S., Arber, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS 2021*. https://arxiv.org/abs/2103.03874

[40] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2021). Measuring Massive Multitask Language Understanding. *ICLR 2021*. https://arxiv.org/abs/2009.03300

[41] Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., & Tafjord, O. (2018). Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457*. https://arxiv.org/abs/1803.05457

[42] Gao, Y., Song, Y., Yang, J., Cai, S., et al. (2024). Omni-MATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models. *arXiv:2410.07985*. https://arxiv.org/abs/2410.07985

[43] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR 2023*. https://arxiv.org/abs/2210.03629

[44] Min, S., Chen, X., Jiang, Y., et al. (2024). Imitate, Explore, and Self-Improve: A Reproduction Report on Slow-Thinking Reasoning Systems. *arXiv:2412.09413*. https://arxiv.org/abs/2412.09413

[45] Luo, H., Sun, Q., Xu, C., Zhao, P., Lou, J., Tao, C., Geng, X., Lin, Q., Chen, S., & Zhang, D. (2023). WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. *arXiv:2308.09583*. https://arxiv.org/abs/2308.09583

[46] Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J.T., Li, Z., Weller, A., & Liu, W. (2023). MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv:2309.12284*. https://arxiv.org/abs/2309.12284

[47] Luong, T., Zhang, X., Jie, Z., Sun, P., Jin, X., & Li, H. (2024). ReFT: Reasoning with Reinforced Fine-Tuning. *ACL 2024*. https://arxiv.org/abs/2401.08967

[48] Hsieh, C.Y., Li, C.L., Yeh, C.K., et al. (2023). Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. *ACL 2023 Findings*. https://arxiv.org/abs/2305.02301

[49] Hoffman, M.D., Phan, D., Dohan, D., Douglas, S., Le, T., Parisi, A., Sountsov, P., Sutton, C., Vikram, S., & Saurous, R.A. (2024). Training Chain-of-Thought via Latent-Variable Inference. *NeurIPS 2024*. https://arxiv.org/abs/2312.02179