

DARTS: Diversity-Aware Reasoning Tree Search for Test-Time Scaling

Kaifeng Liu^{1,2}, Shijie Wang^{2*}, Yuanxing Liu¹, Yang Liu³,
Xinyi Wang¹, Haozheng Hou¹, Biqing Qi^{2*}, Weinan Zhang^{1*}, Ting Liu¹,

¹Research Center for Social Computing and Interactive Robotics,
Harbin Institute of Technology, Heilongjiang, China

²Shanghai AI Laboratory

³Institute of Automation

Correspondence: kfliu@ir.hit.edu.cn

Abstract

Test-time tree search improves the reasoning capability of large language models under fixed inference budgets. However, existing solution-level MCTS often suffers from **local homogenization**: once a promising solution pattern is found, subsequent search tends to refine nearby variants rather than explore distinct reasoning paths. This wastes computation on redundant candidates and limits solution-space coverage. To address this problem, we propose **Diversity-Aware Reasoning Tree Search (DARTS)**, a Bayesian tree-search framework that treats reasoning-path diversity as an explicit search objective. DARTS separates expansion into local refinement and orthogonal regeneration, and updates action posteriors using a joint reward over task quality and reasoning-path diversity. On LiveCodeBench and Code-Contests, DARTS achieves the best exploration performance and produces more diverse correct solutions. With half the budget, it matches or exceeds the strongest tree-search baseline in four of six settings. DARTS also transfers to mathematical reasoning and molecular optimization, demonstrating its generalizability beyond code generation. Code is available at <https://github.com/Semonlzf/DARTS>.

1 Introduction

Large language models (LLMs) have shown strong performance in complex reasoning tasks such as code generation and mathematical reasoning (Wang et al., 2026; Guo et al., 2025; Hui et al., 2024). Test-time scaling serves as a crucial paradigm to overcome the bottleneck of inherent model capabilities (Liu et al., 2025). Tree search methods, including Monte Carlo Tree Search (Browne et al., 2012; Inoue et al., 2026), effectively enhance model performance on complex tasks by conducting systematic breadth exploration and depth refinement within the candidate solution

*Corresponding authors.

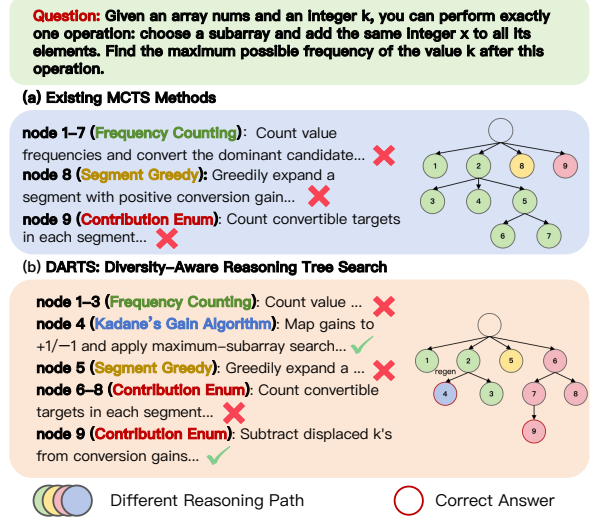


Figure 1: Conceptual illustration of local homogenization in solution-level tree search. Existing MCTS methods tend to expand variants of the same reasoning pattern, whereas DARTS explicitly encourages different reasoning paths.

space. By constructing search trees, these methods balance exploration and exploitation under a limited computational budget to improve the coverage and quality of candidate solutions.

However, existing solution-level tree search methods often suffer from **local homogenization** (Lee et al., 2025; Shypula et al., 2025): candidate solutions gradually drift toward a narrow set of similar reasoning paths. This wastes computation on redundant candidates and limits solution-space coverage under a fixed budget. Figure 1 illustrates this issue with a code-generation example, where the model must choose one subarray operation to maximize the frequency of a target value `k`. Although this problem admits multiple algorithmic strategies, existing tree-search methods tend to repeatedly explore variants of the same reasoning pattern. We provide a quantitative diagnosis of this phenomenon in Appendix A, showing that

parent-child algorithmic agreement increases during search and that diversity loss is associated with weaker exploration success.

These examples reveal two limitations of existing methods. **(1) Refinement-based expansion inherits the parent reasoning structure.** By generating child nodes through local refinements, the search repeatedly explores variants of the same algorithmic idea and may remain locked in a suboptimal solution paradigm. **(2) Quality-only rewards amplify local homogenization.** By rewarding locally successful solutions without considering redundancy, they allocate more search budget to similar reasoning paths and further reduce reasoning-path diversity.

These limitations cannot be addressed by simply generating more candidates. The key challenge is to preserve high-quality solutions while expanding the coverage of distinct reasoning paths. Inspired by Quality-Diversity Optimization (Chatzilygeroudis et al., 2021), we treat reasoning-path diversity as an explicit search objective, enabling tree search to balance the refinement of promising solutions with the exploration of distinct, high-quality alternatives.

Following this principle, we propose **Diversity-Aware Reasoning Tree Search (DARTS)**. DARTS makes solution-level tree search diversity-aware by coupling two mechanisms inside a Bayesian search loop. First, it introduces a diversity-aware expansion space with two actions: local refinement improves the current reasoning path, while orthogonal regeneration diverges to explore alternative solution paradigms. Second, a diversity-augmented evaluation signal combines task quality with reasoning-path diversity measured against previously explored paths. The resulting joint reward updates the Bayesian posteriors of the expansion actions, allowing the search to adaptively decide whether to refine or regenerate based on quality and diversity.

We evaluate DARTS primarily on code generation tasks. In this setting, DARTS aims to improve the quality and diversity of the candidate set explored under a fixed inference budget without altering the final-solution selection protocol. We therefore use Pass@ K and the diversity of functionally correct solutions to evaluate exploration, and report Pass@1 to assess convergence performance. On LiveCodeBench V6 and CodeContests, DARTS achieves stronger exploration-oriented performance while maintaining competitive final-

solution quality and generating the most diverse correct solutions among tree-search baselines. Notably, at a budget of 64, DARTS matches or exceeds the strongest tree-search baseline at a budget of 128 in four of six main code-generation settings. Cross-domain studies on AIME 2024/2025 and molecular optimization further show that the same search principle transfers to mathematical and scientific reasoning. A component-level cost profile confirms that the added extraction, embedding, and Sinkhorn computations remain lightweight relative to generation.

In summary, our primary contributions are as follows:

- We introduce a diversity-aware search principle that explicitly optimizes for both solution correctness and reasoning-path diversity in solution-level tree search.
- We instantiate this principle as DARTS, a Bayesian search framework that combines local refinement, orthogonal regeneration, and a reasoning-path diversity reward for adaptive action selection.
- We evaluate DARTS on code generation, mathematical reasoning, and molecular optimization, demonstrating stronger exploration and cross-domain transfer with modest auxiliary overhead.

2 Related Work

2.1 Test-Time Scaling for LLM Reasoning

Test-time Scaling has become an effective paradigm for improving LLM reasoning without updating model parameters. Early methods such as Chain-of-Thought and Self-Consistency improve reasoning by eliciting or aggregating intermediate reasoning paths (Wei et al., 2022; Wang et al., 2022). Later approaches extend this idea to interactive refinement and structured search, including Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), Tree of Thoughts (Yao et al., 2023), RAP (Hao et al., 2023), Graph of Thoughts (Besta et al., 2024), and LATs (Zhou et al., 2023).

MCTS-based methods provide a principled way to balance exploration and exploitation by expanding candidate reasoning states, estimating their values, and backpropagating feedback. Recent solution-level variants, such as standard MCTS and Adaptive Branching MCTS, further adapt this

framework to LLM inference under limited compute budgets (Browne et al., 2012; Inoue et al., 2026). However, these methods primarily optimize final-answer quality and often overlook the risk of concentrating search around locally similar reasoning paths. DARTS differs by treating reasoning-path diversity as an explicit search objective during test-time search.

2.2 Homogenization and Diversity in Test-Time Reasoning

Balancing homogenization and diversity is critical in test-time reasoning. While prior work broadens reasoning coverage through diverse sampling, prompt variations, and adaptive branching (Jiang et al., 2026; Zhang et al., 2025; Lee et al., 2025; Shypula et al., 2025; Wang et al., 2022; Naik et al., 2023; Fu et al., 2022), these methods typically treat diversity merely as a static sampling procedure, an external evaluation metric, or a training-time signal.

DARTS incorporates diversity directly into the search process. Unlike branching strategies guided only by solution quality, such as ABMCTS, which may spend computation on similar high scoring refinements, DARTS uses reasoning-path diversity as an online feedback signal. Within a unified Bayesian search loop, it combines orthogonal regeneration with a diversity reward and dynamically updates action posteriors according to both solution quality and reasoning path diversity.

3 Problem Formulation

We consider test-time search over complete candidate solutions. Given an input problem x and an inference budget B , a search method incrementally builds a candidate set $\mathcal{H} = \{\tau_1, \dots, \tau_T\}$, where each τ_i is a complete solution object, such as a program for code generation or a molecule for black-box optimization. Unlike token-level decoding, solution-level search evaluates and expands whole candidates. Each candidate receives a task-dependent quality signal $R_{\text{qual}}(\tau_i)$, which is used to guide refinement, expansion, or selection. After the budget is exhausted, the benchmark evaluates either the best candidate in $\mathcal{H}$ or a selected final output $\hat{\tau}$, depending on the task protocol.

From a quality-diversity perspective, effective test-time search should not only find high-quality candidates, but also cover diverse reasoning paths. Let $R_{\text{div}}(\tau, \mathcal{H}_t)$ measure how different candidate

τ is from previously explored candidates in the current archive $\mathcal{H}_t$. We formulate the search utility as

$$U(\tau; \mathcal{H}_t) = \lambda R_{\text{qual}}(\tau) + (1 - \lambda) R_{\text{div}}(\tau, \mathcal{H}_t), \quad (1)$$

where $\lambda \in [0, 1]$ balances task quality and reasoning-path diversity. Standard solution-level tree search mainly uses R_{qual} for expansion, which can repeatedly exploit locally similar candidates. DARTS instead follows the quality-diversity objective in Equation 1, favoring candidates that are both high-quality and distinct in their reasoning paths.

4 Method

The formulation above suggests two intervention points in solution-level tree search. First, expansion should sometimes depart from the parent reasoning path instead of only refining it. Second, feedback should reward reasoning-path diversity in addition to solution quality. Diversity-Aware Reasoning Tree Search (DARTS) implements these two interventions within a Bayesian tree-search framework. We first formalize the overall search process in Section 4.1, then describe diversity-aware expansion in Section 4.2 and the Sinkhorn-based (Watanabe and Isobe, 2024) diversity reward in Section 4.3.

4.1 Framework Overview

DARTS implements the quality-diversity objective in Section 3 through a Bayesian solution-level tree search. Each node $v \in \mathcal{T}$ stores a complete candidate solution τ_v , its tree relations, and posterior parameters for action selection. During search, the tree determines where computation should be allocated, while the global history pool $\mathcal{H}$ acts as the archive used to evaluate reasoning-path diversity. As shown in Figure 2, each iteration updates this archive through four stages: selection, diversity-aware expansion, diversity-augmented evaluation, and Bayesian backpropagation.

Following AB-MCTS, each node v maintains two meta-actions:

- GEN (Go Wider): Directly generates a new candidate child node based on the context of v .
- CONT (Go Deeper): Does not directly generate content, but serves as a downward routing mechanism to facilitate deep refinement.

DARTS uses Thompson sampling over Beta posteriors to adaptively allocate search effort. Each

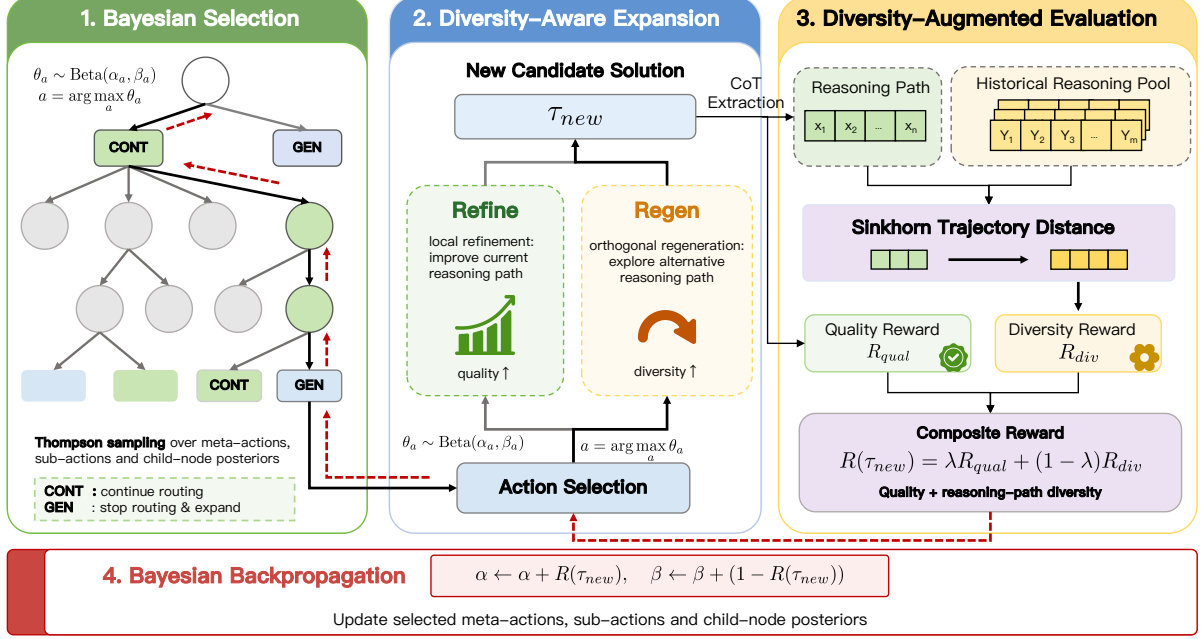


Figure 2: **Overview of DARTS.** DARTS performs solution-level Bayesian tree search over complete candidate solutions. Each iteration selects a node via Thompson sampling, expands it through either local refinement or orthogonal regeneration, evaluates the new candidate with a reward combining task quality and reasoning-path diversity, and backpropagates the reward to update action posteriors for subsequent search decisions.

iteration follows four stages: selection, diversity-aware expansion, diversity-augmented evaluation, and Bayesian backpropagation.

Bayesian Selection. Starting from the root, DARTS routes through the search tree $\mathcal{T}$ by Thompson sampling over Beta posteriors. At each visited node, DARTS first samples the posterior values of its meta-actions. If GEN is selected, routing stops and the current node is chosen for expansion. If CONT is selected, DARTS further selects the next child node by Thompson sampling over the Beta posteriors maintained by the child nodes. Thus, both meta-action selection and child-node routing are guided by posterior feedback.

Diversity-aware Expansion. At the selected node, DARTS generates a new candidate solution τ_{new} . The generation action is chosen between local refinement and orthogonal regeneration. Local refinement improves the current reasoning path, while orthogonal regeneration encourages the search to move toward an alternative solution paradigm. This expansion design is detailed in Section 4.2.

Diversity-augmented Evaluation. The new candidate is evaluated by a composite reward. This reward combines verifier-based solution quality with

reasoning-path diversity, so that the search favors candidates that are both valid and distinct from previously explored paths. The reward design is described in Section 4.3.

Bayesian Backpropagation. After evaluation, the composite reward is propagated along the current search path. DARTS updates the Beta posterior parameters of all selected components on this path, including the selected meta-actions, the selected generation sub-action, and the routed child nodes:

$$\alpha \leftarrow \alpha + R(\tau_{new}), \quad \beta \leftarrow \beta + (1 - R(\tau_{new})). \quad (2)$$

The complete pseudocode is provided in Appendix B.

4.2 Diversity-Aware Expansion

To reduce the tendency for expansion to inherit the parent reasoning path, we decouple GEN into two semantically distinct sub-actions: local refinement and orthogonal regeneration. This directly targets the parent-child inheritance risk in refinement-based tree search. Given a selected node v with candidate solution τ_v , the language model generates a new candidate as $\tau_{new} \sim P_\theta(\cdot | x, \mathcal{C}_a)$, where the context constraint $\mathcal{C}_a$ is determined by the chosen generation sub-action $a \in \mathcal{A}_{gen} = \{a_{ref}, a_{regen}\}$:

- **Refinement** (a_{refine}): This action exploits the current solution τ_v , preserving its underlying reasoning path while correcting local logic flaws.
- **Regeneration** (a_{regen}): This action treats the current solution τ_v as an exclusion constraint rather than a pattern to follow. Unlike repeated sampling from the original prompt, it asks the model to identify the parent’s reasoning strategy and generate an alternative route, discouraging parent-strategy inheritance.

For each node v , DARTS maintains separate Beta posteriors for a_{refine} and a_{regen} . When GEN is selected at v , the generation sub-action is chosen by Thompson sampling:

$$a^* = \arg \max_{a \in \mathcal{A}_{\text{gen}}} \theta_a, \quad \theta_a \sim \text{Beta}(\alpha_{v,a}, \beta_{v,a}). \quad (3)$$

After τ_{new} receives the composite reward $R(\tau_{\text{new}})$, the posterior of the selected sub-action at v is updated together with the meta-action posteriors sampled during selection.

4.3 Diversity-Augmented Reward via Sinkhorn Distance

After generating a new candidate solution τ_{new} , DARTS evaluates both its task quality and its diversity relative to previously explored reasoning paths. Since final outputs alone often fail to capture differences in underlying solution strategies, we extract the reasoning path of each candidate (detailed in Appendix C.4). Different strategies may contain variable-length and non-aligned reasoning steps, making conventional sequence-level distances unsuitable. We therefore use the Sinkhorn distance to align reasoning steps and quantify reasoning-path diversity.

Feature Extraction and Cost Matrix. To make reasoning paths comparable, we extract the intermediate reasoning steps of each candidate solution τ and encode them into a sequence of normalized vectors $X \in \mathbb{R}^{n \times d}$ using a fixed embedding model. Given two candidate solutions τ_A and τ_B with n and m steps (corresponding to feature matrices X and Y), we define the cost matrix $C \in \mathbb{R}^{n \times m}$ using squared Euclidean distance:

$$C_{i,j} = \|X_i - Y_j\|^2 \quad (4)$$

Sinkhorn Distance for Reasoning Paths. To compare reasoning paths with unequal lengths and misaligned step structures, we treat their steps as uniform empirical marginal distributions $\mathbf{u} = \frac{1}{n} \mathbf{1}_n$ and $\mathbf{v} = \frac{1}{m} \mathbf{1}_m$. The Sinkhorn distance is defined via the following entropy-regularized optimal transport problem:

$$D_{\text{Sinkhorn}}(\tau_A, \tau_B) = \min_{P \in U(\mathbf{u}, \mathbf{v})} \langle P, C \rangle_F + \epsilon \sum_{i,j} P_{i,j} \log P_{i,j} \quad (5)$$

The transport matrix P acts as a soft alignment between the steps of two reasoning paths. Here, $U(\mathbf{u}, \mathbf{v}) = \{P \in \mathbb{R}_+^{n \times m} \mid P \mathbf{1}_m = \mathbf{u}, P^\top \mathbf{1}_n = \mathbf{v}\}$ denotes the set of admissible soft-alignment matrices with the prescribed uniform marginals, and ϵ is the entropy regularization coefficient.

Diversity via Reasoning-Path Distance. Given the historical solution pool $\mathcal{H}$, the diversity score of the new solution τ_{new} is defined as its minimum Sinkhorn distance to existing reasoning paths.

$$R_{\text{div}}(\tau_{\text{new}}, \mathcal{H}) = \min_{\tau_k \in \mathcal{H}} D_{\text{Sinkhorn}}(\tau_{\text{new}}, \tau_k) \quad (6)$$

We use the clipped nearest-neighbor Sinkhorn distance as the diversity reward, where a larger value indicates a more distinct reasoning path; details are provided in Appendix C.4.

We combine this diversity score with the scalar quality reward R_{qual} from the verifier to form a composite reward:

$$R(\tau_{\text{new}}) = \lambda \cdot R_{\text{qual}}(\tau_{\text{new}}) + (1 - \lambda) \cdot R_{\text{div}}(\tau_{\text{new}}, \mathcal{H}) \quad (7)$$

where $\lambda \in [0, 1]$ balances correctness and exploration. The composite reward $R(\tau_{\text{new}}) \in [0, 1]$ is then backpropagated along the search path using Equation 2. It updates both the posterior parameters of the Beta distribution for the selected action (as described in Section 4.2) and the value distributions of the current node and all its ancestors, thereby driving the adaptive expansion of the search tree.

5 Experiments

We evaluate DARTS along five questions: **RQ 1:** whether DARTS improves test-time search, **RQ 2:** whether DARTS discovers more diverse correct algorithmic solutions, **RQ 3:** which design choices matter, **RQ 4:** whether the diversity-aware principle transfers beyond code generation, and **RQ 5:** how much computational overhead it introduces.

Method	LiveCodeBench v6			CodeContests			Avg. Rank
	Pass@64	Pass@128	Pass@1	Pass@64	Pass@128	Pass@1	
Qwen2.5-7B-Instruct							
Repeated Sampling	33.0 ± 0.5	35.2 ± 0.3	29.7 ± 0.8	15.2 ± 0.9	18.8 ± 1.0	15.8 ± 0.5	1.5
Sequential Refinement	23.1 ± 0.7	23.0 ± 0.7	21.7 ± 0.0	5.2 ± 0.3	5.2 ± 0.3	5.2 ± 0.3	6.0
Standard MCTS	29.1 ± 0.0	29.9 ± 0.3	26.7 ± 1.0	11.5 ± 0.0	11.5 ± 0.0	10.3 ± 0.0	5.0
ABMCTS-A	30.7 ± 0.5	32.2 ± 0.7	27.2 ± 1.0	13.4 ± 0.5	16.8 ± 1.1	15.0 ± 1.1	3.5
ABMCTS-M	30.6 ± 0.3	32.6 ± 0.6	27.7 ± 0.2	14.6 ± 0.9	16.8 ± 0.8	14.3 ± 1.1	3.5
DARTS (ours)	32.8 ± 1.5	34.5 ± 0.3	27.8 ± 0.3	17.6 ± 0.9	20.4 ± 1.2	16.2 ± 1.0	1.5
Qwen3-8B							
Repeated Sampling	72.8 ± 2.2	74.7 ± 1.4	54.9 ± 1.2	61.7 ± 2.0	63.1 ± 1.1	50.5 ± 2.2	3.3
Sequential Refinement	66.3 ± 0.5	69.0 ± 0.3	60.2 ± 1.4	57.2 ± 1.6	58.9 ± 1.0	51.7 ± 2.3	5.1
Standard MCTS	70.1 ± 1.5	72.0 ± 0.9	60.2 ± 0.7	58.7 ± 0.9	60.3 ± 1.2	51.3 ± 1.0	4.6
ABMCTS-A	72.0 ± 0.8	73.3 ± 0.3	58.5 ± 1.0	59.8 ± 1.0	62.0 ± 0.3	52.7 ± 0.9	3.8
ABMCTS-M	72.4 ± 1.2	73.3 ± 0.7	59.8 ± 1.4	60.9 ± 0.9	62.5 ± 1.1	53.1 ± 0.9	2.9
DARTS (ours)	73.7 ± 0.8	76.0 ± 0.8	60.4 ± 0.7	62.2 ± 0.3	64.8 ± 0.9	52.5 ± 1.9	1.3
Qwen3-14B							
Repeated Sampling	77.4 ± 0.3	79.7 ± 0.3	64.4 ± 0.7	67.8 ± 0.9	69.4 ± 0.5	54.0 ± 1.0	3.2
Sequential Refinement	74.9 ± 0.8	76.2 ± 0.7	67.4 ± 1.2	62.7 ± 0.8	65.0 ± 0.6	54.6 ± 0.5	5.0
Standard MCTS	76.6 ± 1.2	77.1 ± 0.9	65.7 ± 2.1	63.0 ± 0.5	65.1 ± 0.8	57.8 ± 0.8	4.0
ABMCTS-A	77.5 ± 0.3	78.7 ± 0.5	65.1 ± 0.8	65.1 ± 0.8	67.9 ± 0.9	56.6 ± 0.6	3.3
ABMCTS-M	77.0 ± 0.7	78.7 ± 1.8	64.2 ± 1.0	66.6 ± 1.5	68.4 ± 1.4	55.7 ± 0.8	3.8
DARTS (ours)	78.7 ± 0.5	80.0 ± 0.0	66.1 ± 0.5	67.5 ± 0.8	70.1 ± 0.8	55.6 ± 0.3	1.8

Table 1: Code-generation performance under the same inference budget. Pass@64 and Pass@128 measure exploration at different budgets, while Pass@1 measures final selected-solution quality. Best results within each model group are boldfaced.

5.1 Experimental Setup

Benchmarks and baselines. For code generation, we use LiveCodeBench V6 (Jain et al., 2025) (175 tasks) and CodeContests (Li et al., 2022) (165 tasks), with Qwen3-8B, Qwen3-14B (Yang et al., 2025), and Qwen2.5-7B-Instruct (Yang et al., 2024) as backbone models. We compare against repeated sampling, sequential refinement, standard MCTS, ABMCTS-A, and ABMCTS-M (Inoue et al., 2026). To test cross-domain generalization, we evaluate Qwen3-8B on AIME 2024 and AIME 2025, both with and without the Math-Shepherd Mistral-7B process reward model (?). We further evaluate PMO-style multi-objective molecular optimization (Gao et al., 2022; Ran et al., 2025) with GPT-4o-mini.

Evaluation and implementation. For code generation, we report Pass@ K to measure exploration under the search budget and Pass@1 to measure convergence, evaluating the last public-test-passing candidate on private tests. We also report EA, DA@ K , and NAUADC (Lee et al., 2025) to measure the diversity of functionally correct solutions. EA measures how evenly correct solutions

are distributed across algorithmic clusters, DA@ K measures the number of distinct correct strategies among top candidates, and NAUADC summarizes diversity coverage across budgets. We evaluate diversity on LiveCodeBench V6 with Qwen3-8B and Qwen3-14B, using the intersection of tasks with functionally correct solutions; this includes 81 tasks for Qwen3-8B and 125 tasks for Qwen3-14B. For AIME, we report Pass@64 and Pass@1 under a budget of 64. For molecular optimization, we report Top-1 score and Diversity_{Top10}, the average pairwise Tanimoto distance among top-10 Morgan fingerprints. Code-generation and molecular experiments use a budget of 128. Each setting is run three times, and we report mean and standard deviation. Additional details are provided in Appendix C.

5.2 RQ1: DARTS Improves Test-Time Search

Table 1 reports Pass@64, Pass@128, and Pass@1 on LiveCodeBench V6 and CodeContests, while Figure 3 presents the budget-scaling curves on representative Qwen3-8B settings. Overall, DARTS achieves the best average rank across all backbone settings, mainly driven by stronger exploration-

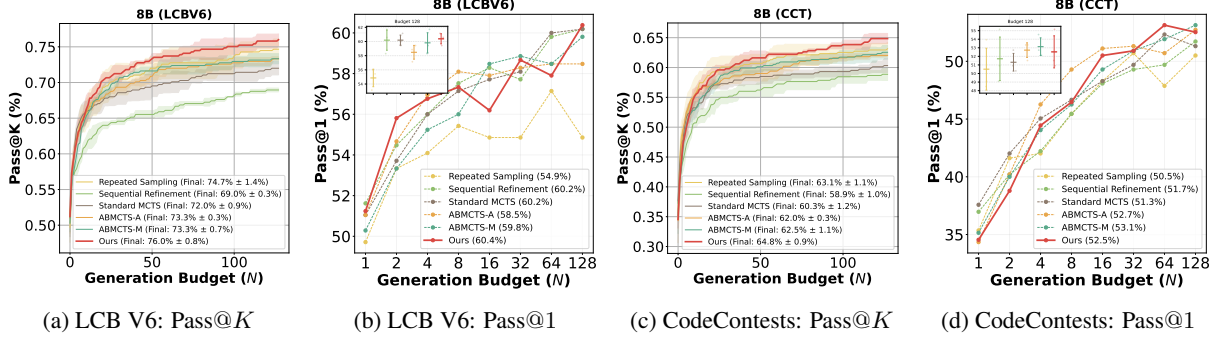


Figure 3: Budget scaling of Pass@K and Pass@1 on representative Qwen3-8B code-generation settings.

Method	Easy			Medium			Hard			All (Avg)		
	EA	DA@K	NAU	EA	DA@K	NAU	EA	DA@K	NAU	EA	DA@K	NAU
Qwen3-8B												
Standard MCTS	1.218	1.436	1.484	1.277	1.591	1.606	1.456	2.014	2.058	1.305	1.652	1.687
ABMCTSA	1.346	1.772	1.815	1.360	1.954	2.024	1.524	2.237	2.322	1.400	1.968	2.033
ABMCTSM	1.396	1.820	1.851	1.413	1.964	1.997	1.635	2.570	2.625	1.468	2.079	2.117
DARTS (ours)	1.407	1.854	1.886	1.486	2.268	2.344	1.739	2.806	2.906	1.527	2.271	2.338
Qwen3-14B												
Standard MCTS	1.172	1.266	1.276	1.429	1.835	1.903	1.520	2.159	2.280	1.369	1.738	1.802
ABMCTSA	1.280	1.626	1.684	1.572	2.235	2.396	1.681	2.586	2.740	1.505	2.132	2.255
ABMCTSM	1.242	1.537	1.572	1.611	2.359	2.482	1.924	3.085	3.252	1.579	2.297	2.403
DARTS (ours)	1.301	1.648	1.704	1.678	2.487	2.631	1.971	3.106	3.297	1.637	2.387	2.515

Table 2: Algorithmic diversity among functionally correct solutions on LiveCodeBench V6. Higher EA, DA@K, and NAUADC indicate broader or more balanced coverage of correct algorithmic strategies. Best results within each model group are boldfaced.

oriented performance.

DARTS improves exploration and budget efficiency. DARTS achieves the best Pass@128 results across the evaluated Qwen3 settings, reaching 76.0% and 64.8% on LiveCodeBench V6 and CodeContests with Qwen3-8B, and 80.0% and 70.1% with Qwen3-14B, respectively. As shown in Figure 3, its advantage generally increases with the search budget in the representative Qwen3-8B settings. Moreover, with a budget of 64, DARTS outperforms the strongest tree-search baseline with a budget of 128 in three of the six main settings and matches it in another. Additional scaling results across backbone models are reported in Appendix Tables 7 and 8. These results suggest that DARTS makes more effective use of additional test-time computation for solution exploration.

DARTS preserves competitive convergence performance. In terms of Pass@1, DARTS remains competitive with the strongest tree-search baselines on both Qwen3 backbones. These results show that, although DARTS explicitly encourages broader exploration, it still converges to high-quality final candidates under the public-test-

guided selection protocol.

5.3 RQ2: DARTS Discovers More Diverse Correct Solutions

Table 2 shows that DARTS produces more diverse functionally correct solutions on LiveCodeBench V6 with Qwen3-8B and Qwen3-14B across difficulty levels.

DARTS covers a broader set of correct algorithms. DARTS achieves the best DA@K and NAUADC on both Qwen3-8B and Qwen3-14B. Compared with the strongest baseline, ABMCTSM, NAUADC increases from 2.117 to 2.338 on Qwen3-8B and from 2.403 to 2.515 on Qwen3-14B. The gains are small on Easy problems, where the space of canonical solutions is limited, but become clearer on Medium and Hard problems. This indicates that DARTS discovers a wider range of valid algorithmic strategies when the task admits multiple solution paths.

DARTS produces a more balanced distribution of correct solutions. Beyond covering more algorithm clusters, DARTS also improves EA. The average EA increases from 1.468 to 1.527 on

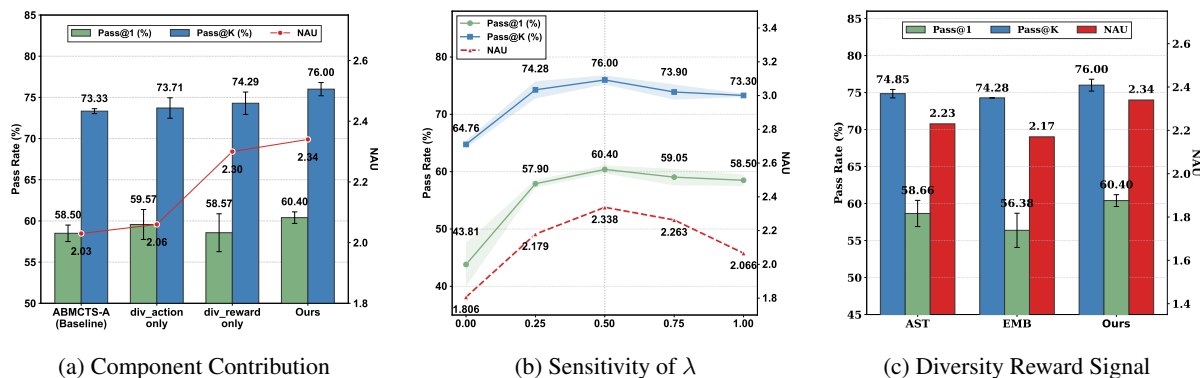


Figure 4: Ablation studies on LiveCodeBench V6 with Qwen3-8B. From left to right, the panels evaluate component contributions, sensitivity to the quality-diversity weight λ , and alternatives for the diversity reward signal.

Qwen3-8B and from 1.579 to 1.637 on Qwen3-14B compared with ABMCTS-M. This suggests that the additional correct solutions are not concentrated in a few dominant strategies, but are spread more evenly across different algorithmic patterns.

5.4 RQ3: Ablation Study

We conduct ablation studies on LiveCodeBench V6 with Qwen3-8B. We ablate DARTS along its two intervention points: orthogonal regeneration in expansion and the reasoning-path diversity reward in evaluation. We also test the quality-diversity trade-off weight λ and the reasoning-path distance signal used in the diversity reward.

Component Contribution. Figure 4a shows that regeneration and the diversity reward both improve performance and diversity. Each component raises Pass@128 over the ABMCTS-A baseline (73.33%), and the full model performs best, reaching 76.00% Pass@128, 60.40% Pass@1, and 2.34 NAU. This indicates that the two components jointly improve search effectiveness and correct-solution diversity.

Quality-Diversity Weight λ . Figure 4b shows that DARTS performs best with a balanced reward at $\lambda = 0.5$. Both pure-diversity ($\lambda = 0$) and pure-quality ($\lambda = 1$) settings perform worse, indicating that the two signals are complementary.

Diversity Reward Signal. Figure 4c shows that the Sinkhorn-based reasoning-path signal provides the most effective diversity feedback. It achieves the best Pass@1, Pass@128, and NAU scores (60.40%, 76.00%, and 2.34). AST-level and embedding-level alternatives perform consistently worse, suggesting that trajectory-level comparison

captures reasoning-path diversity more effectively than surface structural or semantic similarity.

5.5 RQ4: Generalization Beyond Code Generation

Mathematical reasoning. Table 3 reports results on AIME 2024 and 2025. DARTS obtains the best or tied-best score in seven of eight evaluation settings. The improvement is clearest on AIME 2025: without a PRM, DARTS raises both Pass@64 and Pass@1 from the strongest baseline’s 85.00% to 88.34%. With PRM guidance, it ties the best Pass@64 at 86.67% and achieves the highest Pass@1 at 68.33%. On AIME 2024, DARTS remains competitive, tying the best score in three of four settings. These results indicate that diversity-aware search transfers to free-form mathematical reasoning and remains compatible with process-reward guidance.

Molecular optimization. Table 4 reports the molecular optimization results. DARTS obtains the highest Top-1 score of 3.801, outperforming Repeated Sampling (3.778), ABMCTS-A (3.772), and Standard MCTS (3.750). It also achieves the highest Diversity score of 0.750, exceeding Sequential Refinement (0.729) and other tree-search baselines. These results indicate that DARTS can identify high-scoring molecules while maintaining a broader structural distribution among top candidates, suggesting that the diversity-aware search mechanism transfers to multi-objective molecular optimization. Full results are provided in Appendix D.3.

5.6 RQ5: Cost Analysis

Tables 5 and 6 provide complementary views of generation and auxiliary computation.

Method	AIME 2024				AIME 2025			
	w/o PRM		w/ PRM		w/o PRM		w/ PRM	
	P@64	P@1	P@64	P@1	P@64	P@1	P@64	P@1
Repeated Sampling	70.00 $\pm$.00	70.00 $\pm$.00	70.00 $\pm$.00	55.00 $\pm$ 2.36	85.00 $\pm$ 2.36	85.00 $\pm$ 2.36	85.00 $\pm$ 2.36	61.67 $\pm$ 2.36
Sequential Refinement	63.33 $\pm$.00	63.33 $\pm$.00	63.33 $\pm$.00	61.67 $\pm$ 2.36	81.67 $\pm$ 2.36	81.67 $\pm$ 2.36	75.00 $\pm$ 2.36	66.67 $\pm$ 4.72
Standard MCTS	66.67 $\pm$.00	66.67 $\pm$.00	65.00 $\pm$ 2.36	58.34 $\pm$ 2.36	83.33 $\pm$.00	83.33 $\pm$.00	80.00 $\pm$.00	65.00 $\pm$ 2.36
ABMCTS-A	66.67 $\pm$.00	66.67 $\pm$.00	68.34 $\pm$ 2.36	60.00 $\pm$.00	85.00 $\pm$ 2.36	85.00 $\pm$ 2.36	86.67 $\pm$.00	65.00 $\pm$ 2.36
ABMCTS-M	66.67 $\pm$.00	66.67 $\pm$.00	66.67 $\pm$.00	53.34 $\pm$ 4.72	85.00 $\pm$ 2.36	85.00 $\pm$ 2.36	83.34 $\pm$ 4.72	61.67 $\pm$ 7.07
DARTS (ours)	70.00 $\pm$.00	70.00 $\pm$.00	70.00 $\pm$.00	60.00 $\pm$.00	88.34 $\pm$ 2.36	88.34 $\pm$ 2.36	86.67 $\pm$ 4.72	68.33 $\pm$ 7.07

Table 3: Mathematical reasoning results with Qwen3-8B. “w/ PRM” uses Math-Shepherd Mistral-7B as the quality reward. P@64 and P@1 denote Pass@64 and Pass@1. Best results are boldfaced.

Method	Top-1	Diversity _{Top10}
Rep. Sampling	3.778 $\pm$.029	0.687 $\pm$.068
Seq. Refinement	3.657 $\pm$.115	0.729 $\pm$.030
Standard MCTS	3.750 $\pm$.025	0.530 $\pm$.144
ABMCTS-A	3.772 $\pm$.070	0.668 $\pm$.048
ABMCTS-M	3.728 $\pm$.020	0.606 $\pm$.100
DARTS (ours)	3.801 $\pm$.051	0.750 $\pm$.015

Table 4: Molecular optimization results. DARTS achieves the best Top-1 score and the highest structural diversity among top candidates.

Method	Tokens
Rep. Sampling	16,696.24
Seq. Refinement	10,600.93
Standard MCTS	10,099.40
ABMCTS-A	11,802.18
ABMCTS-M	12,461.68
DARTS (ours)	12,322.52

Table 5: Average output-token consumption per LiveCodeBench V6 task at a search budget of 128, estimated post hoc over the full evaluation.

Generation cost. At a search budget of 128, DARTS averages 12,322.52 output tokens per task, comparable to ABMCTS-A (+4.4%) and ABMCTS-M (−1.1%), and 26.2% below repeated sampling. Because the budget fixes expansions rather than response length, variation reflects candidate and refinement-trajectory lengths. DARTS’s improved coverage therefore does not rely on substantially more generation than the adaptive tree-search baselines.

Auxiliary computation. We profile five representative Easy, Medium, and Hard tasks at 16 expansions. Reasoning-path extraction, embedding, and Sinkhorn computation add 16.39–19.23 seconds in these cases. This represents 0.62%–2.38% of end-to-end task time. Their absolute time remains stable while total task latency increases by roughly $4.1\times$, so the relative share falls on longer tasks dominated by LLM generation. Extraction

Task	Total	Extract.	Embed.	Sink.	Extra
E-83	687.72	14.30	1.72	0.37	2.38%
M-69	1017.22	15.66	1.58	0.24	1.72%
M-80	1747.37	15.09	1.49	0.20	0.96%
H-137	1845.28	17.45	1.53	0.24	1.04%
H-87	2819.66	16.15	1.19	0.21	0.62%

Table 6: DARTS latency profile on five representative LiveCodeBench V6 tasks for 16 expansions. E, M, and H denote Easy, Medium, and Hard; component latencies are in seconds, and Extra is their combined share of wall-clock time.

is the main component, while each Sinkhorn comparison takes approximately 1.6–3.1 ms. Within this profile, DARTS’s gains are not explained by substantially more hidden computation. These five-task results characterize the current execution stack rather than provide a universal latency guarantee; larger budgets may require caching, subsampling, or approximate transport.

6 Conclusion

We study the local homogenization problem in solution-level MCTS for LLM reasoning and propose DARTS, which couples orthogonal regeneration with a Sinkhorn-based reasoning-path diversity reward inside a Bayesian search framework. Compared with existing tree search methods, DARTS uses reasoning-path diversity as an intermediate search objective to broaden solution-space coverage. This leads to more diverse successful outputs in code generation and transfers to mathematical reasoning and molecular optimization while preserving competitive convergence behavior. Our cost analysis further shows that diversity-specific computation is small relative to generation at the tested budgets. These observations suggest that explicitly modeling reasoning-path diversity within inference-time tree search is a general and practical lever for improving test-time scaling.

Limitations

DARTS has two main limitations. First, although its measured auxiliary overhead is small at budgets up to 128, the all-pairs Sinkhorn comparison scales with reasoning-path length and history-pool size; very large budgets may therefore require cached embeddings, subsampled histories, or approximate transport. Second, our cross-domain evaluation is limited to competition mathematics and molecular optimization. Formal proof, tool-using agents, and adaptive selection of the diversity weight λ remain open directions.

Acknowledgments

This work was supported by the Shanghai Artificial Intelligence Laboratory. This work was also supported by the National Science and Technology Major Project (Grant Nos. 2025ZD1606200 and 2025ZD1606203), the National Natural Science Foundation of China (Grant No. 92470205), and the Suzhou Science and Technology Development Program (SYG2024037).

References

- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.
- Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfschagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.
- Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. 2021. Quality-diversity optimization: a novel branch of stochastic optimization. In *Black box optimization, machine learning, and no-free lunch theorems*, pages 109–135. Springer.
- Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*.
- Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. 2022. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35:21342–21357.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. 2026. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. *Advances in Neural Information Processing Systems*, 38:35448–35484.
- Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.
- Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. 2026. Artificial hivemind: The open-ended homogeneity of language models (and beyond). *Advances in Neural Information Processing Systems*, 38.
- Seonghyeon Lee, Heejae Chon, Joonwon Jang, Dongha Lee, and Hwanjo Yu. 2025. How diversely can language models solve problems? exploring the algorithmic diversity of model-generated code. *arXiv preprint arXiv:2503.00691*.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.
- Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

- Ranjita Naik, Varun Chandrasekaran, Mert Yuksekgonul, Hamid Palangi, and Besmira Nushi. 2023. Diversity of thought improves reasoning abilities of llms. *arXiv preprint arXiv:2310.07088*.
- Nian Ran, Yue Wang, Xiaoyuan Zhang, Zhongzheng Li, Qingsong Ran, Wenhao Li, and Richard Allmendinger. 2025. Exllm: Experience-enhanced llm optimization for molecular design and beyond. *arXiv preprint arXiv:2502.12845*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.
- Alexander Shypula, Shuo Li, Botong Zhang, Vishakh Padmakumar, Kayo Yin, and Osbert Bastani. 2025. Evaluating the diversity and quality of llm generated content. *arXiv preprint arXiv:2504.12522*.
- Peng-Yuan Wang, Tian-Shuo Liu, Chenyang Wang, Ziniu Li, Yidi Wang, Shu Yan, Chengxing Jia, Xu-Hui Liu, Xinwei Chen, Jiacheng Xu, and 1 others. 2026. A survey on large language models for mathematical reasoning. *ACM Computing Surveys*, 58(8):1–35.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Kazuki Watanabe and Noboru Isobe. 2024. Sinkhorn algorithm for sequentially composed optimal transports. *arXiv preprint arXiv:2412.03120*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Yiming Zhang, Harshita Diddee, Susan Holm, Hanchen Liu, Xinyue Liu, Vinay Samuel, Barry Wang, and Daphne Ippolito. 2025. Noveltybench: Evaluating language models for humanlike diversity. *arXiv preprint arXiv:2504.05228*.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

A Local Homogenization Diagnostics

This appendix provides the diagnostic analysis behind the local homogenization discussion in the main paper. We examine how algorithmic patterns evolve during solution-level tree expansion. Existing tree-search methods typically expand a node by refining its current solution and use quality-oriented rewards to guide future search. These two design choices can narrow the explored solution space: refinement may cause child nodes to inherit the reasoning structure of their parents, while quality-only feedback may repeatedly reinforce a small number of early successful patterns.

We study this behavior on LiveCodeBench V6 using Qwen3-8B and Qwen3-14B under the same search budget as the main experiments. For each generated candidate, we assign an algorithmic cluster label according to its solution strategy and track the ordered sequence of clusters during expansion. This allows us to analyze parent-child similarity, reward concentration over algorithmic clusters, global diversity loss, and final search success.

This section provides the detailed definitions of the diversity diagnostics used in this analysis. For each task i , we record the candidate solutions generated during search in expansion order and assign each candidate to an algorithmic-equivalence cluster, yielding a cluster-label sequence $\mathbf{c}_i = (c_{i,1}, \dots, c_{i,T_i})$. Here, T_i is the number of expanded candidates for task i , and $c_{i,t}$ denotes the cluster label of the t -th candidate. For tree-search methods, we additionally record the parent-child relation, node depth, quality reward $r_{i,t}$, and final test outcome of each node. Figure 5 is computed from this common set of records.

Relative local homogenization. We first measure whether child nodes inherit their parents’ algorithmic clusters more often than expected from the tree topology and depth distribution alone. Given a node budget k , let $E_i(k)$ be the set of parent-child edges induced by the first k expanded nodes of task i . The observed parent-child cluster agreement rate is

$$R_{\text{same}}^{\text{obs}}(i, k) = \frac{1}{|E_i(k)|} \sum_{(u,v) \in E_i(k)} \mathbb{I}[c_u = c_v],$$

where c_u and c_v are the cluster labels of parent node u and child node v , respectively. We construct a depth-controlled permutation null model by fixing the search-tree skeleton and randomly

permuting cluster labels only among nodes at the same depth. This preserves both the tree structure and the cluster distribution within each depth level. Let $R_{\text{same}}^{\text{depth-null}}(i, k)$ denote the expected agreement rate under this null model. The relative local homogenization score used in Panel (a) is

$$H_{\text{local}}(i, k) = \frac{R_{\text{same}}^{\text{obs}}(i, k) - R_{\text{same}}^{\text{depth-null}}(i, k)}{R_{\text{same}}^{\text{depth-null}}(i, k)}.$$

A larger $H_{\text{local}}(i, k)$ indicates stronger parent-child algorithmic agreement beyond the depth-controlled baseline. Figure 5a shows that child nodes are more likely to fall into the same algorithmic cluster as their parents than expected under a depth-controlled permutation baseline. The gap also increases as search proceeds, indicating that refinement-based expansion tends to preserve the parent reasoning pattern rather than producing independent alternatives.

Reward concentration. We next quantify whether quality rewards are concentrated in a small number of algorithmic clusters. For each cluster c in task i , we accumulate the rewards of candidates assigned to that cluster:

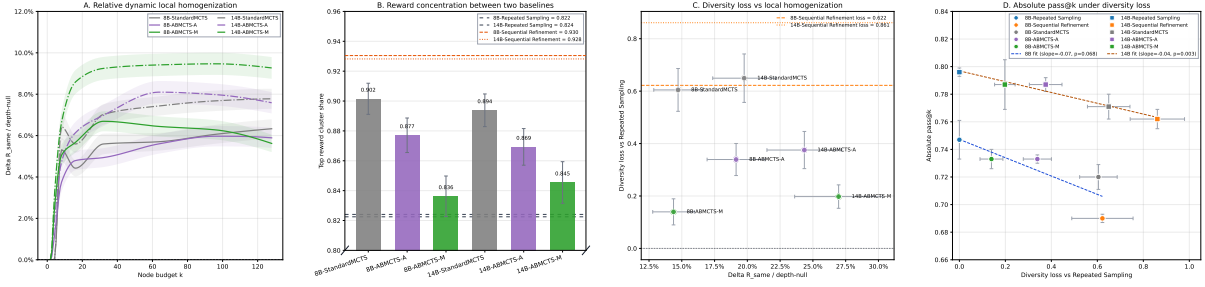
$$R_{i,c} = \sum_{t=1}^{T_i} r_{i,t} \mathbb{I}[c_{i,t} = c].$$

The reward concentration score is the largest cluster-level reward share:

$$C_{\text{reward}}(i) = \frac{\max_c R_{i,c}}{\sum_c R_{i,c}}.$$

Values close to 1 indicate that the search reward is dominated by one or a few algorithmic clusters, whereas lower values indicate a more balanced reward distribution across solution strategies. Figure 5b reports this metric to compare reward allocation in MCTS-based methods against Repeated Sampling. It shows that rewards in standard MCTS are concentrated on a small number of algorithmic clusters, suggesting that quality-only feedback may amplify dominant solution patterns once they receive early positive rewards.

Global diversity loss. Third, we measure the global algorithmic diversity covered by the search and compare it against Repeated Sampling. Let $p_{i,c}$ be the proportion of candidates in task i assigned to cluster c . We define the effective number of



(a) Relative local homogenization (b) Reward concentration between different algorithms (c) Diversity loss versus local homogenization (d) Search success under diversity loss

Figure 5: Diagnostic analysis of local homogenization in solution-level MCTS on LiveCodeBench V6. The panels examine parent-child cluster inheritance, reward concentration across clusters, the relation between local homogenization and global diversity loss, and the relation between diversity loss and Pass@K.

algorithms as the exponential of the cluster-label entropy:

$$N_{\text{eff}}(i) = \exp \left(- \sum_c p_{i,c} \log p_{i,c} \right).$$

This quantity can be interpreted as the number of equally frequent algorithmic clusters that would yield the same entropy. It increases when the search covers more clusters and when the distribution across clusters is more balanced. Using Repeated Sampling as the task-level diversity reference, we define the diversity loss of method m as

$$D_{\text{loss}}(i, m) = N_{\text{eff}}^{\text{RS}}(i) - N_{\text{eff}}^m(i),$$

where $N_{\text{eff}}^{\text{RS}}(i)$ is the effective number of algorithms produced by Repeated Sampling on the same task, and $N_{\text{eff}}^m(i)$ is the corresponding value for method m . Figure 5c compares $D_{\text{loss}}(i, m)$ with $H_{\text{local}}(i, k)$ at the final search budget to examine whether local homogenization is accompanied by a reduction in global algorithmic coverage.

Exploration success under diversity loss. Finally, we relate diversity loss to exploration success. For task i , $\text{Pass}@K(i)$ indicates whether the search finds at least one successful candidate within the first K expansions:

$$\text{Pass}@K(i) = \max_{t \leq K} \mathbb{I}[\text{reward_success}_{i,t} \wedge \text{test_success}_{i,t}].$$

We average $\text{Pass}@K(i)$ over tasks to obtain the method-level $\text{Pass}@K$. To analyze how diversity loss relates to final exploration performance, we fit the following linear model on LiveCodeBench separately for each model scale:

$$\text{Pass}@K = \beta_0 + \beta_1 D_{\text{loss}}.$$

Figure 5d reports this relationship. A negative slope indicates that larger global diversity loss is associated with weaker search success. Together, Figures 5c and 5d connect local search homogenization to global exploration outcomes, providing diagnostic support for introducing diversity-aware expansion and diversity-augmented rewards in DARTS.

Together, these metrics form a progressive diagnostic chain: H_{local} measures parent-child homogenization during local expansion, C_{reward} measures whether quality rewards further concentrate on a few algorithmic clusters, D_{loss} measures whether this local bias corresponds to reduced global algorithmic coverage, and the $\text{Pass}@K$ regression examines whether such diversity loss is associated with weaker search success. This diagnostic chain provides the empirical motivation for introducing diversity-aware expansion and diversity-augmented rewards in DARTS.

B DARTS Algorithm

Algorithm 1 expands the method description in Sections 4.1–4.3. The input consists of a problem x , a language model P_θ , a task verifier V , and the search hyperparameters used by the quality-diversity reward. DARTS first samples an initial solution as the root node and maintains a global history pool $\mathcal{H}$ containing all generated candidates. Each search iteration traverses the current tree with Thompson sampling over the meta-actions GEN and CONT: CONT moves the search to an existing child, while GEN stops traversal and expands the selected node. During expansion, DARTS again uses Thompson sampling to choose between local refinement and orthogonal regeneration. The new candidate is evaluated by a composite reward that

Algorithm 1 Diversity-Aware Reasoning Tree Search

Require: Input problem x , language model P_θ , verifier V , search budget B , quality-diversity weight λ , entropy coefficient ϵ

Ensure: Final candidate solution τ^*

```
1: Initialize root solution  $\tau_0 \sim P_\theta(\cdot | x)$  and search tree  $\mathcal{T}$  with root node  $v_0$ 
2: Initialize global historical pool  $\mathcal{H} \leftarrow \{\tau_0\}$ 
3: Initialize all meta-action and sub-action posteriors with Beta(0.5, 0.5)
4: for  $b = 1$  to  $B$  do
5:    $v \leftarrow v_0, \mathcal{P} \leftarrow []$ 
6:   while  $v$  has children do
7:     Sample  $\theta_a \sim \text{Beta}(\alpha_{v,a}, \beta_{v,a})$  for each  $a \in \{\text{GEN}, \text{CONT}\}$ 
8:      $a^* \leftarrow \arg \max_{a \in \{\text{GEN}, \text{CONT}\}} \theta_a$ 
9:     Append  $(v, a^*)$  to  $\mathcal{P}$ 
10:    if  $a^* = \text{GEN}$  then
11:      break
12:    else
13:      Select child  $u \in \text{Children}(v)$  by Thompson sampling over child posteriors
14:       $v \leftarrow u$ 
15:    end if
16:  end while
17:  Sample  $\theta_s \sim \text{Beta}(\alpha_{v,s}, \beta_{v,s})$  for each  $s \in \{a_{\text{ref}}, a_{\text{regen}}\}$ 
18:   $s^* \leftarrow \arg \max_{s \in \{a_{\text{ref}}, a_{\text{regen}}\}} \theta_s$ 
19:  Append  $(v, s^*)$  to  $\mathcal{P}$ 
20:  if  $s^* = a_{\text{ref}}$  then
21:    Build context  $\mathcal{C}_{s^*}$  to repair  $\tau_v$  while preserving its reasoning trajectory
22:  else
23:    Build context  $\mathcal{C}_{s^*}$  using  $\tau_v$  as an exclusion constraint for orthogonal regeneration
24:  end if
25:  Generate  $\tau_{\text{new}} \sim P_\theta(\cdot | x, \mathcal{C}_{s^*})$  and add it as a child node of  $v$ 
26:   $R_{\text{qual}} \leftarrow V(\tau_{\text{new}})$ 
27:  Extract reasoning steps from  $\tau_{\text{new}}$  and compute the nearest transport cost
28:   $d_{\text{nn}} \leftarrow \min_{\tau_k \in \mathcal{H}} D_{\text{OT}}(\tau_{\text{new}}, \tau_k)$ 
29:   $R_{\text{div}}(\tau_{\text{new}}, \mathcal{H}) \leftarrow \text{clip}(d_{\text{nn}}, 0, 1)$ 
30:   $R \leftarrow \lambda R_{\text{qual}} + (1 - \lambda) R_{\text{div}}(\tau_{\text{new}}, \mathcal{H})$ 
31:   $\mathcal{H} \leftarrow \mathcal{H} \cup \{\tau_{\text{new}}\}$ 
32:  for all  $(u, a) \in \mathcal{P}$  do
33:     $\alpha_{u,a} \leftarrow \alpha_{u,a} + R$ 
34:     $\beta_{u,a} \leftarrow \beta_{u,a} + (1 - R)$ 
35:  end for
36: end for
37: return  $\tau^* = \arg \max_{\tau \in \mathcal{H}} V(\tau)$ 
```

combines verifier-based task quality with Sinkhorn-based reasoning-path diversity against the history pool. This reward is then backpropagated to update all selected Bayesian posteriors, allowing later iterations to adaptively allocate budget to actions that produce both high-quality and non-redundant solutions. After the budget is exhausted, the algorithm returns the best candidate according to the verifier.

C Implementation Details

This appendix provides the implementation details for the experiments in Section 5. We first summarize the shared decoding, search, and evaluation settings, and then report the prompts used for code generation and PMO molecular optimization. Template variables in braces denote fields filled at run-time using the current problem, parent candidate, or execution feedback.

Code-generation tasks, models, and baselines.

For code generation, we use LiveCodeBench V6 with 175 tasks and CodeContests with 165 tasks. We evaluate Qwen3-8B, Qwen3-14B, and Qwen2.5-7B-Instruct as backbone models in the main experiments. To further test Pass@ K scaling across model families, we additionally report tree-search results on Qwen3-4B, Areal-8B, and Areal-14B in Appendix D.1. The compared methods are repeated sampling, sequential refinement, standard MCTS, ABMCTS-A, ABMCTS-M, and DARTS.

Molecular optimization task. For molecular optimization, we use the Practical Molecular Optimization (PMO) evaluation framework. The model starts from the task description rather than an initial molecule and generates candidate molecules in SMILES format. Each candidate is evaluated by PMO using five objectives: synthetic accessi-

bility (SA), DRD2 activity, QED, GSK3 β activity, and JNK3 activity. The optimization directions are min for SA, min for DRD2, max for QED, min for GSK3 β , and max for JNK3. PMO normalizes and aggregates these objective scores to provide the scalar quality signal used during search. We use GPT-4o-mini as the backbone model and apply the same test-time search strategies as in code generation, including repeated sampling, sequential refinement, standard MCTS, ABMCTS-A, ABMCTS-M, and DARTS.

C.1 Shared Configuration

All experiments are conducted using the vLLM framework. The maximum generation length is set to 32,768, and the decoding temperature is set to 0.6 for both the Qwen series and GPT-4o-mini. The search budget is limited to 128 iterations per problem. Each experiment is run independently three times, and we report the mean and standard deviation.

For standard MCTS, the branching factor is fixed at 5. For AB-MCTS-A and DARTS, the priors are initialized with a symmetric Beta distribution Beta(0.5, 0.5). For the composite reward, the quality-diversity weight is set to $\lambda = 0.5$, and the entropy regularization coefficient in the Sinkhorn distance is set to $\epsilon = 0.1$.

For MCTS-based methods, the quality reward is binary: $R_{\text{qual}} \in \{0, 1\}$. A candidate receives $R_{\text{qual}} = 1$ if it passes the verifier available during search, and 0 otherwise. In code-generation tasks, $\text{reward_success}_{i,t}$ denotes whether the t -th candidate for task i passes the public tests or execution checks used as search feedback, while $\text{test_success}_{i,t}$ denotes whether the same candidate passes the final evaluation tests.

For code-generation metrics, we report Pass@ K with $K = 128$. We compute Pass@1 using the final candidate code that passes the public test cases. Algorithmic diversity metrics are computed only on functionally correct solutions to avoid conflating diversity with pass rate. Algorithmic-equivalence clustering is performed using Qwen2.5-7B-Instruct with temperature 0. We set $K = 40$ for DA@ K and $K_{\text{max}} = 128$ for NAUADC.

Task intersections for diversity evaluation. For the diversity results in Table 2, we compute EA, DA@ K , and NAUADC on the intersection of LiveCodeBench V6 tasks available for all compared methods under each backbone. Because the set

of tasks with functionally correct candidates differs by backbone, the intersections are computed separately for Qwen3-8B and Qwen3-14B. The Qwen3-8B intersection contains 81 tasks, including 28 Easy, 31 Medium, and 22 Hard tasks. The Qwen3-14B intersection contains 125 tasks, including 43 Easy, 44 Medium, and 38 Hard tasks.

Qwen3-8B task IDs by difficulty

Easy (28): 9, 153, 66, 79, 20, 171, 83, 30, 33, 164, 41, 13, 52, 57, 6, 31, 138, 167, 113, 37, 1, 21, 43, 71, 51, 45, 145, 156
Medium (31): 95, 119, 8, 136, 67, 174, 139, 55, 122, 4, 80, 49, 128, 12, 108, 59, 154, 142, 133, 63, 69, 146, 36, 24, 168, 127, 118, 77, 5, 123, 42
Hard (22): 39, 70, 62, 23, 68, 16, 53, 26, 105, 78, 160, 84, 40, 38, 101, 144, 170, 54, 61, 175, 50, 152

Qwen3-14B task IDs by difficulty

Easy (43): 82, 158, 20, 167, 33, 121, 13, 30, 164, 171, 161, 135, 138, 9, 1, 43, 149, 83, 27, 21, 113, 37, 131, 57, 79, 51, 14, 66, 52, 45, 156, 3, 60, 41, 124, 173, 76, 31, 117, 6, 153, 145, 71
Medium (44): 42, 119, 95, 67, 24, 47, 136, 147, 169, 77, 162, 123, 174, 128, 150, 5, 140, 114, 12, 8, 108, 69, 122, 133, 36, 80, 22, 127, 146, 55, 143, 125, 154, 115, 118, 168, 59, 49, 63, 72, 139, 75, 142, 4
Hard (38): 16, 17, 44, 53, 100, 54, 65, 170, 46, 68, 101, 129, 126, 38, 28, 18, 84, 48, 2, 62, 144, 29, 105, 40, 152, 141, 50, 39, 166, 70, 7, 61, 78, 175, 172, 160, 120, 23

C.2 Code-Generation Prompts

Initial candidate generation. The initial generation prompt is used to produce the root candidate for each code-generation problem. It provides the problem statement and starter code, and asks the model to return a complete Python solution in the required code-block format.

Code Initial Generation Prompt

You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.

Question:

{self.problem}

You will use the following starter code to write the solution to the problem and enclose your code within delimiters.

{self.starter_code}

Answer: (use the provided format with back-ticks)

Local refinement. The refinement prompt is used when DARTS chooses the local refinement action. It exposes the execution report from the parent candidate and asks the model to infer the root cause of the failure, repair the underlying

logic, and return a generalized corrected program. The same refinement prompt is used for the Sequential Refinement baseline, so refinement-based comparisons use aligned prompt instructions. The `execution_report` field contains the parent candidate code together with a summary of its execution results, including passed and failed public tests when available.

Code Local Refinement Prompt

TASK: Deep Code Repair and Generalization

Here is the test report from the last execution:
{execution_report}

WARNING: Do not write special-case code (e.g., using an `if` statement for a specific input) just to fix the failures listed above. This “patching” approach will fail on unseen test cases.

Your Goal: Treat the failed cases as **clues** to find and fix the **fundamental algorithmic or logical flaw** in the code.

Please follow this thinking framework to provide a generalized solution:

1. Pattern Analysis:

- Examine all failed test cases. Are they edge cases (e.g., empty lists, zero, max values)? Data type issues (e.g., integer vs. float)? Or inputs with a specific pattern?
- Compare successful vs. failed cases and ask: “Under what conditions does my algorithm break?”

2. Root Cause Deduction:

- Based on the pattern, pinpoint the specific defect in the code. Is it a wrong loop condition? Improper variable initialization? A missing base case in recursion? Or a flaw in the overall algorithm logic?

3. General Fix Strategy:

- Formulate a plan that corrects the root cause. This fix must naturally pass all known failed cases and generalize to similar, unseen ones.

4. Provide Final Code:

- Based on your general fix strategy, provide the **complete and immediately runnable** corrected code.
- **Do not** add any explanations, comments, or dialogue before or after the code.

Answer: (use the provided format with back-ticks)

it explicitly asks the model to identify the current algorithmic paradigm and replace it with a meaningfully different solution strategy. As in local refinement, `execution_report` contains the parent candidate code and its summarized execution feedback.

Code Orthogonal Regeneration Prompt

TASK: Algorithmic Paradigm Shift & Re-implementation

Here is the test report from the last execution:
{execution_report}

Context: The current implementation uses a specific algorithmic approach that may be flawed, inefficient, or overly complex.

WARNING: Do not write special-case code (e.g., using an `if` statement for a specific input) just to pass specific test cases. This “patching” approach will be rejected.

Your Goal: Solve the problem by implementing a **fundamentally different algorithmic approach** from the one currently used. You must replace the underlying logic, not just refactor the syntax.

Please follow this thinking framework to provide the new solution:

1. Current Logic Identification:

- Briefly analyze the provided code to identify its core algorithm (e.g., “It currently uses nested loops to check all pairs”).
- Identify why this approach might be failing or suboptimal (e.g., complexity issues, logic gaps in edge cases).

2. Algorithm Selection:

- Select a **new, distinct algorithm** to solve the problem.
- Ensure this new approach is robust and mathematically sound. It should naturally handle edge cases without hard-coded patches.

3. Implementation Strategy:

- Plan the structure of the new code.
- Verify that the time and space complexity are acceptable (or improved).

4. Provide Final Code:

- Based on your new strategy, provide the **complete and immediately runnable** code.
- **Do not** add any explanations, comments, or dialogue before or after the code.

Answer: (use the provided format with back-ticks)

Orthogonal regeneration. The regeneration prompt is used when DARTS chooses the orthogonal regeneration action. Unlike local refinement,

C.3 Molecular Optimization Prompts

Initial candidate generation. For the PMO molecular optimization benchmark, the initial prompt asks the model to generate candidates from scratch according to the benchmark description and optimization goals. The output contains short design reasoning followed by SMILES strings wrapped in <candidate> tags.

PMO Initial Generation Prompt

You are optimizing molecules for a Practical Molecular Optimization benchmark.

{description}

Optimization goals:

- Minimize synthetic accessibility (SA).
- Minimize DRD2 activity.
- Maximize QED.
- Minimize GSK3 β activity.
- Maximize JNK3 activity.

{other_requirements}

Return {N} candidate(s).

{example_output}

Do not output Python code.

First provide your reasoning as numbered steps. Then provide the final answer.

Final Output Structure

Step 1: [Strategic Action]

Step 2: [Strategic Action]

Step K: [Strategic Action]

<candidate>SMILES</candidate>

Local refinement. The PMO refinement prompt improves a parent molecular proposal using only the parent reasoning, previous candidates, and evaluation summary as context. This mirrors local refinement in code generation: the model should improve the current design direction rather than reset it.

PMO Local Refinement Prompt

Refine the previous proposal and produce improved molecule candidates.

Optimization goals:

- Minimize synthetic accessibility (SA).
- Minimize DRD2 activity.
- Maximize QED.

- Minimize GSK3 β activity.
- Maximize JNK3 activity.

Parent reasoning:

{parent_reasoning}

Previous candidates: {previous_candidates}

Evaluation summary: {evaluation_summary}

Use only the parent proposal above as context. Improve these parent candidates.

The output molecules should be valid.

Return {N} refined candidate(s).

First provide your reasoning as numbered steps. Then provide the final answer.

Final Output Structure

Step 1: [Strategic Action]

Step 2: [Strategic Action]

Step K: [Strategic Action]

<candidate>SMILES</candidate>

Orthogonal regeneration. The PMO regeneration prompt asks the model to reset the molecular design strategy. It uses the parent evaluation report to identify bottlenecks, then encourages a different scaffold, ring system, substituent pattern, heteroatom placement, polarity profile, or molecular size when appropriate.

PMO Orthogonal Regeneration Prompt

TASK: Molecular Design Strategy Reset

Here is the evaluation report from the parent proposal only:

{evaluation_report}

Context: The parent molecular proposal is valid for evaluation, but the present design direction may be trapped in a weak local region, over-reusing poor motifs, or failing to balance the objectives.

WARNING: Do not make trivial edits that only preserve the current scaffold with cosmetic substitutions unless they are clearly justified by the objective trade-offs. Avoid simply repeating previously proposed or invalid molecules.

Your Goal: Propose a **meaningfully different molecular design direction** that still respects the optimization goals, while remaining chemically plausible and valid in SMILES format.

Please follow this thinking framework to provide the new proposal:

1. Failure Pattern Identification:

- Briefly infer which objectives are currently bottlenecking the candidates.
- Note whether the current proposal appears invalid, duplicated, or dominated by weak tradeoffs.

2. Design Strategy Shift:

- Select a different molecular modification strategy from the previous round.
- Consider changing scaffold, ring system, polarity, substituent pattern, heteroatom placement, or molecular size if this is likely to improve the weak objectives.

3. Tradeoff Control:

- Preserve any strong properties only when they do not block improvement on the weakest objectives.
- Prefer candidates that are diverse from the previous proposal but still aligned with the target objectives.

4. Provide Final Candidates:

- Please provide your design logic first as numbered steps, followed by the final candidate.
- Each candidate must be enclosed in `<candidate>SMILES</candidate>`.

The output molecules should be valid.

Final Output Structure

Step 1: [Strategic Action]

Step 2: [Strategic Action]

Step K: [Strategic Action]

`<candidate>SMILES</candidate>`

C.4 Reasoning-Path Extraction

DARTS computes reasoning-path diversity from an ordered sequence of intermediate reasoning steps for each candidate. In code generation, candidates are executable programs, so we use a fixed extraction prompt to summarize each program into a high-level algorithmic plan. The extraction model is the same backbone model used in the corresponding test-time search run; when the backbone is a reasoning model, we explicitly use its non-thinking mode for extraction. In molecular optimization, the generation prompts require the model to output design reasoning followed by SMILES candidates, so the reasoning path is parsed directly from the generated reasoning steps. Each extracted step is encoded with Qwen3-Embedding-0.6B, which is kept fixed across methods, backbones, and tasks; the deployed embedding service returns L2-normalized vectors. Pairwise squared Euclidean costs are computed from these step embeddings before applying the Sinkhorn distance. In both settings, the extracted reasoning path is used only as an internal search signal for the diversity reward and is not exposed to the final evaluator.

Extraction efficiency. The reasoning-path extraction step adds limited overhead in code-generation tasks because it only asks the test backbone to produce a short high-level algorithmic plan from an already generated program. For reasoning backbones, this extraction is performed in non-thinking mode, which avoids long deliberation traces. In molecular optimization, no additional extraction call is required: the generation prompt already asks the model to output design reasoning before the final SMILES candidates, and DARTS reuses this generated reasoning as the reasoning path for diversity computation.

Sinkhorn reward calibration. The entropy-regularized Sinkhorn objective is used to compute a soft alignment plan between two reasoning paths. For the diversity reward, we use the transport-cost term induced by the resulting plan, rather than the signed regularized objective value:

$$D_{\text{OT}}(\tau_A, \tau_B) = \langle P^*, C \rangle_F, \quad (8)$$

where P^* is the Sinkhorn alignment plan and $C_{ij} = \|X_i - Y_j\|_2^2$. Although the L2-normalized embeddings make individual squared Euclidean costs bounded, the nearest-neighbor transport cost is not assumed to be analytically confined to $[0, 1]$. Empirically, almost all observed nearest-neighbor Sinkhorn transport costs fall within this interval. When an out-of-range value occurs, we clip it before using it as the diversity reward:

$$R_{\text{div}}(\tau_{\text{new}}, \mathcal{H}) = \text{clip}(d_{\text{nn}}(\tau_{\text{new}}), 0, 1), \\ d_{\text{nn}}(\tau_{\text{new}}) = \min_{\tau_k \in \mathcal{H}} D_{\text{OT}}(\tau_{\text{new}}, \tau_k). \quad (9)$$

This calibration ensures that the composite reward remains in $[0, 1]$ and is compatible with the Beta posterior update used by DARTS.

Code reasoning-path extraction prompt. Before computing reasoning-path diversity for code generation, we use the following prompt to extract the high-level algorithmic strategy from each candidate program. The extracted ordered steps serve as the reasoning path used by the diversity reward.

Code Reasoning-Path Extraction Prompt

Role

You are a Senior Algorithm Architect. You possess the ability to look past syntax and see the underlying mathematical and structural logic of any implementation.

Task

Analyze the provided Python code and extract its **high-level algorithmic strategy**. You must translate “how the code is written” into “how the problem is solved conceptually.”

Strict Constraints

1. **Formatting:** Output ONLY a numbered list. Each item MUST start with Step i : (e.g., Step 1:, Step 2:).
2. **No Code Talk:** DO NOT use words like “loop,” “if-statement,” “variable,” “array,” “index,” “function,” or “return.”
3. **Architectural Verbs:** Use high-level verbs such as *Decompose*, *Prune*, *Traverse*, *Aggregate*, *Distill*, *Normalize*, *Optimize*, or *Map*.
4. **No Meta-talk:** Do not provide any introductory text like “Here is the strategy” or “Based on the code.” Start directly with Step 1:.

Comparison Guide (For your internal alignment)

Bad (Code-level): “Initialize a dictionary to store counts and loop through the list.”

Good (Strategy-level): “Establish a frequency mapping to track the distribution of elements across the input space.”

Code to Analyze

{code}

Final Output Structure

Step 1: [Strategic Action]

Step 2: [Strategic Action]

Step K : [Strategic Action]

D Additional Experimental Results

This appendix provides the full experimental results referenced in Section 5. Appendix D.1 reports detailed code-generation scaling curves across budgets, Appendix D.2 gives a task-level search-tree case study, and Appendix D.3 reports the full molecular optimization results.

D.1 Code-Generation Pass@ K Scaling

Tables 7 and 8 supplement the RQ1 analysis by reporting Pass@ K at all evaluated budgets for LiveCodeBench V6 and CodeContests. The tables include the tree-search methods evaluated across model families and parameter scales. The runs column in the raw export is omitted for compactness, and the best result within each model group and budget is boldfaced.

D.2 Search-Tree Case Study

We present a representative LiveCodeBench V6 case, output_11 (*Simultaneous Kagamimochi*), to illustrate how diversity-aware search changes the algorithmic structure of the explored tree. The problem gives N mochi sorted by size and asks for the maximum number K of pairs that can be formed, where a smaller mochi of size a can be

placed on a larger mochi of size b only if $2a \leq b$. Each mochi can be used at most once.

Figures 6–9 visualize the search trees produced by different tree-search methods. All three baselines collapse into the first algorithmic cluster: greedy two-pointer sequential matching. This strategy scans small and large mochi with two pointers, commits a pair whenever the current large mochi is at least twice the current small mochi, and otherwise moves the large pointer to search for a feasible partner. Although this local construction is intuitive, it does not explicitly separate the candidate region for small mochi from the candidate region for large mochi, so it can allocate resources incorrectly in this case.

In contrast, DARTS generates three distinct algorithmic clusters through orthogonal regeneration. The first cluster is the same greedy two-pointer matching strategy explored by the baselines. The second cluster performs binary search over the answer K and checks whether the smallest K mochi can be matched with the largest K mochi. This converts the problem into a global feasibility test over two disjoint regions of the sorted sequence and corresponds to the correct core algorithm. The third cluster uses a bisect-based greedy matching strategy, which searches for the first feasible large mochi for each small mochi and then continues matching from the next position. This case shows that adaptive budget allocation alone may still reinforce a single dominant reasoning path, while DARTS can escape local homogenization by rewarding reasoning-path diversity and regenerating orthogonal candidates.

Diversity of correct solutions. We further present a diversity-focused case study on output_97 (*Complement Interval Graph*), a hard AtCoder task from LiveCodeBench V6. The problem defines a graph over weighted intervals, where two vertices are connected when their intervals do not intersect; for each query, the model must determine reachability and, if reachable, compute the minimum vertex-weight sum along a path. This task admits multiple correct algorithmic formulations, including explicit complement-graph construction, endpoint-sorted neighbor enumeration, component filtering, and on-the-fly Dijkstra variants.

Figures 10–13 compare the correct-answer clusters discovered by different tree-search methods. The legends count only clusters that produce cor-

Method	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16	Pass@32	Pass@64	Pass@128
Qwen3-4B								
Standard MCTS	49.33±1.89	54.86±0.47	59.81±0.71	61.90±0.27	62.48±0.27	64.76±0.54	66.86±0.47	68.95±0.97
ABMCTS-A	51.54±1.62	55.71±0.76	59.72±1.01	63.72±1.69	65.32±2.25	67.71±1.82	70.10±1.88	71.71±2.80
ABMCTS-M	48.57±0.93	53.16±0.82	57.17±0.50	60.42±0.76	64.06±0.54	67.30±0.50	69.60±0.62	71.13±1.03
DARTS (ours)	48.09±1.33	54.77±1.56	59.16±0.63	63.17±0.70	66.22±0.55	68.89±0.46	70.04±0.66	72.90±0.66
Qwen3-8B								
Standard MCTS	51.05±2.69	55.81±0.71	61.33±1.43	65.14±1.68	66.86±1.23	68.57±1.62	70.10±1.50	72.00±0.93
ABMCTS-A	51.05±2.20	56.38±1.43	60.57±0.47	64.19±0.71	67.43±0.47	69.33±0.54	72.00±0.81	73.33±0.27
ABMCTS-M	50.29±1.23	55.43±0.81	58.48±0.54	63.62±0.71	67.43±0.00	70.67±0.27	72.38±1.17	73.33±0.71
DARTS (ours)	51.24±0.27	57.33±1.17	60.57±1.23	64.38±1.17	68.38±1.17	71.24±1.17	73.71±0.81	76.00±0.81
Qwen3-14B								
Standard MCTS	56.57±1.68	64.00±0.00	69.33±0.27	72.19±0.54	73.14±0.93	75.24±1.17	76.57±1.23	77.14±0.93
ABMCTS-A	57.52±2.65	63.62±1.43	68.38±0.97	71.81±1.08	74.29±0.47	76.38±0.27	77.52±0.27	78.67±0.54
ABMCTS-M	57.14±0.93	61.14±1.23	66.67±2.10	71.24±1.08	73.52±0.54	76.00±0.47	76.95±0.71	78.67±1.77
DARTS (ours)	60.57±1.23	64.95±1.89	68.57±2.03	71.24±2.57	74.29±1.62	76.76±1.89	78.67±0.54	80.00±0.00
Qwen2.5-7B								
Standard MCTS	19.43±0.81	21.52±1.17	24.95±1.17	27.05±0.27	27.62±0.27	28.95±0.27	29.14±0.00	29.90±0.27
ABMCTS-A	20.00±2.03	21.33±2.20	23.81±1.50	26.48±0.54	27.43±0.47	29.90±0.54	30.67±0.54	32.19±0.71
ABMCTS-M	18.29±1.71	20.57±1.71	22.57±0.29	24.86±0.86	27.43±0.57	29.43±0.29	30.57±0.29	32.57±0.57
DARTS (ours)	20.00±0.93	21.52±0.54	24.95±0.54	27.24±0.27	28.38±0.97	31.05±0.71	32.76±1.50	34.48±0.27
Areal-8B								
Standard MCTS	52.76±0.97	56.38±0.71	58.67±0.71	61.33±1.43	61.71±1.68	63.05±2.20	64.19±1.43	65.52±1.43
ABMCTS-A	53.33±0.54	56.00±0.47	59.24±0.54	61.14±1.40	62.86±2.03	64.57±1.23	67.81±1.64	69.71±1.40
ABMCTS-M	51.43±2.86	56.57±0.00	59.71±0.29	62.57±0.86	65.43±0.86	67.14±2.00	69.14±2.29	70.86±1.14
DARTS (ours)	52.95±1.94	55.43±2.14	57.52±1.64	60.57±1.68	64.38±1.17	66.67±1.08	69.71±0.81	72.19±0.71
Areal-14B								
Standard MCTS	60.38±1.17	67.24±1.35	70.86±0.47	72.57±0.00	73.71±0.47	75.24±0.71	77.14±0.81	78.29±1.23
ABMCTS-A	60.95±0.97	66.29±1.40	70.86±1.23	74.86±1.40	76.76±1.77	78.67±0.71	79.62±0.97	80.57±1.23
ABMCTS-M	60.57±3.06	65.71±1.62	69.71±1.68	72.38±1.08	75.05±1.50	77.90±1.08	78.86±0.81	80.76±0.71
DARTS (ours)	61.14±1.14	64.86±0.29	68.00±1.14	70.29±0.00	74.57±0.29	76.86±0.86	80.00±0.00	81.43±0.29

Table 7: Detailed Pass@ K results on LiveCodeBench V6 for tree-search methods. The runs column from the raw export is omitted. Within each model group, the best result in each column is boldfaced.

rect answers, while non-correct candidates are shown in gray. Standard MCTS discovers three correct clusters, but most correct candidates concentrate in SortedEndpointDijkstra (106 candidates), with only a small number assigned to explicit complement-adjacency or bisect-component variants. ABMCTS-A is even more concentrated: 121 correct candidates fall into the ExplicitPairDijkstra cluster, while only three candidates belong to the remaining two correct clusters. ABMCTS-M improves the spread by producing five correct clusters, but its correct candidates are still dominated by explicit adjacency and component-then-Dijkstra families.

DARTS discovers seven correct-answer clusters in the same case. Beyond explicit-adjacency Dijkstra variants, it also finds on-the-fly pairwise Dijkstra, binary-search adjacency construction, sorted-endpoint Dijkstra, and component-graph Dijkstra. This case illustrates the intended behavior of the diversity reward: DARTS does not merely increase the number of correct candidates, but expands the

set of distinct correct reasoning paths explored under the same search budget.

D.3 Full Molecular Optimization Results

Table 9 supplements the RQ4 analysis by reporting all molecular optimization metrics. The main text reports Top-1 and Diversity_{Top10} for compactness; this table additionally includes Top-10, AUC_{Top10}, and average rank.

Method	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16	Pass@32	Pass@64	Pass@128
Qwen3-4B								
Standard MCTS	33.27±1.28	39.20±1.04	44.91±0.95	47.56±0.87	48.78±0.25	52.66±0.91	55.73±0.81	57.95±0.32
ABMCTS-A	31.32±1.11	37.50±1.99	42.41±1.14	47.47±1.79	51.82±2.23	55.06±1.82	57.54±0.85	59.97±0.99
ABMCTS-M	31.10±1.22	37.55±2.12	41.85±0.91	46.55±1.14	49.46±1.07	53.22±0.36	55.49±0.74	57.96±1.04
DARTS (ours)	31.57±1.17	37.07±0.36	42.98±1.71	47.66±0.30	50.92±1.00	54.38±0.59	57.64±1.03	61.31±0.55
Qwen3-8B								
Standard MCTS	37.65±0.41	43.73±1.40	47.98±1.11	52.43±1.63	54.25±1.24	57.49±0.74	58.71±0.89	60.33±1.16
ABMCTS-A	34.34±3.75	41.01±3.71	49.49±1.59	54.14±0.29	56.77±0.29	58.18±0.99	59.80±1.03	62.02±0.29
ABMCTS-M	35.22±3.06	41.09±2.66	46.36±1.50	52.03±1.76	55.26±0.52	59.11±1.07	60.93±0.87	62.55±1.14
DARTS (ours)	34.55±1.48	41.01±0.76	47.27±0.99	52.32±1.87	56.97±1.78	59.80±0.76	62.22±0.29	64.85±0.86
Qwen3-14B								
Standard MCTS	41.82±0.49	48.69±0.76	54.75±0.57	57.78±0.76	59.60±0.29	61.62±0.29	63.03±0.49	65.05±0.76
ABMCTS-A	40.00±3.46	45.05±2.90	51.92±2.00	56.16±1.51	59.60±0.76	62.42±0.49	65.05±0.76	67.88±0.86
ABMCTS-M	40.69±1.59	47.17±0.52	52.63±1.11	56.88±0.93	60.32±1.92	63.56±1.08	66.60±1.49	68.42±1.37
DARTS (ours)	40.81±1.03	45.86±1.03	51.31±1.51	57.78±2.23	61.01±2.00	64.65±1.43	67.47±0.76	70.10±0.76
Qwen2.5-7B								
Standard MCTS	3.64±0.00	4.24±0.00	5.45±0.00	7.27±0.00	8.48±0.00	10.91±0.00	11.52±0.00	11.52±0.00
ABMCTS-A	2.63±0.75	4.25±0.49	5.26±0.77	7.89±0.50	9.72±0.47	11.74±1.13	13.36±0.46	16.80±1.08
ABMCTS-M	3.03±0.99	4.04±0.76	5.25±1.14	8.28±0.76	9.49±0.76	11.31±1.03	14.55±0.86	16.77±0.76
DARTS (ours)	3.03±1.31	4.44±0.76	6.46±0.29	9.09±0.49	11.11±0.76	14.55±0.99	17.58±0.86	20.40±1.25
Areal-8B								
Standard MCTS	42.83±1.59	49.70±1.98	55.56±0.76	58.18±0.00	60.61±0.49	61.82±0.86	62.22±0.76	62.83±0.29
ABMCTS-A	42.63±1.25	47.47±1.25	51.11±1.03	54.75±1.14	58.59±1.03	62.22±0.29	63.64±0.86	64.85±0.49
ABMCTS-M	40.40±1.51	46.46±0.29	51.31±1.51	55.15±1.98	59.60±1.14	62.63±1.03	64.44±1.03	65.66±1.51
DARTS (ours)	39.80±0.76	45.25±0.76	49.90±1.43	55.15±1.31	60.00±0.49	62.02±1.03	65.05±1.59	67.07±1.25
Areal-14B								
Standard MCTS	47.88±0.99	52.73±0.49	57.58±0.86	64.44±1.03	65.66±1.25	67.27±0.86	69.90±1.25	71.11±1.43
ABMCTS-A	45.86±2.49	51.72±2.06	56.36±1.71	61.21±1.31	65.25±1.03	67.68±1.03	69.29±1.03	70.91±0.86
ABMCTS-M	46.46±2.54	52.93±1.25	58.38±1.51	62.63±0.29	65.05±1.03	66.87±0.29	68.69±0.76	71.52±0.99
DARTS (ours)	45.86±2.23	50.71±2.34	57.78±0.57	63.84±1.87	67.47±0.76	68.89±1.03	70.51±1.14	72.73±0.86

Table 8: Detailed Pass@K results on CodeContests for tree-search methods. The runs column from the raw export is omitted. Within each model group, the best result in each column is boldfaced.

Method	Top-1	Top-10	AUC _{Top-10}	Diversity	Avg. Rank
Repeated Sampling	3.778 ± 0.029	3.707 ± 0.009	3.646 ± 0.005	0.687 ± 0.068	3.00
Sequential Refinement	3.657 ± 0.115	3.593 ± 0.095	3.551 ± 0.089	0.729 ± 0.030	5.00
Standard MCTS	3.750 ± 0.025	3.717 ± 0.040	3.658 ± 0.037	0.530 ± 0.144	3.00
ABMCTS-A	3.772 ± 0.070	3.704 ± 0.021	3.651 ± 0.007	0.668 ± 0.048	3.25
ABMCTS-M	3.728 ± 0.020	3.688 ± 0.002	3.641 ± 0.018	0.606 ± 0.100	5.00
DARTS (ours)	3.801 ± 0.051	3.709 ± 0.001	3.649 ± 0.006	0.750 ± 0.015	1.75

Table 9: Full results on the multi-objective molecular optimization task. Best results are boldfaced.

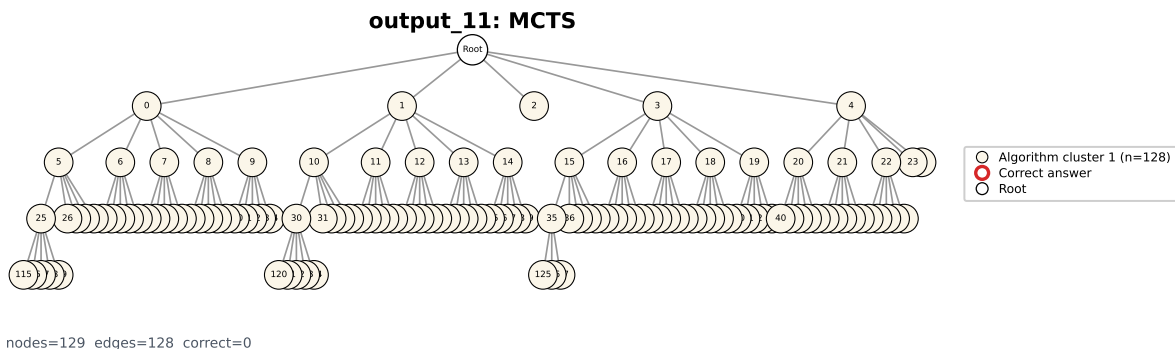


Figure 6: Search tree of Standard MCTS on output_11. The generated candidates remain in the greedy two-pointer matching cluster.

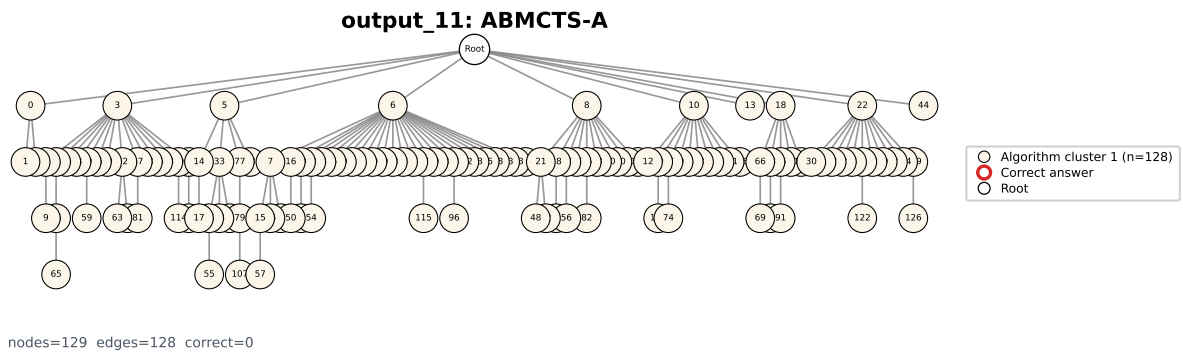


Figure 7: Search tree of ABMCTS-A on output_11. Adaptive branch allocation still concentrates the search on the greedy two-pointer matching cluster.

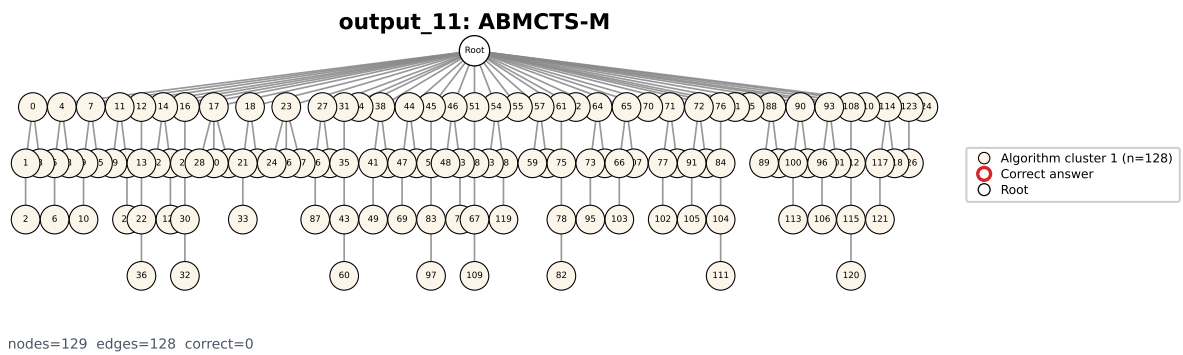


Figure 8: Search tree of ABMCTS-M on output_11. The method also remains within the same greedy two-pointer matching cluster.

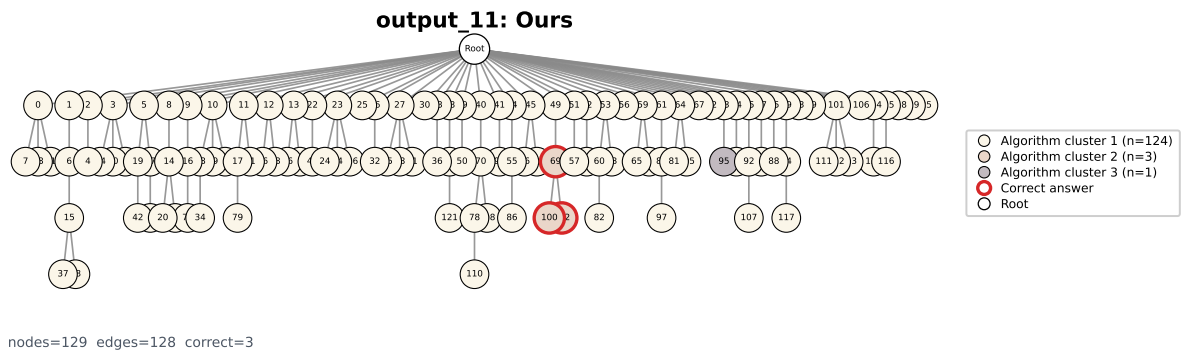


Figure 9: Search tree of DARTS on output_11. DARTS explores multiple algorithmic clusters and discovers the binary-search feasibility formulation.

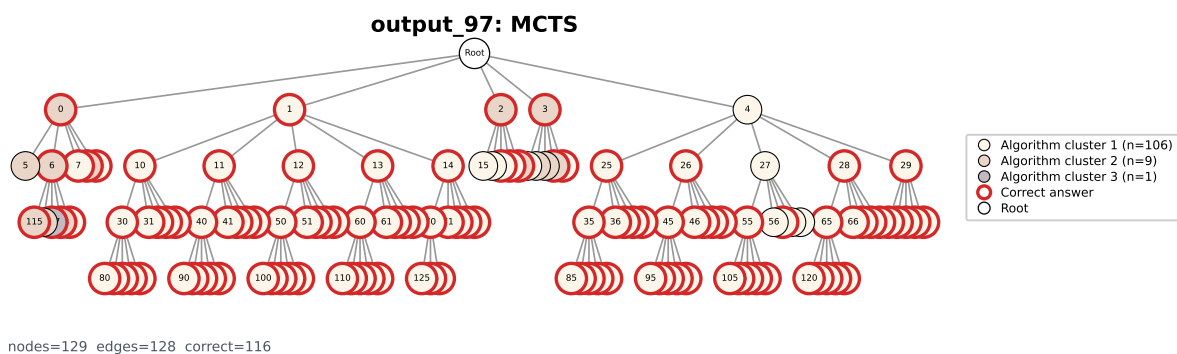


Figure 10: Search tree of Standard MCTS on output_97. Correct candidates are dominated by the SortedEndpoint-Dijkstra cluster.

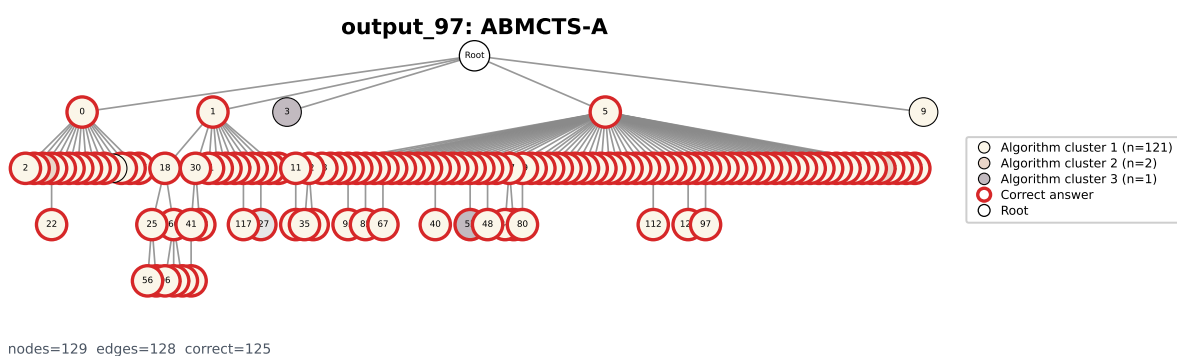


Figure 11: Search tree of ABMCTS-A on output_97. Most correct candidates remain in the ExplicitPairDijkstra cluster.

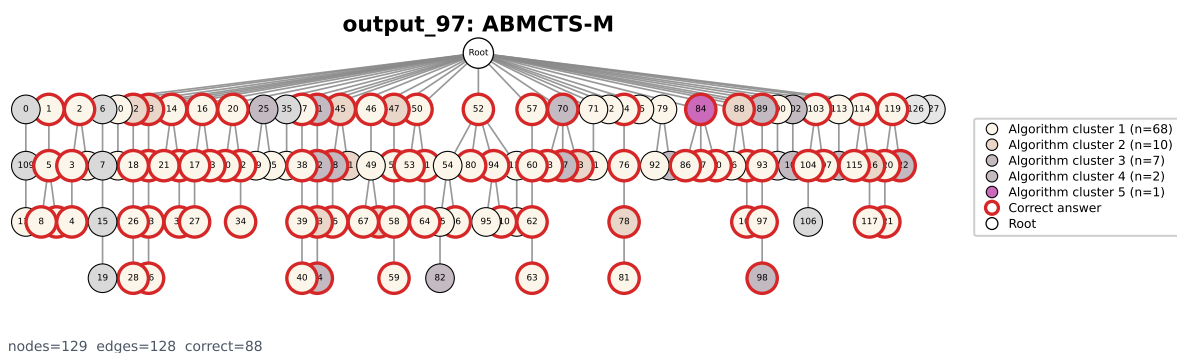


Figure 12: Search tree of ABMCTS-M on output_97. The method discovers several correct clusters but still concentrates on explicit adjacency and component-based Dijkstra variants.

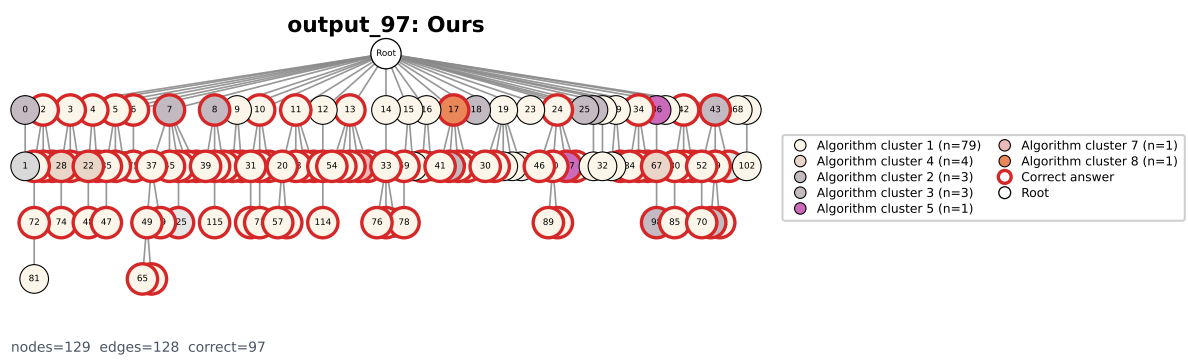


Figure 13: Search tree of DARTS on output_97. DARTS discovers a broader set of correct algorithmic clusters, including on-the-fly, endpoint-sorted, binary-search, and component-graph Dijkstra variants.