SCULPT:Constraint-Guided Pruned MCTS that Carves Efficient Paths for Mathematical Reasoning

Anonymous ACL submission

Abstract

Automated agent workflows can enhance the problem-solving ability of large language models (LLMs), but common search strategies rely on stochastic exploration and often traverse implausible branches. This occurs because current pipelines sample candidate steps from generic prompts or learned policies with weak domain priors, yielding near-random walks over operators, units, and formats. We instead pursue ordered exploration by imposing lightweight mathematical constraints that prioritize type- and unit-consistent steps while suppressing implausible branches. We present SCULPT, a constraint-guided pruning approach for Monte Carlo Tree Search (MCTS) that injects mathematical domain knowledge into selection, expansion, simulation, and backpropagation. SCULPT scores and prunes actions using dimensional and type checks, pattern similarity, operator diversity, and magnitude sanity, thereby steering the search toward plausible reasoning paths. When integrating SCULPT with the same LLMs configuration as the baseline, stable improvements were achieved on multiple datasets; we further report results with GPT-5-mini to assess executor transferability. These results indicate that lightweight, domain-aware constraints can improve accuracy while maintaining efficiency.

1 Introduction

With the rapid progress of LLMs, automated agent workflows have become a practical way to structure complex problem solving (Yu et al., 2025; Tan et al., 2025; Ferrag et al., 2025). However, many systems still depend on largely stochastic exploration, because candidate steps are sampled from generic prompts or weakly informed policies with little mathematical prior. This causes compute to spread over implausible branches (e.g., type or units violations), slowing convergence.

We instead pursue constraint-guided exploration: lightweight mathematical checks provide local structure to the search space, prioritizing type/units-consistent steps and deprioritizing unlikely branches while preserving diversity. In this view, constraints act as soft priors that convert unguided sampling into ordered exploration over plausible operations and compositions.

We therefore propose SCULPT, a domain-aware pruning and shaping approach that formalizes dimensional, type, pattern and magnitude checks and integrates them throughout the workflow search loop via compliance-aware selection, thresholded expansion, and reward shaping. The resulting framework reduces ineffective exploration while remaining compatible with existing components of AFlow(Zhang et al., 2024).

Challenges in mathematical workflow optimization. Mathematical reasoning exhibits heterogeneous structure across topic areas (e.g., arithmetic, number theory, geometry). Workflows must respect typing rules, units and angle conventions, and algebraic identities, while geometry often requires composing local constraints into global constructions. Naive stochastic search suffers from: (i) wide branching factors created by many admissible but implausible actions; (ii) long-horizon dependencies that make rollout estimates noisy; and (iii) sensitivity to formatting conventions (e.g., divisors vs. positive divisors; degree vs. radian; annulus area as outer minus inner). These factors slow convergence and lead to avoidable errors even with strong executors.

Design principles. SCULPT follows three principles. First, *early hard pruning*: depth-aware thresholds filter actions with low compliance scores before expansion, shrinking the tree. Second, *soft shaping*: compliance scores modulate selection and simulation, biasing search toward type/units-consistent steps without forbidding exploration. Third, *compatibility*: constraints are expressed as lightweight checks and pattern priors, so the

method slots into existing AFlow components and can evolve as new priors become available.

Problem statement. Given a dataset of problems and an initial workflow, the goal is to find a program (workflow instance) that maximizes accuracy under a fixed optimizer–executor budget. Let C_i denote constraint scores (depth, diversity, dimensional, types, pattern, magnitude). SCULPT computes an aggregate compliance $C_{\text{total}} = \prod_i w_i C_i$ and uses it to (a) prune expansions by a dynamic threshold $\tau(d)$ that decreases with depth d, and (b) reweight the Upper Confidence Bound (UCB) and rollout rewards. We study whether such shaping improves accuracy under equal budgets.

Overview. Figure 1 summarizes how SCULPT interacts with the standard optimization–execution workflow. The Instruction Hub computes constraint scores and forwards a compliance vector to the Routine Stack, where selection and expansion are shaped by $\tilde{u} = u \exp(\lambda C_{\text{total}})$ and a depth-aware gate $\tau(d)$. Rollouts are mildly nudged toward compliant actions and credit is weighted by compliance. The Representation Layer exposes alternative views (graph/code/network) without changing APIs, so the constraints act purely as side signals that guide exploration. For diagram readability in Figure 1, subscripts use full words (e.g., dim, type, pattern, mag, depth, diversity); in the text we use symbolic indices $\mathcal{U}$ (dimension/unit), $\mathcal{T}$ (type), $\mathcal{P}$ (pattern), $\mathcal{M}$ (magnitude), $\mathcal{D}$ (depth), and $\mathcal{V}$ (diversity).

Research questions. We examine whether lightweight domain constraints improve accuracy under matched compute on GSM8K, MATH and GSM-Hard, which error categories benefit most (e.g., divisor conventions, ring/annulus areas, right–triangle ratios), and whether depth-aware pruning reduces ineffective exploration without harming diversity.

Empirical protocol. We hold the optimizer and executor fixed across methods (Claude-3.5-sonnet and GPT-4o-mini-0718), use the same data splits, and report accuracy on GSM8K, MATH, and GSM-Hard where applicable; cohort-specific coverage follows Section 5. SCULPT adds only the constraint shaping described above; no additional external tools are introduced. Section 5 details datasets, baselines and metrics.

Our contributions are as follows: (1) Problem formulation: we incorporate mathematical domain constraints into automated agent workflows and formalize their use within MCTS. (2) SCULPT: we propose a constraint-guided MCTS that integrates dimensional consistency, type compatibility, pattern similarity, and operator diversity into selection, expansion, simulation and backpropagation, thereby guiding search toward mathematically plausible programs. (3) Comparative evaluation: We conducted a comparative assessment on the GSM8K, MATH, and GSM-Hard datasets, using the same optimizer-executor setup, against representative baselines, and reported consistent improvements.

2 Related work

Recent work on automating agent workflows spans: prompt-level optimization (Fernando et al., 2023; Yuksekgonul et al., 2024; Tang et al., 2023; Khattab et al., 2024), system/hyperparameter optimization (Saad-Falcon et al., 2024), and end-to-end workflow optimization (Li et al., 2024; Zhou et al., 2024; Zhuge et al., 2024; Hu et al., 2024). The latter line aims to synthesize and refine full workflows rather than single prompts (Zhang et al., 2024).

Hu et al. (2024) represents workflows in code and maintains linear histories, while Zhuge et al. (2024) combines graph structures with reinforcement learning. Zhang et al. (2024) further employs named nodes and MCTS to scale automated workflow generation. Our work builds on this direction by introducing mathematically grounded constraints into the MCTS loop, providing domain-aware pruning and guidance without changing the underlying interface.

SCULPT, building upon the foundation of representing workflows with code and implementing automated workflow optimization using the standard Monte Carlo Tree Search algorithm that includes Upper Confidence bounds applied to Trees (UCT) selection strategy, node expansion, simulation, and backpropagation, multiplicatively reweights the Upper Confidence Bound (UCB) with node compliance in exponential form, and smoothly updates compliance using constraint satisfaction scoring to suppress noise and unstable branches. It multiplicatively modulates confidence and simulation rewards through mathematical domain constraints, enhancing the efficiency and robustness of LLM mathematical reasoning through constraint-driven pruning and scoring.

Beyond workflow optimizers, a large body of

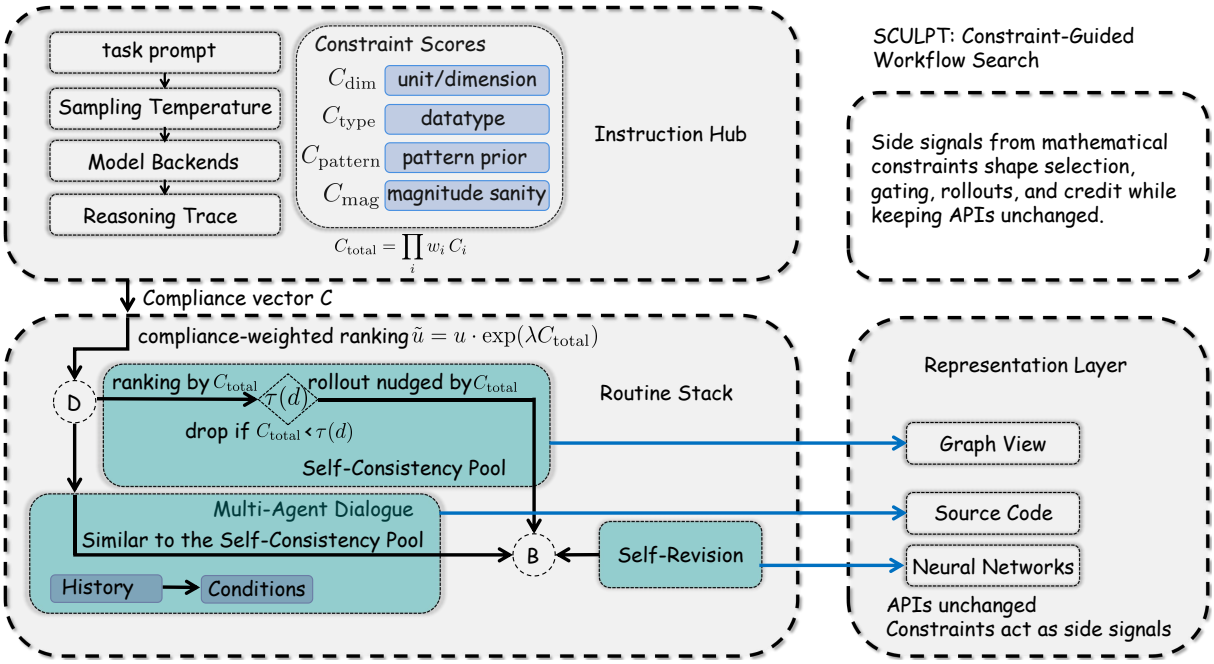

Figure 1: SCULPT overview with constraint guidance. Left: the Instruction Hub records the task setup and computes constraint scores C_{dim}, C_{type}, C_{pattern}, and C_{mag}, aggregated as $C_{\text{total}} = \prod_i w_i C_i$. Middle: routines (Self-Consistency Pool, Multi-Agent Dialogue, Self-Revision) are ranked and gated by compliance via $\tilde{u} = u \exp(\lambda C_{\text{total}})$ and a depth-dependent threshold $\tau(d)$; rollouts and credit updates are lightly shaped by C_{total}. Right: results are rendered as Graph View, Source Code, and Neural Networks; interfaces remain unchanged.

sampling-driven deliberation methods diversifies reasoning paths. The concrete baselines we evaluate are enumerated in Section 5 (Baselines). Our approach is orthogonal: SCULPT serves as a side signal that attaches constraint scores to selection, expansion, and credit updates, and thus can be plugged into those pipelines without changing their interfaces. In parallel, structure-aware controllers—e.g., symbolic or graph-based search (Zhou et al., 2024)—illustrate the benefit of injecting domain structure; SCULPT follows this spirit while keeping a lightweight implementation compatible with AFlow.

3 Methodology

3.1 Overview

We present SCULPT, a domain-specific constraint-guided Monte Carlo Tree Search (MCTS) framework designed to accelerate the convergence of automated workflow optimization for mathematical reasoning tasks. Unlike previous approaches that rely heavily on randomness and general-purpose heuristics, SCULPT integrates mathematical domain knowledge directly into the search process through targeted constraints that reflect the unique properties of mathematical problem-solving.

Our approach operates within the two-phase workflow of the existing AFlow framework: (1) an optimization phase where workflows are iteratively improved using an optimizer model, and (2) an execution phase where optimized workflows are evaluated using an executor model. SCULPT enhances this process by introducing mathematically-informed constraints during the optimization phase to guide the search more efficiently toward high-quality solutions.

An overview is given in Figure 1. The Instruction Hub computes constraint scores, the Routine Stack applies compliance-weighted selection and depth-aware gating, and the Representation Layer exposes views while keeping interfaces unchanged.

3.2 Mathematical Domain Constraints

We identify three key categories of mathematical domain constraints that guide our pruning strategy:

For completeness and to match the implementation, we summarize the aggregate compliance and the depth-aware pruning schedule used throughout the workflow search. Let the constraint index set be $\{\mathcal{D}, \mathcal{V}, \mathcal{U}, \mathcal{T}, \mathcal{P}, \mathcal{M}\}$ for depth, diversity, unit, type, pattern, and magnitude, respectively:

$$C_{\text{total}}(w) = \prod_{\mathcal{I} \in \{\mathcal{D}, \mathcal{V}, \mathcal{U}, \mathcal{T}, \mathcal{P}, \mathcal{M}\}} w_{\mathcal{I}} \, C_{\mathcal{I}}(w). \quad (1)$$

The expansion gate follows a linear schedule with a lower bound:

$$\tau(d) = \max\big(\tau_{\min},\, \tau_0 - k\, d\big), \tag{2}$$

where $(\tau_0, \tau_{\min}, k)$ are configuration constants and d denotes search depth. The mapping between symbolic indices and the figure's full-word subscripts is given in the Overview paragraph above.

3.2.1 Structural Complexity Constraints

Mathematical solutions typically follow principles of elegance and parsimony. We therefore adopt two complementary complexity controls. The solution depth constraint penalizes overly deep paths:

$$C_{\text{depth}}(w) = \max\left(0,\, 1 - \alpha\, \log\frac{d_{\max}}{d(w)}\right) \tag{3}$$

where $d(w)$ is the depth of workflow w, $d_{\max}$ is a domain-specific maximum, and α is a scaling parameter. Complementarily, the operator diversity constraint encourages an appropriate spread of operators based on problem type:

$$C_{\text{div}}(w) = \begin{cases} 1 - \beta\left|1 - \frac{H}{H_t}\right|, & H \le 2H_t, \\ 0.5 - \gamma\,(H - 2H_t), & \text{otherwise}, \end{cases} \tag{4}$$

where $H = H(w)$ denotes the operator-usage entropy and H_t the target entropy; β, γ are scaling factors.

3.2.2 Mathematical Consistency Constraints

Mathematical reasoning must maintain internal consistency. We enforce two checks. Dimensional consistency is a hard constraint,

$$C_{\text{dim}}(w) = \mathbf{1}\{\text{dim-ok}(w)\}, \tag{5}$$

where $\text{dim-ok}(w)$ holds iff arithmetic operations respect dimensions (addition/subtraction require the same dimensions; multiplication adds and division subtracts dimensions), trigonometric functions are dimensionless, and calculus operations change order by ± 1 (integration increases by 1, differentiation decreases by 1). Type compatibility is measured by the mean fraction of operations with admissible inputs,

$$C_{\text{type}}(w) = \frac{1}{|O_w|} \sum_{o \in O_w} \mathbf{1}\{\text{type-ok}(o)\}, \tag{6}$$

covering numeric versus boolean operators and special cases such as square root (nonnegative input) and logarithm (positive input).

3.2.3 Domain-Specific Pattern Constraints

Different mathematical subdomains exhibit characteristic solution patterns. We first classify the problem and compute a pattern prior

$$C_{\text{pattern}}(w, p) = \text{sim}\big(w, P_{\text{class}(p)}\big) \tag{7}$$

where $\text{class}(p)$ identifies the problem class, $P_{\text{class}(p)}$ is the set of effective patterns for that class, and sim is a weighted cosine similarity augmented with a Jaccard term over operator types. Workflow features include operation counts, depth, branching factor, operation-type distributions, and structural motifs (e.g., loops and conditionals). To suppress pathological scalings, we also regularize solution magnitudes:

$$C_{\text{mag}}(w) = \max\{0,\, 1 - \delta\, \tfrac{M^*(w) - M_{\text{th}}}{M_{\text{th}}}\} \tag{8}$$

where $M^*(w) = \max_{v \in \text{val}(w)} |v|$, M_{th} is a reasonable magnitude threshold, and δ scales the penalty.

3.3 Constraint-Guided MCTS

We integrate constraints across the search loop as follows. During selection, confidence estimates are softly modulated by a node's compliance score so that type/units-consistent branches are preferred without forbidding exploration. During expansion, candidates whose compliance falls below a depth-aware threshold are discarded, which reduces branching on implausible operations early while relaxing constraints deeper in the search. During simulation, rollouts are lightly biased toward compliant actions to stabilize estimates without incurring heavy overhead. During backpropagation, rewards are propagated with compliance-aware weighting, improving credit assignment and suppressing noisy trajectories. This design improves sample efficiency with minimal engineering and keeps interfaces unchanged. As illustrated in Figure 1, the depth-aware threshold $\tau(d)$ governs early pruning and the compliance scores shape selection and credit assignment while preserving the original interfaces. Concretely, we form a compliance-shaped ranking score $\tilde{u} = u \cdot \exp(\lambda C_{\text{total}})$, where u is the base selection utility (e.g., UCB), $C_{\text{total}} = \prod_i w_i C_i$ is the aggregate compliance from Section 3, and $\lambda > 0$ controls shaping strength.

3.4 Integration with Optimization-Execution Workflow

SCULPT operates within the two-phase workflow of AFlow:

1. **Optimization Phase**: SCULPT guides the search for effective workflows by applying mathematical constraints during MCTS, focusing the optimizer model on mathematically sound modifications.

2. **Experience Accumulation**: Successful and unsuccessful workflow modifications are stored along with their constraint satisfaction scores, building a domain-specific knowledge base.

3. **Execution Phase**: Optimized workflows are evaluated using the executor model, with results feeding back to refine constraint weights.

This integration ensures that SCULPT's mathematical constraints guide the search process without disrupting the overall workflow, leading to more efficient convergence while maintaining compatibility with the existing framework.

4 Design Details of SCULPT

This section provides an engineering-oriented view of SCULPT: what is new, how the procedure flows, how Figure 1 ties components together, and which concrete settings we adopt to ensure reproducibility.

4.1 What is new and how it fits the workflow

SCULPT contributes a lightweight, domain-aware guidance signal that is injected as side information into the standard optimization–execution workflow used by AFlow. Rather than changing any optimizer/executor interface, SCULPT computes a compliance vector $\mathbf{C} = (C_{\text{dim}}, C_{\text{type}}, C_{\text{pattern}}, C_{\text{mag}}, C_{\text{depth}}, C_{\text{div}})$ and aggregates it into $C_{\text{total}} = \prod_i w_i C_i$. This aggregate modulates four MCTS touch points that mirror the routine stack in AFlow: selection, expansion, simulation, and backpropagation. In effect, SCULPT acts as a soft, mathematically grounded prior that preserves exploration while filtering implausible branches.

4.2 End-to-end flow and Figure reference

Figure 1 ("Constraint-augmented overview") illustrates how the Instruction Hub computes constraint scores that travel with proposals into the Routine Stack and shape ranking, gating, rollout, and credit update. The process begins with initialization, where dataset splits are loaded following Section 5, the random seed is fixed to **42** across Python, NumPy, and framework components, and the cohort-specific prompt templates, sampling temperatures, and model backends are frozen. An experience store is prepared to record proposals, scores, and outcomes. During each optimization round, the optimizer proposes routine edits or workflow candidates; for every candidate, SCULPT computes the compliance vector $\mathbf{C}$ and its aggregate C_{total}. Candidates whose compliance falls below the depth-aware threshold $\tau(d)$ are filtered early, while the rest are prioritized by the compliance-shaped score $\tilde{u} = u \cdot \exp(\lambda C_{\text{total}})$ during selection, and the subsequent rollouts are gently nudged toward higher-compliance actions without sacrificing stochasticity. Backpropagation multiplies simulation returns by C_{total} when updating node statistics, and the experience store logs (workflow, $\mathbf{C}$, C_{total}, reward, and metadata). Constraint weights w_i are updated conservatively using the correlation-based rule referenced in Section 5. At the end of each round, the top workflows are evaluated by the designated executor for the cohort, and the outcomes feed back into both the optimization loop and the adaptive weighting. The search proceeds for **15** optimization rounds (Section 5); optional early stopping is triggered if validation accuracy plateaus within a pre-set window, and all methods use matched optimizer–executor settings to enable apples-to-apples comparisons.

4.3 Experimental controls and splits

We follow the dataset protocol in Section 5: GSM8K, MATH, and GSM-Hard (v2) are split into validation and test in a $1:4$ ratio; for MATH we use difficulty level 5 problems from four representative categories. Unless otherwise specified by the cohort, all preprocessing, tokenization (implicit in API usage), and formatting rules are kept identical across methods. Random seed is fixed to **42** throughout data loading, shuffle order, and any internal sampling components exposed by the framework to enhance reproducibility.

4.4 Training-time procedure

Although SCULPT does not train new model parameters, it *does* perform an iterative search procedure whose settings matter. Each round collects a batch of candidate routines from the optimizer under the same temperature and stop conditions used by baselines; proposals are evaluated in a rate-limit-aware queue, and caching avoids duplicate executions. The threshold $\tau(d)$ decreases with

depth so that early stages emphasize dimensional and type feasibility while deeper nodes relax gating to preserve diversity. Operator-usage entropy contributes to C_{div}, and when entropy falls below a target range the sampler is encouraged to surface under-explored operator families in subsequent batches. To stabilize rollouts, we bound the rollout horizon and the retry logic; when API calls are nondeterministic, retries are capped and the first successful response is recorded to keep budget accounting consistent. Constraint weights w_i are updated with a small learning rate and an exponential moving average of correlations to prevent oscillations. Determinism is promoted by propagating seed 42 to Python and NumPy, fixing request-level parameters (e.g., temperature schedules) per cohort, and recording full metadata with hashed prompts where the backend exposes non-determinism.

4.5 Interaction with other sections

This section operationalizes the ideas introduced in Section 3 by specifying how constraint scores shape the four MCTS stages and how they are integrated into AFlow without interface changes. The cohort definitions, round budgets, and evaluation details correspond to Section 5. Figure 1 should be read alongside these settings for a complete picture of execution.

5 Experiment

5.1 Datasets

In our experiments, we utilize three publicly available mathematical datasets: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and GSM-Hard (Gao et al., 2023) as our benchmark datasets. Following established practices in workflow optimization research Saad-Falcon et al. (2024), we split our benchmark datasets into validation and test sets in a 1:4 ratio. For the MATH(Hendrycks et al., 2021) dataset, we follow the approach outlined in (Hong et al., 2024), selecting problems of difficulty level 5 from four typical categories (combinatorics and probability, number theory, prealgebra, and precalculus).

5.2 Baselines

We compare the workflows generated by our method on the executor against the following baselines: Chain-of-Thought (Wei et al., 2022), Self Consistency CoT (5 answers) (Wang et al., 2022), MultiPersona Debate (Wang et al., 2023), Self-Refine (max 3 iteration rounds) (Madaan et al., 2023), MedPrompt (3 answers and 5 votes) (Nori et al., 2023), ADAS(Hu et al., 2024),and IO (direct LLM invocation). Given that our design relies on Monte Carlo Tree Search (MCTS), these baselines are selected from those compared by Zhang et al. (2024), including both automated workflow optimization methods and manually designed approaches, to provide a clear contrast.

5.3 Evaluation Metrics

For the mathematical datasets GSM8K, MATH, and GSM-Hard (v2 split), we use accuracy (%) as the primary evaluation metric. We also collect the computational costs incurred during execution to compare the performance-cost trade-offs across different methods, providing an intuitive comparison of both performance and cost-effectiveness.

5.4 Experimental Setup

We evaluate two experimental cohorts to disentangle the effect of executors and to ensure apples-to-apples comparisons.

Cohort A (GPT-4o-mini executor). We follow the AFlow baseline configuration and use Claude-3.5-sonnet as the optimizer and GPT-4o-mini-0718 as the executor. Datasets include GSM8K and MATH. For budget considerations, we focus on these two datasets and do not evaluate GSM-Hard in this cohort. SCULPT runs for 15 optimization rounds, while AFlow uses 20 and ADAS uses 30.

Cohort B (GPT-5-mini executor). To assess transfer and stronger executors, we adopt GPT-5-mini both as the optimizer and the executor for SCULPT. Baselines are reproduced or taken from matched settings where the executor is GPT-5-mini. Due to practical constraints in open-source reproduction, not all baselines are included under GPT-5-mini; we report those that are publicly available or reproducible with matched settings. Data splits are aligned with Cohort A unless otherwise noted for GSM-Hard (v2). We report each cohort independently and compute cross-executor averages of relative gains (with respect to each baseline) for summary.

5.5 Implementation Details

We implement constraint-aware selection, thresholded expansion, light rollout shaping, and compliance-weighted backpropagation. Depth and diversity constraints are instantiated with simple, dataset-agnostic formulas; dimensional and type

checks verify operator compatibility; pattern priors are computed from workflow features with weighted similarity. Operator usage histograms are extracted recursively from workflow code. Constraint weights are tuned by small-grid search on validation and updated conservatively using correlation trends to avoid instability. All components integrate with AFlow without modifying optimizer/executor interfaces.

5.6 Adaptive Constraint Weighting

To reflect the varying importance of constraints across tasks, we adopt an adaptive weighting rule that is updated during optimization on the validation split:

$$w_i \leftarrow w_i \cdot \exp(\eta \cdot \mathrm{corr}(C_i, \mathcal{A})) \tag{9}$$

where $\mathcal{A}$ denotes the task accuracy used as the performance metric and $\mathrm{corr}(\cdot, \cdot)$ is the running correlation; η controls the adaptation rate. This smooth update emphasizes helpful constraints while avoiding instability.

5.7 Results and Analysis

Figure 2 provides an overall comparison across methods under the two executor settings (left: Cohort A with GPT-4o-mini; right: Cohort B with GPT-5-mini). We then report detailed numbers in the cohort-specific tables below. We compute relative improvements with respect to each baseline and also discuss compute efficiency.

Results and analysis. Under Cohort A (GPT-4o-mini executor), SCULPT attains 93.8 on GSM8K and 57.2 on MATH, exceeding AFlow (93.5/56.2) while using 25% fewer optimization rounds (15 vs. 20). Relative to prompting ensembles such as CoT-SC, improvements are modest on GSM8K but more pronounced on MATH (+6.8 points over CoT-SC), suggesting that constraint guidance especially helps harder, type/units-sensitive items. Under Cohort B (GPT-5-mini executor), SCULPT reaches 97.6 on GSM8K, 75.8 on MATH, and 84.4 on GSM-Hard, outperforming AFlow across datasets with a +1.4 average gain (85.9 vs. 84.5). Gains over ADAS are larger, consistent with pruning reducing ineffective exploration without the cost of extensive rollouts.

Accuracy improvements concentrate where dimensional sanity and datatype compatibility are pivotal: divisor conventions, annulus/ring area computations (outer minus inner), and trigonometric ratios in right triangles. In these cases, constraints down-weight branches that violate units or admissible input types, allowing search to allocate budget to feasible programs earlier. The effect is reflected by higher MATH and GSM-Hard scores—datasets where such local checks are more often decisive—while GSM8K, already near a ceiling, still benefits from small but consistent gains.

Compute-wise, SCULPT reaches stronger results with fewer rounds (15) than AFlow (20) and ADAS (30), indicating better sample efficiency. Because constraints act as side signals rather than hard replacements for scoring, diversity is preserved: the Routine Stack continues to explore multiple routines, but selection pressure favors compliant branches. Overall, the quantitative results support that lightweight, domain-aware pruning yields reliable accuracy gains under matched budgets and executors.

Table 1: Cohort A (GPT-4o-mini executor): accuracy (%). Optimizer is Claude-3.5-sonnet; executor is GPT-4o-mini-0718. Datasets: GSM8K and MATH. SCULPT runs 15 optimization rounds; AFlow and ADAS use 20 and 30, respectively; other baselines follow standard settings. Numbers align with the left panel of Figure 2.

Method	GSM8K	MATH
IO (GPT-4o-mini)	92.7	48.6
CoT (Wei et al., 2022)	92.4	48.8
CoT SC (5-shot) (Wang et al., 2022)	92.7	50.4
MedPrompt (Nori et al., 2023)	90.0	50.0
MultiPersona (Wang et al., 2023)	92.8	50.8
Self Refine (Madaan et al., 2023)	89.6	46.1
ADAS (Hu et al., 2024)	90.8	35.4
AFlow (Zhang et al., 2024)	93.5	56.2
Ours	**93.8**	**57.2**

Table 2: Cohort B (GPT-5-mini executor): accuracy (%). Optimizer and executor are GPT-5-mini. Datasets: MATH, GSM8K, and GSM-Hard (v2). SCULPT uses 15 optimization rounds; AFlow and ADAS use 20 and 30, respectively. Numbers align with the right panel of Figure 2.

Method	MATH	GSM8K	GSM-Hard	Average
IO	68.9	93.8	76.7	79.8
Self-Refine	65.4	91.7	73.8	77.0
MultiPersona	71.5	97.4	80.6	83.2
ADAS	60.2	92.9	65.8	73.0
AFlow	74.7	95.6	83.2	84.5
Ours†	**75.8**	**97.6**	**84.4**	**85.9**

Overall trends are consistent across executors: SCULPT outperforms workflow and prompting baselines while using fewer optimization rounds.

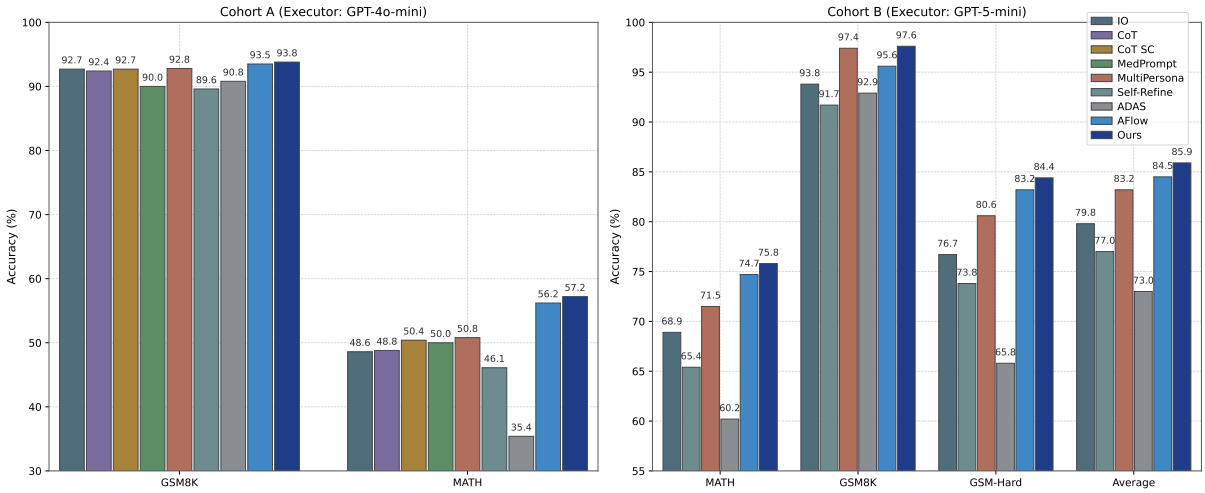

Figure 2: Overall comparison across executors. Left panel: Cohort A uses Claude-3.5-sonnet as optimizer and GPT-4o-mini-0718 as executor, evaluated on GSM8K and MATH. Right panel: Cohort B adopts GPT-5-mini as both optimizer and executor, evaluated on MATH, GSM8K, and GSM-Hard (v2 split). Ours (SCULPT) runs with 15 optimization rounds, while AFlow and ADAS use 20 and 30 rounds, respectively; other prompting baselines follow their standard settings. See Tables 1 and 2 for exact values.

Qualitatively, divisor conventions, ring/annulus areas, and right–triangle ratios particularly benefit from dimensional, type, and pattern checks.

Compute efficiency. SCULPT uses **15** optimization rounds versus **20** for AFlow (**25%** fewer iterations) and **30** for ADAS (**50%** fewer iterations), yet attains higher accuracy. Compared with prompting-only baselines (e.g., CoT, CoT-SC), SCULPT improves accuracy under matched executor and data, indicating that constraint signals provide complementary guidance to search beyond sampling-based diversity. Compared with methods that rely on heavy ensembling, SCULPT achieves improvements with lightweight pruning and verification, preserving efficiency.

Limitations. Improvements on MATH are moderate, reflecting the difficulty of geometry-heavy items where symbolic construction is required. Our integration does not introduce new geometry solvers; instead, we rely on constraint checks and selective verification. Extending the pattern base and incorporating stronger geometric primitives are promising directions.

5.7.1 Cohort A: GPT-4o-mini executor

Table 1 lists the exact numbers corresponding to the left panel of Figure 2. Under the matched optimizer–executor setting (Claude-3.5-sonnet/GPT-4o-mini) and identical data splits, SCULPT achieves the best accuracy on both GSM8K and MATH while using only 15 optimization rounds.

5.7.2 Cohort B: GPT-5-mini executor

Table 2 reports the right panel of Figure 2. Using GPT-5-mini as both optimizer and executor, SCULPT maintains its advantage on all datasets, including GSM-Hard (v2 split).

6 Conclusion

This paper presented SCULPT, a constraint-guided MCTS that attaches lightweight mathematical checks to workflow search and uses an aggregate compliance score to shape selection, expansion, rollouts, and backpropagation while keeping interfaces unchanged. Under matched settings on GSM8K, MATH, and GSM-Hard, SCULPT achieves higher accuracy than prompting and workflow baselines with fewer optimization rounds, with the largest benefits in categories sensitive to unit/type discipline and localized structural priors. These findings indicate that modest, domain-aware constraints can turn stochastic exploration into ordered search without trading off diversity or efficiency, offering a practical path to more reliable agentic reasoning systems.

Limitations

Our study focuses on mathematical reasoning; generalization to other domains will require task-specific constraint libraries, though the integration remains interface-preserving. The current checks

are intentionally lightweight and may miss deeper semantic relations, especially in geometry; expanding the pattern base and optional links to symbolic tools would help. Finally, compliance scoring adds small overhead and its weighting is tuned conservatively; more principled adaptation and profiling are left for future work.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, and 1 others. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, and Keshav Santhanam. 2024. Sri vardhamanan a, saiful haq, ashutosh sharma, thomas t. joshi, hanna moazam, heather miller, matei zaharia, and christopher potts. dspy: Compiling declarative language model calls into stateof-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, 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.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, and 1 others. 2023. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452*.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, and 1 others. 2024. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*.

Xiaoyu Tan, Bin Li, Xihe Qiu, Chao Qu, Wei Chu, Yinghui Xu, and Yuan Qi. 2025. Meta-agent-workflow: Streamlining tool usage in llms through workflow construction, retrieval, and refinement. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 458–467.

Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Halevy. 2023. Verifai: verified generative ai. *arXiv preprint arXiv:2307.02796*.

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*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*.

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.

Chaojia Yu, Zihan Cheng, Hanwen Cui, Yishuo Gao, Zexu Luo, Yijin Wang, Hangbin Zheng, and Yong Zhao. 2025. A survey on agent workflow–status and future. In *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 770–781. IEEE.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2024.

Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, and 1 others. 2024. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.