CARVE: CREDIT ASSIGNMENT VIA ROUTED VERIFICATION OF STATE-CONDITIONED BEHAVIORAL UNITS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) provides outcome-level supervision for reasoning models, but many reasoning and decision-making tasks expose intermediate verification signals during generation. Effectively exploiting these signals requires identifying appropriate intermediate credit units, matching behaviors with informative verification, and translating heterogeneous feedback into localized policy updates. We propose CARVE, a framework for credit assignment via routed verification of state-conditioned behavioral units. CARVE organizes trajectory tokens into structurally segmented and behaviorally annotated units, each associated with its preceding reasoning, execution, or interaction state. It then applies behavior-specific verification to assess local validity and continuation-based verification to estimate how selected units change the state-dependent downstream utility. Both evidence sources are converted into separate behavior-level credits and localized to the corresponding tokens, while the final task reward remains the primary optimization signal. CARVE outperforms the strongest evaluated baselines across two model families using only 400 curated training problems per domain. It achieves gains of up to 7.00 percentage points in mathematical reasoning, 3.66 in code generation and 2.50 in multi-turn tool use. These results demonstrate its effectiveness on challenging tasks with multi-step dependencies and state-dependent decisions.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has emerged as an effective paradigm for improving the reasoning and decision-making capabilities of large language models (Guo et al., 2025; Shao et al., 2024; Yu et al., 2026). Most RLVR methods optimize responses using a task-level scalar reward for final correctness or task success. Such outcome-level supervision is natural when feedback is available only at the end of a trajectory. Yet many tasks also expose verifiable information during generation, often associated with identifiable intermediate behaviors. Intermediate calculations can be checked symbolically or numerically in mathematical reasoning (Cobbe et al., 2021; Lightman et al., 2024; Uesato et al., 2022); code can be executed against test cases (Le et al., 2022b); and tool calls can be assessed through their selection, arguments and execution outcomes (Schick et al., 2023). These signals offer opportunities to supervise decisions more precisely than a final outcome alone.

Turning intermediate verification into policy credit remains challenging (Setlur et al., 2024). In multi-step reasoning and sequential interaction, the value of a behavior depends on the preceding state and the decisions that follow it. A locally valid step need not improve the eventual outcome, while an unsuccessful trajectory may contain steps worth reinforcing. Verification signals also differ in what they assess: an arithmetic check, a code test, and a tool execution result cannot be treated as interchangeable measures of progress (Le et al., 2022a; Qian et al., 2026). Assigning useful intermediate credit therefore requires identifying the behavior to be evaluated, applying an appropriate check, and relating the resulting evidence to downstream task success.

The unit of credit is central to this problem. Response-level credit cannot distinguish intermediate behaviors, whereas token-level or coarse segment-level credit may split tokens implementing one operation or group operations with different roles. We use behavioral units as an intermediate granularity: each unit covers a structurally identified span, is annotated with its functional role, and is associated with its preceding state. Such units can represent planning, derivation, calculation, code

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

Table 1: Comparison of representative reinforcement learning methods along progressively richer forms of credit assignment and verification. ✓ = supported; △ = partially supported; × = not explicitly supported in the original formulation.

Method	Verif. Outcome Reward	Interm. Credit	Struct. Credit Loc.	Behav.-spec. Verif.	Unified Tasks/Verif.
GRPO (Shao et al., 2024)	✓	×	×	×	×
VinePPO (Kazemnejad et al., 2024)	✓	✓	△	×	×
SPO (Guo et al., 2026)	✓	✓	✓	×	×
ToolRL (Qian et al., 2026)	✓	✓	△	✓	×
rStar-Math (Guan et al., 2025)	✓	✓	△	✓	×
CARVE (ours)	✓	✓	✓	✓	✓

implementation and tool invocation (Schick et al., 2023). Their roles determine which local checks apply, while their preceding states provide the context needed to evaluate subsequent effects.

Different units require different evidence. Where explicit criteria exist, direct verification assesses local properties through symbolic or numerical checks, code execution, or tool-specific constraints (Chen et al., 2024; Yang et al., 2023). For selected units, Monte Carlo continuations (Kazemnejad et al., 2024) estimate changes in task-success probability between states before and after each unit. These forms of evidence answer related but distinct questions: whether a behavior satisfies an applicable local criterion and whether it is associated with progress toward task completion.

Prior work addresses parts of this problem from different directions. Outcome-based methods such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024) optimize response-level rewards. VinePPO (Kazemnejad et al., 2024) and SPO (Guo et al., 2026) provided intermediate-state values and segment-level advantages. Task-specific methods additionally exploit process or search-derived signals in mathematics, as in rStar-Math (Guan et al., 2025), and execution- or tool-specific rewards (Qian et al., 2026). However, outcome-level rewards provide limited resolution for intermediate credit, while finer-grained value estimation does not by itself identify which verification signals are appropriate for each behavioral role. Task-specific feedback provides useful local checks, but their scope and semantics vary across behaviors and domains, and local validity alone does not establish downstream utility. These limitations motivate a framework that jointly aligns state-conditioned credit units, applicable verification, and localized policy updates, combining evidence of local validity with estimates of contribution to final task success.

We propose CARVE, a framework for credit assignment via routed verification of state-conditioned behavioral units, to make this connection. CARVE decomposes each trajectory into behavioral units, annotates their functional roles, and associates each unit with its preceding intermediate state. It then selects locally applicable verification according to behavioral role, task domain, and available verification resources, while evaluating selected units through Monte Carlo continuations. Direct verification captures local validity, whereas continuation-based evaluation estimates state-dependent downstream utility. CARVE converts these two evidence sources into complementary behavior-level credit channels and localizes them only to the corresponding policy-generated tokens. The resulting local credit is combined with the task-level advantage, allowing behaviors with supported positive continuation evidence to receive positive updates even when the trajectory fails. This enables CARVE to distinguish useful intermediate progress from final outcomes in tasks with dependencies across reasoning steps or tool interactions. **Our main contributions are as follows:**

- We formulate state-conditioned behavioral units through structural segmentation and role annotation, linking each unit to its preceding state and aligning verification with policy credit.
- We propose CARVE, combining routed local verification with continuation-based evaluation to assess local validity and downstream utility. Confidence- and outcome-aware credits are localized to corresponding tokens, enabling useful behaviors in failed trajectories to be reinforced.
- Experiments with only 400 curated training problems per domain support CARVE's effectiveness on tasks requiring state-dependent reasoning and multi-step planning. On Qwen2.5-7B-Instruct, it outperforms the strongest evaluated baselines by 3.3, 3.0, and 2.5 percentage points on AIME, APPS, and BFCL-MT, respectively, with consistent gains on Llama-3.1-8B-Instruct.

2 Related Work

Reinforcement Learning for Reasoning. Reinforcement learning is widely used to improve LLM reasoning. PPO (Schulman et al., 2017) optimizes policies using advantage estimates from a learned value function, while GRPO (Shao et al., 2024) estimates relative advantages from grouped responses without an explicit critic. In reasoning tasks, these methods commonly use outcome-level verifiable rewards reflecting final correctness or task success. Such supervision provides a reliable task objective but limited information about the contribution of individual intermediate behaviors.

Fine-Grained Credit Assignment. Recent work has explored credit assignment beyond response-level rewards. VinePPO (Kazemnejad et al., 2024) estimates intermediate-state values using rollout-based evaluation, while SPO (Guo et al., 2026) assigns segment-level advantages for more localized policy updates. These methods improve the granularity of credit estimation and application. CARVE addresses a complementary aspect by organizing trajectories into behaviorally coherent units, associating each with its preceding state, and applying behavior-specific verification so that localized credit aligns with the functional role of each behavior.

Intermediate Verification and Task-Specific Feedback. Prior work explores domain-specific intermediate supervision. For mathematical reasoning, Math-Shepherd (Wang et al., 2024) learns process rewards from automatic step-level labels, while rStar-Math (Guan et al., 2025) uses search-derived process supervision. For tool use, ToolRL (Qian et al., 2026) uses fine-grained rewards for tool selection and argument correctness. CARVE routes behavioral units to applicable verifiers, combines local-validity evidence with continuation-based estimates of downstream utility, and localizes credit to the corresponding tokens.

3 Methodology

We first formalize the problem setting and behavioral representation, then introduce behavior decomposition, verification routing, and credit construction.

3.1 Problem Setup

We represent a generated response through its intermediate states and behavioral units, using intermediate evidence to derive localized token-level credit alongside the final task reward. Let $x \sim \mathcal{D}$ denote an input prompt and $y = (y_1, \ldots, y_T) \sim \pi_\theta(\cdot \mid x)$ a response generated by the policy π_θ. In standard reinforcement learning with verifiable rewards, the response is evaluated by a task-level verifier $R_{\text{task}}(x, y)$, which provides a scalar signal reflecting final correctness or task success. Beyond this response-level outcome, we define the intermediate state at generation step t as

$$s_t = (x, y_{<t}, e_t),$$

where $y_{<t}$ denotes the generated prefix and e_t represents optional task-dependent state, such as program execution results or tool-interaction history.

CARVE organizes a response into a sequence of behavioral units

$$\mathcal{B}(y) = (B_1, \ldots, B_K).$$

Each unit is represented as

$$B_k = (I_k, z_k, s_{l_k}), \qquad I_k = [l_k, r_k],$$

where I_k specifies the token span, z_k denotes its behavioral role, and s_{l_k} is the intermediate state preceding the behavior. The behavioral units define the scope of intermediate credit assignment, while the associated state preserves the context in which each behavior is generated.

At a high level, CARVE associates each behavioral unit with intermediate evidence, transforms this evidence into behavior-level credit, and localizes the resulting credit to the corresponding tokens. The overall process can be summarized as

$$B_k \xrightarrow{\text{evidence acquisition}} \mathcal{E}_k \xrightarrow{\text{credit transformation}} c_k \xrightarrow{\text{localization}} \{c_t\}_{t \in I_k},$$

where $\mathcal{E}_k$ denotes the intermediate evidence associated with behavioral unit B_k, c_k denotes the resulting behavior-level credit, and c_t denotes the localized token-level credit.

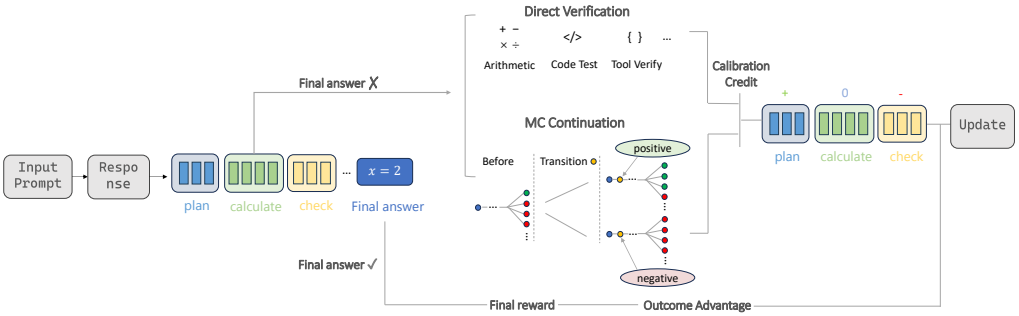


Figure 1: Overview of CARVE.

3.2 Behavioral Decomposition and Annotation

Behavioral Decomposition. A model response typically contains multiple intermediate operations with distinct functional roles, such as planning, derivation, calculation, verification, code implementation, or tool invocation. CARVE organizes the generated trajectory into contiguous behavioral units whose boundaries follow the structural organization of these operations.

Formally, let d_x denote the task domain associated with input x, and let η denote observable structural information provided by the task format or interaction protocol. A decomposition operator $\mathcal{G}$ produces an ordered sequence of token spans

$$\mathcal{G}(y; d_x, \eta) = (I_1, \ldots, I_K), \qquad I_k = [l_k, r_k].$$

In our implementation, $\mathcal{G}$ is deterministic and identifies segment boundaries using explicit trajectory structure, including paragraph and reasoning-step boundaries, code blocks, and tool-interaction protocol boundaries.

Behavioral Annotation. CARVE identifies the functional role of each segmented span using its content, task-domain information, and structural context. Formally, the behavioral annotation operator $\mathcal{S}$ assigns each span a functional behavior type:

$$z_k = \mathcal{S}\left(y_{I_k}; d_x, \eta_k\right),$$

where η_k denotes the structural metadata associated with the k-th span. The annotation operator combines lexical cues, task-domain information, and explicit reasoning, code, or tool-interaction structure to assign a discrete behavioral role, such as calculation, derivation, verification, planning, code implementation, or tool invocation.

The operator $\mathcal{G}$ determines where a behavioral unit begins and ends, while $\mathcal{S}$ determines the functional type of the resulting span. Each annotated span is then associated with its preceding intermediate state, $B_k = (I_k, z_k, s_{l_k})$, which records the context in which the behavior was generated.

The resulting behavioral units define the structural scope of intermediate evidence and credit assignment. Their behavioral roles determine the locally applicable verification criteria, while the associated intermediate states provide the context used in subsequent verification and continuation-based evaluation.

3.3 Behavior-Specific Verification and Continuation Evaluation

Verifier Routing. CARVE selects locally applicable verification programs for each behavioral unit based on its functional role, task domain, and available verification resources.

The behavioral role of each unit determines which local properties can be explicitly verified. CARVE maintains a library of task-specific verification programs $\mathcal{V} = \{\nu_1, \ldots, \nu_M\}$, covering arithmetic validity, algebraic equivalence, answer consistency, code execution, and tool-interaction constraints. For each behavioral unit B_k, the verification router selects applicable verifiers based on its behavioral role, task domain, and available resources:

$$(\mathcal{A}_k, \mathbf{w}_k) = \rho\left(z_k, d_x, \chi_k; \mathcal{V}\right),$$

where $\mathcal{A}_k \subseteq \mathcal{V}$ is the selected verifier set, $\mathbf{w}_k$ contains their configured weights, and χ_k collects routing conditions such as segment eligibility and the availability of task-specific references, tests, or tool specifications.

Local Verification. The selected verifiers evaluate locally checkable properties of each behavioral unit and produce verification signals with associated confidence scores when available.

The router uses a predefined role-to-verifier mapping followed by task- and resource-dependent filtering. A behavioral unit may therefore be associated with multiple local verifiers, or with an empty verifier set when no applicable local check is available. For each selected verifier $\nu_j \in \mathcal{A}_k$, CARVE obtains

$$(q^{\mathrm{ver}}_{k,j}, \gamma_{k,j}) = \nu_j(B_k, s_{l_k}, \chi_k),$$

where $q^{\mathrm{ver}}_{k,j}$ denotes the local verification signal and $\gamma_{k,j}$ denotes the verifier confidence when provided. These signals characterize locally checkable properties of the behavioral unit under the corresponding verification criterion.

Monte Carlo Evaluation. CARVE estimates the state-dependent contribution of selected behavioral units by comparing continuation-based task-success probabilities before and after each unit.

CARVE additionally evaluates selected behavioral units through Monte Carlo continuations. The units included in this evaluation are determined by the MC anchor protocol according to the configured behavioral eligibility, task difficulty, and continuation budget. Let s_{r_k+1} denote the state after generating B_k. Starting from the prefix ending at r_k, CARVE samples N continuations from the current policy and estimates the subsequent task-success probability as

$$\widehat{p}^{\,\mathrm{post}}_k = \frac{1}{N}\sum_{n=1}^{N} R_{\mathrm{task}}\left(x,\, y_{\le r_k} \oplus \widetilde{y}^{(n)}_{>r_k}\right).$$

Similarly, starting from the prefix preceding the behavioral unit, CARVE estimates

$$\widehat{p}^{\,\mathrm{pre}}_k = \frac{1}{N}\sum_{n=1}^{N} R_{\mathrm{task}}\left(x,\, y_{<l_k} \oplus \widetilde{y}^{(n)}_{\ge l_k}\right).$$

The estimated change in continuation value is $\widehat{\Delta}_k = \widehat{p}^{\,\mathrm{post}}_k - \widehat{p}^{\,\mathrm{pre}}_k$, and we denote its estimated standard error by $\widehat{\sigma}_k$. The pair $(\widehat{\Delta}_k, \widehat{\sigma}_k)$ provides state-dependent evidence about the behavioral unit's contribution to subsequent task success.

For each behavioral unit, CARVE therefore collects intermediate evidence of the form

$$\mathcal{E}_k = \left(\{q^{\mathrm{ver}}_{k,j}, \gamma_{k,j}\}_{\nu_j \in \mathcal{A}_k}, \widehat{\Delta}_k, \widehat{\sigma}_k\right),$$

with unavailable components omitted when the local verifier or MC evaluation is not applicable.

3.4 Localized Credit Assignment and Policy Optimization

CARVE transforms the intermediate evidence associated with each behavioral unit into two complementary credit signals: direct-verification credit and continuation-based credit. The former summarizes locally checkable evidence, while the latter reflects the estimated effect of the behavior on subsequent task success.

Direct-Verification Credit. For the k-th behavioral unit of response i, let $\mathcal{A}^{\mathrm{cert}}_{i,k}$ denote the routed certification verifiers. We aggregate their outputs according to the configured verifier weights and confidence scores:

$$u^{\mathrm{ver}}_{i,k} = \sum_{\nu_j \in \mathcal{A}^{\mathrm{cert}}_{i,k}} w_{i,k,j}\gamma_{i,k,j} q^{\mathrm{ver}}_{i,k,j}.$$

Positive certification evidence is further regulated by trajectory outcome and continuation evidence. When the MC-support gate is enabled, a failed response retains positive certification credit only if the corresponding behavioral unit has an available MC estimate satisfying $\widehat{\Delta}_{i,k} > \delta_{\mathrm{gate}}$. Detector penalties are processed separately by the configured penalty rules. After weighting, gating, and magnitude constraints, we denote the resulting direct-verification credit by $c^{\mathrm{ver}}_{i,k}$.

Continuation-Based Credit. The continuation channel converts the estimated value difference into signed credit while accounting for Monte Carlo estimation uncertainty. Given the estimated continuation-value change $\widehat{\Delta}_{i,k}$ and its standard error $\widehat{\sigma}_{i,k}$, we first compute

$$b_{i,k} = \left[\left|\widehat{\Delta}_{i,k}\right| - z_{\mathrm{mc}}\widehat{\sigma}_{i,k}\right]_+$$

where $[v]_+ = \max(v, 0)$. The initial MC credit is then

$$\widetilde{c}^{\mathrm{mc}}_{i,k} = \operatorname{sgn}\left(\widehat{\Delta}_{i,k}\right) \min\left(C_{\mathrm{mc}}, b_{i,k}\right) \mathbf{1}\left[b_{i,k} \geq \max\left(\tau_{\mathrm{abs}}, \tau_{\mathrm{rel}}\widehat{\sigma}_{i,k}\right)\right].$$

This transformation preserves the direction of continuation-value change, discounts uncertainty, and assigns zero credit below the confidence threshold.

CARVE further conditions continuation-based credit on the final task outcome. For successful responses, supported positive MC credit may be retained, while negative MC credit is suppressed under the training configuration used in our experiments. For unsuccessful responses, selected positive MC credit is scaled and capped to reinforce useful intermediate behaviors, whereas selected negative MC credit penalizes behaviors that reduce estimated continuation success. After retaining the selected positive and negative anchors and applying outcome-dependent scaling and magnitude constraints, we denote the resulting training credit by $c^{\mathrm{mc}}_{i,k}$. Unselected, unavailable, or inconclusive MC evidence contributes zero credit. The detailed anchor-selection and outcome-dependent adjustment rules are provided in Appendix A.

Both credit channels are localized to the tokens of their corresponding behavioral unit. For $a \in \{\mathrm{ver}, \mathrm{mc}\}$, we define $C^a_{i,t} = M_{i,t} \sum_{k=1}^{K_i} \mathbf{1}\{t \in I_{i,k}\} \frac{c^a_{i,k}}{s_a}$, where $M_{i,t}$ indicates a policy-generated token eligible for optimization and $s_a > 0$ is a fixed channel scale. Tokens without corresponding intermediate evidence receive zero credit from that channel. Behavioral credit is applied uniformly as a per-token coefficient to the policy-generated tokens within each span, without equalizing aggregate weight across behavioral units.

Credit-Guided Policy Optimization. CARVE combines the task-level advantage with localized behavioral credits, applies a sign-preserving constraint to active MC supervision, and optimizes the resulting token-level advantages using a clipped policy objective. Given a group of G responses sampled for the same prompt, let R_i denote the task-level outcome reward of response i. The task-level advantage is computed only from outcome rewards:

$$A^{\mathrm{task}}_i = \frac{R_i - \mu_R}{\sigma_R + \epsilon},$$

where μ_R and σ_R are computed within the response group. Intermediate credits enter only through the localized channels. We first combine the three signals as

$$\widetilde{A}_{i,t} = M_{i,t}\lambda_{\mathrm{task}}A^{\mathrm{task}}_i + \lambda_{\mathrm{ver}}C^{\mathrm{ver}}_{i,t} + \lambda_{\mathrm{mc}}C^{\mathrm{mc}}_{i,t}.$$

CARVE then applies a sign-preserving token-advantage constraint to active MC supervision:

$$A^{\mathrm{CARVE}}_{i,t} = \begin{cases} \max\left(\widetilde{A}_{i,t}, \lambda_{\mathrm{mc}}C^{\mathrm{mc}}_{i,t}\right), & C^{\mathrm{mc}}_{i,t} > 0, \\ \min\left(\widetilde{A}_{i,t}, \lambda_{\mathrm{mc}}C^{\mathrm{mc}}_{i,t}, 0\right), & C^{\mathrm{mc}}_{i,t} < 0, \\ \widetilde{A}_{i,t}, & C^{\mathrm{mc}}_{i,t} = 0. \end{cases}$$

This constraint allows a behavior with supported positive continuation credit to receive a positive local update even when the sampled response is unsuccessful, while preserving negative supervision for behaviors with harmful continuation evidence.

The resulting token-specific advantage is incorporated into clipped policy optimization. Let $s_{i,t}$ denote the recorded generation state and define

$$r_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid s_{i,t})}{\pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid s_{i,t})}, \qquad \bar{r}_{i,t} = \operatorname{clip}\left(r_{i,t}(\theta), 1 - \epsilon_c, 1 + \epsilon_c\right), \qquad \text{with } Z = \sum_{i,t} M_{i,t}.$$

The CARVE objective is

$$\mathcal{L}_{\text{CARVE}} = -\frac{1}{Z}\sum_{i=1}^{G}\sum_{t} M_{i,t}\min\left(r_{i,t}(\theta)A_{i,t}^{\text{CARVE}}, \bar{r}_{i,t}A_{i,t}^{\text{CARVE}}\right) + \beta\mathcal{L}_{\text{KL}},$$

where $\mathcal{L}_{\text{KL}}$ regularizes the policy toward a frozen reference model. The objective is normalized over eligible generated tokens across the response group.

4 Experiments

4.1 Experimental Settings

Models and Baselines. We evaluate CARVE with Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct, training a separate policy for each task domain. We compare against the instruction-tuned base model, GRPO (Shao et al., 2024), VinePPO (Kazemnejad et al., 2024), and SPO (Guo et al., 2026), with rStar-Math (Guan et al., 2025) and ToolRL (Qian et al., 2026) included for their applicable domains. Trainable methods use the same backbone, training data, and evaluation protocol.

Data and Evaluation. To construct the training sets, we sample 10 responses per candidate example from the corresponding base model and retain examples with empirical success rates between 0.25 and 0.75, focusing training on nontrivial but learnable instances. For mathematics, we train on 400 problems consisting of 300 MATH problems (Hendrycks et al., 2021b) and 100 historical AIME problems, and evaluate on a held-out MATH subset selected using the same filtering procedure as the training set, as well as AIME 2024–2025 (Zhang & Math-AI, 2024; 2025). For code generation, we train on 400 APPS (Hendrycks et al., 2021a) and APPS+ (Dou et al., 2024) problems and evaluate on held-out 200 APPS and Humaneval. For tool use, we train on 400 BFCL-derived examples and evaluate on 200 held-out BFCLv3 (Patil et al., 2025) tasks, both consisting of multi-turn tool-calling tasks. All evaluation examples are disjoint from and unseen during RL training. All evaluation examples are disjoint from and unseen during RL training. We report Greedy@1 for mathematics, Pass@1 for code generation, and Task Success@1 for tool use.

Training. All methods use rank-16 LoRA with $G = 8$ responses per prompt, AdamW with learning rate 1×10^{-6}, and PPO clipping threshold 0.2. CARVE uses 32, 16, and 16 MC continuations per selected anchor for mathematics, code generation, and tool use, respectively. Additional optimization and credit-processing hyperparameters are given in the Appendix C.

4.2 Main Results

Table 2 summarizes the results across mathematical reasoning, code generation, and multi-turn tool use. On Qwen2.5-7B-Instruct, CARVE achieves the best performance on all five benchmarks, reaching 61.00 on MATH, 16.67 on AIME 2024–2025, 16.00 on APPS, 73.17 on HumanEval, and 11.50 on the multi-turn subset of BFCLv3 (BFCL-MT). Compared with the strongest baseline on each benchmark, these represent gains of 7.00, 3.34, 3.00, 1.83, and 2.50 percentage points, respectively. The gains extend across both general credit-assignment methods and domain-specific approaches. CARVE outperforms VinePPO and SPO across all evaluated domains, as well as rStar-Math on mathematical reasoning and ToolRL on multi-turn tool use. These comparisons support the effectiveness of organizing heterogeneous verification signals around behavioral units and translating them into localized policy credit.

The gains on AIME, APPS, and BFCL-MT highlight CARVE's effectiveness on challenging tasks that require multi-step reasoning or sequential interaction. In these settings, the utility of an intermediate behavior depends on the preceding state and its consequences for subsequent reasoning or actions, rather than local correctness alone. These benchmarks cover complementary forms of such dependencies: successive mathematical deductions in AIME, algorithm design and implementation in APPS, and tool-call sequences with evolving environment states in BFCL-MT. CARVE combines behavior-specific local checks with MC estimates of how selected behaviors change the probability of eventual success. Localizing this evidence to the corresponding tokens allows useful steps in failed trajectories to be reinforced and steps with negative continuation evidence to be penalized. This distinction between local validity and downstream utility helps explain CARVE's gains across these forms of state-dependent problem solving.

The results on Llama-3.1-8B-Instruct show a consistent pattern: CARVE improves over the baseline methods across the evaluated domains. This consistency across two model families provides further evidence for the applicability of the framework.

Table 2: Main results across mathematical reasoning, code generation, and tool use. We report Greedy@1 for MATH and AIME 2024–2025, Pass@1 for APPS and HumanEval, and Task Success@1 for the multi-turn subset of BFCLv3 (BFCL-MT). Best results are shown in **bold** and second-best results are underlined.

Method	Math		Code		Tool Use
	MATH	AIME24&25	APPS	HumanEval	BFCL-MT
Qwen2.5-7B-Instruct					
Base	52.00	10.00	12.50	69.51	8.00
GRPO	51.00	10.00	13.00	70.73	7.50
VinePPO	50.00	13.33	11.50	71.34	8.00
SPO	54.00	11.67	13.00	71.34	8.50
rStar-Math	54.00	13.33	—	—	—
ToolRL	—	—	—	—	9.00
CARVE	**61.00**	**16.67**	**16.00**	**73.17**	**11.50**
Llama-3.1-8B-Instruct					
Base	25.00	5.00	7.50	62.80	2.00
GRPO	26.00	5.00	8.00	63.41	2.50
VinePPO	25.50	6.67	7.50	64.63	3.50
SPO	28.00	5.00	8.50	63.41	3.00
rStar-Math	27.50	6.67	—	—	—
ToolRL	—	—	—	—	3.50
CARVE	**29.50**	**8.33**	**10.00**	**68.29**	**5.00**

Generalization. After training on APPS and APPS+, CARVE achieves 73.17 on HumanEval with Qwen2.5-7B-Instruct, without further fine-tuning or adaptation, outperforming the base model by 3.66 points, GRPO by 2.44 points, and both VinePPO and SPO by 1.83 points. Together with its improvement on held-out APPS, this supports cross-benchmark transfer in code generation.

For mathematics, after training on MATH and historical AIME problems, CARVE achieves 16.67 on AIME 2024–2025, compared with 10.00 for both the base model and GRPO. This supports generalization to held-out competition problems from later years, reflecting cross-year transfer within the AIME benchmark family and complementing the cross-benchmark transfer in code generation.

4.3 Ablation Study

We evaluate five ablations on Qwen2.5-7B-Instruct, as shown in Table 3. Random Segments replaces behavioral boundaries with random token segments and retains only MC-based intermediate credit. w/o Continuation Evaluation uses direct verification without MC evaluation. w/o Direct Verification retains behavioral segmentation and MC-based intermediate credit while removing the direct-verification channel. w/o Credit Localization (Broadcast) assigns the mean final token advantage to all eligible generated tokens within each response. Outcome-only RL removes all intermediate credit. All variants retain the same task-level outcome rewards.

Full CARVE achieves the best performance across all three benchmarks. Removing direct verification reduces performance by 3.34, 3.00, and 2.50 percentage points on AIME, APPS, and BFCL-MT, respectively, indicating that local verification complements continuation-based credit. This variant still outperforms Random Segments across all three benchmarks, supporting behavioral segmentation under MC-only intermediate supervision.

Removing continuation evaluation reduces performance from 16.67 to 8.33 on AIME, from 16.00 to 13.50 on APPS, and from 11.50 to 7.50 on BFCL-MT. Its lower scores than outcome-only RL on AIME and BFCL-MT suggest that local verification alone does not consistently improve learning, highlighting the importance of assessing downstream utility.

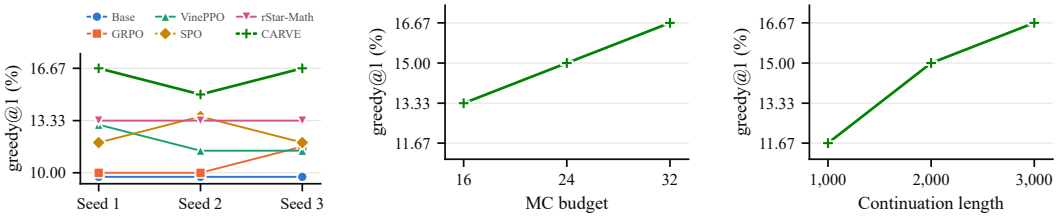
Broadcasting advantages reduces performance by 5.00, 3.50, and 3.00 percentage points on AIME, APPS, and BFCL-MT, respectively, supporting the localization of credit to corresponding behavioral spans. Compared with outcome-only RL, full CARVE improves performance by 6.67, 4.00, and 3.50 percentage points, respectively, supporting the benefit of supplementing final-outcome supervision with localized intermediate credit.

Table 3: Ablation study of CARVE.

Variant	AIME	APPS	BFCL-MT
Random Segments	11.67	12.50	8.50
w/o Cont. Eval.	8.33	13.50	7.50
w/o Direct Verif.	13.33	13.00	9.00
w/o Credit Loc.	11.67	12.50	8.50
Outcome-only RL	10.00	12.00	8.00
CARVE	**16.67**	**16.00**	**11.50**

Table 4: Math: time and performance.

Method	Time (h)	MATH	AIME
GRPO	13.0	51.50	10.00
VinePPO	12.7	50.00	13.33
SPO	12.4	54.00	11.67
rStar-Math	13.5	54.00	13.33
CARVE	13.1	**61.00**	**16.67**



(a) Training stability. (b) MC budget sensitivity. (c) Continuation length sensitivity.

Figure 2: Analysis of training stability and sensitivity to Monte Carlo evaluation configurations.

4.4 Performance and Training Cost

CARVE introduces additional computation through behavior-level verification and Monte Carlo continuation evaluation. To examine whether its performance gains can be explained simply by increased training time, we compare the performance and actual wall-clock training time of CARVE and representative baselines on mathematical reasoning.

VinePPO and SPO use MC continuations for intermediate credit estimation, while rStar-Math uses MCTS-based trajectory generation and process-level scoring. Under our implementation and hardware setting, their training times are comparable to CARVE's. We reuse their checkpoints and results from Table 2 and report actual training times. Since GRPO has lower per-update overhead, we extend its training to approximately match CARVE's wall-clock time. All methods share the same backbone, training data, hardware allocation, and evaluation protocol.

As shown in Table 4, CARVE maintains its performance advantage at comparable training cost. Extending GRPO training yields no substantial improvement over Table 2, suggesting that longer outcome-only training provides limited additional benefit.

4.5 Additional Analysis

All analyses in this section are conducted on the math reasoning task with Qwen2.5-7B-Instruct.

Training stability. CARVE outperforms all baselines across three math training seeds (Figure 2a), suggesting its gains are not seed-specific.

MC budget sensitivity. With other settings fixed, increasing continuations per prefix from 16 to 32 improves Greedy@1 from 13.33% to 16.67% (Figure 2b), consistent with more reliable credit estimation.

Continuation length sensitivity. Extending continuations from 1,000 to 3,000 tokens improves Greedy@1 from 11.67% to 16.67% (Figure 2c), suggesting that more complete reasoning aids downstream utility estimation, with diminishing performance gains.

5 Conclusion and Outlook

We introduced CARVE, a credit-assignment framework for RLVR that combines behavior-specific verification with continuation-based evaluation and localizes intermediate credit to corresponding tokens. Across mathematical reasoning, code generation, and multi-turn tool use, CARVE outperforms outcome-level and fine-grained baselines using small, curated training sets. Its gains on AIME, APPS, and BFCL-MT support its applicability to complex tasks involving state-dependent decisions, multi-step dependencies, and delayed outcomes. Future work will explore more flexible behavioral decomposition and verifier routing, and extend CARVE to longer-horizon agentic tasks.

AI USE STATEMENT

In this work, we used generative AI tools for language translation. We have not used generative AI tools for developing research methodology, designing experiments, implementing methods, interpreting experimental results, formulating or modifying research hypotheses, or deriving theoretical results, and the remaining tasks requiring disclosure are not applicable to this work. Additionally, we used generative AI tools for language editing to improve the clarity and readability of the manuscript. We have reviewed all AI-assisted work and verified the final text for correctness and consistency with the authors' intended meaning. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a credit-assignment framework for language model training and evaluates it on mathematical reasoning, code generation, and simulated tool-use tasks. The study does not involve human subjects, personal data, or deployment in real-world environments. We therefore identify no direct ethical concerns arising from the experimental procedures. Its broader societal implications are primarily indirect, through advances in language model training and reasoning capabilities.

REPRODUCIBILITY STATEMENT

The methodology is described in Section 3, including behavioral decomposition, verifier routing, and localized credit assignment. Section 4.1 specifies the models, baselines, datasets, and evaluation protocols. Appendix B provides pseudocode for credit construction and the training procedure, and Appendix C reports the experimental hardware, implementation details, and training hyperparameters.

REFERENCES

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, et al. Stepcoder: improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4571–4585, 2024.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *Advances in Neural Information Processing Systems*, 38:114399–114431, 2026.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms. *arXiv preprint arXiv:2410.01679*, 2024.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022a. URL https://arxiv.org/abs/2207.01780.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022b.

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

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning, 2024. URL https://arxiv.org/abs/2410.08146.

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

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

A Detailed Credit Construction

This section details the construction of the two intermediate credit channels used by CARVE and their integration into localized policy updates. Experiment-specific optimization schedules and hyperparameter settings are reported separately in the experimental settings. Throughout this section, $R_i \in \{0, 1\}$ denotes the final task-success indicator of response i. Direct verification provides local evidence about explicitly checkable properties of a behavioral unit, while Monte Carlo continuations estimate its effect on subsequent task success. The two credit channels follow distinct processing rules, with continuation evidence additionally gating positive certification credit in unsuccessful responses. Both channels are subsequently localized to their corresponding token spans.

A.1 Direct-Verification Credit

Let $\mathcal{A}_{i,k}^{\mathrm{cert}}$ denote the set of certification verifiers routed to behavioral unit $B_{i,k}$. Each verifier $\nu_j \in \mathcal{A}_{i,k}^{\mathrm{cert}}$ returns a signed verification score $q_{i,k,j}^{\mathrm{ver}}$ and, when available, a confidence value $\gamma_{i,k,j}$. For rule-based verifiers, $\gamma_{i,k,j}$ is assigned by fixed scoring rules and defaults to one when unspecified. CARVE aggregates the certification evidence as

$$u_{i,k}^{\mathrm{ver}} = \sum_{\nu_j \in \mathcal{A}_{i,k}^{\mathrm{cert}}} w_{i,k,j} \gamma_{i,k,j} q_{i,k,j}^{\mathrm{ver}},$$

where $w_{i,k,j}$ is the configured weight associated with verifier ν_j.

We separate the positive and negative components as

$$u_{i,k}^{+} = \max\left(u_{i,k}^{\mathrm{ver}}, 0\right), \qquad u_{i,k}^{-} = \min\left(u_{i,k}^{\mathrm{ver}}, 0\right).$$

Let

$$e_{i,k}^{\mathrm{elig}} \in \{0, 1\}$$

indicate whether $B_{i,k}$ is eligible for positive certification credit. Positive certification evidence is further regulated by the final trajectory outcome and, when enabled for the corresponding task domain, by MC support. We define

$$h_{i,k} = e_{i,k}^{\mathrm{elig}} \begin{cases} 1, & R_i = 1 \text{ or MC-support gating is disabled,} \\ \mathbf{1}\{\text{MC available}\}\, \mathbf{1}\left\{\widehat{\Delta}_{i,k} > \delta_{\mathrm{gate}}\right\}, & \text{otherwise.} \end{cases}$$

The support gate uses the raw continuation-value difference $\widehat{\Delta}_{i,k}$. It is therefore evaluated separately from the uncertainty filtering used to construct MC training credit below.

For unsuccessful responses, CARVE constrains the total retained positive certification evidence within the trajectory. Let

$$S_i^{+} = \sum_{k=1}^{K_i} h_{i,k} u_{i,k}^{+}.$$

The trajectory-level scaling factor is

$$a_i = \begin{cases} \min\left(1, \dfrac{C_{\mathrm{cert}}^{\mathrm{fail}}}{S_i^{+}}\right), & R_i = 0 \text{ and } S_i^{+} > 0, \\ 1, & \text{otherwise.} \end{cases}$$

In our implementation,

$$C_{\text{cert}}^{\text{fail}} = 0.15.$$

This budget is applied to the retained positive certification evidence before multiplication by the difficulty-dependent certification weight.

Violation detectors are processed separately from certification evidence. Let

$$d_{i,k} \leq 0$$

denote the weighted and confidence-adjusted detector penalty assigned to $B_{i,k}$ after the configured penalty filtering. Diagnostic-only checks contribute zero training credit.

The resulting direct-verification credit is

$$c_{i,k}^{\text{ver}} = \omega_i \left(u_{i,k}^{-} + a_i h_{i,k} u_{i,k}^{+} \right) + d_{i,k},$$

where ω_i denotes the configured difficulty-dependent certification weight.

A.2 Continuation-Based Credit

For each behavioral unit selected by the MC anchor protocol, CARVE estimates the change in continuation value

$$\widehat{\Delta}_{i,k} = \widehat{p}_{i,k}^{\text{post}} - \widehat{p}_{i,k}^{\text{pre}},$$

together with its estimated standard error

$$\widehat{\sigma}_{i,k}.$$

To account for finite-sample uncertainty, CARVE first computes

$$b_{i,k} = \left[\left| \widehat{\Delta}_{i,k} \right| - z_{\text{mc}} \widehat{\sigma}_{i,k} \right]_{+},$$

where

$$[v]_{+} = \max(v, 0),$$

and z_{mc} controls the uncertainty adjustment.

We further define an uncertainty-dependent credit threshold

$$\delta_{i,k}^{\text{mc}} = \max\left(\tau_{\text{abs}}, \tau_{\text{rel}} \widehat{\sigma}_{i,k} \right).$$

The initial signed MC credit is

$$\widetilde{c}_{i,k}^{\text{mc}} = \text{sgn}\left(\widehat{\Delta}_{i,k} \right) \min\left(C_{\text{mc}}, b_{i,k} \right) \mathbf{1}\left\{ b_{i,k} \geq \delta_{i,k}^{\text{mc}} \right\}.$$

Thus, estimates whose uncertainty-adjusted magnitude falls below the configured threshold contribute zero initial MC credit, while retained estimates preserve the sign of $\widehat{\Delta}_{i,k}$.

CARVE then selects the strongest positive and negative MC anchors within each response according to their initial signed credits. Let

$$\mathcal{K}_i^{\text{mc}} \subseteq \{1, \ldots, K_i\}$$

denote the set of retained MC anchors. The numbers of retained positive and negative anchors are specified by the training configuration.

The retained MC credits are subsequently adjusted according to the final task outcome:

$$c_{i,k}^{\text{mc}} = \mathbf{1}\left\{ k \in \mathcal{K}_i^{\text{mc}} \right\} \begin{cases} \alpha_{\text{succ}}^{+} \max\left(\widetilde{c}_{i,k}^{\text{mc}}, 0 \right), & R_i = 1, \\ \min\left(\alpha_{\text{fail}}^{+} \widetilde{c}_{i,k}^{\text{mc}}, C_{\text{fail}}^{+} \right), & R_i = 0 \text{ and } \widetilde{c}_{i,k}^{\text{mc}} > 0, \\ \widetilde{c}_{i,k}^{\text{mc}}, & R_i = 0 \text{ and } \widetilde{c}_{i,k}^{\text{mc}} \leq 0. \end{cases}$$

Here, α_{succ}^{+} and α_{fail}^{+} are the configured positive-credit scaling factors, and C_{fail}^{+} caps positive MC credit in unsuccessful responses.

For successful responses, supported positive MC credit may therefore be retained, while negative MC credit is suppressed. For unsuccessful responses, selected positive MC credit reinforces intermediate behaviors associated with increased continuation success, while selected negative MC credit penalizes behaviors associated with reduced continuation success. Unselected anchors, unavailable MC estimates, and estimates that do not pass the credit threshold receive

$$c_{i,k}^{\text{mc}} = 0.$$

A.3 TOKEN-LEVEL LOCALIZATION

Both credit channels are localized to policy-generated tokens belonging to their corresponding behavioral units. Let

$$M_{i,t} \in \{0, 1\}$$

indicate whether token t of response i is eligible for policy optimization. For

$$a \in \{\text{ver}, \text{mc}\},$$

we define

$$C^a_{i,t} = M_{i,t} \sum_{k=1}^{K_i} \mathbf{1}\{t \in I_{i,k}\} \frac{c^a_{i,k}}{s_a},$$

where $s_a > 0$ is the fixed scale associated with credit channel a. The implementation uses

$$s_{\text{ver}} = s_{\text{mc}} = 0.10.$$

Prompt tokens, padding tokens, external-observation tokens, and tokens without a resolved behavioral span receive no localized intermediate credit. Intermediate credits are not mean-centered across responses, so the absence of intermediate evidence remains neutral:

$$C^a_{i,t} = 0.$$

A.4 TASK-LEVEL ADVANTAGE AND CREDIT CONSOLIDATION

The global task channel is computed only from final task outcomes. Let

$$\overline{\mathbf{R}} = \mathcal{N}_{d_x}(\mathbf{R})$$

denote the task-reward vector after the configured domain-specific reward preprocessing. For G responses sampled for the same prompt, CARVE computes

$$A^{\text{task}}_i = \frac{\overline{R}_i - \mu_{\overline{R}}}{\sigma_{\overline{R}} + \epsilon},$$

where $\mu_{\overline{R}}$ and $\sigma_{\overline{R}}$ are computed within the response group. Intermediate verification and MC credits are excluded from this global task channel and enter optimization only through their localized token-level signals.

Before imposing the local sign constraint, CARVE combines the task-level and intermediate signals as

$$\widetilde{A}_{i,t} = M_{i,t}\lambda_{\text{task}} A^{\text{task}}_i + \lambda_{\text{ver}} C^{\text{ver}}_{i,t} + \lambda_{\text{mc}} C^{\text{mc}}_{i,t}.$$

For

$$\lambda_{\text{mc}} > 0,$$

CARVE preserves the direction of active continuation-based supervision through the token-level constraint

$$A^{\text{CARVE}}_{i,t} = \begin{cases} \max\left(\widetilde{A}_{i,t}, \lambda_{\text{mc}} C^{\text{mc}}_{i,t}\right), & C^{\text{mc}}_{i,t} > 0, \\ \min\left(\widetilde{A}_{i,t}, \lambda_{\text{mc}} C^{\text{mc}}_{i,t}, 0\right), & C^{\text{mc}}_{i,t} < 0, \\ \widetilde{A}_{i,t}, & C^{\text{mc}}_{i,t} = 0. \end{cases}$$

This constraint allows an intermediate behavior with supported positive continuation evidence to receive a positive local update even when the sampled response is unsuccessful. Active negative MC evidence analogously preserves a negative update direction for behaviors associated with reduced downstream success.

B TRAINING PSEUDOCODE

Algorithm 1 details the construction of the two intermediate credit channels, including uncertainty filtering, anchor retention, and outcome-dependent gates and caps. Algorithm 2 summarizes the complete CARVE training procedure, including behavioral decomposition, routed verification, continuation evaluation, and localized policy updates. Here, $\mathcal{H}$ collects the configured credit-processing parameters, and the resulting token advantages are held fixed during the policy updates within each rollout round.

Algorithm 1 CARVE Behavior-Level Credit Construction

function CONSTRUCTCREDITS($\mathcal{E}_i, R_i, \mathcal{H}$)
  Set $C_{\text{cert}}^{\text{fail}} = 0.15$; initialize $\widetilde{c}_{i,k}^{\text{mc}} = c_{i,k}^{\text{mc}} = 0$ for all k.
  For rule-based verifiers, use confidence $\gamma_{i,k,j} = 1$ when unspecified.
  for $k = 1, \ldots, K_i$ **do**
    Aggregate certification evidence: $u_{i,k}^{\text{ver}} \leftarrow \sum_{\nu_j \in \mathcal{A}_{i,k}^{\text{cert}}} w_{i,k,j} \gamma_{i,k,j} q_{i,k,j}^{\text{ver}}$.
    Split $u_{i,k}^{+} \leftarrow \max(u_{i,k}^{\text{ver}}, 0)$ and $u_{i,k}^{-} \leftarrow \min(u_{i,k}^{\text{ver}}, 0)$.
    Initialize $h_{i,k} \leftarrow e_{i,k}^{\text{elig}}$.
    if $R_i = 0$ and MC-support gating is enabled **then**
      Set $h_{i,k} \leftarrow 0$ unless MC is available and the raw difference satisfies $\widehat{\Delta}_{i,k} > \delta_{\text{gate}}$.
    end if
    Obtain weighted, confidence-adjusted detector penalty $d_{i,k} \leq 0$ after configured filtering; diagnostic-only checks contribute zero.
    if MC evidence is available **then**
      Discount uncertainty: $b_{i,k} \leftarrow \max(|\widehat{\Delta}_{i,k}| - z_{\text{mc}} \widehat{\sigma}_{i,k}, 0)$.
      if $b_{i,k} \geq \max(\tau_{\text{abs}}, \tau_{\text{rel}} \widehat{\sigma}_{i,k})$ **then**
        Set $\widetilde{c}_{i,k}^{\text{mc}} \leftarrow \text{sgn}(\widehat{\Delta}_{i,k}) \min(C_{\text{mc}}, b_{i,k})$.
      end if
    end if
  end for
  Finalize direct-verification credit
  Compute $S_i^{+} \leftarrow \sum_k h_{i,k} u_{i,k}^{+}$; initialize $a_i \leftarrow 1$.
  if $R_i = 0$ and $S_i^{+} > 0$ **then**
    Set $a_i \leftarrow \min(1, C_{\text{cert}}^{\text{fail}} / S_i^{+})$.
  end if
  For all k, assign $c_{i,k}^{\text{ver}} \leftarrow \omega_i (u_{i,k}^{-} + a_i h_{i,k} u_{i,k}^{+}) + d_{i,k}$.
  Finalize continuation-based credit
  Form $\mathcal{K}_i^{\text{mc}}$ by retaining up to K^{+} largest positive and K^{-} most negative initial MC credits within response i.
  for $k \in \mathcal{K}_i^{\text{mc}}$ **do**
    if $R_i = 1$ **then**
      $c_{i,k}^{\text{mc}} \leftarrow \alpha_{\text{succ}}^{+} \max(\widetilde{c}_{i,k}^{\text{mc}}, 0)$.
    else if $\widetilde{c}_{i,k}^{\text{mc}} > 0$ **then**
      $c_{i,k}^{\text{mc}} \leftarrow \min(\alpha_{\text{fail}}^{+} \widetilde{c}_{i,k}^{\text{mc}}, C_{\text{fail}}^{+})$.
    else
      $c_{i,k}^{\text{mc}} \leftarrow \widetilde{c}_{i,k}^{\text{mc}}$.
    end if
  end for
  return $(\mathbf{c}_i^{\text{ver}}, \mathbf{c}_i^{\text{mc}})$
end function

C IMPLEMENTATION DETAILS AND HYPERPARAMETERS

Experiment Hardware. Experiments were conducted on a server equipped with four NVIDIA H100 GPUs and an Intel Xeon Platinum 8468 CPU. GPU resources were used for policy generation and optimization, while CPU resources supported symbolic verification, unit-test execution, and tool-environment simulation.

Optimization and sampling. We use AdamW with a learning rate of 1×10^{-6}, zero weight decay, and a five-step linear warm-up. For each training prompt, we sample eight responses with temperature 1.0 and top-$p = 1.0$. Policy updates use a clipping threshold of 0.2 and gradient-norm clipping at 1.0. A frozen reference policy provides KL regularization. The number of MC continuations per evaluated prefix is fixed within each task domain.

Detail hyperparameters. Table 5 summarizes the core optimization and generation hyperparameters used across the three domains. Unless otherwise specified, shared hyperparameters are held fixed across mathematics, code generation, and tool use, while domain-specific settings mainly concern KL regularization, generation limits, and Monte Carlo continuation budgets.

Table 5: Core training, credit-assignment, and evaluation hyperparameters. Generation limits are measured in tokens; for tool use, they apply to each assistant turn. Anchor counts are per-response upper bounds.

Hyperparameter	Mathematics	Code generation	Tool use
Optimizer	AdamW	AdamW	AdamW
Learning rate	1×10^{-6}	1×10^{-6}	1×10^{-6}
Weight decay	0	0	0
Warm-up steps	5	5	5
LoRA rank / α	16 / 32	16 / 32	16 / 32
LoRA dropout	0	0	0
Responses per prompt	8	8	8
Rollout temperature / top-p	1.0 / 1.0	1.0 / 1.0	1.0 / 1.0
Policy-ratio clipping threshold	0.2	0.2	0.2
Maximum gradient norm	1.0	1.0	1.0
KL coefficient	0.005	0.01	0.005
Rollout generation limit	3,000	768	512 per turn
MC continuations per evaluated prefix	32	16	16
MC generation limit	3,000	512	512 per turn
MC temperature / top-p	0.7 / 0.95	0.7 / 0.95	0.7 / 0.95
Maximum evaluated MC anchors per response	1	3	1
Maximum retained positive / negative anchors	1 / 1	1 / 1	1 / 1
MC uncertainty coefficient z_{mc}	1.0	1.0	1.0
Absolute MC threshold τ_{abs}	0.03	0.03	0.03
Relative MC threshold coefficient τ_{rel}	1.0	1.0	1.0
MC credit cap C_{mc}	0.10	0.15	0.10
Positive MC scale on failure $\alpha^{+}_{\mathrm{fail}}$	0.35	0.50	0.50
Positive MC cap on failure C^{+}_{fail}	0.035	0.05	0.05
MC-support gate threshold δ_{gate}	0.05	0.05	0.05
Failed-response certification budget $C^{\mathrm{fail}}_{\mathrm{cert}}$	0.15	0.15	0.15
Credit scales $s_{\mathrm{ver}}/s_{\mathrm{mc}}$	0.10 / 0.10	0.10 / 0.10	0.10 / 0.10
Weight of the final task outcome λ_{task}	1.0	1.0	1.0
Weight of direct behavioral verification λ_{ver}	0.5	0.5	0.5
Weight of the MC continuation-value change λ_{mc}	0.7	0.7	0.7
Evaluation temperature / top-p	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0
Evaluation generation limit	2,048	512 per turn	512 per turn

Behavioral Segmentation and Verifier Routing. Mathematical trajectories are split at reasoning steps and paragraph boundaries. Step cues such as `Step` or `Then` identify derivations, which use arithmetic validity (weight 0.4), algebraic equivalence (0.3), and reference-answer anchoring (0.3). Arithmetic cues identify computations, which use arithmetic validity (1.0). `Verify` or `Check` cues identify verification segments, which use final-answer substitution (0.5) and answer anchoring (0.5). Arithmetic and algebraic scores lie in $[0, 1]$; substitution returns 0 or 1; answer anchoring returns 1 for a match, -0.3 for an explicit contradiction, and 0 otherwise.

Strict code segmentation follows the ordered `Plan:`, `Implementation:`, `Check:`, and `<answer>` fields. Planning and checking have no default direct verifier. Implementation uses syntax, entry-point, empty-function, and expression-only defect detectors, each with weight 1.0 and a score of 0 or -0.5. The final program also receives a public-test pass fraction in $[0, 1]$ with weight 1.0. A failed strict-protocol parse produces no process-level MC anchors.

Tool trajectories are segmented at individual call boundaries. Invalid tool names or arguments incur a penalty in $[-0.5, 0]$ with weight 1.0, whereas schema validity is diagnostic only and has weight 0. Cross-call redundancy and execution-error checks run at trajectory scope, each with weight 1.0.

Each active direct verifier contributes its route weight multiplied by its raw score and confidence. If a required answer, test, or tool specification is unavailable, the corresponding positive check contributes zero, while applicable negative detectors remain. Unresolved segments receive no localized credit. For a literal `medium` routing label, positive verifier weights are capped at 0.3; penalty weights are unchanged.

MC Anchor Selection. MC uses the task's difficulty label when one is supplied in the manifest. In calibrated probes, success rate $p \geq 0.75$ is labeled `too_easy`, $0.30 \leq p < 0.75$ is `learnable_medium`, and $0 < p < 0.30$ is `learnable_hard`. A zero-success task receives a hard label only when its partial progress passes the configured magnitude and variability thresholds; otherwise it is labeled `out_of_capability`. Without a manifest label, difficulty combines task prior, rollout uncertainty, evaluator disagreement, and trajectory complexity.

MC first filters segments by the domain's allowed difficulties and behavior types. Among eligible segments, those selected by direct routing take priority; candidates are then ranked by importance, with earlier segments breaking ties, and truncated to the configured per-response budget. For tool use, routed calls are used when available; otherwise parsed calls are considered in trajectory order. Tool MC evaluates eligible successful and failed trajectories alike. Its candidate limit applies per trajectory, not per rollout group.

MC Standard Error. Let q_i^- and q_i^+ be continuation scores before and after a behavior, respectively. The estimated value change is $\widehat{\Delta} = \bar{q}^+ - \bar{q}^-$. For N shared-seed paired samples, define $d_i = q_i^+ - q_i^-$. The standard error is

$$\widehat{\sigma}_{\text{paired}} = \sqrt{\frac{\sum_{i=1}^{N}(d_i - \bar{d})^2}{N(N-1)}}.$$

When pairing is unavailable, the implementation uses the independent Bernoulli plug-in estimate

$$\widehat{\sigma}_{\text{ind}} = \sqrt{\frac{\bar{q}^-(1-\bar{q}^-) + \bar{q}^+(1-\bar{q}^+)}{N_{\text{eff}}}},$$

where $N_{\text{eff}} = \min(N_-, N_+)$, and N_- and N_+ are the observed pre- and post-behavior sample counts, respectively. For binary continuation rewards, this expression equals the standard independent Bernoulli plug-in estimate when the counts are equal and is conservative relative to the separate-denominator estimate when they differ. In the evaluated runs, mathematics and code generation used paired differences, whereas batched tool-use MC used the independent estimate.

D Qualitative Examples

We provide qualitative examples to illustrate how CARVE performs behavior-level credit assignment in practical scenarios. All examples are obtained from the Qwen2.5-7B-Instruct backbone and cover the three evaluated domains: mathematical reasoning, code generation, and tool use. Each case presents the complete generation trajectory and highlights how intermediate behaviors are evaluated and assigned credit through CARVE. These examples serve to illustrate the interaction between behavioral decomposition, verification signals, and localized credit assignment.

Mathematical Reasoning. Figure 3 shows a complete response from a model post-trained with CARVE to a mathematical reasoning problem. The model proceeds through problem reduction, candidate verification, and final computation, illustrating its ability to organize intermediate reasoning steps and arrive at the correct answer.

Code Generation. Figure 4 presents a CARVE-generated response to an APPS programming task. Its plan, implementation, check, and final-answer sections illustrate the semantic units used for behavior-level credit assignment.

Tool Use. Figure 5 shows a complete interaction of a model post-trained with CARVE on a BFCL vehicle-control task. The model uses tool feedback to correct failed actions and complete the requested operations, illustrating its ability to adjust its actions and recover from intermediate errors.

Algorithm 2 CARVE Training

Require: Training set $\mathcal{D}$, policy π_θ, frozen reference π_{ref}, group size G, continuation count N, update count U

Require: Segmentation $\mathcal{G}$, role annotation $\mathcal{S}$, router ρ, verifier library $\mathcal{V}$, and credit configuration $\mathcal{H}$

Set $s_{\text{ver}} = s_{\text{mc}} = 0.10$.
for each rollout round **do**
  Set $\theta_{\text{old}} \leftarrow \theta$; fix credit weights for this round.
  Sample $x \sim \mathcal{D}$ and $\{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$; record generation states and old log-probabilities.
  Construct optimization masks $M_{i,t}$, excluding prompt, padding, and external-observation tokens.
  for $i = 1, \ldots, G$ **do**
    Evaluate final task success $R_i \leftarrow R_{\text{task}}(x, y_i) \in \{0, 1\}$.
    Obtain behavioral spans $(I_{i,1}, \ldots, I_{i,K_i}) \leftarrow \mathcal{G}(y_i; d_x, \eta_i)$.
    Initialize evidence $\mathcal{E}_i$; mark all MC estimates as unavailable.
    for $k = 1, \ldots, K_i$ **do**
      Assign $z_{i,k} \leftarrow \mathcal{S}(y_{i,I_{i,k}}; d_x, \eta_{i,k})$ and form $B_{i,k} \leftarrow (I_{i,k}, z_{i,k}, s_{i,l_{i,k}})$.
      Route $(\mathcal{A}_{i,k}, \mathbf{w}_{i,k}) \leftarrow \rho(z_{i,k}, d_x, \chi_{i,k}; \mathcal{V})$.
      Evaluate each $\nu_j \in \mathcal{A}_{i,k}$: $(q^{\text{ver}}_{i,k,j}, \gamma_{i,k,j}) \leftarrow \nu_j(B_{i,k}, s_{i,l_{i,k}}, \chi_{i,k})$; store the outputs in $\mathcal{E}_i$.
    end for
    Select MC evaluation anchors $\mathcal{J}_i$ according to behavioral eligibility, task difficulty, valid boundaries, and the evaluation budget.
    for $k \in \mathcal{J}_i$ **do**
      From each boundary state $s^{\text{pre}}_{i,k} = s_{i,l_{i,k}}$ and $s^{\text{post}}_{i,k} = s_{i,r_{i,k}+1}$, sample N suffixes under $\pi_{\theta_{\text{old}}}$.
      Combine each suffix with its boundary prefix to obtain complete trajectories $Y^{b,n}_{i,k}$, $b \in \{\text{pre}, \text{post}\}$.
      Compute $\widehat{p}^{\,b}_{i,k} \leftarrow \frac{1}{N}\sum_{n=1}^{N} R_{\text{task}}(x, Y^{b,n}_{i,k})$, $b \in \{\text{pre}, \text{post}\}$.
      Set $\widehat{\Delta}_{i,k} \leftarrow \widehat{p}^{\,\text{post}}_{i,k} - \widehat{p}^{\,\text{pre}}_{i,k}$; estimate its standard error $\widehat{\sigma}_{i,k}$ with the domain-specific continuation evaluator.
      Store $(\widehat{\Delta}_{i,k}, \widehat{\sigma}_{i,k})$ and its availability in $\mathcal{E}_i$.
    end for
    $(\mathbf{c}^{\text{ver}}_i, \mathbf{c}^{\text{mc}}_i) \leftarrow$ ConstructCredits$(\mathcal{E}_i, R_i, \mathcal{H})$.
  end for
  Preprocess task rewards: $\overline{\mathbf{R}} \leftarrow \mathcal{N}_{d_x}((R_1, \ldots, R_G))$.
  Compute $A^{\text{task}}_i \leftarrow (\overline{R}_i - \mu_{\overline{R}})/(\sigma_{\overline{R}} + \epsilon)$ using group task rewards only.
  for each response i and token t **do**
    Localize both channels: $C^a_{i,t} \leftarrow M_{i,t}\sum_{k=1}^{K_i} \mathbf{1}\{t \in I_{i,k}\}\, c^a_{i,k}/s_a$, $a \in \{\text{ver}, \text{mc}\}$.
    Combine $\widetilde{A}_{i,t} \leftarrow M_{i,t}\lambda_{\text{task}}A^{\text{task}}_i + \lambda_{\text{ver}}C^{\text{ver}}_{i,t} + \lambda_{\text{mc}}C^{\text{mc}}_{i,t}$.
    Apply the MC sign constraint:

$$A^{\text{CARVE}}_{i,t} \leftarrow \begin{cases} \max\left(\widetilde{A}_{i,t}, \lambda_{\text{mc}}C^{\text{mc}}_{i,t}\right), & \lambda_{\text{mc}} > 0,\ C^{\text{mc}}_{i,t} > 0, \\ \min\left(\widetilde{A}_{i,t}, \lambda_{\text{mc}}C^{\text{mc}}_{i,t}, 0\right), & \lambda_{\text{mc}} > 0,\ C^{\text{mc}}_{i,t} < 0, \\ \widetilde{A}_{i,t}, & \text{otherwise.} \end{cases}$$

  end for
  Cache trajectories, states, masks, old/reference log-probabilities, and final advantages; hold A^{CARVE} fixed during the updates.
  Set $Z \leftarrow \sum_{i,t} M_{i,t}$.
  for $u = 1, \ldots, U$ **do**
    Compute $r_{i,t}(\theta) \leftarrow \pi_\theta(y_{i,t} \mid s_{i,t})/\pi_{\theta_{\text{old}}}(y_{i,t} \mid s_{i,t})$, and $\overline{r}_{i,t} \leftarrow \text{clip}(r_{i,t}(\theta), 1 - \epsilon_c, 1 + \epsilon_c)$.
    Compute the clipped objective with KL regularization toward π_{ref}:

$$\mathcal{L}_{\text{CARVE}} = -\frac{1}{Z}\sum_{i,t} M_{i,t} \min\left(r_{i,t}(\theta)A^{\text{CARVE}}_{i,t}, \overline{r}_{i,t}A^{\text{CARVE}}_{i,t}\right) + \beta\mathcal{L}_{\text{KL}}.$$

    Update trainable policy parameters θ using AdamW to minimize $\mathcal{L}_{\text{CARVE}}$.
  end for
end for
return π_θ

Problem

Find the sum of all positive integers n such that $n+2$ divides the product $3(n+3)(n^2+9)$.

CARVE

To solve the problem, we need to find the sum of all positive integers n such that $n+2$ divides the product $3(n+3)(n^2+9)$. First, let's denote $f(n) = 3(n+3)(n^2+9)$. We need to check when $n+2$ divides $f(n)$.

Step 1: Simplify the expression
We start by expanding $f(n)$:

$$f(n) = 3(n+3)(n^2+9) = 3(n+3)(n^2+9)$$

Step 2: Apply the Polynomial Division Theorem
By the polynomial division theorem, if $n+2$ divides $f(n)$, then $f(-2) \equiv 0 \pmod{n+2}$.

Step 3: Evaluate $f(-2)$
Substitute $n = -2$ into $f(n)$:

$$f(-2) = 3(-2+3)((-2)^2+9) = 3(1)(4+9) = 3 \cdot 13 = 39$$

Step 4: Determine the Divisors of 39
Since $n+2$ must divide 39, we list the divisors of 39:

$$39 = 3 \times 13 \quad \text{and} \quad 39 = (-3) \times (-13)$$

Thus, the divisors are $\pm 1, \pm 3, \pm 13, \pm 39$.

Step 5: Solve for n
We solve $n+2 = d$ for each divisor d:

$$n+2 = 1 \implies n = -1 \quad \text{(not positive)}$$
$$n+2 = 3 \implies n = 1 \quad \text{(positive)}$$
$$n+2 = 13 \implies n = 11 \quad \text{(positive)}$$
$$n+2 = 39 \implies n = 37 \quad \text{(positive)}$$
$$n+2 = -1 \implies n = -3 \quad \text{(not positive)}$$
$$n+2 = -3 \implies n = -5 \quad \text{(not positive)}$$
$$n+2 = -13 \implies n = -15 \quad \text{(not positive)}$$
$$n+2 = -39 \implies n = -41 \quad \text{(not positive)}$$

Step 6: Sum the Positive Solutions
The positive solutions are $n = 1, 11, 37$. Sum them:

$$1 + 11 + 37 = 49$$

Thus, the sum of all positive integers n such that $n+2$ divides the product $3(n+3)(n^2+9)$ is $\boxed{49}$. <answer>49</answer>

Problem

Mister B once received a gift: it was a book about aliens, which he started read immediately. This book had c pages.

At first day Mister B read v_0 pages, but after that he started to speed up. Every day, starting from the second, he read a pages more than on the previous day (at first day he read v_0 pages, at second — $v_0 + a$ pages, at third — $v_0 + 2a$ pages, and so on). But Mister B is just a human, so he physically wasn't able to read more than v_1 pages per day.

Also, to refresh his memory, every day, starting from the second, Mister B had to reread last l pages he read on the previous day. Mister B finished the book when he read the last page for the first time.

Help Mister B to calculate how many days he needed to finish the book.

-----Input-----

First and only line contains five space-separated integers: c, v_0, v_1, a and l ($1 \le c \le 1000$, $0 \le l < v_0 \le v_1 \le 1000$, $0 \le a \le 1000$) — the length of the book in pages, the initial reading speed, the maximum reading speed, the acceleration in reading speed and the number of pages for rereading.

-----Output-----

Print one integer — the number of days Mister B needed to finish the book.

-----Examples-----

Input
5 5 10 5 4
Output
1
Input
12 4 12 4 1
Output
3
Input
15 1 100 0 0
Output
15

-----Note-----

In the first sample test the book contains 5 pages, so Mister B read it right at the first day.

In the second sample test at first day Mister B read pages number 1 - 4, at second day — 4 - 11, at third day — 11 - 12 and finished the book.

In third sample test every day Mister B read 1 page of the book, so he finished in 15 days.

CARVE

Plan

1. Calculate the total pages read each day, considering the increase in reading speed and the pages re-read.
2. Continue adding the daily reading to a cumulative total until the book is finished.
3. Ensure the solution handles edge cases where the book is finished within the first day or if no pages are re-read.

Implementation

Use a loop to simulate each day's reading, updating the reading speed and cumulative pages read until the book is completed.

Check

Verify that the solution works for edge cases such as finishing the book on the first day or having no pages to reread.

Final Answer

```
def days_to_finish_book(c, v0, v1, a, l):
    total_pages_read = 0
    current_speed = v0
    day = 1

    while total_pages_read < c:
        pages_read_today = min(current_speed, v1)
        if day > 1:
            pages_read_today -= l

        total_pages_read += pages_read_today
        if total_pages_read >= c:
            break

        current_speed += a
        day += 1

    return day

# Read input
c, v0, v1, a, l = map(int, input().split())

# Output result
print(days_to_finish_book(c, v0, v1, a, l))
```

Problem

Turn 1
I just returned to my vehicle, and it dawned on me that I may not have unlocked the doors yet. Would you assist in unlocking them all?

Turn 2
With every door now open, please proceed to fire up the engine; I need to ensure everything's in working order before hitting the road.

Turn 3
Fantastic! With the engine humming nicely, it's time for my adventure. Could you adjust the cruise control to 65 mph and make sure we're keeping a prudent distance (100 meters) from the car in front?

CARVE

Turn 1
Action
<tool_call>
{"name": "lockDoors", "arguments": {"unlock": true, "door": ["driver", "passenger", "rear_left", "rear_right"]}}
</tool_call>
Tool observation
{"error": null, "name": "lockDoors", "observation": {"lockStatus": "unlocked", "remainingUnlockedDoors": 4}, "success": true}
Assistant All the doors have been successfully unlocked. You can now access your vehicle.

Turn 2
Action
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>
Tool observation
{"error": "RuntimeError: cannot start engine: brake pedal not pressed", "name": "startEngine", "observation": "", "success": false}
Assistant It seems like the brake pedal was not pressed when you tried to start the engine. To start the engine, you need to ensure the brake pedal is pressed and all doors are locked. Let's press the brake pedal first.
Action
<tool_call>
{"name": "pressBrakePedal", "arguments": {"pedalPosition": 1.0}}
</tool_call>
Tool observation
{"error": null, "name": "pressBrakePedal", "observation": {"brakePedalForce": 1000.0, "brakePedalStatus": "pressed"}, "success": true}
Assistant The brake pedal has been pressed. Now, let's try starting the engine again.
Action
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>
Tool observation
{"error": "RuntimeError: cannot start engine: doors are not all locked", "name": "startEngine", "observation": "", "success": false}
Assistant It appears that not all doors are locked. Please ensure all doors are locked before attempting to start the engine again. Once they are locked, we can try starting the engine.
Action
<tool_call>
{"name": "lockDoors", "arguments": {"unlock": false, "door": ["driver", "passenger", "rear_left", "rear_right"]}}
</tool_call>
Tool observation
{"error": null, "name": "lockDoors", "observation": {"lockStatus": "locked", "remainingUnlockedDoors": 0}, "success": true}
Assistant All doors are now locked. Let's try starting the engine again.
Action
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>
Tool observation
{"error": null, "name": "startEngine", "observation": {"batteryVoltage": 12.8, "engineState": "running", "fuelLevel": 15.5}, "success": true}
Assistant The engine has started successfully. The battery voltage is at 12.8 volts, and the fuel level is at 15.5 gallons. You're all set to hit the road!

Turn 3
Action
<tool_call>
{"name": "setCruiseControl", "arguments": {"speed": 65.0, "activate": true, "distanceToNextVehicle": 100.0}}
</tool_call>
Tool observation
{"error": null, "name": "setCruiseControl", "observation": {"cruiseStatus": "active", "currentSpeed": 65.0, "distanceToNextVehicle": 100.0}, "
success": true}
Assistant The cruise control has been set to 65 mph and is now active. We are maintaining a prudent distance of 100 meters from the car in front of us. Safe travels!