Countdown as an Agentic Optimization Problem

Extended Abstract

Tao Sun
taosun@stanford.edu

1. Motivation and Problem Statement

The Countdown problem is a simple yet versatile playground that could help study multi-step agentic reasoning in language models. Typically, language models work in a text-based, full-response paradigm where a model outputs an entire arithmetic expression and optionally an extensive Chain-of-Thought (CoT) reasoning text in a single turn. This formulation suffers from severe sparse learning signals as it is difficult to check for the validity of the format, correctness of intermediate steps, and achievement of the objective until the full sequence is generated. This implies that mistakes done early on in the process significantly impact the sequence, leading to errors being compounded throughout the process. Equivalently, this kind of error compounding collapses many different failure modes into a single negative signal at the end, yielding an uninformative and weak learning signal for both supervised finetuning and reinforcement-based policy optimization.

2. Method and Novelty

To overcome this structural bottleneck, we introduce the **Agentic Countdown Framework**, an agentic framework that reformulates Countdown as a *finite-horizon, deterministic Markov Decision Process (MDP)*. Instead of generating the final text response at once, the language model acts as an interactive agent predicting a single executable arithmetic action at each step (*i.e.*, selecting two operands and an operator, *e.g.*, $10 - 7 = 3$). An external environment executes the action, updates the remaining multiset of numbers, and returns the simplified state back to the agent for the next round. The major contribution is the reformation, which results in dense action-level supervision and local solution re-use.

3. Implementation Details and Results

Our optimization pipeline begins with a state-action Supervised Fine-Tuning (SFT) phase using successful trajectories from an exact solver, followed by preference alignment via Action DPO. To introduce a strong inductive bias, we regularize ActionDPO with a soft value function (Q_{soft}) that penalizes path depth while rewarding states containing a high density of alternative solution paths.

At inference time, a beam search strategy tracks partial trajectories while maintaining an evaluation budget. Evaluated on the standard test set, our final model achieves a strong **86%** Pass@1 accuracy. On a rigorous, 576-prompt custom test set across nine distinct difficulty groups, it still achieves an **81%** Pass@1 accuracy. These results represent a significant improvement over the strongest text-based full-answer baseline (RLOO), which achieves only 56% and 26% on the respective test sets.

4. Discussion, Limitations, and Conclusion

Shifting Countdown from full-text responses to interactive state-action MDPs allows explicit reusing of the local state-to-action knowledge. However, error analysis reveals clear performance boundaries in harder configurations (hardest 2 groups in the Custom test set). The policy's performance declines sharply on four-number tasks requiring exact division, dropping to a 26.6% pass rate due to the inherent difficulty in action ranking under those cases. Future works should integrate learned value networks or adaptive search sampling to mitigate this bottleneck. In conclusion, our a agentic framework provides a highly effective and verifiable paradigm for complex arithmetic reasoning tasks.

Team contributions: This is a solo project.

Countdown as an agentic Optimization Problem

Tao Sun [1]

Abstract

Countdown is an arithmetic task that requires language models to precisely manipulate a given multiset of numbers using basic arithmetic operations to achieve a given target. The conventional text-based full-answer model often suffers from sparse feedback, since syntactic validity, arithmetic correctness, and goal alignment can only be verified after generating the entire response. To address this limitation, we reformulate Countdown as an **agentic framework under finite-horizon Markov Decision Process (MDP)**. In this framework, the language model predicts a single executable arithmetic action (*e.g.*, $10 - 2$), after which an external executor applies the action and updates the task state by replacing the operands with their result (*e.g.*, 8). This MDP formulation enables local solution reuse and provides dense, action-level supervision. Combined with supervised fine-tuning (SFT), direct preference optimization (DPO), and soft value function for action ranking, our pipeline achieves a Pass@1 accuracy of 86% on the standard test set and 81% on a custom test set, significantly outperforming RLOO text-based full-answer baselines, which score 56% and 26% respectively.

1. Introduction

The Countdown task requires language models (LMs) generating an arithmetic expression that reaches a target value T from an input multiset N_0 using the basic operations $+, -, \times, /$ such that each input number is used exactly once. This task is small enough to be completely solved using an exact solver, yet complex enough that certain models (especially small LMs) still fail to achieve satisfactory performance. Countdown is therefore an ideal testbed for language-model reasoning under strict symbolic verification.

[1]Stanford University. Correspondence to: Tao Sun <taosun@stanford.edu>.

Preprint. June 9, 2026.

Conventional text-based models are typically prompted to generate a full expression or a comprehensive chain-of-thought, which a verifier evaluates only at the end. It suffers from sparse feedback, as diverse failure modes are collapsed into a single negative label at the end. Consequently, the learning signal becomes weak, particularly in early thinking or decision steps where a single sub-optimal operation can inadvertently destroy all paths to a valid completion.

In this paper, we explore an alternative formulation of the same problem. The model takes the current task state as input and predicts a single action, such as `<pick>7, 3, *</pick>`. An executor then parses the action, validates it, computes the result, and updates the task state. Here, the policy model remains the same language model, while the environment manages the task state, executes transitions, and provides labels if the task state reaches the target.

Our main contributions are:

- We formalize Countdown as a deterministic finite-horizon action MDP and prove that accepted successful trajectories coincide with valid Countdown expression trees.
- We derive exact binary successor values and show how local ActionDPO implements a KL-regularized policy improvement under Bradley–Terry action preferences.
- We design a comprehensive training pipeline combining state-action SFT, ActionDPO, and distillation. Our final model significantly outperforms conventional text-based baselines.

2. Related Work

LMs with reasoning and search. To solve complex multi-step problems, recent work focuses on breaking tasks down into smaller pieces. Chain-of-Thought (CoT) prompting guides language models (LMs) to generate intermediate reasoning text before producing a final answer (Wei et al., 2022). Building on this, Self-Consistency samples multiple reasoning paths and selects the most common final answer (Wang et al., 2023), while Tree of Thoughts allows LMs to explicitly search over a tree of natural-language states (Yao et al., 2023a).

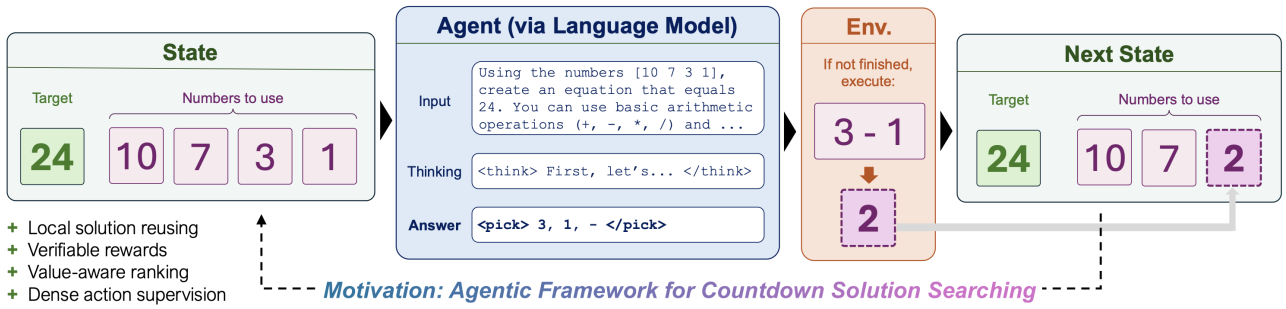

Figure 1. **Our state-action agentic framework for Countdown**: Our framework converts a full arithmetic response into a sequence of executable local choices. The policy proposes an arithmetic action (*e.g.*, $3 - 1$), and the environment checks and applies the transition exactly (*e.g.*, replacing 3 and 1 with 2). By doing so, our framework allows local solution reusing and more dense step-level supervison, compared to standard text-basd full-response models. (*Best viewed in color.*)

Other approaches use interactive intermediate tracking. For example, verifier-based systems demonstrate that providing feedback on intermediate steps significantly improves mathematical reliability (Nye et al., 2021; Cobbe et al., 2021; Lightman et al., 2024). Compared to those methods, our state-action Countdown formulation allows for a stricter and more compact state representation. Because every intermediate state is a concrete, executed multiset of numbers, our system can immediately catch and reject illegal actions before they break later steps. Furthermore, an exact solver can instantly evaluate whether a state can still successfully reach the target.

Preference optimization for LMs. To better facilitate LM optimization, recent work explores the use of preference data (Christiano et al., 2017). Direct Preference Optimization (DPO) gives a closed-form pairwise objective relative to a reference policy (Rafailov et al., 2023), and recent theory connects preference optimization with regularized policy improvement (Gheshlaghi Azar et al., 2024). Compared to this work, our action DPO's comparisons are conducted in a local level. The preferred and rejected items are two actions from the same arithmetic state. This removes many style and length confounders that appear in response-level comparisons in common LMs.

LMs with an agentic framework for arithmetic tasks. A growing line of work integrates language models with external computational tools/environments to overcome standard reasoning limitations. For instance, *Toolformer* demonstrates how language models can teach themselves to call simple helper APIs to handle arithmetic or factual lookups (Schick et al., 2023). Similarly, the ReAct framework interleaves thought reasoning and system actions to guide models through multi-step environmental environments (Yao et al., 2023b), while Program-Aided Language models offload calculations entirely by generating a block of Python code and passing it to a programmatic runtime (Gao et al., 2023).

Our work differs from these agentic framework methods in how the environment and the model interact. Our system relies on the executor as a strict state-transition engine that completely drives a finite-horizon MDP.

3. Method

3.1. Countdown as an agentic framework under MDP

Our agentic framework makes the Countdown game as a step-by-step Markov decision problem. Imagine that we want to reach a specific target number $T \in \mathbb{Q}$. At any step t, we have a set of remaining numbers available to use, represented as a multiset N_t. If we start with a total of m numbers, then a **task state** s_t can be defined as a tuple:

$$s_t = (T, N_t, h_t), \quad \text{where } |N_t| = m - t. \tag{1}$$

Here, h_t is a multiset of history actions ($h_0 = \emptyset$), used to reconstruct the answer expression. The legal action set is

$$\mathcal{A}(s_t) = \{(n_i, n_j, o) \mid i \neq j,\ n_i, n_j \in N_t,\ o \in \{+, -, \times, /\},\ \text{valid}(n_i, n_j, o)\}. \tag{2}$$

Here, the function $\text{valid}(\cdot, \cdot, \cdot)$ helps to exclude division by zero. When a legal action $a_t = (n_i, n_j, o)$ is taken, the environment updates deterministically according to a transition function:

$$\mathcal{T}(s_t, a_t) = \big(T, ((N_t \setminus \{n_i, n_j\}) \cup \{n_i \, o \, n_j\}),\ h_t \cup \{a_t\}\big). \tag{3}$$

The game ends when only one number remains in N. Thus, a terminal state succeeds when reward $R(s_t)$ is 1, where

$$R(s_t) := \mathbf{1}\{|N| = 1\} \cdot \mathbf{1}\{N = \{T\}\}. \tag{4}$$

To interact with this environment, our language model reads a text string representing the current state s_t and generates a short text as the action, such as `<pick>7,3,*</pick>`.

An external parser $\phi(y, s)$ processes this text. If the text is formatted correctly and represents a legal move, it maps to a valid action a. If it is illegal, it maps to an error. Note that the actual probability of executing a valid action a is the sum of the probabilities of all text strings y that parse into that action:

$$\pi_\theta(a \mid s) = \sum_{y:\phi(y,s)=a} \pi_\theta(y \mid p(s)). \tag{5}$$

Separating the raw text generation from the actual game action is a benefit of our formulation, because it allows our evaluation system to point exactly why a model failed, whether it wrote a corrupted text action, chose a number it already used, attempted an invalid math operation, etc.

Example. To further illustrate our formulation, let us look at a game where the target is 24 and the initial numbers are $N_0 = \{10, 7, 3, 1\}$. If the model chooses the action $(3, 1, -)$, the new N_1 becomes $\{10, 7, 2\}$. A full rollout example can be found in Table 1.

Table 1. Rollout example for $T = 24$ and $N_0 = \{10, 7, 3, 1\}$.

t	Action output	Operation	Successor multiset
0	`10,3,*`	$10 \times 3 = 30$	$\{30, 7, 1\}$
1	`30,7,-`	$30 - 7 = 23$	$\{23, 1\}$
2	`23,1,+`	$23 + 1 = 24$	$\{24\}$

3.2. Value functions and action ranking

To help the model choose the best path, we need a way to measure how favorable a task state is. Let $H(s) = |N| - 1$ represent the number of steps left in the game. We define an exact solvability function $\mathrm{Solv}(s)$, which outputs 1 if an exact solver can find at least one winning path from state s, and 0 otherwise:

$$\mathrm{Solv}(s) = \mathbf{1}\{\text{some legal continuation from } s \text{ reaches reward } 1\}. \tag{6}$$

Using this, we can determine the optimal binary value of a state ($V^\star$) and the value after taking an action (i.e., Q_{bin}):

$$\begin{aligned} V^\star(s) &= \mathrm{Solv}(s), \\ Q_{\mathrm{bin}}(s, a) &= V^\star(\mathcal{T}(s, a)). \end{aligned} \tag{7}$$

Proposition 3.1 (Bellman correctness of Q_{bin})**.** *For every non-terminal state, the following recursive relationships hold true:*

$$\begin{aligned} V^\star(s) &= \max_{a \in \mathcal{A}(s)} V^\star(\mathcal{T}_{\mathrm{env}}(s, a)), \\ Q_{\mathrm{bin}}(s, a) &= V^\star(\mathcal{T}_{\mathrm{env}}(s, a)). \end{aligned} \tag{8}$$

Therefore, checking whether a move leads to a solvable next state gives us the mathematically optimal binary action value for our Countdown MDP.

Proof. If a state is solvable, there must be a first correct move that leads to a solvable next state. Conversely, if a move leads to a solvable next state, we can complete the solution by following the solver's remaining path. Because the game has a fixed, finite number of steps, this logic holds perfectly at every step, turning our search for a solution into a maximum over available actions. □

To give our model a more informative inductive bias, we introduce a soft-version of the value called $Q_{\mathrm{soft}}(s, a)$. Let $s' = \mathcal{T}_{\mathrm{env}}(s, a)$ be the state we land on. We ask our exact solver for: (1) $d_{\min}(s')$, which is the shortest number of steps needed to finish the game, and (2) $c(s')$, which is the total number of valid ways to complete the game from the current s. We combine these into the soft value:

$$Q_{\mathrm{soft}}(s, a) = \begin{cases} 1 - \alpha d_{\min}(s') & \\ \quad + \beta \log(1 + c(s')), & \mathrm{Solv}(s') = 1, \\ 0, & \mathrm{Solv}(s') = 0. \end{cases} \tag{9}$$

In other words, it rewards actions that keep the solution path short and leave rooms for enough other options for later steps.

Finally, we combine the model's text probabilities with this external value guidance to calculate a ranking score G_λ:

$$\begin{aligned} G_\lambda(h_t, a) &= L_\theta(h_t) + \ell_\theta(a \mid s_t) + \lambda Q(s_t, a), \\ L_\theta(h_t) &= \sum_{j<t} \ell_\theta(a_j \mid s_j). \end{aligned} \tag{10}$$

3.3. Beam search strategy

During inference or data generation, we use beam search of a beam width B and sample size K. It tracks the top B best partial paths at the same time. At each step, it generates K candidate action tags for each path. Any invalid actions are ignored. The remaining valid actions are ranked using our ranking score (Eq. 10).

The following lemma guarantees that as long as our model has a non-zero chance of generating good moves, then the search budget will exponentially increase our odds of finding the solution:

Lemma 3.2 (Success rate's lower bound)**.** *Suppose there exists a valid path of H steps passing through states $s_0, \ldots, s_{H-1}$. Let $\mathcal{A}_{\mathrm{good}}(s_t) \subseteq \mathcal{A}(s_t)$ be the set of moves that keep the problem solvable, and let $p_t = \pi_\theta(\mathcal{A}_{\mathrm{good}}(s_t) \mid s_t)$ be the model's probability of choosing one of them. If we sample K candidate actions independently at each step, then:*

$$\Pr(\textit{path is solvable}) \geq \prod_{t=0}^{H-1} \left[1 - (1 - p_t)^K\right]. \tag{11}$$

Proof. At any single step t, the probability that all K independent samples fail to find a winning move is $(1-p_t)^K$. Therefore, the probability that at least one sample is successful is $1-(1-p_t)^K$. Because the beam search preserves successful prefixes, these step-by-step probabilities accumulate multiplicatively over the H steps. □

3.4. Learning objectives

To train our policy model, we employ two distinct training objectives that teach the model both **(1)** the distribution of actions-to-take conditioning on the current state, via supervised finetuning, and **(2)** how to rank action choices via direct performance optimization.

Supervised finetuning (SFT). This objective lets the model to learn successful action trajectories generated by the exact solver $\mathcal{D}_{\text{sft}}$:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(s,a^*)\sim\mathcal{D}_{\text{sft}}}\ell_\theta(a^* \mid s). \tag{12}$$

Direct performance optimization (DPO). Instead of just mimicking a single good choice, we also want the model to learn why one action is better than another from the exact same state. We train action-level DPO (ActionDPO) using localized state-action triplets (s, a^+, a^-), where a^+ is a higher-ranked action and a^- is the lower-ranked one.

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{\mathcal{D}_{\text{pref}}}\left[\log\sigma\left(\beta_{\text{dpo}}\left[\Delta_\theta - \Delta_{\text{ref}}\right]\right)\right], \tag{13}$$

where the policy preference margins Δ_θ and reference model margins Δ_{ref} are computed as:

$$\begin{aligned}\Delta_\theta &= \ell_\theta(a^+ \mid s) - \ell_\theta(a^- \mid s),\\ \Delta_{\text{ref}} &= \ell_{\text{ref}}(a^+ \mid s) - \ell_{\text{ref}}(a^- \mid s).\end{aligned} \tag{14}$$

Proposition 3.3 (ActionDPO as local KL-regularized improvement)**.** *Fix an intermediate state s and a reference policy π_{ref}. The solution to the localized optimization problem:*

$$\max_{\pi\in\Delta(\mathcal{A}(s))}\sum_a \pi(a \mid s)Q(s,a) - \eta\,\text{KL}\big(\pi(\cdot \mid s)\,\|\,\pi_{\text{ref}}(\cdot \mid s)\big) \tag{15}$$

solves by the exact closed form:

$$\pi_\eta(a \mid s) = \frac{\pi_{\text{ref}}(a \mid s)\exp(Q(s,a)/\eta)}{Z_s}. \tag{16}$$

If the underlying preference labels mirror the true environment margins via $P^*(a \succ b \mid s) = \sigma(\kappa(Q(s,a) - Q(s,b)))$, optimizing the ActionDPO loss (Eq. 13) at scale yields a policy that satisfies:

$$\log\frac{\pi_\theta(a \mid s)}{\pi_\theta(b \mid s)} = \log\frac{\pi_{\text{ref}}(a \mid s)}{\pi_{\text{ref}}(b \mid s)} + \frac{\kappa}{\beta_{\text{dpo}}}(Q(s,a) - Q(s,b)). \tag{17}$$

Hence, ActionDPO mathematically recovers the exact odds ratio of a reward-tilted local policy with an implicit temperature of $\eta = \beta_{\text{dpo}}/\kappa$.

Proof. The optimal policy shape comes directly from constructing a standard Lagrange multiplier problem, taking first-order derivatives, and normalizing the distribution. For the preference objective, the overall cross-entropy loss is minimized when our predicted model probabilities match the real data preferences. Since the sigmoid function σ maps inputs to unique outputs, equating these probabilities yields the precise log-odds relationship in Eq. 17. □

4. Experiments and Results

4.1. Setup

Evaluation data. We use two held-out evaluation sets. The **Test set** contains standard Countdown prompts for model comparison. The **Custom test set** contains 576 additionally generated prompts and is designed to stress harder reasoning cases. It has nine difficulty groups with 64 prompts each. The groups differ by the number of starting numbers, whether multiplication or division is needed, and how many exact solution paths exist. The hardest groups are the four-number multiplication and division groups with few valid solution paths. The detailed definition can be found in Table 3.

Metrics. We measure performance using three primary metrics:

1. **Pass@1 accuracy:** the fraction of prompts where the selected trajectory reaches the exact target.
2. **Average score:** a scalar score where a correct solution earns 1.0, a valid path that reaches the wrong target earns partial credit of 0.1, and an invalid or broken path earns 0.0.
3. **Action rejection rate:** the fraction of sampled action tags that cannot be parsed or cannot be legally executed in the current state.

For state-action methods, evaluation returns one searched trajectory per prompt. Thus Pass@1 measures whether the selected trajectory solves the prompt. The invalid action rate is computed over all sampled action tags before illegal tags are removed by the executor.

Search budget. For state-action methods, a budget B/K means that beam search keeps B partial trajectories and samples K action tags for each active state. The standard budget is $16/16$. Greedy action decoding is $1/1$, and broad search is $32/32$. All state-action rows use the same parser, executor, and exact final verifier.

Method name	Meaning
Full-answer SFT	A text model trained to write a complete final expression in one response. It has no intermediate executor state. This is from **Milestone 1**.
Full-answer RLOO	The strongest text-space baseline. It is trained with RLOO in the **Milestone 2**.
State-to-action SFT	A state-action policy trained on exact solver actions. At test time it predicts tags like `<pick>7,3,*</pick>`.
Action DPO	Pairwise training on two actions from the same state. The chosen action keeps a solution path open, while the rejected action has lower exact value.
Hard-case Action DPO	Action DPO with extra training weight on prompts shaped like the hardest Custom test set groups L7–L9. Held-out prompts are never used for training.
Binary or Q_{soft} rerank	Inference-only action reranking using Eq. 10. Binary rerank uses Q_{bin}; Q_{soft} rerank uses Eq. 9.
Action DPO with Q_{soft}	Training variants where the teacher uses the Q_{soft} value instead of the binary solvability value.
Final model	The Action DPO with Q_{soft} checkpoint, evaluated with $16/16$ search, temperature 0.9, top-$p = 0.95$, and top-$k = 50$.

Table 2. **Methods used in the experiment.**

Evaluated learning strategies. Table 2 gives the meaning of every method name used in the result tables. The most important distinction is between text-based full-answer models (from Milestone 1 and 2) and the state-action models using our proposed agentic framework.

4.2. Quantitative results and analysis

Does a larger search budget improve performance? As shown in Figure 2 and Table 4, giving the state-to-action SFT model more search budget directly improves accuracy. On the Test set, accuracy rises from 0.220 with greedy $1/1$ decoding to 0.820 with $32/32$ search. On the Custom test set, the same model rises from 0.325 at $1/8$ to 0.764 at $16/16$ and 0.832 at $32/32$.

This trend matches Lemma 3.2. The model already assigns some probability to useful actions. A larger candidate set makes those useful actions easier to find and preserve. The gain is also larger than a formatting gain. The invalid action rate stays near 1%–2% for the larger search budgets, so the main bottleneck is action ranking.

Which training strategy helps the policy most? Table 4 shows that moving from full-answer generation to the state-action MDP gives the largest gain. The full-answer valid-

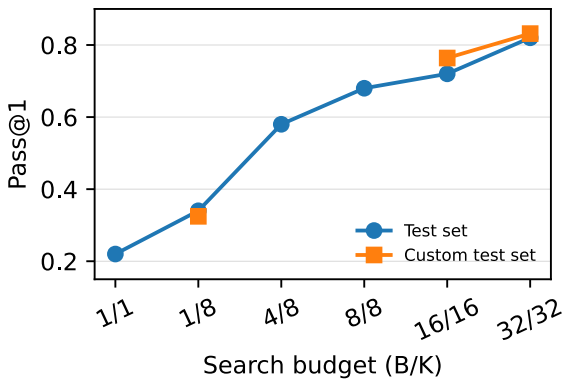

Figure 2. **Increasing the search budget improves state-to-action SFT**. The Test set has all test budgets. The Custom test set is evaluated at the main budgets used in later comparisons.

wrong DPO model reaches 0.560 Pass@1 on the Test set and 0.260 on the Custom test set. The first state-to-action SFT model already reaches 0.720 and 0.764 with the same $16/16$ budget.

Action DPO improves the Test set to 0.740, which shows that local action preferences can sharpen the policy. Its Custom test set score is lower than the state-to-action SFT score. Hard-case Action DPO moves the Custom test set back up to 0.762 and gives a clear gain on difficult multiplication and division states. Its Test set score drops to 0.700, so focusing too much on rare hard states can shift the policy away from the standard test distribution.

How do Q value's hyperparameters affect performance? The value-guided search in Eq. 10 uses three hyperparameters: λ controls how strongly the external value changes the model ranking, α penalizes longer remaining solutions in Q_{soft}, and β rewards states with many remaining solution paths. Table 6 summarizes the main value ablation.

At the $1/8$ budget, plain state-to-action SFT reaches only 0.325 on the Custom test set. Adding exact binary successor values raises this to 0.705. Using Q_{soft} raises it further to 0.719, with the best setting at $\lambda = 1.0$, $\alpha = 0.2$, and $\beta = 0.05$. This means the solver value is useful when search is small. It helps the system pick actions that keep a solution path open.

Figure 3 gives a clearer view of the Q_{soft} sweep. Two patterns stand out. First, value guidance helps across a fairly wide range of settings, which means the gain does not depend on one narrow hyperparameter choice. Second, the best results appear at moderate value strength. Very small λ does not use the value strongly enough, while very large λ can overrule the policy score too much. From those results, we chose $\lambda = 1$, $\alpha = 0.2$, and $\beta = 0.05$ for the later experiments regarding Q_{soft}.

Group	Definition	Pass@1	Failures
L1	3 numbers, shortest solver expression uses only addition/subtraction, at least 3 unique solutions	0.984	1
L2	3 numbers, multiplication without division, at least 3 unique solutions	0.984	1
L3	3 numbers, multiplication without division, at most 2 unique solutions	0.906	6
L4	3 numbers, shortest solver expression uses division	0.750	16
L5	4 numbers, shortest solver expression uses only addition/subtraction, at least 4 unique solutions	1.000	0
L6	4 numbers, multiplication without division, at least 6 unique solutions	0.984	1
L7	4 numbers, multiplication without division, at most 5 unique solutions	0.578	27
L8	4 numbers, shortest solver expression uses division, at least 6 unique solutions	0.812	12
L9	4 numbers, shortest solver expression uses division, at most 5 unique solutions	0.266	47

Table 3. **Custom test set definitions and final-model results.** Each group has 64 prompts. The final column counts unsolved prompts.

		Test set		Custom test set		
Learning strategy	**Budget**	Pass@1	Score	Pass@1	Score	Rejection rate
Full-answer SFT	text	0.320	0.351	0.200	0.230	N/A
Full-answer RLOO	text	0.560	0.540	0.260	0.276	N/A
State-to-action SFT	$1/8$	0.340	0.406	0.325	0.392	0.4%
State-to-action SFT	$16/16$	0.720	0.748	0.764	0.788	1.5%
State-to-action SFT	$32/32$	0.820	0.838	**0.832**	**0.848**	2.0%
Action DPO	$16/16$	0.740	0.766	0.731	0.758	1.4%
Hard-case Action DPO	$16/16$	0.700	0.730	0.762	0.786	1.1%
Final model (Action DPO, temp 0.9)	$16/16$	**0.860**	**0.874**	0.807	0.827	1.1%

Table 4. **Main experimental results.** Full-answer baselines write the final expression in one response. State-action methods use our MDP and symbolic executor. The search budget B/K means that beam search keeps B active paths and samples K action tags per state.

Sampling setting	**Pass@1**	**Score**
Temperature 0.3	0.600	0.640
Temperature 0.5	0.740	0.766
Temperature 0.7	0.800	0.820
Temperature 0.9	**0.860**	**0.874**

Table 5. **Decoding calibration for the final model checkpoint on the Test set.** All rows use the same $16/16$ search budget, top-$p = 0.95$, and top-$k = 50$.

Setting	**Budget**	**Test**	**Custom**
No value guidance	$1/8$	0.340	0.325
Binary value rerank	$1/8$	0.660	0.705
Q_{soft} rerank	$1/8$	0.660	0.719
Action DPO	$16/16$	0.740	0.731
Action DPO with Q_{soft}	$16/16$	0.780	0.741
Final model	$16/16$	**0.860**	**0.807**

Table 6. **Value guidance ablation.** Reranking changes only the action score during search. The checkpoint stays fixed. The best Q_{soft} inference setting uses $\lambda = 1.0$, $\alpha = 0.2$, and $\beta = 0.05$.

Notably, the value signal is more effective as a training target for Action DPO. It improves over the plain search setting, showing that baking soft value advantages directly into the policy during offline training allows the model to inherently choose more robust action.

How does sampling temperature affect the performance? Table 5 explains the last step of model selection. A low temperature makes the policy repeat its highest-probability actions, which is safe on easy states yet misses useful alternatives. A higher temperature exposes more candidate actions to the same executor and beam search. Since the executor filters illegal actions, the added diversity mainly helps the search find a solvable path.

Final model choice. Based on all the avove analysis, the final system is the Action DPO policy with Q_{soft} training. At evaluation time, the final system uses the trained policy with the same $16/16$ search budget as the other main state-action rows. Its action sampling temperature is set to 0.9, with top-$p = 0.95$ and top-$k = 50$. This setting increases proposal diversity inside the fixed search budget, while the executor still rejects illegal actions and verifies the final expression.

4.3. Qualitative results

Table 7 shows typical behavior of the final model. In the success cases, the model finds short local steps that keep

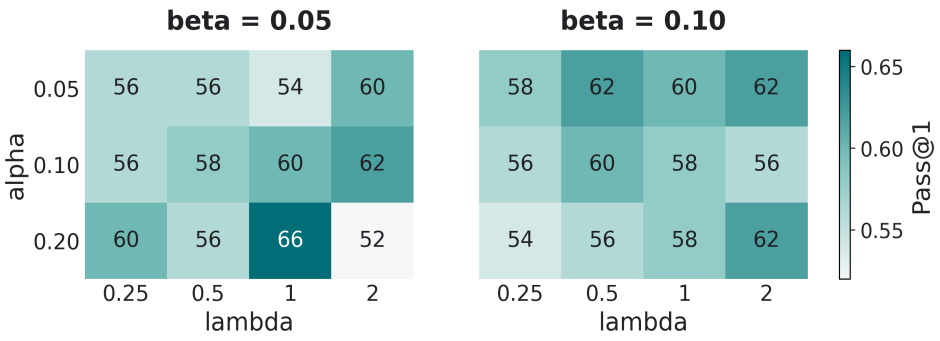

Figure 3. **Test set Pass@1 (%) for the Q_{soft} reranker at $1/8$ search.** Each heatmap fixes one value of β and sweeps over the value weight λ and the depth penalty α. The left panel uses $\beta = 0.05$ and the right panel uses $\beta = 0.10$. The best setting is $\lambda = 1.0$, $\alpha = 0.2$, $\beta = 0.05$, which reaches 66% Pass@1 on the Test set. The main trend is stable: moderate value weight works best, and the setting with $\beta = 0.05$ is slightly stronger than $\beta = 0.10$.

Type	Group	Numbers → Target	Final model output	Reference solution / note
Success	L7	$[24, 92, 65, 5] \rightarrow 37$	`(65 - ((24 * 5) - 92))`	Uses multiplication and subtraction to reach the target exactly. A valid reference solution is `((92 + 65) - (24 * 5))`.
Success	L9	$[62, 43, 11, 38] \rightarrow 13$	`((62 + (38 + 43)) / 11)`	A hard four-number division case. A valid reference solution is `((38 + (62 + 43)) / 11)`.
Failure	L4	$[78, 68, 73] \rightarrow 2$	`(68 / (78 - 73))` $= 68/5$	Valid intermediate actions, wrong final value. A correct solution is `((78 + 68) / 73)`.
Failure	L9	$[41, 65, 65, 17] \rightarrow 67$	`(65 + ((65 - 41) - 17))` $= 72$	The model follows a simple subtraction path and misses the useful division step. A correct solution is `(65 + ((65 + 17) / 41))`.

Table 7. **Success and failure examples from the final model.** The success cases show that the model can solve sparse multiplication and division problems with short action sequences. The failure cases are valid action paths that end at the wrong target, which matches the main error mode in our evaluation.

the state easy to solve. In the failure cases, the model still produces valid executable actions, yet it commits to a path that drifts away from the target. This is why the invalid action rate stays low even when the final answer is wrong.

Where do errors still occur? Figure 4 and Table 3 show that the final model solves almost all easy and medium groups, in L1, L2, L5, and L6. The remaining errors are concentrated in sparse four-number tasks. Out of 111 total Custom test set failures, 47 come from sparse four-number division, 27 from sparse four-number multiplication, and 16 from three-number division.

The final model improves several hard groups compared with state-to-action SFT at the same $16/16$ budget. Four-number multiplication rises from 0.406 to 0.578, and dense four-number division rises from 0.656 to 0.812. Sparse four-number division remains the hardest group at 0.266. The $32/32$ SFT run reaches 0.438 on this group, so more search can still recover solutions that the final $16/16$ model fails.

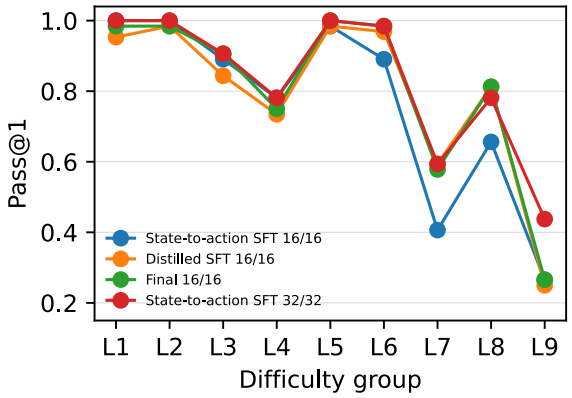

Figure 4. **Pass@1 by Custom test set difficulty group.** The final $16/16$ model improves the hard multiplication and dense division groups, while L7 and L9 (four-number division) remain the main failure case.

5. Conclusion

We presented a state-action formulation of Countdown that turns full-answer arithmetic generation into a short deterministic decision process. The model predicts one executable arithmetic action at a time, and the symbolic executor checks the action, updates the multiset, and verifies success exactly. This gives dense action-level supervision and makes failures easier to identify. The main weakness is still at the harder questions. Future work should improve with learned value models, stricter action filtering, or adaptive search budgets that spend more samples only when the current state is hard.

References

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. In *International Conference on Machine Learning*, 2023.

Gheshlaghi Azar, M., Rowland, M., Piot, B., Guo, D., Calandriello, D., Valko, M., and Munos, R. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, 2024.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., Sutton, C., and Odena, A. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023a.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.