BACKPROPAGATING VERIFIER REWARDS THROUGH CONTINUOUS FLOW LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) has substantially improved the reasoning of autoregressive and discrete-diffusion language models. Language models that generate continuous token embeddings are emerging as a strong alternative, but RL post-training for them remains understudied. We propose CFRL, a post-training framework for continuous flow language models. Such a model denoises token embeddings over multiple steps and decodes them into discrete tokens only at the last step. To obtain pathwise gradients, which backpropagate through continuous actions and have much lower variance than REINFORCE, we introduce a critic that estimates the value of intermediate token embeddings, bypassing the non-differentiable decoding step. The critic must also keep track of the value function as the policy changes; to this end, we design a critic that predicts a value from every token representation and improves policy optimization. Across four reasoning tasks, CFRL reliably improves a continuous flow language model through RL post-training and outperforms baseline methods of the same type.

1 INTRODUCTION

Reinforcement learning (RL) post-training has become a central mechanism for turning pretrained autoregressive (AR) language models into reasoners (Ouyang et al., 2022; Shao et al., 2024; DeepSeek-AI, 2025). The explicit token probabilities of AR models provide a natural RL interface: each token choice is an action that an advantage can reinforce or suppress. However, AR generation requires one sequential model evaluation per token, tying latency to sequence length. Diffusion-based language models ease this bottleneck by jointly refining all token positions over a fixed number of denoising steps, which can substantially speed up generation (Nie et al., 2025; Sahoo et al., 2024; Lou et al., 2023). RL has recently been applied to diffusion-based language models with promising results (Zhao et al., 2025; Zhu et al., 2025; Shao et al., 2026). Because these models operate on discrete tokens and take discrete actions during generation, RL methods for them, including PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), rely mainly on score-function (REINFORCE) gradients (Williams, 1992).

Continuous language models (CLMs) have historically lagged behind discrete counterparts in language generation (Li et al., 2022; Gong et al., 2022; Dieleman et al., 2022). The recent ELF (Hu et al., 2026), a CLM based on flow matching (Lipman et al., 2022; Liu et al., 2022), closes this gap: it achieves lower generative perplexity with fewer sampling steps while using ten times fewer training tokens. ELF progressively denoises Gaussian noise into real-valued token embeddings, then decodes the final embeddings into discrete tokens. Its continuous generation trajectory is differentiable, which makes it possible to backpropagate reward signals through the denoising process.

In this work, we propose **C**ontinuous **F**low **R**einforcement learning for **L**anguage models (CFRL), a post-training framework for improving CLM reasoning. We first adapt group-relative REINFORCE (Yao et al., 2026) to CLM post-training, using a verifier applied to the decoded response as the reward. We then exploit the differentiability of the continuous trajectory. Because the CLM's decoder is not differentiable, we train a critic to predict the expected verifier outcome from intermediate denoising states. Near the end of the trajectory, the critic can predict this outcome accurately because the state contains nearly all the information needed to determine the decoded response. Its prediction is a differentiable reward estimate that can be backpropagated through a flow transition. We further design a novel critic architecture that propagates reward feedback to each token representation: it

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.

predicts a value from every token representation and trains each prediction against the sequence-level verifier outcome. This design turns the verifier signal into position-specific gradients, enabling more effective policy optimization.

We evaluate our framework under a unified protocol on four program-verifiable reasoning tasks: Countdown, Sudoku, Knights-and-Knaves, and Game24. Our models perform comparably to or better than baselines of similar or larger size. Our contributions are as follows.

- To the best of our knowledge, we demonstrate the first reliable improvement in the reasoning capabilities of a CLM through pathwise reinforcement learning.
- We propose a novel critic architecture that consistently improves RL performance and can be readily adapted to other denoising-based generative models.
- We evaluate our approach on four program-verifiable reasoning tasks and demonstrate its effectiveness.

2 Related Work

Diffusion and flow language models. Discrete diffusion language models, including SEDD, MDLM, and LLaDA (Lou et al., 2023; Sahoo et al., 2024; Nie et al., 2025), generate through discrete or masked-token trajectories. Continuous-state language generation instead diffuses token embeddings, beginning with Diffusion-LM (Li et al., 2022). Although earlier continuous models lagged behind discrete diffusion, ELF (Hu et al., 2026) closes this gap and outperforms discrete counterparts by using flow matching (Lipman et al., 2022; Liu et al., 2022; Albergo et al., 2023) in a continuous embedding space and decoding tokens only at the trajectory endpoint.

RL for denoising generative models. RL for image diffusion commonly casts denoising as a sequential decision process and applies score-function policy gradients, as in DDPO and DPOK (Black et al., 2023; Fan et al., 2023). Flow-GRPO, DanceGRPO, and MixGRPO adapt this approach to flow-matching models (Liu et al., 2025; Xue et al., 2025; Li et al., 2025), while FPO uses the flow-matching objective as a likelihood surrogate within PPO (McAllister et al., 2025). RL for discrete diffusion language models relies on score-function updates using one-step likelihood approximations, variational objectives, or actions at intermediate denoising states (Zhao et al., 2025; Zhu et al., 2025; Shao et al., 2026).

RL for autoregressive language models. RL post-training has enabled autoregressive language models, including OpenAI-o1 (OpenAI, 2024) and DeepSeek-R1 (DeepSeek-AI, 2025), to become strong reasoners. These methods optimize categorical token actions with score-function methods such as PPO and GRPO (Shao et al., 2024; Schulman et al., 2017), with recent variants modifying clipping and sampling (Yu et al., 2025; Zheng et al., 2025).

3 Preliminaries: Continuous Flow Language Modeling and RL

Continuous flow language modeling. We build our work on ELF (Hu et al., 2026), a flow-based language model. ELF represents text tokens using embeddings produced by a frozen T5 encoder (Raffel et al., 2020). Let $\mathbf{x} = (x_1, \ldots, x_L)$ denote the embeddings of the tokens in a target sequence $\mathbf{w} = (w_1, \ldots, w_L)$. When the target sequence is conditioned on additional context, such as a prompt, we denote the corresponding context-token embeddings by $\mathbf{c}$.

ELF models the conditional distribution $p_\theta(\mathbf{x} \mid \mathbf{c})$ using flow matching (FM), where a neural network learns to transform noisy states into clean token embeddings. At time $t \in [0, 1]$, the noisy state is defined as $\mathbf{z}_t = (1 - t)\,\epsilon + t\,\mathbf{x}$, where $\epsilon \sim \mathcal{N}(0, c_\sigma^2 I)$ is Gaussian noise with fixed scale c_σ^2, and $\mathbf{x}$ denotes the clean target embeddings. ELF adopts an x-prediction parameterization (Li & He, 2026; Han et al., 2026): given a noisy state $\mathbf{z}_t$, the denoiser predicts an estimate $\mathbf{x}_t$ of the clean embeddings $\mathbf{x}$. Let $0 = t_0 < \cdots < t_K = 1$, denote a discretization of the flow time. The corresponding denoising update is

$$\mathbf{x}_{t_k} = \mathrm{d}_\theta(\mathbf{z}_{t_k}, t_k, \mathbf{x}_{t_{k-1}}, \mathbf{c}) \tag{1}$$

We explicitly incorporate the self-conditioning (Chen et al., 2022), whereby the denoiser d_θ takes the previous clean-embedding estimate $\mathbf{x}_{t_{k-1}}$ as an additional input. The denoiser d_θ is implemented as a diffusion Transformer (Peebles & Xie, 2023), with θ denoting its learnable parameters. It is trained by minimizing the mean squared error (MSE) between the predicted clean embedding $\mathbf{x}_t$ and the corresponding ground-truth embedding $\mathbf{x}$.

At inference time, we consider the SDE sampling of ELF, which injects stochastic noise at each generation step. This stochastic sampling is particularly useful for the reinforcement learning framework introduced later, as it induces a distribution over generation trajectories rather than a single deterministic path.

$$\mathbf{z}_{t_k} = \lambda_k \mathbf{z}_{t_{k-1}} + \eta_k \mathbf{x}_{t_{k-1}} + \psi_k \epsilon, \qquad k = 1, \ldots, K \tag{2}$$

The scalars λ_k, η_k, and ψ_k are determined by the step size and the noise reinjection scale specified by the ELF sampling algorithm. Sampling is initialized with $\mathbf{z}_0 \sim \mathcal{N}(0, c_\sigma^2 I)$ and $\mathbf{x}_{t_{-1}} = \mathbf{0}$. After K denoising steps, the procedure produces $\mathbf{z}_1$, which corresponds to the clean embedding $\mathbf{x}$.

ELF further learns a decoder $p_\theta(\mathbf{w} \mid \mathbf{x})$ that maps the final denoised embeddings $\mathbf{z}_1$ to the discrete token sequence $\mathbf{w}$. Specifically, each embedding in $\mathbf{z}_1$ is projected to the token vocabulary through a linear output layer, yielding a distribution over discrete tokens.

$$w_i \sim \mathrm{softmax}(W \mathbf{z}_1[i]), \qquad i = 1, \ldots, L \tag{3}$$

Here, W denotes the linear output layer that maps each embedding to a vector of vocabulary logits. We include W in the set of learnable parameters θ. Together, $p_\theta(\mathbf{x} \mid \mathbf{c})$ and the decoder $p_\theta(\mathbf{w} \mid \mathbf{x})$ define the conditional language distribution $p_\theta(\mathbf{w} \mid \mathbf{c})$. A derivation of the stochastic ELF sampler is provided in Appendix B.1.

4 METHOD

The generation procedure of ELF introduced in Section 3 proceeds over a fixed horizon of $K + 1$ steps. At each generation step $k \in \{1, \ldots, K\}$, we define the MDP state as $s_k = (\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})$. The initial state is $s_0 = (\epsilon, \mathbf{0})$, where ϵ is the Gaussian noise defined above. We include $\mathbf{x}_{t_{k-1}}$ in the state because it is required by the self-conditioning mechanism of ELF. We omit $\mathbf{c}$ and t_k from the state s_k for notational simplicity. The terminal state $s_{K+1} = \mathbf{w}$ corresponds to the decoded response.

The actions consist of the denoising predictions followed by the final decoding step. Specifically, for $k \in \{0, \ldots, K-1\}$, we define the action as $a_k = (\mathbf{x}_{t_k}, \epsilon)$. The corresponding policy $\pi_\theta(a_k | s_k)$ defined by the denoiser d_θ and the noise distribution. The final action is $a_K = \mathbf{w}$, and the policy at this step is $\pi_\theta(a_K | s_K) = p_\theta(\mathbf{w} | \mathbf{z}_{t_K})$.

The transition probability is defined by the denoising process in (2). A terminal reward $R(s_{K+1})$ is provided by a verifier that evaluates the decoded response $\mathbf{w}$. The reinforcement learning objective is to maximize the expected terminal reward over trajectories induced by the policy:

$$\max_\theta \mathbb{E}_{s_0 \sim p_0, a_k \sim \pi_\theta(\cdot|s_k), s_{k+1} \sim p(\cdot|s_k, a_k)}[R(s_{K+1})]. \tag{4}$$

4.1 CFRL-GRR: LEARNING WITH GROUP-RELATIVE REINFORCE

Similar to other language models, an ELF model can be post-trained using policy-gradient methods (Sutton et al., 1998), without concerning the last non-differentible decoding step. We start with a simple method and consider a variant of the group-relative REINFORCE method (Yao et al., 2026). For each context $\mathbf{c}$, we sample a group of G responses, $(s_{K+1}^1, \ldots, s_{K+1}^G)$. For each response, we additionally record the corresponding generation trajectory. Let $r^g = R(s_{K+1}^g)$, then the group advantage is computed as $A^g = \frac{r(s_{K+1}^g) - \mathrm{mean}(r^{1:G})}{\mathrm{std}(r^{1:G})}$. The maximization objective is:

$$\mathcal{J}_{\mathrm{GRR}}(\theta) = \frac{1}{G} \sum_{g=1}^{G} A^g \Big[\sum_{k=0}^{K-1} \log \pi_\theta\big(a_k^g \mid s_k^g\big) + \frac{1}{L} \log \pi_\theta\big(a_K^g \mid s_K^g\big) \Big]. \tag{5}$$

The last step, $\pi_\theta\big(a_K^g \mid s_K^g\big)$, decodes all L token embeddings with a shared output matrix. Because this decoding step is comparatively easy to optimize, we scale its term by $\frac{1}{L}$ to slow its updates.

We also experimented with GRPO (Shao et al., 2024) with various settings and found that neither ratio clipping nor a KL penalty was necessary in our setting. This observation that the KL-penalty has no benefit is consistent with findings reported in DAPO (Yu et al., 2025). One important distinction is that our denoiser operates over a continuous state space of dimension dL, whereas an AR language model defines a discrete distribution over a vocabulary at each generation step. These differences in the underlying generation process and policy parameterization may call for different RL optimization strategies. We leave an investigation of such training techniques to future work.

4.2 CFRL-SVG-1D: Learn with pathwise RL with a critic

In our MDP formulation, only the last decoding step blocks gradient flow from the reward to the policy. We use stochastic value gradients (SVG) (Heess et al., 2015) and learn a critic to back-propagate the reward signal in denoising steps.

The value function $V_k^\star(s_k) = \mathbb{E}[R(s_{K+1}) \mid s_k]$ of a state s_k is the expected reward of all continuations from that state. Since the reward $R(s_{K+1})$ is binary, the value $V_k^\star(s_k)$ is the probability $p_\theta(R(s_{K+1})|s_k)$ of success from s_k. Note that the MDP problem here has a finite horizon so the value function is relatively easy to learn.

We train a critic V_ϕ to predict the value function $V_k^\star(s_k)$ from the state s_k. The critic is trained with a binary cross-entropy loss against the verifier reward $R(s_{K+1})$ at the end of the trajectory. The minimization objective is:

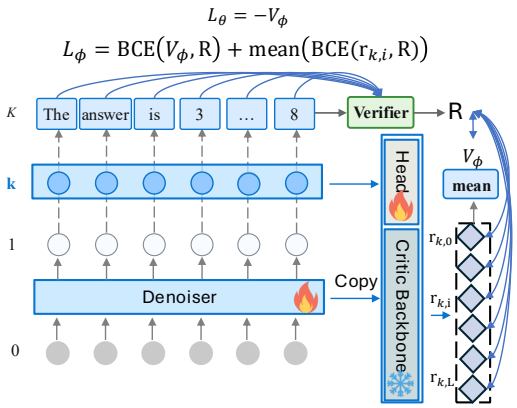


Figure 1: **Joint actor–critic training.** The denoiser serves as the actor. The critic uses a frozen copy of the denoiser backbone and a value head trained jointly with the actor.

$$\mathcal{L}_{\text{critic}}(\phi) = \mathbb{E}_{s_k, s_{K+1}}\Big[\mathrm{BCE}(V_\phi(s_k), R(s_{K+1}))\Big]. \tag{6}$$

Here $\mathrm{BCE}(V_\phi(s_k), R(s_{K+1}))$ is the binary cross-entropy loss of $V_\phi(s_k)$ with the true binary label $R(s_{K+1})$. The loss is consistent here: the true probability $p_\theta(R(s_{K+1})|s_k)$ receives the lowest expected loss.

The critic can learn the value function fairly accurately. Since the FM model is trained to generate accurate token embeddings that can be decoded without much ambiguity. We further freeze the decoder so that denoised token embeddings keep the same semantic. The critic can read out the information about the response from these embeddings at the later denoising stage.

Now we can directly backpropagate the reward information to denoising actions. We don't have reward in non-terminal states, so the maximization objective is

$$\mathcal{J}_{\text{SVG}}(\theta) = \mathbb{E}_{s_{k-1}, s_k}\Big[V_\phi(s_k)\Big] = \mathbb{E}_{\mathbf{z}_{t_{k-1}}, \mathbf{x}_{t_{k-2}}, \varepsilon}\Big[V_\phi(\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})\Big] \quad \text{with} \tag{7}$$

$$\mathbf{x}_{t_{k-1}} = \mathrm{d}_\theta(\mathbf{z}_{t_{k-1}}, \mathbf{x}_{t_{k-2}}), \quad \mathbf{z}_{t_k} = \lambda_k \mathbf{z}_{t_{k-1}} + \eta_k \mathbf{x}_{t_{k-1}} + \psi_k \varepsilon \tag{8}$$

In the calculation of $\nabla_\theta \mathcal{J}_{\text{SVG}}(\theta)$, we only use the gradient from one transition and do not take the derivative of state $s_{k-1} = (\mathbf{z}_{t_{k-1}}, \mathbf{x}_{t_{k-2}})$. Then the calcualtion of $\mathcal{J}_{\text{SVG}}(\theta)$ can be viewed reparameterization, which allows the gradient to propagate from V_ϕ to d_θ directly.

In our architecture design, the critic V_ϕ takes Transformer architecture from the trained EFL model but keeps its own copy. The Transformer takes $(\mathbf{z}_{t_k}, t_k, \mathbf{x}_{t_{k-1}}, \mathbf{c})$ as input and produce a sequence of L token-level hidden representations, $\mathbf{h} = (h_1, \ldots, h_L)$. Then mean pooling is applied over these hidden representations to give a sequence-level representation, which is then mapped to a scalar value prediction:

$$V_\phi(s_k) = \mathrm{sigmoid}(f_\phi(\mathrm{mean}(\mathbf{h}))) \tag{9}$$

Here f_ϕ is an MLP, whose parameters are included in ϕ.

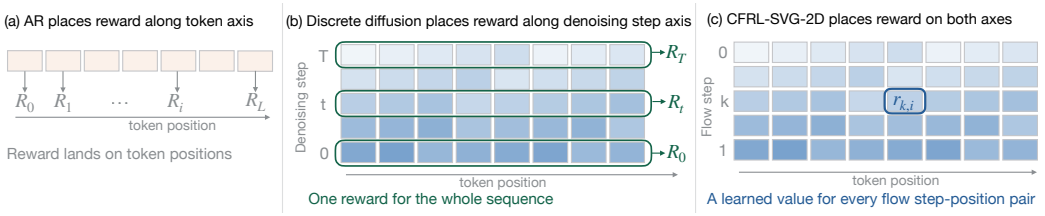


Figure 2: Where the reward lands: (a) on the token axis in an AR model; (b) on the step axis in RL for diffusion and flow models; (c) on both in CFRL-SVG-2D: $r_{k,i}$ for flow step k and position i, each an estimate of the whole response's outcome.

4.3 CFRL-SVG-2D: Token-wise reward feedback

A plain design of the critic architecture needs to aggregate transformer outputs $\mathbf{h} = (h_1, \ldots, h_L)$ into a single vector and then compute the value from it. The aggregation can be mean pooling (our method) or weighted average with attention weights from a special class token. In either case, the aggregation creates a bottleneck for information flow. From the perspective of gradient calculation, the signal from the reward is distributed to multiple tokens. In this subsection, we design a special architecture that allows reward signal to directly pass to single tokens.

In the new neural architecture, we compute a prediction of the reward $r_{k,i}$ for each token and aggregate these token-wise predictions to get the critic output.

$$V_\phi(s_k) = \text{sigmoid}\left(\sum_{i=1}^{L} f_\phi(h_i)\right), \quad r_{k,i} = \text{sigmoid}(f_\phi(h_i)). \tag{10}$$

Here h_i is the output embedding for token i when the transformer takes s_k as the input. The function $f_\phi(\cdot)$ is an MLP head.

During model training, we compare both the token prediction $r_{k,i}$ and the final prediction $V_\phi(s_k)$ against the reward $R(s_k)$. The loss for fitting the new critic is:

$$\mathcal{L}_{\text{2D}}(\phi) = \mathbb{E}_{s_k, s_{K+1}}\Big[\text{BCE}\big(V_\phi(s_k), R(s_{K+1})\big) + \tfrac{1}{L}\textstyle\sum_{i=1}^{L} \text{BCE}\big(r_{k,i}, R(s_{k+1})\big)\Big]. \tag{11}$$

In this new design, each token vector receives the reward signal directly.

4.4 Architecture and model training

For each task, we first perform supervised fine-tuning (SFT) on the pre-trained ELF model to obtain a task-specific policy. We then pre-train the critic on this policy's own rollouts: we generate full trajectories with the task-SFT sampler, evaluate each decoded response with the task verifier to obtain its terminal reward $R(s_{K+1})$, and pair a subset of intermediate states s_k of every trajectory with that reward. The critic starts from a copy of the task-SFT denoiser backbone with an added value head f_ϕ, and backbone and head are both trained on these state–reward pairs with the BCE loss (6) or (11).

We then train the policy and the critic jointly, alternating one policy update with a few critic updates (Algorithm 1); the critic backbone stays frozen and only its head is updated. A policy update picks one denoising step from the middle of the trajectory, where the state already carries information about the outcome but the generation has not yet committed to it. We roll out the current policy up to that step without gradients, then differentiate through the single denoiser call at that step and update θ to increase V_ϕ: the policy learns to move the trajectory toward states of higher predicted reward. A critic update uses fresh rollouts of the updated policy, decoded and verified, and refits the head to their states and outcomes.

After RL, we add an optional self-training loop. Each round samples the RL policy on unlabelled problems, keeps the answers the verifier accepts, and fine-tunes the model on them with ELF's supervised loss. Since that loss trains the whole denoiser, the gain becomes a property of the model rather than of the sampler used in RL, so faster samplers benefit too.

Algorithm 1 Joint training of the policy and critic.

Require: policy parameters θ; pre-trained critic V_ϕ with trainable head parameters ϕ on a fixed copy of the denoiser; verifier R; sampler (γ, K); window $\mathcal{W}$ with top step k_{hi}; buffer $\mathcal{B}$ and its size $N_{\mathcal{B}}$; iterations T, prompts per iteration B, critic minibatch B' and steps m_{c}, AUC check every n_q iterations on the newest N_q states with floor a_0 and m'_{c} extra steps; learning rates η_θ, η_ϕ

1: **for** iteration $= 1, \ldots, T$ **do**
2: **Actor.** For B prompts and one $k \sim \mathcal{U}(\mathcal{W})$: roll out under (γ, K) to s_{k-1} without gradients, take the denoiser call at s_{k-1} with gradients to form s_k, and set $\theta \leftarrow \theta + \eta_\theta \,\mathrm{AdamW}\big(\nabla_\theta \frac{1}{B} \sum_{g=1}^{B} V_\phi(s_k^g)\big)$ (eq. 9; eq. 10; gradient-norm clipped)
3: **Buffer.** Roll out B fresh prompts under (γ, K) to the end, decode, verify; push each $(s_{k_{\text{hi}}}, R(S_{K+1}))$ into $\mathcal{B}$, keeping the newest $N_{\mathcal{B}}$
4: **Critic.** m_{c} steps $\phi \leftarrow \phi - \eta_\phi \,\mathrm{AdamW}(\nabla_\phi \mathcal{L}_{2\text{D}})$ on minibatches of B' states of $\mathcal{B}$; every n_q iterations, m'_{c} more if the AUC of V_ϕ on the newest N_q states is below a_0 (eq. 11; eq. (6))
5: **end for**

Model	Countdown	Sudoku	K&K	Game24
GPT-2 (124M, autoregressive)				
released model	0.0	0.0	0.0	0.0
+ task SFT	45.1	0.8	92.7	28.0
+ token GRPO/PPO	60.9 (+15.8)	0.8 (+0.0)	92.8 (+0.1)	46.7 (+18.7)
LLaDA-8B-Instruct (masked diffusion)				
released model	1.4	0.0	5.4	1.6
+ task SFT	12.1	0.5	61.4	15.9
+ diffu-GRPO	14.9 (+2.8)	0.8 (+0.3)	63.2 (+1.8)	18.8 (+2.9)
ELF-B (105M, continuous flow)				
released model	0.0	0.0	0.0	0.0
+ task SFT (base)	42.8	64.0	68.2	44.0
+ CFRL-GRR	42.6 (−0.2)	67.9 (+3.9)	71.4 (+3.2)	47.4 (+3.4)
+ CFRL-SVG-1D	47.4 (+4.6)	64.1 (+0.1)	76.5 (+8.3)	51.2 (+7.2)
+ CFRL-SVG-2D	**49.7** (+6.9)	**68.7** (+4.7)	**77.5** (+9.3)	**58.2** (+14.2)

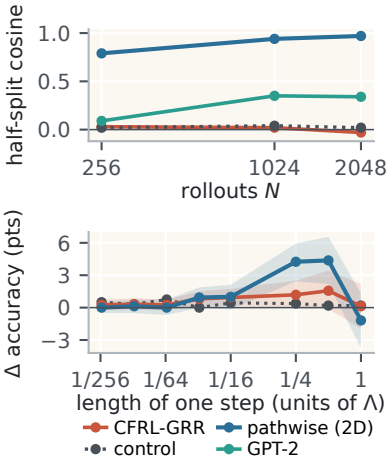


Table 1: **Main results. Left:** accuracy (%) under the deployment sampler, with the gain over task SFT in parentheses; shaded: recommended; bold: best flow row; blue: best per task. **Right:** gradient estimates on Game24 base rollouts: half-split cosine (top) and accuracy after one step (bottom).

5 Experiments

We evaluate CFRL on four reasoning tasks with program verifiers. **Countdown**: combine four or five given integers with arithmetic operations to reach a target. **Sudoku**: complete a 9×9 grid from its givens. **Knights-and-Knaves** (K&K): from the statements of four to six characters who each always tell the truth or always lie, decide who is which. **Game24**: reach 24 from four given numbers, each used once.

Setup. Every model is first fine-tuned on each task (task SFT); the ELF-B (105M) task-SFT model is the base of all flow runs. Held-out prompts are split into a validation half, on which checkpoints are selected, and a test half, on which every accuracy is reported. The deployment sampler is $\gamma 56@64$ (churn γ=56, K=64 steps; $\gamma 28@32$ on Sudoku) and the cheap sampler $\gamma 4@16$, both through the same denoiser. As references from other architectures we train GPT-2 (Radford et al., 2019) with token-level GRPO/PPO and LLaDA-8B-Instruct (Nie et al., 2025) with d1's diffu-GRPO (Zhao et al., 2025) on the same data. More details are in Appendices A and D.

Main results. Table 1 compares the three CFRL variants with the two references. Every variant raises the accuracy of the base model over the four tasks; CFRL-GRR, which needs no critic, provides the least improvement and CFRL-SVG-2D is the best flow row on every task. Across architectures, GPT-2 leads on Countdown and K&K, which require longer derivations, whereas the flow model leads on Game24 and Sudoku, the latter of which neither reference model learns. The 8B masked-diffusion model stays far below the flow model on every task, and CFRL-SVG-2D gains

Critic	Pearson	pick@V_ϕ	max.	final
critic input and initialisation				
fine-tuned copy, reads $(y_k, x_{t_{k-1}})$	0.93	0.94	**58.2**	**57.5**
without $x_{t_{k-1}}$ in the input	0.53	0.59	47.0	12.9
copy not fine-tuned	0.65	0.82	51.7	13.4
random initialisation	0.43	0.67	37.9	4.1
same critic, trained differently in RL				
copy also trained in RL	0.93	0.94	49.7	49.3
head not refit in RL	0.93	0.94	49.3	0.5

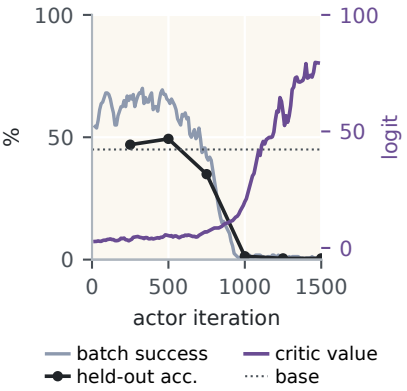


Table 2: **Critic design. Left:** the 2D critic on Game24 (shaded) and variants. Offline: correlation of the value with the outcome, and success of its top-ranked rollout (chance 0.55). Online: RL accuracy (%), max. and final. **Right:** the last row over training.

more than diffu-GRPO on every task. Two patterns emerge. Score-function methods appear capped: CFRL-GRR on the flow model and diffu-GRPO on the 8B masked model improve by only a few points on any task, regardless of model size, whereas the critic-based variants achieve substantially larger, task-dependent gains. Critic resolution also matters: the 1D critic does not consistently outperform CFRL-GRR, whereas the 2D critic outperforms both methods across all four tasks. The gains are substantial: CFRL-SVG-2D improves Game24 by 14.2 points and reaches 68.7 on Sudoku, making the 105M flow model the strongest model on both tasks.

Effect of the gradient estimator. The right panels of Table 1 test the two estimators at the same point, before any training moves the policy. We take the Game24 task-SFT model, sample N complete rollouts under the deployment sampler and score them with the verifier. On these rollouts we form, at one fixed flow step (24 of 64), one gradient per rollout with respect to both the score-function gradient of CFRL-GRR and the pathwise gradient of 2D critic. A control sets every advantage to 1, which gives an estimator whose expectation is exactly 0. GPT-2 is measured the same way on its own task-SFT model, with its token-level gradient. *Top:* the N rollouts are split at random into two halves, the gradients of each half are averaged, and the cosine between the two averages is plotted. Two halves of an estimator with a non-zero expectation agree more as N grows, and the cosine rises towards 1; two halves of noise are unrelated, and the cosine stays near 0. *Bottom:* from the base model we take one AdamW-normalized parameter update along each estimator's mean gradient over 1,024 rollouts; the unit Λ is the distance in parameter space that the 2D-critic policy of Table 1 travels from the base over its training. Each updated model is evaluated on the held-out prompts with the same random numbers as the base, and the change in accuracy is shown with a paired 95% interval; a random direction of the same lengths is the control.

The score-function gradient of the flow model does not separate from the zero-mean control at any N up to 2,048, whereas the pathwise gradient agrees with itself from 256 rollouts on. GPT-2's token gradient separates from the control at 1,024 rollouts: indicating that the limitation lies not only in the score-function estimator itself but also in the flow model's action space. The one-step test says the same in accuracy: a single step along the pathwise direction, at a quarter or a half of Λ, raises held-out accuracy by about 4 points, whereas the score-function step and the random step stay within about 1 point of the base at every length. The two panels tell a consistent story. Relative to its mean, the flow model's score-function gradient has high variance that independent estimates disagree and a single update has no measurable effect, even at practical rollout budgets. By contrast, pathwise estimates agree within a few hundred rollouts, and one update recovers a substantial fraction of the full-training gain. Because this difference appears before training, it reflects the estimators rather than their training recipes.

Effect of the critic design. Table 2 varies one critic component at a time. Removing the estimate $x_{t_{k-1}}$ or omitting backbone fine-tuning reduces offline outcome prediction and causes the policy to improve briefly before collapsing; random initialization never surpasses the base. Thus, the outcome

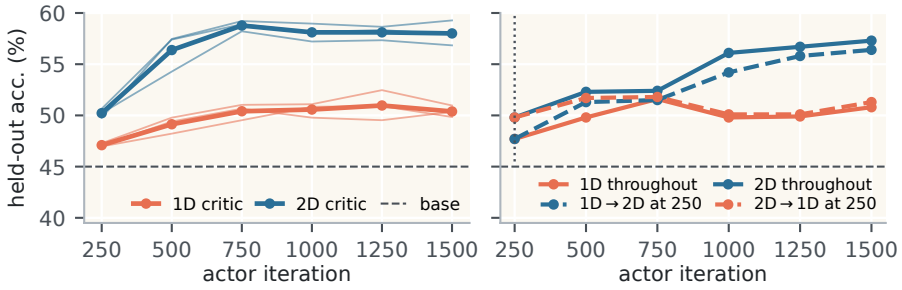


Figure 3: **1D versus 2D critic.** Game24 accuracy (%) over training. **Left:** 1D- and 2D-critic runs (thin) and their means (thick). **Right:** critics exchanged at iteration 250 (dashed) or not (solid); colour: the critic in charge afterwards.

Table 3: **Effect of the objective, the pooling and the sampler** on Game24: change in accuracy (%) at the best and the final checkpoint relative to the full recipe; blue: the largest loss.

Choice	**Variant**	**Δ max.**	**Δ final**
objective	actor ascends V_ϕ instead of $\mathrm{logit}(V_\phi)$	-4.0	-6.1
pooling	actor ascends the minimum over positions instead of the mean	-3.1	-2.5
sampler	trained under the cheap sampler, read at deployment	-9.4	-9.4

is encoded in $x_{t_{k-1}}$ and requires pretrained denoiser features to interpret. Updating the backbone during RL provides no benefit, so joint training refits only the head. Without refit, the actor exploits the critic: its predicted value approaches certainty while the same batch's true success falls to zero.

1D versus 2D critic. Figure 3 follows the 1D- and 2D-critic runs of Table 1 on Game24; the two recipes differ only in the critic's head, one reading of the pooled features against one reading per position. *Left:* the 2D critic rises faster from the first checkpoint on and plateaus well above the 1D critic, which levels off early. *Right:* to tell whether this is the critic's doing or a better start, the two critics are exchanged at iteration 250, while the two policies are still close. Each exchanged run ends where the critic now in charge takes it, whichever policy it inherited: the run that switches to the 2D critic climbs to the 2D level, the run that switches to the 1D critic falls to the 1D level. The advantage is therefore made during joint training, by the gradient the critic hands the actor, not by a head start; one reading per position steers the policy further than one reading of the whole state.

Effect of the objective, the pooling and the sampler. Table 3 takes the full 2D-critic recipe on Game24 and changes one choice at a time, keeping the base, the critic, the window and the budget fixed; each row reports how much the change costs at the best checkpoint and at the last one. Each choice matters. Ascending the success probability instead of its log-odds lowers both numbers, and the final accuracy more than the best: the probability's gradient fades on states whose outcome is nearly decided, which is exactly where the log-odds still amplify it, so the policy stops learning from its most informative states and levels off earlier. Pooling the positions by their minimum instead of their mean costs less: the gradient of a minimum reaches only the single lowest position of a state, so each update teaches fewer positions, but the positions it teaches are the right ones. Training under the cheap 16-step sampler and evaluating under the deployment sampler costs the most, two thirds of the full gain: the critic's value is defined by the sampler that produces the states, and a policy improved for one sampler is not thereby improved for another. This is why actor and critic are trained under the sampler used at deployment, and why carrying the gain to cheaper samplers needs the loop of the next paragraph rather than a change of training sampler.

Effect of the amortization loop. Table 4 starts from the 2D-critic checkpoint of Table 1 on Countdown and K&K and applies the amortization loop (Algorithm 2) with a pool of 50k unlabelled problems: each round samples the policy on the pool, keeps the responses the verifier accepts, and fine-tunes the model on them with ELF's supervised loss, for 4 rounds on Countdown and 5 on K&K. The left table reads each model under the cheap 16-step sampler and the 64-step deployment

Model	Countdown		K&K	
	cheap	deploy	cheap	deploy
base	23.8	42.8	58.6	68.2
2D-critic ckpt (loop start)	24.3	50.5	68.8	79.8
+ amortization loop	**45.6**	51.4	**93.8**	**99.1**

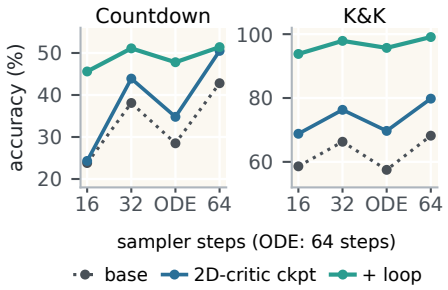


Table 4: **Amortization loop. Left:** accuracy (%) under the cheap and the deployment sampler before and after the loop; bold: over 1.0 point above the loop start; shaded: the full pipeline. **Right:** the same three models across sampler cost.

sampler. RL alone improves the deployment sampler, by about 8 points on Countdown and 12 on K&K, but on Countdown it leaves the cheap sampler where the base was. The loop moves the cheap sampler most: on both tasks it ends above the accuracy the base had under the deployment sampler, a gain over the loop start of about 20 points on Countdown and 25 on K&K, while the deployment sampler holds on Countdown and keeps rising on K&K to 99.1, above every model of Table 1. The reason is where the two stages write their gains: RL changes the denoiser only along the transitions of the sampler it trains under, whereas ELF's supervised loss trains the denoiser at every noise level, so fine-tuning on the policy's own verified samples spreads what RL taught the policy over the whole denoiser, and every sampler reads it.

Effect of the loop across sampler cost. The right panel of Table 4 reads the same three models under four samplers ordered by cost: 16 and 32 stochastic steps, 64 deterministic ODE steps, and the 64-step deployment sampler. The base and the RL policy trace steep curves: accuracy climbs with the number of steps, and on Countdown RL's gain sits almost entirely at the expensive end, the sampler it was trained under, while on K&K RL lifts every sampler by a similar amount. After the loop both curves are nearly flat: accuracy varies by 5 to 6 points across the four samplers. The deployment sampler is still the best, but the gap between the cheapest and the most expensive shrinks from about 20 points to about 6 on Countdown and from about 10 to about 5 on K&K. In practice this means the 16-step sampler delivers nearly the deployment accuracy at a quarter of the denoiser calls: the loop turns a gain that belonged to one sampler into a property of the model.

6 Conclusion and Limitations

We propose CFRL, RL post-training for continuous flow language models in three variants and an amortisation loop. CFRL-GRR is the score-function estimator on the chain's exact density; CFRL-SVG-1D and CFRL-SVG-2D are stochastic value gradients with one value per state, or per state and token position. CFRL-GRR gains little: one reward bit is spread over the chain's noise coordinates, and its gradient agrees with itself no better than a zero-mean control, whereas GPT-2's gradient, with one score term per token decision, does. Differentiating the chain through a critic removes the noise factor. The critic must read the denoiser's estimate through a copy of the pre-trained denoiser and be refit after every actor step, or it is exploited. With one value per position the policy rises faster and further than with a scalar value, and exchanging the critics mid-training shows that the critic in charge decides. The loop carries the gain into a cheap chain.

Limitations. Our results cover one 105M flow model and four synthetic reasoning tasks with exact verifiers; larger models, natural-language tasks and learned rewards remain untested. The RL gain is tied to the sampler used in training, and carrying it to cheaper samplers requires the amortization loop and a pool of unlabelled problems. Finally, the autoregressive reference still leads on two of the four tasks, so the advantage of a flow model with RL is task-dependent rather than uniform.

REPRODUCIBILITY STATEMENT

Code, configurations and job scripts for every run in this paper will be released; all datasets are generated programmatically or drawn from public collections (Appendix A) and the exact splits, seeds and verifier implementations are included. Appendix A defines the samplers, Appendix D lists every hyperparameter, the evaluation protocol and the compute used; Appendix E reports the full trajectories and the supporting results behind the numbers in the main text.

AI USE STATEMENT

In this work, we used generative AI tools for writing assistance (editing prose), for writing and debugging experiment code. We have not used generative AI tools for generating research ideas that are presented as contributions, nor for producing any experimental result, number or figure that was not computed by the released code. We have reviewed all AI-assisted work: all code was tested against the reported runs, all citations were verified against the cited sources, and all claims were checked against the experiment logs. The authors take full responsibility for all content of this paper.

REFERENCES

Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. DPOK: Reinforcement learning for fine-tuning text-to-image diffusion models. *arXiv preprint arXiv:2305.16381*, 2023.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Xu Han, Jiajing Hu, and Liping Liu. Self-consistent flow: Unifying velocity and endpoint prediction for rectified flow models. *Transactions on Machine Learning Research*, 2026.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. ELF: Embedded language flows. *arXiv preprint arXiv:2605.10938*, 2026.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Jaewoo Lee, Minsu Kim, Sanghyeok Choi, Inhyuck Song, Sujin Yun, Hyeongyu Kang, Woocheol Shin, Taeyoung Yun, Kiyoung Om, and Jinkyoo Park. Diffusion alignment as variational expectation-maximization. In *International Conference on Learning Representations*, 2026.

Junzhe Li, Yutao Cui, Tao Huang, Yinping Ma, Chun Fan, Miles Yang, and Zhao Zhong. MixGRPO: Unlocking flow-based GRPO efficiency with mixed ODE-SDE. *arXiv preprint arXiv:2507.21802*, 2025.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 36115–36125, 2026.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM improves controllable text generation. *arXiv preprint arXiv:2205.14217*, 2022.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. *arXiv preprint arXiv:2505.05470*, 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Wang, and Angjoo Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

OpenAI. Learning to reason with LLMs. `https://openai.com/index/learning-to-reason-with-llms/`, 2024.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*, 2024.

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

Kulin Shah, Nishanth Dikkala, Xin Wang, and Rina Panigrahy. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. *arXiv preprint arXiv:2409.10502*, 2024.

Yawen Shao, Jie Xiao, Kai Zhu, Yu Liu, Hongchen Luo, Xueyang Fu, Yang Cao, Wei Zhai, and Zheng-Jun Zha. Back on track: Aligning rewards and states for reasoning in diffusion large language models. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Richard S Sutton, Andrew G Barto, and Andrew Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. On memorization of large language models in logical reasoning. *arXiv preprint arXiv:2410.23123*, 2024.

Zeyue Xue, Jie Wu, Yu Gao, Fangyuan Kong, Lingting Zhu, Mengzhao Chen, Zhiheng Liu, Wei Liu, Qiushan Guo, Weilin Huang, and Ping Luo. DanceGRPO: Unleashing GRPO on visual generation. *arXiv preprint arXiv:2505.07818*, 2025.

Chaorui Yao, Yanxi Chen, Yuchang Sun, Yushuo Chen, Wenhao Zhang, Xuchen Pan, Yaliang Li, and Bolin Ding. Group-relative REINFORCE is secretly an off-policy algorithm: Demystifying some myths about GRPO and its friends. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/6cad062f78120689e8a678b8af9c6d92-Abstract-Conference.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A DATASETS AND VERIFIERS

All tasks are generated programmatically; every prompt has a program verifier that decides correctness of a model output without reference to the gold answer, so rewards are exact and label-free.

Countdown. Following the Countdown formulation of Gandhi et al. (2024), a prompt is four or five integers and a target; the model emits a derivation as a sequence of binary operations (`12-5=7 , 4*7=28 , 28-3=25` for the numbers 12, 5, 4, 3 and the target 25). The verifier parses the derivation, checks that every operation is arithmetically correct, that each input number is used at most once, and that the last result equals the target. The supervised set is 500k examples, one gold derivation each; the loop's pool is a disjoint set of 50k unlabelled problems; evaluation uses 2,000 held-out problems. Sequence length is 72 embedding positions (16 for the condition).

Sudoku. Puzzles are drawn from a public collection of rated Sudoku puzzles (with solutions); the model reads the 81-cell puzzle (blanks as $\star$) and emits all 81 cells. The verifier checks that the output is a valid grid consistent with the givens; since solutions are unique this equals exact match. The supervised set is 400k puzzles, evaluation 2,000 held-out puzzles; the "fresh puzzle" pool for the loop is 50k further puzzles with no overlap with training, validation or test. Sequence length is 164 (82 condition, 82 target).

Knights-and-Knaves. Following Xie et al. (2024), each puzzle has $n \in \{4, 5, 6\}$ characters who each make one statement ("A: C knave", "B: A same C", "C: D diff B"); the model outputs an assignment ("A knight B knave ..."). Our generator enumerates assignments and keeps only puzzles with a unique solution; the verifier re-parses the statements and checks that the assignment satisfies every one. Training uses 300k puzzles; evaluation 2,000; the loop's first round samples the 50k training puzzles without their labels, and every later round a fresh 50k pool disjoint from training, validation and each other. The supervised model reaches 96.1/93.3/99.4/99.7 (ODE-64/$\gamma 4$@16/$\gamma 28$@32/$\gamma 56$@64) at epoch 8, so the RL base is the epoch-4 checkpoint (57.5/58.6/66.3/68.2) and the epoch-8 model is the supervised oracle. Sequence length is 96 (56 condition).

Game24. Following the Game of 24 of Yao et al. (2023), reach 24 from four numbers in $\{1, \ldots, 13\}$ with each number used once. Only 1,345 unordered number sets are solvable, so we split by set: the training set contains 13,150 orderings of 1,076 sets with one gold derivation each, the loop pool a further 4,693 orderings of the training sets, and the evaluation set 1,601 orderings of the 269 remaining sets, which the model never sees in any order. The verifier is the Countdown verifier with target 24. Sequence length is 72 (16 condition). The supervised model is initialised from the Countdown model and trained for 40 epochs; held-out accuracy peaks at epoch 20 (36.2/35.6/43.5/44.0 at ODE-64/$\gamma 4$@16/$\gamma 28$@32/$\gamma 56$@64) and falls to 30.4/29.0/32.4/34.1 by epoch 40, so the epoch-20 checkpoint is the RL base and there is no supervised oracle above it.

Samplers. ODE-64 is the 64-step deterministic sampler of Hu et al. (2026). $\gamma 56$@64 denotes the stochastic sampler of equation 2 with $K{=}64$ uniform steps and noise re-injection scale (churn) $\gamma{=}56$, and likewise for the other values; $\gamma 56$@64, $\gamma 28$@32 and $\gamma 14$@16 all have per-step retention $\alpha \approx 0.125$, which we found to be the best sampling configuration independently of K (more steps still help at fixed α); $\gamma 4$@16 is a cheap low-churn configuration. All evaluations use classifier-free guidance 2 and self-conditioning guidance 1 as in the ELF release. Throughout, the *deployment sampler* is $\gamma 56$@64 ($\gamma 28$@32 on Sudoku) and the *cheap sampler* is $\gamma 4$@16 ($\gamma 14$@16 on Sudoku).

B DERIVATIONS

B.1 THE STOCHASTIC SAMPLER AND ITS TRANSITION DENSITY

The sampler of (2) is the stochastic sampler released with ELF (Hu et al., 2026). At every step it renoises the current state, queries the denoiser at the re-noised state and at the correspondingly lowered time, and moves part of the way from the re-noised state towards the estimate; the coefficients λ_k, η_k and ψ_k of (2) follow from these three operations, as derived below. The state of the MDP,

$s_k = (\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})$, is the pair the denoiser reads at its $(k+1)$-th call: the re-noised embeddings and the previous estimate carried by self-conditioning.

Derivation from the released implementation. The official implementation of ELF uses the following stochastic sampler with churn γ (Hu et al., 2026). The implementation is slightly different from Algorithm 6 in the original paper. Here we provide the derivation based on the code, and the notations are mostly consistent with the original paper.

At step k, let the current time be t_{k-1} and the step size be h, so that $t_k = t_{k-1} + h$. Let $\gamma \geq 0$ be the noise re-injection scale, and $t_{\text{eps}} = 0.05$ is the lower bound used in the velocity conversion. Define

$$\alpha = \text{clip}(1 - \gamma h,\ 0,\ 1)\,, \quad \tilde{t}_{k-1} = \alpha t_{k-1}, \quad D = \max\big(1 - \tilde{t}_{k-1},\ t_{\text{eps}}\big) = \max(1 - \alpha t_{k-1},\ t_{\text{eps}})\,. \tag{12}$$

With $\epsilon \sim \mathcal{N}(0, c_\sigma^2 I)$, one step performs

$$\tilde{z}_{k-1} = \alpha\, z_{k-1} + (1-\alpha)\,\epsilon, \tag{13}$$

$$x_{k-1} = \mathrm{d}_\theta\big(\tilde{z}_{k-1},\ \tilde{t}_{k-1}, x_{k-2}, c\big)\,, \tag{14}$$

$$v_{k-1} = \frac{x_{k-1} - \tilde{z}_{k-1}}{D}, \tag{15}$$

$$z_k = \tilde{z}_{k-1} + \big(t_k - \tilde{t}_{k-1}\big)\, v_{k-1}. \tag{16}$$

$$\tilde{z}_k = \alpha\, z_k + (1-\alpha)\,\epsilon. \tag{17}$$

Note that x_{k-1} is the prediction at the re-noised input $(\tilde{z}_{k-1}, \tilde{t})$, not at (z_{k-1}, t). At each step, the code record (z_k, x_{k-1}) at each step. At $k = 1$, we use $z_0 = \varepsilon$ and $x_{-1} = 0$.

In our derivation, we use $(\tilde{z}_k, x_{k-1})$ as the state, which is $(\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})$ in our notation. We write:

$$\tilde{z}_k = \lambda_k \tilde{z}_{k-1} + \eta_k x_{k-1} + \psi_k \epsilon \tag{18}$$

with

$$\lambda_k = \alpha\left(1 - \frac{t + h - \tilde{t}}{D}\right), \quad \eta_k = \frac{\alpha(t + h - \tilde{t})}{D}, \quad \psi_k = 1 - \alpha, \tag{19}$$

for $k = 1, \ldots, K-1$. When $k = K$, we set $\alpha = 1$ and eliminate the re-noising step,

$$\lambda_K = \left(1 - \frac{t + h - \tilde{t}}{D}\right), \quad \eta_K = \frac{(t + h - \tilde{t})}{D}, \quad \psi_K = 0, \tag{20}$$

so that $\tilde{z}_K = z_K$, matching the ELF implementation.

To derive these, substitute equation 15 into equation 16 and group the terms by $\tilde{z}_{k-1}$ and x_{k-1}:

$$\begin{aligned} z_k &= \tilde{z}_{k-1} + \frac{t + h - \tilde{t}}{D}\,(x_{k-1} - \tilde{z}_{k-1}) \\ &= \left(1 - \frac{t + h - \tilde{t}}{D}\right)\tilde{z}_{k-1} + \frac{t + h - \tilde{t}}{D}\, x_{k-1}. \end{aligned}$$

Substituting this into equation 17 gives

$$\tilde{z}_k = \alpha\left(1 - \frac{t + h - \tilde{t}}{D}\right)\tilde{z}_{k-1} + \frac{\alpha\,(t + h - \tilde{t})}{D}\, x_{k-1} + (1-\alpha)\,\epsilon,$$

from which λ_k, η_k and ψ_k are read off. The noise enters only through the re-noising after the update, so ψ_k carries no factor from the step.

Two consequences are used in the main text. First, the time fed to the denoiser together with $\mathbf{z}_{t_k}$ is the lowered time $\tilde{t}_k = \alpha\, t_k$ of the re-noised state, which the main text writes as t_k in (1) for brevity. Second, given s_{k-1}, the next embeddings are Gaussian: for $k = 1, \ldots, K-1$ and every generated position $i \in \mathcal{G}$,

$$p_\theta\big(\mathbf{z}_{t_k}[i] \,\big|\, s_{k-1}\big) = \mathcal{N}\big(\mathbf{z}_{t_k}[i];\ \lambda_k \mathbf{z}_{t_{k-1}}[i] + \eta_k\, \mathbf{x}_{t_{k-1}}[i],\ \psi_k^2 c_\sigma^2 I_d\big), \qquad \mathbf{x}_{t_{k-1}} = \mathrm{d}_\theta(s_{k-1}), \tag{21}$$

independently across the generated positions $i \in \mathcal{G}$ (the response positions), while the prompt positions are held fixed; the last step has $\psi_K = 0$ and is deterministic, and the decoder gives $\pi_\theta(a_K \mid s_K) = \prod_i \mathrm{softmax}(W\mathbf{z}_{t_K}[i])_{w_i}$. The mean of (21) depends on θ only through the estimate $\mathrm{d}_\theta(s_{k-1})$, so the θ-dependent part of $\log \pi_\theta(a_k \mid s_k)$ in (5) is the log-density (21) of the next state, and the trajectory log-likelihood is the sum of these terms and the decoder term, with no bound involved. We verified this on CPU: recomputing the log-likelihood of a stored trajectory under the parameters that produced it agrees with the value recorded at rollout to relative error 7×10^{-5} (denoising steps) and 10^{-3} (decoder). One caveat matters in practice: $\gamma\Delta_k = 1$ makes $\alpha = 0$, in which case the state is replaced by fresh noise and every parameter gradient of that step vanishes, so we require $\alpha > 0$.

B.2 Score-function and pathwise gradients of one transition

From (21), the score of transition k is

$$\nabla_\theta \log p_\theta(\mathbf{z}_{t_k} \mid s_{k-1}) = \frac{\eta_k}{\psi_k^2 c_\sigma^2} \sum_i \epsilon_k[i]^\top \frac{\partial\, \mathrm{d}_\theta(s_{k-1})[i]}{\partial\theta}, \tag{22}$$

where $\epsilon_k \sim \mathcal{N}(0, c_\sigma^2 I)$ is the noise injected at that step: the advantage multiplies the injected noise before it passes through the denoiser's Jacobian. Summed over the steps, (22) is the transition part of the gradient of (5). A trajectory therefore contributes $(K-1)\cdot|\mathcal{G}|\cdot d$ terms, each proportional to an independent Gaussian coordinate and weighted by an advantage that depends on the trajectory only through its final reward. A coordinate adds to the expected gradient only through the covariance of the reward with that coordinate's noise, but it adds to the variance regardless; with N trajectories per update the signal grows like N times these covariances and the noise like $\sqrt{N(K-1)|\mathcal{G}|d}$. In an autoregressive model the analogous sum has one term per token and every term is a decision the reward depends on; in the flow model the covariances are, as far as we can measure, undetectable (the half-split cosine of Table 1 stays at the level of a zero-mean control up to $2{,}048$ rollouts), which is the origin of the gap between the two estimators in Table 1. A finer advantage, per step or per position, changes which terms receive weight but leaves the factor $\epsilon_k/(\psi_k c_\sigma)$ in place.

The pathwise estimator of $\mathcal{J}_{\mathrm{SVG}}$ uses the same reparameterisation, $\mathbf{z}_{t_k} = \lambda_k \mathbf{z}_{t_{k-1}} + \eta_k\, \mathrm{d}_\theta(s_{k-1}) + \psi_k\epsilon_k$ and $\mathbf{x}_{t_{k-1}} = \mathrm{d}_\theta(s_{k-1})$, and differentiates the critic through it with s_{k-1} and ϵ_k held fixed:

$$\nabla_\theta V_\phi(s_k) = \Big[\eta_k\, \nabla_{\mathbf{z}} V_\phi(s_k) + \nabla_{\mathbf{x}} V_\phi(s_k)\Big] \frac{\partial\, \mathrm{d}_\theta(s_{k-1})}{\partial\theta}, \tag{23}$$

the first channel through the re-noised embeddings, the second through the estimate that self-conditioning feeds to the next call. No noise factor appears: the weight on each coordinate is the critic's derivative there, which vanishes on every coordinate the predicted outcome does not depend on. One random k per update suffices (next section), so the sampler need not be differentiated end to end and memory is constant in K.

B.3 Why differentiating one transition suffices

Fix a prompt and write $V_k^\star(s) = \mathbb{E}[R(s_{K+1}) \mid s_k = s]$ as in Section 4.2. Since the objective (4) is $J(\theta) = \mathbb{E}[V_k^\star(s_k)]$ for every k by the tower property, and s_k is produced from s_{k-1} by the call $\mathrm{d}_\theta(s_{k-1})$ and the injected noise, the value satisfies the one-step recursion

$$V_{k-1}^\star(s) = \mathbb{E}_{\epsilon_k}\big[V_k^\star(s_k)\big], \qquad s_k = \big(\lambda_k \mathbf{z}_{t_{k-1}} + \eta_k\, \mathrm{d}_\theta(s) + \psi_k\epsilon_k,\ \mathrm{d}_\theta(s)\big), \qquad s = (\mathbf{z}_{t_{k-1}}, \mathbf{x}_{t_{k-2}}), \tag{24}$$

and $V_k^\star$ depends on θ only through the calls at s_k and later. The parameters enter J once per denoiser call. Writing $\nabla_\theta^{(k)}$ for the gradient taken through the call at s_{k-1} alone, with every other call held fixed, the chain rule gives

$$\nabla_\theta J = \sum_{k=1}^{K+1} \nabla_\theta^{(k)} J, \qquad \nabla_\theta^{(k)} J = \mathbb{E}\big[\nabla_\theta^{(k)} V_k^\star(s_k)\big] \quad (1 \le k \le K-1), \tag{25}$$

where the second equality exchanges derivative and expectation, and $\nabla_\theta^{(k)} V_k^\star(s_k)$ is (23) with $V_k^\star$ in place of V_ϕ. The exchange needs $V_k^\star$ to be locally Lipschitz in s_k with an integrable gradient. For

$k \leq K-1$ this holds in the first argument because, by (24), $V_k^\star$ is a Gaussian average of $V_{k+1}^\star$ over the noise injected at step $k{+}1$; for the second argument, the estimate carried by self-conditioning, we assume the same regularity. It fails at the last call: $V_K^\star$ is the reward of the decoded response, an $\arg\max$ followed by a program, hence piecewise constant with zero gradient almost everywhere, so a pathwise estimator sees no signal through the final decode. This is why the pathwise route needs a value on *intermediate* states rather than the reward on the final one, and why $\mathcal{J}_{\mathrm{SVG}}$ is written with a critic at s_k. Each term of (25) is the one-step value gradient of SVG (Heess et al., 2015) with the sampler as the transition model; drawing one k at random and applying (23) is an unbiased estimate of the window's share of the sum, up to the constant $|\mathcal{W}|$, when $V_\phi = V_k^\star$.

The first channel of (23) and the score-function term (22) estimate the same quantity. By Gaussian integration by parts, $\mathbb{E}[\psi(\mu{+}\sigma\epsilon)\,\epsilon] = \sigma\,\mathbb{E}[\nabla\psi(\mu{+}\sigma\epsilon)]$ for standard Gaussian ϵ and locally Lipschitz ψ with integrable gradient; applied to $\psi = V_k^\star(\cdot, \mathbf{x}_{t_{k-1}})$ it turns the factor $V_k^\star(s_k)\,\epsilon_k/(\psi_k c_\sigma)$ of the score-function form into the factor $\nabla_{\mathbf{z}} V_k^\star(s_k)$ of the pathwise form, conditionally on s_{k-1}. The two have the same expectation and differ only in variance, coordinate by coordinate: the score-function factor at a coordinate has variance of order $\mathbb{E}[(V_k^\star)^2]/(\psi_k c_\sigma)^2$ whether or not the coordinate affects the reward, whereas the pathwise factor is identically zero on every coordinate the value does not depend on. The second channel of (23) has no counterpart in (5), which holds the stored estimates fixed. With a learned critic in place of $V_k^\star$ the estimate is biased by the difference of the two slopes on the states the current policy visits; both the visited states and $V_k^\star$ itself move with θ, which is why the critic is refit on the current policy's own rollouts after every policy update (Section 4.3).

B.4 Head before or after the mean

The 1D critic (9) applies the head to the mean of the position features, the 2D critic (10) applies it to every position and pools the readings. The following statement makes precise the bottleneck of Section 4.3: with the features held fixed, a head on the mean can weigh the positions only linearly.

Proposition 1. *Let $m \geq 2$, $\mathbf{h} = (h_1, \ldots, h_m)$ with $h_i \in \mathbb{R}^{d'}$, $\bar{h} = m^{-1}\sum_i h_i$, and let $g_1, g_2 : \mathbb{R}^{d'} \to \mathbb{R}$ be differentiable.* (a) *If $g_1(\bar{h}) = m^{-1}\sum_i g_2(h_i)$ for all $\mathbf{h}$, then $g_1 = g_2$ and both are affine.* (b) *For $U = g_1(\bar{h})$, $\partial U/\partial h_j = m^{-1}\nabla g_1(\bar{h})$ for every j; for $U = m^{-1}\sum_j g_2(h_j)$, $\partial U/\partial h_j = m^{-1}\nabla g_2(h_j)$.*

Proof. (a) Differentiating the identity with respect to h_1 gives $\nabla g_1(\bar{h}) = \nabla g_2(h_1)$ for all $\mathbf{h}$. Fix h_1 and vary h_2: $\bar{h}$ ranges over all of $\mathbb{R}^{d'}$ while the right-hand side is fixed, so ∇g_1 is constant and g_1 is affine. Setting $h_1 = \cdots = h_m = u$ in the identity gives $g_2(u) = g_1(u)$. (b) is the chain rule. □

Part (b) is the gradient statement: through a head on the mean, every position's feature is pushed along one direction, the head's slope at $\bar{h}$; through a head per position, position j is pushed along the head's slope at its own reading. The proposition holds the features fixed; in the critic they are computed by a transformer that attends over the whole state, so neither order limits what the critic can represent as a function of the state. The two critics also share their population target: in expectation over the states at step k and their outcomes, both terms of (11) are minimised by the position-independent field $f_\phi(h_i) = \operatorname{logit} V_k^\star(s_k)$, whose pooled value is the minimiser of (6) as well. Their difference is therefore one of parametrisation and of behaviour under joint training, which is how Table 1 and Figure 3 report it.

B.5 Token commitment along the trajectory

For each generated position we record the first step at which the estimate $\mathbf{x}_{t_k}$ decodes to the final token and never changes afterwards. On Countdown, 86% of positions commit at the first denoiser call (median the first call, 90th percentile the third), identically for $\gamma \in \{4, 8, 14\}$; the remaining contested positions commit uniformly along the trajectory. Most word decisions are therefore made in the first forward pass and carried by the self-conditioned estimate. This statistic describes the decoded tokens, not the coordinates' influence on the reward (Appendix B.2).

C METHOD DETAILS

This appendix holds the parts of the method that the main text states briefly: the choices behind the pathwise estimator and the critic (Sections 4.2 and 4.3), the design of the joint training (Algorithm 1), and the amortization loop in full. Notation is that of the main text: $s_k = (\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})$, the denoiser d_θ, the sampler coefficients $\lambda_k, \eta_k, \psi_k$ of (2), the reward $R(s_{K+1})$ of the decoded response, and $\mathcal{G}$ for the set of generated positions.

C.1 THE PATHWISE ESTIMATOR AND THE CRITIC

One transition. The estimator of $\mathcal{J}_{\mathrm{SVG}}$ draws one transition k uniformly from a window $\mathcal{W} = \{k_{\mathrm{lo}}, \dots, k_{\mathrm{hi}}\}$ of the sampler, with $1 \le k_{\mathrm{lo}}$ and $k_{\mathrm{hi}} \le K-1$, and differentiates the critic through that transition alone:

$$\hat{g}_{\mathrm{PW}} = \nabla_\theta V_\phi(s_k), \qquad k \sim \mathcal{U}(\mathcal{W}), \qquad s_k = \big(\lambda_k \mathbf{z}_{t_{k-1}} + \eta_k\, \mathrm{d}_\theta(s_{k-1}) + \psi_k \epsilon_k,\ \mathrm{d}_\theta(s_{k-1})\big),$$

with s_{k-1} and ϵ_k held fixed (eq. 23). It is computed by rolling the sampler to s_{k-1} without gradients and taking the denoiser call at s_{k-1} with gradients, so its memory is that of the backward passes through the actor's call (two evaluations under classifier-free guidance) and through the critic, independent of K: the dependence of the reward on s_k through the remaining steps is carried by the value in expectation rather than by back-propagation through them (Appendix B.3). The window is the middle of the trajectory, $\mathcal{W} = [K/8,\ 5K/8]$ on every task: it excludes the first steps, where the state is noise and the value flat, and the last, where the state is committed and the reward of the decoded response is piecewise constant; its ends are hyperparameters, not derived. With a learned critic the update is biased by the difference between the critic's slope and the true value's slope on the states the current policy visits, which is why the critic is refit on those states after every actor step (Algorithm 1). The update has no baseline, advantage or trust region: a baseline is a function of s_{k-1}, which the differentiated call holds fixed, and so drops from a pathwise gradient, and the policy's drift is controlled by the step size and by the refit of the critic.

Training target. No intermediate labels are needed. States are taken from rollouts of the sampler together with the rollout's reward $R(s_{K+1})$, and the critic is trained by binary cross-entropy, (6) for the 1D critic and (11) for the 2D critic; the minimiser of either over all functions of the state is the success probability $V_k^\star(s_k) = p_\theta(R(s_{K+1}) = 1 \mid s_k)$. *Pre-training.* Before RL the critic is trained on rollouts of the task-SFT policy, with states stored at several steps spanning the trajectory and each state labelled by its rollout's reward; the copy of the denoiser and the head f_ϕ are trained together for a fixed number of epochs, and the epoch is chosen by a held-out score of the critic (Appendix D). The rollouts are drawn under the training sampler, or, when its early states carry no information about the outcome, under a cheaper sampler of the same denoiser, whose states the critic reads at their own time. *During RL* the copy is frozen and only the head is trained. This is a cost and stability trade-off: refitting the copy after every actor step would cost a full fine-tuning step and would move the actor's objective between actor steps, whereas the head is a small correction on fixed features that a few steps refit.

Per-position dependence is not restricted to be affine. With the features held fixed, if a head on the mean feature could be written as a mean of per-position readings, it would be affine (Proposition 1(a)): a head on the mean weighs the positions only linearly, through the single bottleneck $\bar{h}$; a head applied per position may depend on each reading through any function, for instance one that saturates when a reading is confident.

C.2 DESIGN CHOICES OF THE JOINT TRAINING

Algorithm 1 refits the head after every actor step. Its remaining choices are the following trade-offs.

Where the head is refitted. Each iteration rolls out a fresh batch of prompts to the end of the trajectory, decodes and verifies them, and pushes their state at the top of the window, $s_{k_{\mathrm{hi}}}$, with the reward into a replay buffer; the head is then trained on minibatches from the buffer with (11) (its first term for the 1D critic). The refit is made at one step only, the top of the window, where the outcome is most determined by the state, so that the head follows the policy's changes rather than the residual randomness of the outcome. The critic at the earlier steps of the window is the pre-trained one,

corrected only through what is shared, the head and its time input, on features the joint phase does not change. This is a design choice for the stability of the refit, not a consequence of the objective, and its cost is that the check below tests the critic at k_{hi} only.

Staleness of the buffer. The buffer holds the newest $N_{\mathcal{B}}$ states, pushed B per iteration, so it spans $N_{\mathcal{B}}/B$ iterations of past policies and the head is fitted to a mixture of their values. This is a second approximation, made for the sake of the sample size of the refit; with the values of Appendix D the span is 512 iterations, a third to a half of a run, so the head is fitted to a genuine mixture of past policies rather than to the current one. What makes it usable is that the actor's step is small and that the check below is made on the newest states only.

The critic stays ahead of the actor. A fixed number m_{c} of head steps per actor step suffices while the policy moves slowly. Every n_q iterations the critic's discrimination is measured on the newest N_q buffer states by the AUC, the area under the ROC curve of V_ϕ as a score for the reward (the probability that a random successful state scores above a random failed one). If it falls below a floor a_0, the critic takes m'_{c} extra steps before the actor moves again. The AUC at k_{hi} is a proxy for a necessary condition, that the critic still separates outcomes under the current policy, not a test of slope agreement across the window.

Step size. The bias of a learned critic is paid at every actor step in proportion to the step, which motivates a decreasing step; the schedule itself, a cosine decay to a tenth of the initial value, is a choice. The critic's learning rate is constant.

One sampler. The value function is defined by the sampler: a different sampler (γ, K) induces different trajectories and different values, so critic and actor are trained under the sampler that will be deployed, with one exception: on Countdown the critic's *pre-training* uses states of the cheap sampler, because the deployment sampler's early and middle states carry almost no information about the outcome (Appendix D); the joint refit and the actor still run on the deployment sampler. Appendix C.3 also uses the critic to rank and nudge candidates on a cheaper sampler; that use rests on the assumption that a state's reading, at its own time, ranks candidates on that sampler too, which the training argument does not cover.

C.3 The amortization loop

The value and the critic are defined by the sampler under which the policy was trained, and the RL improvement is made along that sampler's trajectories; a cheaper sampler, with fewer steps or $\gamma = 0$, runs different trajectories through the same denoiser, and an improvement along one need not carry over to the other. The loop moves the improvement into the model's supervised objective, which trains the denoiser at every time t rather than along one sampler, and uses the critic of CFRL-SVG-2D a second time, to steer generation before the verifier can.

One round. Let $p_\theta(\mathbf{w} \mid \mathbf{c})$ be the distribution of decoded responses under the sampler that draws the round's samples. A round takes a pool $\mathcal{P}$ of prompts without labels, disjoint from the evaluation prompts, draws n responses per prompt, keeps the verified ones, and fine-tunes the model on them:

$$\mathcal{A}(\theta) = \Big\{(\mathbf{c}, \mathbf{w}^{(j)}) :\ \mathbf{c} \in \mathcal{P},\ 1 \le j \le n,\ R(\mathbf{w}^{(j)}) = 1\Big\}, \qquad \mathbf{w}^{(1)}, \dots, \mathbf{w}^{(n)} \overset{\text{iid}}{\sim} p_\theta(\cdot \mid \mathbf{c}), \quad (26)$$

$$\theta' = \text{the point reached by descending } \mathcal{L}_{\mathrm{SFT}}\big(\vartheta; \mathcal{A}(\theta)\big) \text{ from } \vartheta = \theta, \quad (27)$$

where $\mathcal{A}(\theta)$ is first reduced in place, by removing duplicate responses of a prompt and keeping at most M per prompt (the reduced set is the one (27) and Algorithm 2 use), the descent runs for a fixed number of epochs, and $\mathcal{L}_{\mathrm{SFT}}$ is ELF's training loss on the accepted pairs, the objective that produced the task-SFT model (Hu et al., 2026): for $(\mathbf{c}, \mathbf{w}) \in \mathcal{A}$ with clean embeddings $\mathbf{x}$ of $\mathbf{w}$, a random time t and noise ϵ on the generated positions, $\mathbf{z}_t = (1-t)\epsilon + t\mathbf{x}$ on $\mathcal{G}$ with the prompt positions held at $\mathbf{c}$,

$$\mathcal{L}_{\mathrm{SFT}}(\vartheta; \mathcal{A}) = \mathbb{E}_{(\mathbf{c},\mathbf{w})\in\mathcal{A}}\Big[\,\mathbb{E}_{t,\epsilon} \sum_{i \in \mathcal{G}} \big\|\mathrm{d}_\vartheta(\mathbf{z}_t, t, \mathbf{x}_{\mathrm{prev}}, \mathbf{c})[i] - \mathbf{x}[i]\big\|^2 - \sum_{i \in \mathcal{G}} \log p_\vartheta(w_i \mid \mathbf{x})\Big], \quad (28)$$

the regression of the clean embeddings on the generated positions plus the cross-entropy of the decoder on the clean embeddings. The self-conditioning input $\mathbf{x}_{\mathrm{prev}}$ is, as in ELF's training, the network's own estimate from a preceding call without gradient for half of the examples and zero

for the rest, and the prompt is dropped at random for guidance training; both are kept as in the base training. Rounds are chained: $\theta^{(\varrho+1)}$ is (27) applied to $\theta^{(\varrho)}$ for rounds $\varrho = 0, \ldots, n_{\mathrm{r}} - 1$, and the final round's model is the result (Algorithm 2).

What a round does. A round is rejection-sampling fine-tuning (Yuan et al., 2023; Singh et al., 2023; Zelikman et al., 2022) with the RL policy as the sampler: the accepted responses of a prompt are samples from the policy conditioned on success, and fitting the model to them raises the likelihood of successful responses under the model's own training objective. Two properties of (28) are the design reasons for using it rather than a sampler-specific loss: it is the objective the model was built with, so a round is ordinary fine-tuning, and it trains the denoiser at all times t, which is the mechanism by which the improvement reaches every sampler rather than only the one that produced the samples. A round gives no direct signal on a prompt with no accepted response, so its coverage, the fraction of the pool with at least one success, bounds what the round can teach directly. This is where the critic enters.

Critic-guided sampling. The 2D critic is a function of partial states, so it can rank and improve candidates before they are decoded. Both operations use the per-position readings $r_{k,i}$ of (10) through the score

$$R_{\min}(s_k) = \min_{i \in \mathcal{G}} r_{k,i}, \tag{29}$$

the most pessimistic of the $|\mathcal{G}|$ readings of the outcome. The minimum is used here, and the mean in training, because the two tasks differ: the actor improves the policy on average and needs a gradient at every position, whereas a single candidate is accepted only if the whole response is right, so what limits it is its least confident reading, and a gradient of the minimum acts on that reading. *Pruning:* for each prompt, n_{g} trajectories are run in parallel to a step k_p; the trajectory with the largest $R_{\min}(s_{k_p})$ continues alone. *Guidance:* at every stochastic step $k = 1, \ldots, K-1$ (it needs $\gamma > 0$), before the denoiser call, the re-noised embeddings are moved along the gradient of the score on the generated positions,

$$\mathbf{z}_{t_k} \;\leftarrow\; \mathbf{z}_{t_k} + \kappa\, \psi_k c_\sigma \sqrt{L}\, \frac{\nabla_{\mathbf{z}} R_{\min}(s_k)}{\|\nabla_{\mathbf{z}} R_{\min}(s_k)\|}, \tag{30}$$

a step whose length is proportional to the noise scale $\psi_k c_\sigma$ of that step, a fraction $\kappa\sqrt{L/(|\mathcal{G}|d)}$ of the norm $\psi_k c_\sigma \sqrt{|\mathcal{G}|d}$ of the noise the sampler injects there, so that the pushed state stays within the spread of the sampler. The gradient with respect to the state in (30) is the same kind of quantity as the critic's slope that the actor multiplies with the denoiser Jacobian in (23); guidance applies it to the sample at inference instead of to the parameters at training. A guided response costs n_{g} trajectories up to step k_p, one afterwards, and one backward pass through the critic per step; guided pools are therefore drawn with a cheap sampler (small K and γ), and the search is meant to compensate the cheap sampler's lower accuracy. Ranking states of a sampler other than the critic's training sampler is an assumption of this use (Appendix C.2). Verification and fine-tuning are unchanged.

Algorithm 2 Amortization loop (with optional critic-guided sampling)

Require: policy $\theta^{(0)}$; optional critic V_ϕ of CFRL-SVG-2D; verifier R; prompt pool $\mathcal{P}$; per round: a sampler and whether guidance is used; responses per prompt n; cap M; rounds n_{r}; guidance constants $(n_{\mathrm{g}}, k_p, \kappa)$

1: **for** round $\varrho = 0, \ldots, n_{\mathrm{r}} - 1$ **do**
2: **for** each prompt $\mathbf{c} \in \mathcal{P}$ and $j = 1, \ldots, n$ **do**
3: Run the round's sampler under $\theta^{(\varrho)}$; with guidance, apply (30) at every step and prune $n_{\mathrm{g}} \to 1$ at step k_p by (29); decode $\mathbf{w}^{(j)}$
4: **end for**
5: $\mathcal{A} \leftarrow$ the verified, distinct responses, at most M per prompt (eq. 26)
6: $\theta^{(\varrho+1)} \leftarrow$ descend $\mathcal{L}_{\mathrm{SFT}}(\vartheta; \mathcal{A})$ from $\vartheta = \theta^{(\varrho)}$ for the round's epochs (eq. 27)
7: **end for**
8: **return** $\theta^{(n_{\mathrm{r}})}$

D EXPERIMENTAL DETAILS

Base models. The released-model rows of Table 1 evaluate each released checkpoint with the task prompt and no task training: GPT-2 (124M), LLaDA-8B-Instruct (256 generated tokens), and

ELF-B's OpenWebText checkpoint under the deployment sampler. ELF-B (Hu et al., 2026): 105M-parameter transformer denoiser, frozen T5-small (Raffel et al., 2020) encoder embeddings (d=512), x-prediction with self-conditioning and training-time classifier-free guidance, PyTorch port of the released code. A denoiser call d_θ combines a prompt-conditioned and an unconditioned evaluation, $\mathbf{x}_{t_k} = \mathbf{x}_\mathrm{u} + \omega(\mathbf{x}_\mathrm{c} - \mathbf{x}_\mathrm{u})$, with guidance weight $\omega = 2$ in every rollout and evaluation (Appendix A); the critic's copy is evaluated without guidance. Supervised fine-tuning uses the released recipe (AdamW, learning rate 2×10^{-3} with warm-up, global batch 256–512, EMA of weights; EMA decay 0.99 for the short Game24 run and 0.999 otherwise). The Countdown base is epoch 60 of a 100-epoch run; Sudoku uses the converged model (76 epochs on 400k puzzles); K&K the epoch-4 checkpoint of a 40-epoch schedule stopped at epoch 9; Game24 the epoch-20 checkpoint of a 40-epoch run initialised from the Countdown model.

Critic. The critic reuses the denoiser as a feature extractor: a copy of the whole network, fine-tuned during pre-training (learning rate 10^{-5}) and frozen during RL, reads the state $s_k = (\mathbf{z}_{t_k}, \mathbf{x}_{t_{k-1}})$ at the time of the corresponding denoiser call, exactly as the sampler's call does, and returns the position features $\mathbf{h} = (h_1, \ldots, h_L)$ of Section 4.3. The MLP head f_ϕ (d+1 $\to$ 256 $\to$ 64 $\to$ 1, GELU; the extra input is the time) produces the per-position readings $r_{k,i}$ of (10), pooled over the generated positions. The 1D critic is the same network with the head applied to the mean of the features (eq. 9), pre-trained on the same rollouts and stored steps with the same epoch budget and selection (kept epochs 10, 3, 10 and 6 on Countdown, Sudoku, K&K and Game24; the 2D heads keep epochs 9, 1, 10 and 6) and trained jointly with the same settings; the two rows of Table 1 differ in nothing else. Pre-training: rollouts of the task-SFT policy with states stored at several denoising steps and the verifier reward, the BCE loss (6) or (11), batch 64, and the epoch with the best held-out field score kept (Table 5); the field score is the mean over the stored steps of half the Pearson correlation between the minimum per-position reading and the reward plus half the within-prompt rank correlation between the pooled value and the reward (for the 1D critic, which has no per-position readings, the pooled value replaces the minimum). The states are sampled under the deployment sampler except on Countdown, where states of the deployment sampler at the early and middle steps carry almost no information about the outcome (held-out AUC about 0.6 for $k \leq 24$ of 64) and the critic is pre-trained on states of the cheap γ4@16 sampler instead; the choice is made by the held-out field score.

Table 5: Critic pre-training per task (Algorithm 1, Require).

task	states from	rollouts	stored steps	epochs (kept)
Countdown	γ4@16	64k	1, 2, 4, 6, 8, 10, 12, 14 of 16	12 (9)
Sudoku	γ28@32	32k	1, 6, 12, 18, 24 of 32	12 (1)
K&K	γ56@64	32k	1, 8, 16, 24, 32, 40, 48 of 64	10 (10)
Game24	γ56@64	16k	1, 8, 16, 32, 48 of 64	6 (6)

CFRL-SVG-1D and CFRL-SVG-2D (Algorithm 1). 32 prompts per iteration, deployment-sampler rollouts without gradients, one random step in the window $\{8, \ldots, 40\}$ of 64 ($\{4, \ldots, 20\}$ of 32) recomputed with gradients, loss $-V_\phi(s_k)$ averaged over prompts, AdamW on the actor with gradient clipping at 1.0, learning rate 10^{-6} decayed on a cosine to 10^{-7} over the run (1500 iterations on Countdown and Game24, 1000 on Sudoku and K&K). Replay buffer of 16,384 states, filled each iteration from 32 fresh prompts rolled out to the end of the trajectory (separately from the actor's rollouts) with the state at the top of the window (k=40 of 64; 20 of 32) and its outcome; 2 critic steps per iteration on minibatches of 64, updating the head at learning rate 10^{-4} with the copy of the denoiser frozen, loss (11); every 10 iterations the pooled value's AUC (area under the ROC curve) against the outcome is computed on the newest 256 buffer states, and if it is below 0.9 the critic takes 8 extra steps. In the symbols of Algorithm 1: T=1500 (Countdown, Game24) or 1000 (Sudoku, K&K), B=32, B'=64, $N_\mathcal{B}$=16,384, m_c=2, m'_c=8, n_q=10, N_q=256, a_0=0.9, $\mathcal{W} = \{8, \ldots, 40\}$ of 64 ($\{4, \ldots, 20\}$ of 32), $\eta_\theta = 10^{-6} \to 10^{-7}$, $\eta_\phi = 10^{-4}$. Checkpoints are saved every 250 iterations (CFRL-GRR: every 100) and evaluated as described under *Evaluation* below. The 1D critic has no per-position readings and minimises (6). The cheap-sampler row of Table 3 and its reference use 1000 iterations at a constant learning rate of 10^{-6}, a buffer of the states at

steps $8, 16, 24, 32, 40$ of 64, the pooled loss term only and no AUC check. Runs take 45–90 minutes for 1000 iterations on one H200 (2–3 hours on an L40).

CFRL-GRR. Group size G=8, 16 prompts per iteration (128 rollouts), $\gamma 4$@16 rollouts, group-normalised advantage as in (5), no KL penalty, no ratio clipping (the clipped variants are in Tables 7 and 8), learning rate 3×10^{-6}, 600 iterations, checkpoints every 100 iterations, evaluated as under *Evaluation* below. The sum over denoising steps in (5) is estimated on 8 transitions drawn uniformly at random per update, with uniform weight over steps and positions.

AR control. GPT-2 (124M) (Radford et al., 2019) is fine-tuned on each task's training set (batch 128, learning rate 3×10^{-4}, warm-up 500; 10 epochs, 20 on Sudoku), decoded greedily and scored by the same verifiers as the flow model: Countdown 44.2% (epoch 8), Sudoku 1.2% (23 of 2,000 puzzles at epoch 20), K&K 92.9% (epoch 10), Game24 26.9% (epoch 2, initialised from the Countdown-fine-tuned weights, as the flow model is). Token-level GRPO/PPO uses 32 prompts $\times$ 8 samples per iteration, clip 0.2, no KL, 2 inner epochs, 600 iterations evaluated every 50, learning rate 10^{-5} on Countdown, Sudoku and Game24 and 3×10^{-6} on K&K, where the supervised policy is already sharp; dropout is disabled during RL, as in standard PPO implementations, so that the two forward passes that form the ratio agree; the log importance ratio is clamped to ± 10 as a numerical safeguard. Curves: Appendix E.1.1.

Amortization loop. The loop of Table 4 runs 4 rounds on Countdown and 5 on K&K (Table 13 adds one Sudoku round; Game24 has no pool of unlabelled problems, since it is split by number set). In each round the current policy samples its pool at the deployment sampler (Countdown: a disjoint unlabelled pool of 50k problems, n=4 samples per problem; Game24, for the check only: the 4,693 other orderings of the training sets, n=8; K&K: the 50k training problems in round 1 and a fresh 50k pool in every later round, n=2; Sudoku: 50k fresh puzzles, n=2), verifier-accepted samples are kept with at most four (Countdown, Game24) or one (unique-solution tasks) per problem, and the policy is fine-tuned on them with the supervised recipe (20 epochs on Countdown, 8 elsewhere) from the previous round's final checkpoint; the tables report the final checkpoint of the last round. Starting policies: a 2D-critic checkpoint from the runs of Table 1, the SFT base, and a CFRL-GRR checkpoint. The gold-label control starts from the 2D-critic policy and fine-tunes, with the same rounds and schedule, on the gold labels of the same pools instead of the policy's samples (one gold derivation per problem; on K&K the round-1 pool is the 50k training problems with their labels). On Countdown, rounds 3–4 of the 2D-policy loop sample the pool with the critic-guided sampler of Appendix C.3: the cheap sampler $\gamma 4$@16, n_g=4 trajectories per prompt pruned at k_p=8 by the score (29), and the push (30) with κ=5 at every step, all with the jointly trained 2D critic of the starting policy; this restores the pool's coverage (Table 13). The controls have no critic and sample plainly in every round.

Evaluation. Every number is one evaluation of the held-out prompts (2,000 per task, 1,601 for Game24) at classifier-free guidance 2. The held-out prompts are split by index into a validation half (even indices) and a test half (odd indices): checkpoints are selected by their validation accuracy at the deployment sampler (CFRL-GRR: every 100 of 600 iterations; the SVG variants: every 250; LLaDA: every 960 inner steps; ties to the earlier checkpoint) and every reported accuracy is measured on the test half.

Rollout budgets. CFRL-GRR: 16 prompts $\times$ 8 samples $= 128$ rollouts per iteration under $\gamma 4$@16 (16 denoiser calls each), 600 iterations, about 1.2M denoiser calls per run. Algorithm 1: 32 actor rollouts and 32 buffer rollouts per iteration under the deployment sampler (64 calls each; 32 on Sudoku), 1,000–1,500 iterations, about 4–6M denoiser calls per run (2M on Sudoku) plus one differentiable call per actor rollout; CFRL-GRR is thus the cheaper of the two by three to five times in denoiser calls (under two on Sudoku), and it gains little under either sampler (Appendix E.1.2). GPT-2's token-level GRPO/PPO: 32 prompts $\times$ 8 samples per iteration, 600 iterations.

Compute. The campaign used about 300 GPU-hours on A100 (40/80 GB), H200 and L40 nodes; a single pathwise run costs about one GPU-hour on an H200, a loop round 15–40 minutes.

E ADDITIONAL RESULTS

E.1 MAIN RESULT: SUPPORTING RESULTS

This group stands behind Table 1: the learning curves of the autoregressive reference (Appendix E.1.1), CFRL-GRR read under both samplers (Appendix E.1.2), and the recipe of the masked-diffusion reference (Appendix E.1.3).

E.1.1 AUTOREGRESSIVE REFERENCE: LEARNING CURVES

Table 6 gives the curves behind the GPT-2 rows of Table 1: the supervised stage by epoch and token-level GRPO/PPO by iteration. Token-level RL climbs steadily on Countdown and Game24, much of it within the first 100 iterations; on K&K, where the supervised policy is already sharp, it holds; on Sudoku, where the supervised model solves about one puzzle in a hundred, the reward is too sparse to move it. The autoregressive policy sees the same problems and verifier as the flow policies and at least as many rollouts per iteration; what differs is the action space the estimator acts on.

Table 6: AR control (GPT-2, 124M), held-out accuracy (%) with greedy decoding. Top: supervised fine-tuning by epoch (Sudoku: epochs 2, 4, ..., 20, scored by the SFT script's solved-grid metric; the verifier gives 1.2 at epoch 20). Bottom: token-level GRPO/PPO by iteration (32 prompts $\times$ 8 samples per iteration). Bold: the highest cell of each row.

SFT epoch	1	2	3	4	5	6	7	8	9	10
Countdown	26.5	30.8	38.1	38.8	41.1	43.9	42.4	**44.2**	43.6	42.8
Sudoku (epochs 2–20)	0.6	0.9	1.1	1.3	1.3	1.4	1.4	1.4	**1.7**	1.7
K&K	28.2	46.6	68.7	82.8	87.1	88.5	91.4	92.3	92.8	**92.9**
Game24 (init. from the Countdown model)	19.6	**26.9**	22.1	21.3	19.9	21.7	21.0	20.8	20.7	21.6
Game24 (init. from vanilla GPT-2)	1.1	9.8	11.4	**12.4**	9.0	9.5	10.3	11.2	9.7	9.7
RL iteration	0	100	150	200	300	400	500	550	600	
Countdown, 10^{-5}	44.2	52.3	55.2	54.8	57.8	59.0	60.6	61.3	**61.7**	
Sudoku, 10^{-5}	1.2	1.1	1.4	**1.6**	1.6	1.5	1.6	1.4	1.6	
K&K, 3×10^{-6}	92.9	93.0	**93.2**	93.1	92.9	92.8	92.7	92.8	92.7	
Game24, 10^{-5}	26.9	39.8	40.2	42.5	45.8	44.2	42.8	**46.2**	46.0	

E.1.2 CFRL-GRR UNDER ITS OWN SAMPLER AND UNDER THE DEPLOYMENT SAMPLER

CFRL-GRR is trained under the 16-step $\gamma 4@16$ sampler (the cheap sampler except on Sudoku) and, like every row of Table 1, read under the deployment sampler. Two checks show that this mismatch is not what limits it. Read under its own sampler, the Countdown runs gain about one point over the base. Trained under the deployment sampler instead, with the same recipe otherwise, CFRL-GRR gains $+0.1$ on Countdown and $+3.5$ on Game24. The estimator gains little under either sampler.

E.1.3 THE MASKED-DIFFUSION REFERENCE: D1 ON OUR TASKS

d1 (Zhao et al., 2025) is the most used RL recipe for masked diffusion language models: LLaDA-8B-Instruct (Nie et al., 2025) with LoRA, a supervised stage, and diffu-GRPO, a GRPO variant whose per-token log-probabilities come from one forward pass in which all completion tokens are masked. That estimator requires a mask token and a discrete masked-diffusion model, so it cannot be applied to a continuous flow model; CFRL-GRR is its counterpart on ELF. We ran d1's own recipe, unchanged, on its own base model and on our four held-out sets, with the authors' code and hyperparameters (LoRA rank 128 on all attention and MLP projections, six generations per prompt, 256-token completions with 128 diffusion steps, block length 32, learning rate 3×10^{-6}, 12 inner iterations, KL weight 0.04, clip 0.5), prompts and rewards in d1's own formats, and accuracy as our exact-success metric. The supervised stage is completion-only masked SFT on the same training data as ELF-B's task SFT (3–20 epochs per task); diffu-GRPO then runs from the merged SFT model for 8,000 inner steps, about 32k rollouts, the budget of our pathwise runs. The comparison is across architectures and training histories: LLaDA-8B-Instruct was never trained on these tasks and its supervised stage is a LoRA run, whereas ELF-B is trained in full, so we make no claim

about the ceiling of masked diffusion models. What Table 1 shows is the behaviour of the estimator: diffu-GRPO moves the supervised model by under three points on every task, consistent with d1's own results on its easier three-number Countdown and 4×4 Sudoku.

E.2 THE ESTIMATOR

This group supports the estimator paragraph of Section 5: the credit-assignment variants of the score-function estimator (Appendix E.2.1), the parts of GRPO put back one at a time (Appendix E.2.2), the two estimators on the same rollouts (Appendix E.2.3), and one step along each gradient (Appendix E.2.4).

E.2.1 SCORE-FUNCTION VARIANTS ON COUNTDOWN

Table 7 lists the credit-assignment variants of the score-function estimator that we ran on Countdown before moving to the pathwise estimator. All keep the objective (5) and change only the advantage or the ratio: *sparse credit* restricts the step sum to the last n transitions; the *oracle advantage* replaces the reward by the verifier's verdict on the greedy decode of every intermediate state, so that each transition is credited by the change in a privileged value; the *advantage grid* gives every transition and position its own advantage from the change of a per-position value between successive states; *per-position PPO ratios* reuse each batch with a clipped ratio per position; and *critic-selected* advantages keep only the rollouts that a jointly trained critic ranks first in their group. The variants differ in which coordinates receive weight; none removes the noise factor of (22), and none lifts the estimator past $+5$ points, where the autoregressive reference with the same estimator gains $+15.8$ (Table 1).

Table 7: Score-function variants on Countdown: best change in held-out accuracy (%) over the base under the deployment sampler. No variant clears $+5$; the autoregressive reference with the same estimator gains $+15.8$.

variant	what changes	best Δ at $\gamma56$@64
CFRL-GRR, uniform credit over the steps	group baseline	$+2.8$
credit to the last n steps only	sparse credit	$+1.6$
oracle advantage (verifier on every intermediate decode)	exact advantage	$+3.1$
per-step $\times$ per-position advantage grid	dense credit	$+3.9$
per-position PPO ratios, clip 0.2	clipped ratios	$+5.0$
jointly trained critic selects advantages (pick@V_ϕ = 1.0)	critic-selected	$+4.6$
GPT-2 (124M) + token-level GRPO/PPO	same estimator, token actions	$+15.8$

E.2.2 GRPO WITH RATIO, CLIPPING AND ANCHOR

CFRL-GRR takes one optimizer step per batch of rollouts and has no KL anchor; at the parameters that drew the batch every importance ratio equals 1, so the first step of clipped GRPO with $\beta = 0$ coincides with a step on (5). Table 8 puts the parts of the original GRPO objective back, on the rollout budget of Table 1: two steps per batch with the ratio taken per denoising step and position (which the diagonal covariance of (21) allows), then the clip at $\epsilon = 0.2$, then the anchor at $\beta = 0.04$ on the closed-form KL of the Gaussian transitions. The second step and the clip add a few points on Game24 and K&K and under a point on Countdown; the anchor at its usual strength takes the clipped two-step form back to about the level of CFRL-GRR. With either row the order of Table 1 is unchanged: the 2D critic stays ahead of the clipped two-step form on every task.

Table 8: GRPO variants on the flow model, held-out accuracy (%) under the deployment sampler. All rows draw the same rollouts per iteration for the same number of iterations as the CFRL-GRR row of Table 1.

	steps per batch	ratio	clip	anchor	Countdown	Game24
base					42.2	45.0
CFRL-GRR	1	—	—	—	43.1	46.8
ratio only	2	yes	—	—	42.6	50.1
ratio and clip	2	yes	0.2	—	44.7	51.1
GRPO, ratio, clip and anchor	2	yes	0.2	0.04	43.2	46.0

	Countdown	Sudoku	K&K	Game24
CFRL-GRR	43.1	69.7	71.6	47.5
ratio and clip, two steps	43.9	68.0	74.1	50.0

E.2.3 The two estimators on the same rollouts: SNR and half-split cosine

The top right panel of Table 1 asks, on one set of Game24 rollouts of the base under the deployment sampler and with one differentiable denoiser call as in Algorithm 1, how well the mean gradients of two random halves of the rollouts agree. The ruler is the $A{=}1$ control, whose true mean is zero. The flow model's score-function gradient is indistinguishable from it under every form of the advantage up to $2{,}048$ rollouts; GPT-2's token-level gradient looks the same at 256 rollouts and then separates as N grows, the signature of a weak but genuine per-sample signal (the estimator is the same, the action space is not); the pathwise gradient through the same denoiser call agrees with itself already at 256 rollouts, with an SNR nearly three times the floor. Table 9 gives the numbers; the derivation of the two gradients is in Appendix B.2.

Table 9: The score-function (22) and pathwise (23) estimators on Game24 rollouts of the task-SFT policy at denoising step $k{=}24$ of 64 (256 rollouts for the SNR, up to 2,048 for the half-split cosine; pathwise term with the pre-trained 2D critic). SNR: norm of the mean per-sample gradient over its root-mean-square deviation; a zero-mean estimator sits at $1/\sqrt{N} = 0.0625$, the $A{=}1$ row. Half-split cosine: agreement between the mean gradients of two disjoint halves of the rollouts (mean$\pm$sd over 10 splits) as the number of rollouts grows; a genuine signal rises with N, a zero-mean estimator stays at 0. Last two rows: the same statistic for GPT-2's token-level GRPO on Game24 (the panel's line) and on Countdown. Bold: the pathwise estimator.

	SNR, $N{=}256$	half-split cosine		
		$N{=}256$	$N{=}1024$	$N{=}2048$
score-function, mean-centred A	0.061	-0.03 ± 0.03	-0.04 ± 0.03	0.00 ± 0.04
score-function, GRPO A (groups of 4)	0.065	0.03 ± 0.03	0.02 ± 0.03	-0.03 ± 0.03
score-function, $A{=}1$ (zero-mean control)	0.063	0.02 ± 0.02	0.04 ± 0.02	0.02 ± 0.03
pathwise, 2D critic	**0.177**	$\mathbf{0.79 \pm 0.03}$	$\mathbf{0.94 \pm 0.01}$	$\mathbf{0.97 \pm 0.00}$
GPT-2, token-level GRPO (Game24)	—	0.09 ± 0.05	0.35 ± 0.04	0.34 ± 0.07
GPT-2, token-level GRPO (Countdown)	—	0.05 ± 0.05	0.17 ± 0.05	0.27 ± 0.02

E.2.4 One step along each gradient

The half-split cosine says that the pathwise gradient is reproducible, not that it points the right way. The lower right panel of Table 1 tests the direction by intervention at iteration 0 of RL, with the task-SFT base and the pre-trained 2D critic and no training. We draw $1{,}024$ rollouts of 128 training prompts under the deployment sampler and compute, on the same rollouts and the same four denoising steps of the window per rollout, the mean score-function gradient of (5) and the mean pathwise gradient of $V_\phi(s_k)$. The policy is then moved once, $\theta' = \theta + \eta\, g/\|g\|$, and evaluated on the held-out prompts under the deployment sampler with the sampler's random stream fixed, so that every θ' sees the same prompts and the same noise as the base. The length η is measured against Λ, the parameter distance that a 2D-critic run of Table 1 travels from the base; each gradient

is applied as AdamW applies a first step (global clipping at 1, then $g/(|g| + \epsilon)$ per coordinate) and, for comparison, raw; a Gaussian random direction is the control. Table 10 gives the paired change in accuracy. In the form the optimizer uses, one pathwise step gains about four points on Game24 at a quarter to a half of Λ and three to four on Countdown from a half of Λ on, with intervals that exclude zero; the score-function step and the random step stay within about a point of the base at every length. The raw gradients are a different matter: over 0.9 of the squared norm of either gradient lies in one percent of the coordinates, so a raw step of any useful length moves a few coordinates far outside the region where the critic was fitted and the raw pathwise step destroys the policy; the per-coordinate normalisation of AdamW is part of what makes the pathwise update work. On Game24 the gain turns over at $\eta{=}\Lambda$: a single direction from a critic that is not refit cannot be followed as far as training goes.

Table 10: One step from the base along each mean gradient of $1{,}024$ rollouts. Each cell is the paired change in held-out accuracy (points) against the base under the deployment sampler with the sampling noise fixed; brackets: 95% bootstrap interval over prompts. *AdamW*: the direction of a first AdamW step of that gradient; *raw*: the gradient itself; all steps have length η.

η/Λ	pathwise, AdamW	score-function, AdamW	random	pathwise, raw	score-function, raw
Game24 (base 43.0, $\Lambda = 1.30$ over 750 iterations, 1601 prompts)					
$1/256$	$+0.0$ [-0.5, +0.5]	$+0.2$ [-0.2, +0.7]	$+0.5$	$+0.5$	$+0.1$
$1/128$	$+0.1$ [-0.5, +0.7]	$+0.3$ [-0.2, +0.9]	$+0.3$	-0.2	$+0.1$
$1/64$	$+0.0$ [-0.7, +0.7]	$+0.2$ [-0.2, +0.7]	$+0.7$	-0.6	$+0.3$
$1/32$	$+0.9$ [+0.0, +1.9]	$+0.8$ [+0.2, +1.5]	$+0.0$	-1.3	$+0.4$
$1/16$	$+1.0$ [-0.2, +2.1]	$+0.9$ [+0.2, +1.7]	$+0.4$	-2.1	$+0.4$
$1/4$	$+4.2$ [+2.4, +5.9]	$+1.2$ [-0.1, +2.5]	$+0.4$	-13.1	$+0.3$
$1/2$	$+4.4$ [+2.2, +6.6]	$+1.6$ [-0.1, +3.4]	$+0.2$	-43.0	-2.1
1	-1.2 [-3.7, +1.3]	$+0.1$ [-1.9, +2.2]	$+0.2$	-43.0	-2.1
Countdown (base 42.2, $\Lambda = 1.39$ over 1500 iterations, 2000 prompts)					
$1/256$	-0.1 [-0.9, +0.5]	$+0.1$ [-0.7, +0.7]	$+0.2$	-0.5	-0.2
$1/128$	$+0.1$ [-0.6, +0.9]	-0.1 [-0.9, +0.5]	$+0.3$	$+0.1$	$+0.2$
$1/64$	-0.1 [-0.8, +0.7]	-0.1 [-0.8, +0.6]	$+0.3$	-0.3	-0.1
$1/32$	$+0.2$ [-0.5, +1.1]	-0.1 [-1.0, +0.7]	$+0.4$	-0.9	$+0.2$
$1/16$	$+0.7$ [-0.4, +1.7]	$+0.4$ [-0.8, +1.5]	$+0.6$	-0.9	$+0.0$
$1/4$	$+2.6$ [+1.0, +4.3]	-0.1 [-1.7, +1.4]	-0.4	-6.7	-0.4
$1/2$	$+3.3$ [+1.4, +5.3]	$+0.2$ [-1.8, +2.1]	$+0.0$	-36.7	-0.7
1	$+3.9$ [+1.6, +6.2]	-2.1 [-4.5, +0.1]	-0.4	-42.2	-4.1

E.3 The critic and the joint training

This group supports the critic paragraphs of Section 5: how the critic's value develops along the trajectory and over positions (Appendices E.3.1 and E.3.2), the two critics over training on all tasks (Appendix E.3.3), the measurements showing that they are interchangeable at a fixed policy (Appendix E.3.4), the frozen critic (Appendix E.3.5), and the training dynamics of the reference runs (Appendix E.3.6).

E.3.1 Critic quality along the trajectory

Table 11 reports, for the pre-trained 2D critic of each task, how well the pooled value V_ϕ of (10) at a *noisy* state at step k predicts the final reward of the trajectory it lies on: the Pearson correlation between V_ϕ and the reward over held-out rollouts, and pick@V_ϕ, the top-1 accuracy of selecting a successful sample among a group of 8 rollouts of the same prompt by ranking on V_ϕ (only mixed groups count; the random baseline is the success rate of a random pick in such groups). The profile is the same on every task: near chance at $k{=}1$, informative from roughly a quarter of the trajectory, and essentially an oracle in the last third. This is the token-commitment profile of the sampler (Appendix B.5); it is the reason the pathwise step is drawn from the middle of the trajectory rather than from its start, where the critic has no gradient to give, or its end, where the state is already committed. For Countdown and Game24 the per-position AUC of $r_{k,i}$ against oracle step labels is also reported; the K&K and Sudoku verifiers do not produce per-position labels.

Table 11: Pre-trained 2D critics on held-out rollouts of the base policy (deployment sampler; Countdown critic trained on $\gamma 4$@16 states). pick@V_ϕ over groups of 8 (random baseline in the header); Pearson correlation of the pooled value with the 0/1 reward.

task (random)	metric	flow step k				
Countdown (0.32), 16 steps	k	1	4	8	14	
	pick@V_ϕ	0.35	0.56	0.97	1.00	
	Pearson	0.14	0.33	0.91	0.98	
	token AUC	0.22	0.27	0.46	0.47	
Sudoku (0.53), 32 steps	k	1	8	16	24	
	pick@V_ϕ	0.55	0.89	0.96	0.98	
	Pearson	0.48	0.85	0.92	0.97	
K&K (0.53), 64 steps	k	1	8	16	32	48
	pick@V_ϕ	0.58	0.74	0.84	0.95	0.99
	Pearson	0.45	0.70	0.82	0.94	0.97
Game24 (0.53), 64 steps	k	1	8	16	32	48
	pick@V_ϕ	0.57	0.61	0.80	0.96	0.99
	Pearson	0.44	0.60	0.76	0.94	0.98
	token AUC	0.71	0.70	0.71	0.72	0.73

E.3.2 PER-POSITION CRITIC VALUES AS A CREDIT MAP

The 2D critic produces one reading per generated position at every step of the trajectory. Figure 4 shows $r_{k,i}$ for held-out rollouts of the jointly trained critics on Game24 and Sudoku. Two things are visible. First, the pooled value separates successful from failed trajectories progressively along the trajectory: the critic has learned when the trajectory commits. Second, the per-position readings carry some localisation but are dominated by the sequence-level signal: on Sudoku, ranking all cells of 96 held-out samples by $r_{k,i}$ separates wrong from right cells well, but most of that comes from wrong samples being uniformly low; restricted to the cells within incorrect samples the AUC is 0.56–0.63 at every k. On Game24 the maps are organised by equation, but they are not a step verifier. The per-position readings serve as a ranking and guidance score (Appendix C.3); the actor ascends their pooled value.

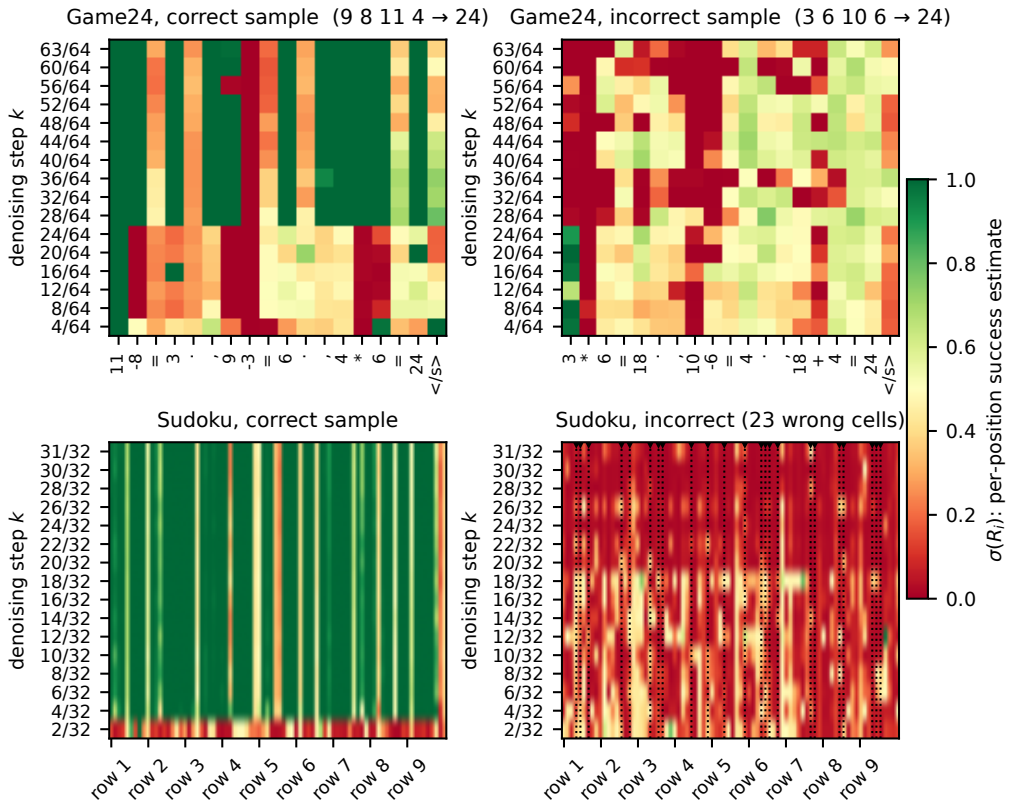


Figure 4: Per-position critic readings $r_{k,i}$ (colour) along the deployment-sampler trajectory (rows: denoising step k; columns: generated positions, labelled by the decoded token for Game24 and by Sudoku row) for one correct and one incorrect held-out sample per task. The correct samples are the median-valued correct rollout; the incorrect Game24 sample is the median-valued failed rollout and the incorrect Sudoku sample the failed rollout with the fewest wrong cells (dotted lines and markers).

E.3.3 1D AGAINST 2D CRITIC OVER TRAINING, ALL TASKS

Figure 5 extends the left panel of Figure 3 from Game24 to the four tasks and adds the AUC of the jointly trained head. On Game24 and K&K both heads keep up with the policy and the 2D critic's policy climbs further; on Countdown the pooled head falls behind the moving policy and its policy turns back; on Sudoku both heads keep their AUC while both policies turn back early, the 2D critic's later and from a higher peak.

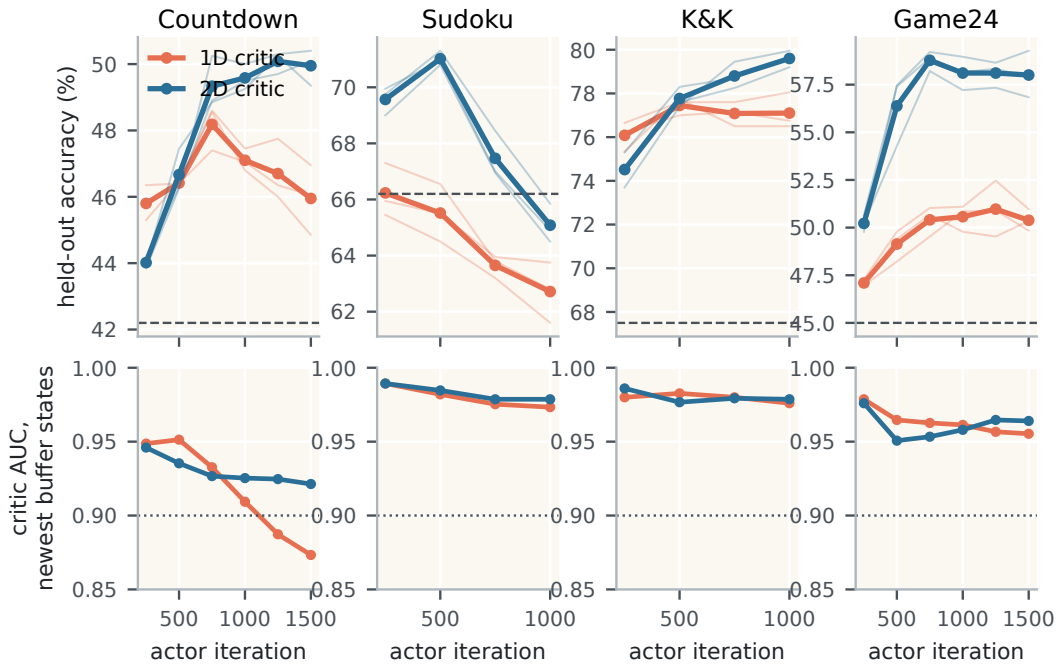


Figure 5: 1D against 2D critic over training: the runs of Table 1, individual runs (thin) and their mean (thick). Top: held-out accuracy (%) at the deployment sampler of the checkpoint after that many actor iterations; dashed: the SFT base. Bottom: AUC of the jointly trained head against the outcome on the newest 256 replay-buffer states, averaged over 250-iteration windows of the training logs; dotted: the floor $a_0 = 0.9$ below which Algorithm 1 gives the critic extra steps. Bottom-row curves are means over the runs. On Countdown the pooled head falls behind the moving policy and its policy turns back; on Sudoku both heads keep their AUC and the 1D critic's policy declines from the start; on Game24 and K&K both heads keep up and the 1D critic's gain is smaller.

E.3.4 The 1D and the 2D critic are interchangeable at a fixed policy

Table 12 compares the pre-trained 1D and 2D critics of the Table 1 runs on the same states of the same policy, and their jointly trained heads on fixed probe sets. Measured this way the two critics discriminate the reward equally well, prescribe parameter updates at cosine 0.83–0.90 of each other, react equally, and weakly, to a single wrong token, and their heads refit equally stably. None of the fixed-policy measurements separates the two critics: whatever distinguishes the policies they train (Figures 3 and 5) is not visible in either critic's value, slope or stability at a fixed policy, which is why the main text attributes the difference to the joint training rather than to what the critics can represent.

Table 12: 1D against 2D critic on the same states. Pre-trained critics unless a row says otherwise; k is the denoising step of the state.

measurement	setting	1D	2D
Pearson correlation of the pooled value with the outcome, held-out rollouts	pre-training, Game24 k=48 / Sudoku k=18 / Countdown k=14	0.97 / 0.94 / 0.99	0.97 / 0.94 / 0.99
pathwise SNR of the actor's parameter gradient (Table 9 statistic)	Game24 k=16 / 24 / 32	0.19 / 0.17 / 0.20	0.18 / 0.18 / 0.15
half-split cosine of the actor's parameter gradient	Game24 k=16 / 24 / 32	0.81 / 0.78 / 0.83	0.81 / 0.79 / 0.70
cosine between the two critics' mean actor gradients	Game24 k=16 / 24 / 32	0.90 / 0.88 / 0.83	
cosine between the two critics' per-position gradient-norm profiles	Game24, real tokens	0.94 / 0.94 / 0.94	
mean pairwise cosine of per-position state gradients $\partial V_\phi / \partial \mathbf{z}_{t_k}[i]$ (shared direction)	Game24 k=24 / Sudoku k=12	0.03 / 0.06	0.04 / 0.07
pooled ΔV_ϕ (logits) for one wrong token in a successful state	Game24 k=24, re-encoded	-0.86	-0.89
the same, corruptions that break the answer	Game24 k=24	-1.76	-2.04
pooled ΔV_ϕ for one wrong cell (never changes the decoded grid)	Sudoku k=12	-0.11	-0.03
jointly trained head: AUC on states of its own peak policy, iteration 0 $\rightarrow$ end	Game24 / Sudoku / Countdown	0.97$\rightarrow$0.96 / 0.98$\rightarrow$0.98 / 0.90$\rightarrow$0.92	0.96$\rightarrow$0.96 / 0.98$\rightarrow$0.98 / 0.90$\rightarrow$0.91
jointly trained head: AUC on the newest buffer states, mean over training	Game24 / Sudoku / K&K / Countdown	0.96 / 0.98 / 0.98 / 0.92	0.96 / 0.98 / 0.98 / 0.93
actor gradient after 20 head refits: cosine to the gradient before them	Game24 k=32, buffer displaced 5%/refit	0.98	0.99

E.3.5 Frozen versus joint critic

The right panel of Table 2 shows the exploit directly on Game24: the recipe of Algorithm 1 with the pre-trained 2D critic held fixed. The pathwise update first helps, then the actor finds the critic's blind spots: the critic's value on the training batch climbs from about 3 logits at iteration 250 to over 80 by 1500 while the true success rate of the same batch falls from 78% to 0%, and the held-out accuracy collapses. The identical run with the head refit after every actor step (Table 1) keeps the critic's value in the calibrated range. Increasing the frozen critic's quality (more pre-training rollouts, the 2D form) does not change this: whatever the critic is, the actor exploits it within a few hundred updates unless the critic is refreshed on the actor's own states.

E.3.6 Training dynamics on Countdown and K&K

Figure 6 shows the Algorithm 1 runs with the 2D critic on the two tasks not covered by Figure 3. On Countdown the held-out accuracy rises for about 750 iterations and then holds while the replay-buffer success rate keeps climbing: the training batch keeps improving after the held-out set has levelled off. On K&K the held-out accuracy rises monotonically and is still at its peak when the schedule ends; its buffer success rate and critic value rise together over the whole run.

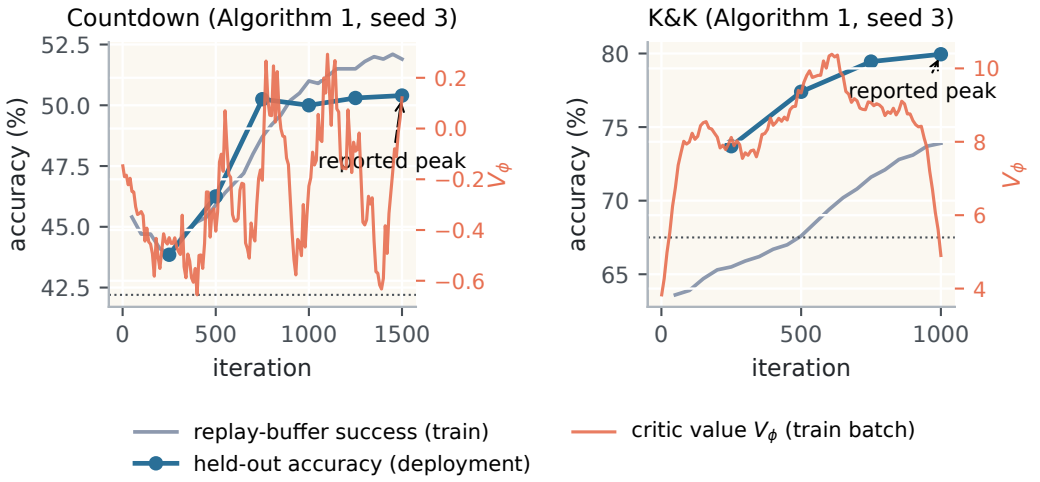


Figure 6: Training dynamics of Algorithm 1 (2D critic) on Countdown (left) and K&K (right): held-out accuracy at the deployment sampler per checkpoint (blue), replay-buffer success rate (grey), and the critic's value on the training batch (red, right axis); dotted line: SFT base.

E.4 The loop

This group stands behind Table 4: every round of every loop with its controls, and the critic-guided sampler that draws the later Countdown rounds.

E.4.1 The loop in full: rounds, controls and the guided sampler

Table 13 gives every round behind Table 4 with the pool's coverage and the controls. On Countdown and K&K the rounds lift the cheap sampler by 17 to 35 points from every start, RL or not. On K&K fresh puzzles give near-full coverage from any start and the rounds close the task; RL adds nothing there that the rounds do not. On Sudoku one round covers three quarters of the pool from every start and moves accuracy by at most 1.4 points at either sampler, as the gold-solution control confirms. On Countdown two plain rounds shrink coverage and give back part of the RL gain; rounds 3 and 4, drawn with the critic-guided cheap sampler, raise coverage past its start and return the deployment sampler to the RL level, 5 to 7 points above sampling plainly from the base or the CFRL-GRR policy. Two further plain rounds continued from the round-2 model (the two control rows after round 4) leave coverage and accuracy at the round-2 level, whereas the critic-guided rounds from the same model cover about six thousand more problems and recover 4.5–5.9 points at deployment; the recovery is therefore due to the critic-guided pools, not to the additional rounds. Table 14 isolates the guided sampler on the first $5{,}000$ pool problems with the loop's starting policy and its jointly trained critic at $\gamma 4@16$: pruning four trajectories at random changes nothing; pruning them by the critic's minimum reading (29) doubles the acceptance rate and raises coverage by a third, and the push (30) adds to both.

Table 13: The amortization loop, held-out accuracy (%) at three samplers. Each block is one task: the SFT base, the 2D-critic policy the loop starts from, the model at the end of each round of Algorithm 2 (Countdown: all four rounds; K&K: first and last of five; Sudoku: its one round), and, indented, the controls at their last round, which run the same rounds without a critic from the SFT base, from the CFRL-GRR policy, and on the gold derivations of the same pool, and, for Countdown, two further plain rounds continued from the round-2 model. *Covered*: pool problems (of 50k) with at least one verified sample in that round. Game24 has no pool of unlabelled problems. Sudoku's cheap sampler is $\gamma 14@16$ and its deployment sampler $\gamma 28@32$. Bold: the last round of each loop where it is over 1.0 point above the start.

	pool sampler	covered	16-step sampler	ODE-64	deployment
Countdown					
SFT base			23.8	28.5	42.8
2D-critic policy (start)			24.3	34.8	50.5
round 1	deployment sampler	35.1k	37.9	42.4	45.8
round 2	deployment sampler	31.2k	40.8	42.6	46.0
round 3	critic-guided, $\gamma 4@16$	37.2k	43.6	46.3	50.0
round 4	critic-guided, $\gamma 4@16$	37.8k	**45.6**	**47.8**	51.4
plain round 3 (control)	deployment sampler	31.2k	41.1	43.7	45.5
plain round 4 (control)	deployment sampler	31.3k	41.1	43.4	45.5
control: from the SFT base	deployment sampler	31.2k	42.7	44.5	46.4
control: from CFRL-GRR	deployment sampler	30.9k	41.1	42.9	44.7
control: gold derivations	(labels)	50.0k	22.5	26.4	40.8
K&K					
SFT base			58.6	57.5	68.2
2D-critic policy (start)			68.8	69.7	79.8
round 1	deployment sampler	41.1k	73.4	76.6	82.1
round 5	deployment sampler	48.8k	**93.8**	**95.7**	**99.1**
control: from the SFT base	deployment sampler	49.4k	93.5	97.0	99.6
control: from CFRL-GRR	deployment sampler	48.3k	91.7	94.6	97.9
control: gold labels	(labels)	50.0k	94.8	98.0	99.6
Sudoku					
SFT base			60.0	60.6	64.0
2D-critic policy (start)			63.4	65.3	68.9
round 1	deployment sampler	39.3k	63.1	65.4	**70.3**
control: from the SFT base	deployment sampler	36.1k	58.7	60.7	63.6
control: from CFRL-GRR	deployment sampler	39.3k	64.2	67.7	69.6
control: gold solutions	(labels)	50.0k	63.9	65.2	69.7

Table 14: The critic as a search heuristic. Each row is one way of drawing samples from the Countdown 2D-critic policy the loop starts from (Table 13) and its jointly trained critic at $\gamma 4@16$ on the first 5,000 pool problems, four samples per problem: *acc.*, the fraction of samples the verifier accepts; *coverage*, problems (of 5,000) with at least one accepted sample; *distinct*, distinct accepted derivations.

sampler	acc. (%)	coverage	distinct
plain	25.4	2,783	4,978
best of 4 at step 8, random	25.2	2,748	4,960
best of 4 at step 8, by the critic	52.8	3,712	10,204
push (30) at every step	28.4	3,069	5,647
best of 4 by the critic + push	**59.3**	**3,933**	**11,753**

F Related work

Actor-critic control. Stochastic value gradients (Heess et al., 2015) differentiate a learned value through a reparameterised policy and a transition model; DDPG and SAC (Lillicrap et al., 2015; Haarnoja et al., 2018) are the model-free cases. The SVG variants of CFRL are this pattern on

a generative sampler whose transition model is the sampler itself, with a value that is a success probability fitted to verifier rewards and, in CFRL-SVG-2D, a reading per position.

Self-training with verifiers. STaR, RFT, ReST-EM and related work (Zelikman et al., 2022; Yuan et al., 2023; Singh et al., 2023; Huang et al., 2022) fine-tune a model on its own verified generations; expert iteration (Anthony et al., 2017) alternates a stronger search-based policy with distillation. The amortization loop is this pattern with an RL policy as the sampler and, in its guided form, with the RL critic as the search heuristic; the controls in Appendix E.4.1 separate what the starting policy contributes from what the rounds contribute. DAV (Lee et al., 2026) casts diffusion alignment as variational expectation-maximisation; a round of Algorithm 2 can be read as the M-step of that view for one prompt, with the RL policy as the sampler.

Tasks. Countdown and Game24 follow Gandhi et al. (2024) and Yao et al. (2023); Sudoku follows Shah et al. (2024); Knights-and-Knaves follows Xie et al. (2024).